ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ (ΑΕΠΠ)

Σημειώσεις Θεωρίας

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ (ΑΕΠΠ)

Σημειώσεις Θεωρίας

Δημιουργία - Συγγραφή

Costas Chatzinikolas
www.CostasChatzinikolas.gr
info@CostasChatzinikolas.gr

Τελευταία Ενημέρωση: 09 Φεβρουαρίου 2018

Οδηγίες

Τα θέματα θεωρίας που περιέχει το παρόν έγγραφο, αποτελούν το μεγαλύτερο τμήμα της θεωρίας που ανήκει στην εξεταστέα ύλη των πανελληνίων εξετάσεων, όπως αυτή είχε δοθεί από το Υπουργείο Παιδείας για το σχολικό έτος 2017 - 2018.

Όταν θα έχετε διαβάσει και εμπεδώσει ήδη πολύ καλά τη θεωρία που περιέχεται στο παρόν έγγραφο και επιθυμείτε να δώσετε μεγαλύτερη βαρύτητα στις λεπτομέρειες της θεωρίας, μπορείτε να κατεβάσετε και να διαβάσετε το έγγραφο με τίτλο **Κρυμμένα Θέματα Θεωρίας**, στη διεύθυνση www.CostasChatzinikolas.gr/aepp.html για μια πιο άρτια προετοιμασία.

Σημειώστε τα θέματα στα οποία αντιμετωπίσατε κάποια δυσκολία και επικοινωνείστε μαζί μου (www.CostasChatzinikolas.gr) για διευκρινίσεις.

ΚΑΛΗ ΠΡΟΕΤΟΙΜΑΣΙΑ

Εξεταστέα Ύλη (2017 - 2018)

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ (ΑΕΠΠ)

Από το βιβλίο «Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον» της Γ΄ τάξης Γενικού Λυκείου των Α. Βακάλη, Η. Γιαννόπουλου, Ν. Ιωαννίδη, Χ.Κοίλια, Κ. Μάλαμα, Ι. Μανωλόπουλου, Π. Πολίτη, έκδοση (Ι.Τ.Υ.Ε.) “Διόφαντος".

2. Βασικές Έννοιες Αλγορίθμων
2.1 Τι είναι αλγόριθμος.
2.3 Περιγραφή και αναπαράσταση αλγορίθμων.
2.4 Βασικές συνιστώσες/ εντολές ενός αλγορίθμου.
2.4.1 Δομή ακολουθίας.
2.4.2 Δομή Επιλογής.
2.4.3 Διαδικασίες πολλαπλών επιλογών. (αφαιρείται η εντολή πολλαπλής επιλογής "Επίλεξε")
2.4.4 Εμφωλευμένες Διαδικασίες.
2.4.5 Δομή Επανάληψης.

3. Δομές Δεδομένων και Αλγόριθμοι
3.2 Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα
3.3 Πίνακες
3.6 Αναζήτηση
3.7 Ταξινόμηση

6. Εισαγωγή στον προγραμματισμό
6.3 Φυσικές και τεχνητές γλώσσες.
6.4 Τεχνικές σχεδίασης προγραμμάτων.
6.4.1 Ιεραρχική σχεδίαση προγράμματος.
6.4.2 Τμηματικός προγραμματισμός.
6.4.3 Δομημένος προγραμματισμός.
6.7 Προγραμματιστικά περιβάλλοντα.

7. Βασικά στοιχεία προγραμματισμού.
7.1 Το αλφάβητο της ΓΛΩΣΣΑΣ.
7.2 Τύποι δεδομένων.
7.3 Σταθερές.
7.4 Μεταβλητές.
7.5 Αριθμητικοί τελεστές.
7.6 Συναρτήσεις.
7.7 Αριθμητικές εκφράσεις.
7.8 Εντολή εκχώρησης.
7.9 Εντολές εισόδου-εξόδου.
7.10 Δομή προγράμματος.

8. Επιλογή και επανάληψη
8.1 Εντολές Επιλογής
8.1.1 Εντολή ΑΝ
8.2 Εντολές επανάληψης
8.2.1 Εντολή ΟΣΟ...ΕΠΑΝΑΛΑΒΕ
8.2.2 Εντολή ΜΕΧΡΙΣ - ΟΤΟΥ
8.2.3 Εντολή ΓΙΑ...ΑΠΟ...ΜΕΧΡΙ

9. Πίνακες
9.1 Μονοδιάστατοι πίνακες.
9.2 Πότε πρέπει να χρησιμοποιούνται πίνακες.
9.3 Πολυδιάστατοι πίνακες.
9.4 Τυπικές επεξεργασίες πινάκων.

10. Υποπρογράμματα

10.1 Τμηματικός προγραμματισμός.
10.2 Χαρακτηριστικά των υποπρογραμμάτων.
10.3 Πλεονεκτήματα του τμηματικού προγραμματισμού.
10.4 Παράμετροι.
10.5 Διαδικασίες και συναρτήσεις.
10.5.1 Ορισμός και κλήση συναρτήσεων.
10.5.2 Ορισμός και κλήση διαδικασιών.
10.5.3 Πραγματικές και τυπικές παράμετροι.
10.6 Εμβέλεια μεταβλητών – σταθερών

Κεφάλαιο 2
Βασικές έννοιες αλγορίθμων

Τι είναι ο αλγόριθμος:
Είναι μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων, οι οποίες εκτελούνται σε πεπερασμένο χρόνο με σκοπό την επίλυση ενός προβλήματος.

Ποια τα κριτήρια που πρέπει να πληροί ένας αλγόριθμος:
1. **Είσοδος:** Καμία, μία ή περισσότερες τιμές δεδομένων πρέπει να δίνονται στον αλγόριθμο ως είσοδοι. Το «καμία» αναφέρεται στην περίπτωση που ο αλγόριθμος δημιουργεί μόνος του τα δεδομένα, π.χ. με γεννήτριες τυχαίων αριθμών.
2. **Έξοδος:** Ο αλγόριθμος πρέπει να εξάγει τουλάχιστον μία τιμή ως αποτέλεσμα προς το χρήστη ή προς άλλο κάποιο πρόγραμμα (π.χ. υποπρόγραμμα)
3. **Καθοριστικότητα:** Για κάθε εντολή δεν πρέπει να υπάρχει καμία αμφιβολία ως προς την εκτέλεση της, π.χ. διαίρεση με το μηδέν.
4. **Αποτελεσματικότητα:** Κάθε εντολή πρέπει, εκτός από καλά ορισμένη, να είναι και απλή, εκτελέσιμη.
5. **Περατότητα:** Πεπερασμένος αριθμός βημάτων (εντολών) και πεπερασμένος χρόνος εκτέλεσης. Διαφορετικά πρόκειται για υπολογιστική διαδικασία.

Τρόποι περιγραφής και αναπαράστασης αλγορίθμων:
1. **Ελεύθερο κείμενο:** ανεπεξέργαστος και αδόμητος τρόπος παρουσίασης σε μορφή έκθεσης (φυσική γλώσσα). Παραβιάζει το κριτήριο της αποτελεσματικότητας.
2. **Φυσική γλώσσα κατά βήματα:** Φυσική γλώσσα με χρονική όμως κατανομή των ενεργειών. Παραβιάζει το κριτήριο της καθοριστικότητας.
3. **Διαγραμματικές τεχνικές:** Γραφικός τρόπος απεικόνισης των βημάτων του αλγορίθμου. Χρησιμοποιείται πλέον σπάνια για εκπαιδευτικούς κυρίως σκοπούς.
4. **Κωδικοποίηση:** Πρόγραμμα γραμμένο σε ψευδογλώσσα ή κάποια γλώσσα προγραμματισμού.

Ποιες είναι οι τρεις προγραμματιστικές δομές:
Οι τρεις προγραμματιστικές δομές είναι:
1. Δομή ακολουθίας
2. Δομή επιλογής
3. Δομή επανάληψης

Αποτελούν και τη βάση του δομημένου προγραμματισμού. Με τη χρήση τους καθώς και συνδυασμών τους μπορούν να κωδικοποιηθούν όλα τα προβλήματα που έχουν λύση στον υπολογιστή.

Τι είναι οι δεσμευμένες λέξεις:
Δεσμευμένες λέξεις ονομάζονται εκείνες οι λέξεις οι οποίες δεσμεύονται από τη γλώσσα προγραμματισμού και χρησιμοποιούνται για μία συγκεκριμένη λειτουργία, π.χ. Αρχή, Τέλος κλπ. Οι δεσμευμένες λέξεις δεν επιτρέπεται να χρησιμοποιούνται ως ονόματα μεταβλητών.

Πότε θεωρείται ως έγκυρο ένα όνομα προγράμματος (αλγορίθμου) ή μεταβλητής:
Ως έγκυρο όνομα προγράμματος (αλγορίθμου) ή μεταβλητής, θεωρείται μία λέξη όταν:
1. Δεν είναι δεσμευμένη λέξη, π.χ. Αν, Αρχή, Όσο κλπ.
2. Δεν είναι ή δεν αρχίζει από αριθμό, π.χ. 3τιμή
3. Δεν έχει ενδιάμεσα κενό, π.χ. ύψος μαθητή. Στην περίπτωση που το όνομα του προγράμματος ή της μεταβλητής πρέπει να αποτελείται από δύο λέξεις, τις ενώνουμε με κάτω παύλα δηλαδή ύψος_μαθητή, ή τις γράφουμε χωρίς κενό με το πρώτο γράμμα της δεύτερης έξης κεφαλαίο δηλαδή ύψοςΜαθητή.

Τι είναι η επικεφαλίδα, το τμήμα δηλώσεων και το κύριο μέρος ή σώμα του προγράμματος:
Η πρώτη σειρά του προγράμματος ονομάζεται επικεφαλίδα, οι μεταβλητές αποτελούν το τμήμα δηλώσεων και ότι υπάρχει μεταξύ των δεσμευμένων λέξεων Αρχή και Τέλος αποτελεί το κύριο μέρος ή σώμα του προγράμματος.

Τι είναι το αναγνωριστικό:
Το όνομα μιας μεταβλητής ονομάζεται και **αναγνωριστικό**, γιατί αναγνωρίζει μία συγκεκριμένη θέση μνήμης.

Σε τι διαφέρει η αναπαράσταση σε ψευδογλώσσα από το διάγραμμα ροής:
Η αναπαράσταση ενός αλγόριθμου σε ψευδογλώσσα δεν διαφέρει σε τίποτα από την αναπαράσταση σε διάγραμμα ροής, εκτός του ότι στο **διάγραμμα ροής δε χρειάζεται να δηλώσουμε τις μεταβλητές που χρησιμοποιούμε**, δηλαδή δεν υπάρχει τμήμα δηλώσεων (το πρόγραμμα ξεκινάει απευθείας με το σώμα – δεσμευμένη λέξη Αρχή).

Υπολογισμός εκφράσεων, μαθηματικών ή λογικών:
Σε αυτές τις ασκήσεις θα πρέπει να προσεχθεί ιδιαίτερα η προτεραιότητα των πράξεων. Οι προτεραιότητες για τους μαθηματικούς και λογικούς τελεστές είναι:

- Μαθηματικοί τελεστές: **^**, [*****, **/**, **div**, **mod**], [**+**, **-**]
- Λογικοί τελεστές: **ΌΧΙ**, [**ΚΑΙ**, **Ή**]

Συγκριτικοί τελεστές:
Χρησιμοποιούνται για το σχηματισμό λογικών εκφράσεων. Είναι οι: **<, >, <=, >=, =, <>**. Σε μία έκφραση που περιέχει συγκριτικό τελεστή, αποτιμάμε την έκφραση αριστερά και δεξιά του συγκριτικού τελεστή και στη συνέχεια κάνουμε τη σύγκριση.

Επιπλέον πρέπει να έχουμε υπόψιν μας τα εξής:

- Για πράξεις που έχουν ίδια προτεραιότητα, η αποτίμηση ξεκινά από **αριστερά**.
- Πάντα προηγείται η αποτίμηση των **συναρτήσεων** και **διαδικασιών**.
- Οι **παρενθέσεις** χρησιμοποιούνται για να αλλάξουμε την προτεραιότητα των πράξεων.
- Εάν σε μία έκφραση υπάρχει **έστω ένας** συγκριτικός τελεστής, π.χ. >= τότε όλη η έκφραση αποτιμάται σαν λογική έκφραση.

Υπολογισμός πίνακα τιμών σε δομή ακολουθίας:
Σχηματίζουμε έναν πίνακα (πρώτα στο πρόχειρο) όπου τοποθετούμε σε στήλες όλες (ανεξαιρέτως) τις μεταβλητές του αλγορίθμου. Ο πίνακας θα έχει τη μορφή:

Βήμα	X	Y	Z	W
1				
2				
3				
4				
5				

- Αφού έχουμε δομή ακολουθίας, θεωρούμε κάθε μία εντολή σαν ένα βήμα.
- Ο πίνακας τιμών θα φτιαχτεί στο πρόχειρο ακόμη και αν δεν είναι το ζητούμενο της άσκησης.
- Συνήθως ζητάνε τι τιμές θα εκτυπωθούν, οπότε θα πρέπει να αναζητήσετε τις εντολές **Γράψε** ή **Εμφάνισε** ή **Εκτύπωσε** να δείτε τι τιμές έχουν οι μεταβλητές εκείνη τη στιγμή και να τις βάλετε σε κύκλο ή να τις γράψετε κάπου ξεχωριστά.

Χρήση των mod και div για διαχωρισμό ψηφίων αριθμού:
Πολλές φορές σε ασκήσεις θα χρειαστεί να:

- χωρίσουμε έναν ακέραιο αριθμό στα επιμέρους ψηφία από τα οποία αποτελείται και να τα αποθηκεύσουμε σε ξεχωριστές μεταβλητές.
- Πάρουμε ένα τμήμα του αριθμού ή κάποιο ψηφίο του που βρίσκεται σε οποιαδήποτε θέση (αρχή, μέση, τέλος)

Σε όλες αυτές τις περιπτώσεις η λύση βασίζεται στην παρακάτω αρχή:

Όταν διαιρούμε έναν οποιοδήποτε ακέραιο με κάποια δύναμη του 10 (10, 100, 1000 κλπ) χρησιμοποιώντας τους τελεστές **mod** και **div**, τότε μπορούμε να φανταστούμε ότι ο αριθμός αυτός χωρίζεται από μία νοητή γραμμή σε δύο τμήματα. Τη γραμμή θα την τοποθετήσουμε τόσες θέσεις από το δεξί άκρο του αριθμού, όσα και τα μηδενικά στη δύναμη του 10. Δηλαδή:

Αν ο αριθμός 86583 διαιρεθεί με **mod** ή **div** με το **100**, τότε η νοητή γραμμή θα τοποθετηθεί **2** ψηφία από το δεξί άκρο του αριθμού, δηλαδή κάπως έτσι:

Διαίρεση με 100 (**mod** ή **div**) --> **865** | **83** (**div** **mod**)

Ανάλογα τώρα με το ποια διαίρεση χρησιμοποιούμε έχουμε και διαφορετικό αποτέλεσμα, δηλαδή:

- Η διαίρεση με το **mod** θα μας δώσει το τμήμα στα **δεξιά της νοητής γραμμής** και…
- Η διαίρεση με το **div** θα μας δώσει το τμήμα **αριστερά της νοητής γραμμής**

π.χ.

865 | **83**

86583 **mod** 100 = 83 και 86583 **div** 100 = 865

Κάνοντας τώρα χρήση αυτής της αρχής, μπορούμε να πάρουμε οποιοδήποτε τμήμα του αριθμού συνδυάζοντας διαιρέσεις με **mod** και **div** και αποθηκεύοντας αν χρειαστεί κάποια αποτελέσματα σε προσωρινές μεταβλητές.

Παράδειγμα: Να χωριστεί ο αριθμός **94578** που περιέχεται στη μεταβλητή x, στα επιμέρους ψηφία του, τα οποία θα αποθηκευτούν αντίστοιχα στις μεταβλητές ψ1, ψ2, ψ3, ψ4, ψ5 έτσι ώστε στο τέλος αυτές να περιέχουν τις τιμές: ψ5 = 8, ψ4 = 7, ψ3 = 5, ψ2 = 4, ψ1 = 9.

```
94578 mod 10 = 8
94578 div 10 = 9457

9457 mod 10 = 7
9457 div 10 = 945

945 mod 10 = 5
945 div 10 = 94

94 mod 10 = 4
94 div 10 = 9
```

Σε εντολές

```
ψ5 ← x mod 10
tmp ← x div 10

ψ4 ← tmp mod 10
tmp ← tmp div 10

ψ3 ← tmp mod 10
tmp ← tmp div 10

ψ2 ← tmp mod 10
ψ1 ← tmp div 10
```

Δομή Επιλογής
Η δομή της επιλογής χρησιμοποιείται όταν το πρόγραμμα που γράφουμε πρέπει να πάρει (όπως λέμε) αποφάσεις. Δηλαδή να εκτελέσει κάποια ενέργεια όταν ικανοποιείται μία συνθήκη ή να εκτελέσει μία άλλη ενέργεια όταν η συνθήκη δεν ικανοποιείται.

Τι εννοούμε όταν λέμε ότι μία συνθήκη ικανοποιείται; Εννοούμε ότι, αν την δούμε ως λογική έκφραση (όπως και πρέπει να είναι), η τιμή της είναι **Αληθής**. Όταν η τιμή της είναι **Ψευδής**, τότε λέμε ότι δεν ικανοποιείται.

Τη συνθήκη της επιλογής εμείς οι άνθρωποι τη χρησιμοποιούμε συνέχεια και συνήθως ασυνείδητα, όταν παίρνουμε κάποιες αποφάσεις που αφορούν την καθημερινότητά μας. Για παράδειγμα συχνά λέμε (ή τουλάχιστον λέω :-)

1. Αν ο καιρός είναι καλός αύριο θα πάω να κολυμπήσω στη θάλασσα.
2. Αν σχολάσω νωρίτερα θα πάω να πληρώσω το λογαριασμό του ΟΤΕ.

Και πολλά άλλα παραδείγματα…

Η δομή της επιλογής στον προγραμματισμό συναντάται σε πολλές μορφές - όλες όμως είναι κατά βάση ίδιες αφού βασίζονται στην πολύ απλή ερώτηση της μορφής «Αν συμβεί κάτι, τότε θα εκτελέσω μία ενέργεια».

Πιο συγκεκριμένα θα συναντήσουμε:

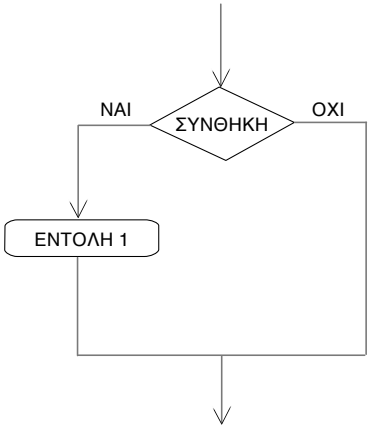
1. **Την απλή επιλογή:**
Εδώ τα πράγματα είναι σχετικά απλά. Όταν ικανοποιείται μία συνθήκη τότε εκτελούμε μία ενέργεια. Σε περίπτωση που δεν ικανοποιείται, αγνοούμε την προς εκτέλεση ενέργεια. Σε κάθε περίπτωση όμως (είτε εκτελέσουμε την ενέργεια είτε όχι), στη συνέχεια προχωράμε στις επόμενες εντολές του προγράμματος.

Η απλή επιλογή συντάσσεται ως εξής:

```
Γράψε 'Δώσε τον αριθμό α'
Διάβασε α
Αν (α > 0) τότε
      Γράψε 'Ο αριθμός', α, 'είναι θετικός'
Τέλος_αν
Γράψε 'Δώσε τώρα τον αριθμό β'
Διάβασε β
…
…
```

Παρατηρήστε ότι δεν κάνουμε τίποτα, όταν το α είναι μηδέν ή αρνητικός αριθμός. Η εντολή που βρίσκεται μέσα στον **Αν … Τέλος_αν** εκτελείται μόνο αν το α είναι θετικός. Σε κάθε περίπτωση πάντως προχωράμε στην επόμενη εντολή, δηλαδή την Γράψε 'Δώσε τώρα τον αριθμό β'.

Διάγραμμα ροής **απλής επιλογής**



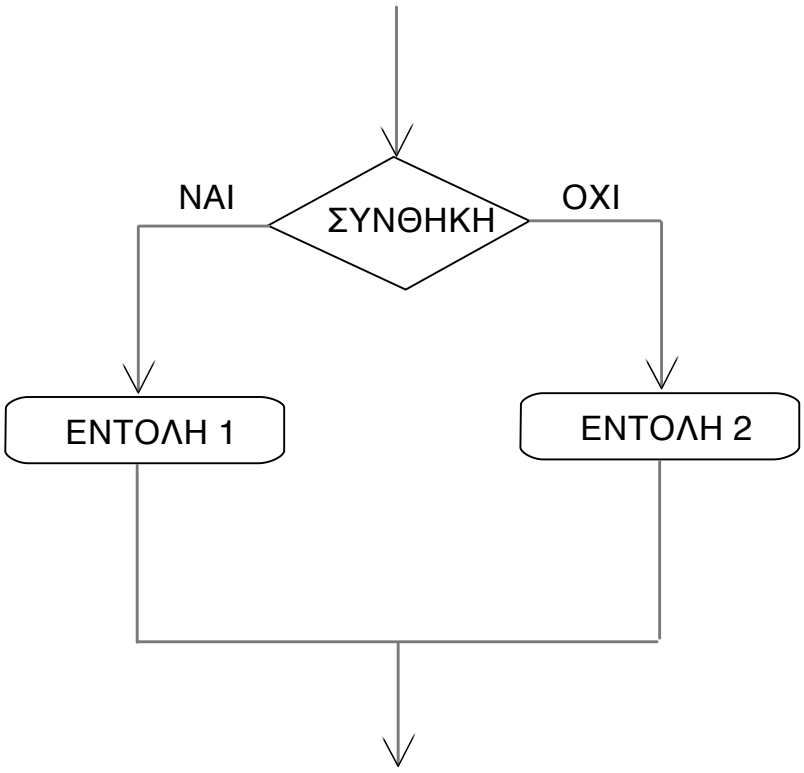
2. **Τη σύνθετη επιλογή:**
Εδώ το πρόβλημα και η επίλυσή του επιβάλλει να εκτελέσουμε κάποια ενέργεια όταν ικανοποιείται μία συνθήκη και αντιστοίχως να εκτελέσουμε μία διαφορετική ενέργεια όταν η συνθήκη δεν ικανοποιείται.

Η σύνθετη επιλογή συντάσσεται ως εξής:

```
Γράψε 'Δώσε τον αριθμό α'
Διάβασε α
Αν (α > 0) τότε
      Γράψε 'Ο αριθμός', α, 'είναι θετικός'
Αλλιώς
      Γράψε Ό αριθμός', α, 'είναι αρνητικός ή μηδέν'
Τέλος_αν
Γράψε 'Δώσε τώρα τον αριθμό β'
Διάβασε β
…
…
```

Παρατηρήστε ότι, <u>όποια και αν είναι η τιμή της συνθήκης (Αληθής ή Ψευδής) κάποια από τις δύο ενέργειες θα εκτελεστεί, πριν προχωρήσουμε στην επόμενη εντολή του προγράμματος. Ποτέ όμως δεν θα εκτελεστούν και οι δύο εντολές και αυτό είναι άλλωστε και το ζητούμενο.</u>

Διάγραμμα ροής **σύνθετης επιλογής**



3. **Την πολλαπλή επιλογή:**

Στη σύνθετη επιλογή, οι περιπτώσεις που έπρεπε να εξετάσουμε ήταν μόνο δύο. Πώς όμως θα αντεπεξέλθει το πρόγραμμά μας σε περιπτώσεις με περισσότερες επιλογές; Για παράδειγμα σκεφθείτε το παρακάτω πρόβλημα:

Πρέπει να διαβάζουμε τον αριθμό των παιδιών ενός δημοσίου υπαλλήλου και να υπολογίζουμε το επίδομα που δικαιούται το οποίο είναι 0 € για κανένα παιδί, 10 € για ένα, 20 € για 2 και 50 € για τρία ή περισσότερα παιδιά.

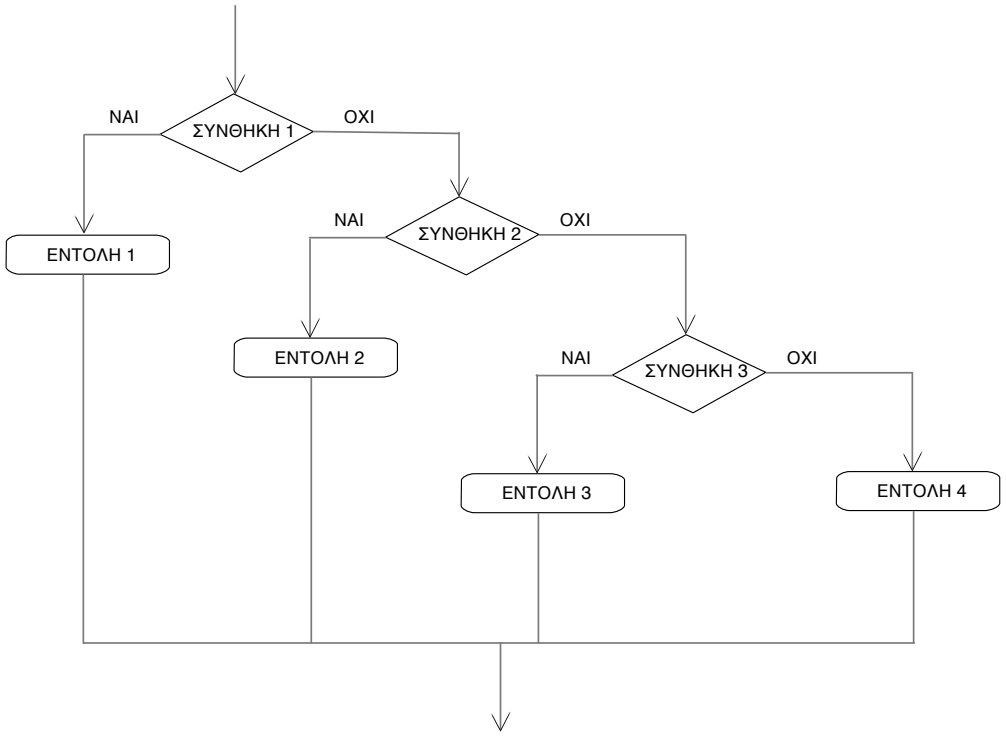
Θα χρησιμοποιήσουμε την εξής δομή:

```
Γράψε 'Δώσε τον αριθμό των παιδιών'
Διάβασε α
Αν (α = 0) τότε
      επ <-- 0
Αλλιώς_αν (α = 1) τότε
      επ <-- 10
Αλλιώς_αν (α = 2) τότε
      επ <-- 20
Αλλιώς
      επ <-- 50
Τέλος_αν
Γράψε 'Το επίδομα που δικαιούται ο υπάλληλος είναι:', επ
```

Παρατηρήστε ότι μόνο μία εντολή θα εκτελεστεί από αυτές που βρίσκονται μέσα στο τμήμα προγράμματος **Αν ... Τέλος_αν**. Στην πρώτη συνθήκη που θα ικανοποιείται, θα εκτελεστεί η ανάλογη εντολή και τότε το πρόγραμμα θα εξέλθει της δομής επιλογής και θα προχωρήσει στις επόμενες εντολές του προγράμματος.

Το ίδιο πρόβλημα θα μπορούσε να λυθεί και με τη χρήση της σύνθετης επιλογής, μπερδεύοντάς την λιγάκι και χρησιμοποιώντας μία σύνθετη επιλογή ενσωματωμένη σε μία άλλη κ.ο.κ. δημιουργώντας αυτή τη δομή που ονομάζουμε εμφωλευμένη επιλογή.

Διάγραμμα ροής **πολλαπλής επιλογής**

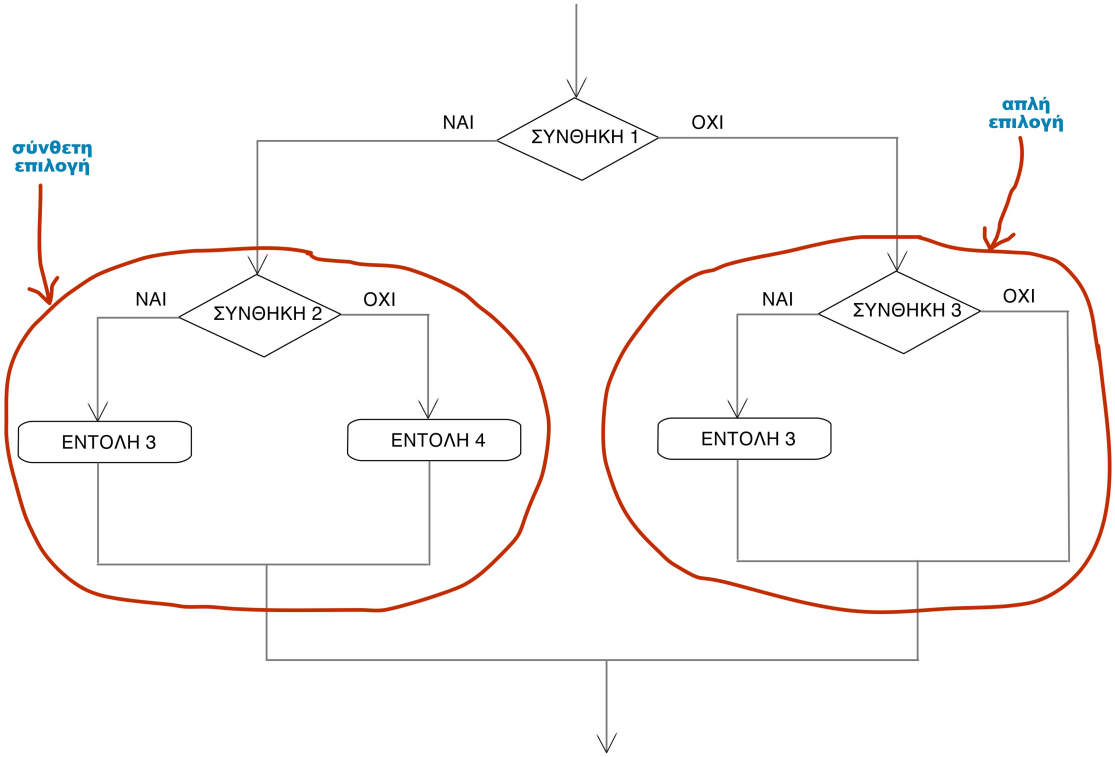


4. **Εμφωλευμένη επιλογή:**
Εδώ θα λύσουμε το προηγούμενο πρόβλημα, χρησιμοποιώντας όμως μόνο σύνθετες επιλογές. Κάθε νέα επιλογή θα ανοίγει μέσα στο τμήμα **Αλλιώς** της προηγούμενης. Προσοχή στις αποστάσεις… θα σας βοηθήσουν να μην κάνετε λάθος.

```
Γράψε 'Δώσε τον αριθμό των παιδιών'
Διάβασε α
Αν (α = 0) τότε
      επ <-- 0
Αλλιώς
      Αν (α = 1) τότε
            επ <-- 10
      Αλλιώς
            Αν (α = 2) τότε
                  επ <-- 20
            Αλλιώς
                  επ <-- 50
            Τέλος_αν
      Τέλος_αν
Τέλος_αν
Γράψε 'Το επίδομα που δικαιούται ο υπάλληλος είναι:', επ
```

Παρατηρήστε ότι κάθε **Αν** που ανοίγει πρέπει να κλείσει. Και εδώ ισχύει ότι στην πρώτη συνθήκη που θα ικανοποιείται, θα εκτελεστεί η ανάλογη εντολή και τότε το πρόγραμμα θα εξέλθει της δομής επιλογής και θα προχωρήσει στις επόμενες εντολές του προγράμματος.

Διάγραμμα ροής **εμφωλευμένης επιλογής**



Εύρεση Μεγίστου - Ελαχίστου (max - min)
Μία βασική χρήση της δομής της επιλογής είναι η εύρεση του μεγίστου (max) ή του ελαχίστου (min) ή και των δύο ταυτόχρονα, από ένας πλήθος τιμών που δίνεται συνήθως ως είσοδος από το πληκτρολόγιο.

Εύρεση του μεγίστου 2 αριθμών
Μία πολύ απλή λύση είναι να συγκρίνουμε τις τιμές μεταξύ τους

```
Γράψε 'Δώσε 2 τιμές'
Διάβασε a, b
Αν (a > b) τότε
      max <— a
Αλλιώς
      max <— b
Τέλος_αν

Γράψε 'Το μέγιστο είναι:', max
```

Εύρεση του μεγίστου περισσοτέρων αριθμών π.χ. 4 αριθμών
Εδώ η προηγούμενη λύση δεν εφαρμόζει καλά, καθώς όπως μπορούμε να δούμε και στον κώδικα που ακολουθεί, οι συνθήκες γίνονται αρκετά πιο μεγάλες και αυξάνει η πιθανότητα λάθους. Επιπλέον η λύση δεν είναι καθόλου κομψή.

```
Γράψε 'Δώσε 4 τιμές'
Διάβασε a, b, c, d
Αν (a > b) και (a > c) και (a > d) τότε
      max <— a
Αλλιώς_αν (b > a) και (b > c) και (b > d) τότε
      max <— b
Αλλιώς_αν (c > a) και (c > b) και (c > d) τότε
      max <— c
Αλλιώς
      max <— d
Τέλος_αν

Γράψε 'Το μέγιστο είναι:', max
```

Χρήση 1 υπόθεσης και απλών συγκρίσεων
Όταν έχουμε να συγκρίνουμε πάνω από 2 αριθμούς, υποθέτουμε ότι ένας οποιοσδήποτε από αυτούς (συνήθως ο πρώτος κατά σειρά) είναι ο μέγιστος και στη συνέχεια συγκρίνουμε διαδοχικά όλους τους υπόλοιπους αριθμούς με το μέγιστο, ψάχνοντας να βρούμε κάποιον μεγαλύτερο.

- Πιο απλή και πιο κομψή μέθοδος.
- Οι συγκρίσεις είναι **αυστηρά απλές επιλογές**. Δεν χρησιμοποιούμε ποτέ **Αλλιώς**.
- Δεν συγκρίνουμε ποτέ 2 αριθμούς μεταξύ τους, αλλά καθε αριθμό με το μέγιστο.

```
Γράψε 'Δώσε 4 τιμές'
Διάβασε a, b, c, d

max <— a          ! Κάνουμε την υπόθεση ότι ο πρώτος κατά σειρά αριθμός,
                  ! δηλαδή ο a είναι ο μέγιστος

Αν (b > max) τότε ! Αν ο b τυγχάνει να είναι μεγαλύτερος από το μέχρι στιγμής
      max <— b    ! μέγιστο, τότε αυτός είναι ο νέος μέγιστος
Τέλος_αν
                  ! Και προχωράμε στην επόμενη σύγκριση, έτσι κι αλλιώς
Αν (c > max) τότε ! Αν ο c τυγχάνει να είναι μεγαλύτερος από το μέχρι στιγμής
      max <— c    ! μέγιστο, τότε αυτός είναι ο νέος μέγιστος
Τέλος_αν
                  ! Και προχωράμε στην επόμενη σύγκριση, έτσι κι αλλιώς
Αν (d > max) τότε ! Αν ο d τυγχάνει να είναι μεγαλύτερος από το μέχρι στιγμής
      max <— d    ! μέγιστο, τότε αυτός είναι ο νέος μέγιστος
Τέλος_αν

Γράψε 'Το μέγιστο είναι:', max
```

Παρόμοια ακριβώς λογική ακολουθούμε όταν έχουμε να βρούμε το ελάχιστο. Το μόνο που αλλάζει είναι ότι στη σύγκριση αντί για **>** χρησιμοποιούμε **<**. Επίσης, το ελάχιστο συνήθως το αποθηκεύουμε σε μεταβλητή με όνομα **min**. Π.χ.

```
Γράψε 'Δώσε 4 τιμές'
Διάβασε a, b, c, d

min <— a
Αν (b < min) τότε
      min <— b
Τέλος_αν

Αν (c < min) τότε
      min <— c
Τέλος_αν

Αν (d < min) τότε
      min <— d
Τέλος_αν

Γράψε 'Το ελάχιστο είναι:', min
```

Στην περίπτωση που μας ζητηθεί να υπολογίσουμε ταυτόχρονα και το μέγιστο και τον ελάχιστο, τότε υπολογίζουμε και τα δύο μαζί, χωρίς να τα μπλέξουμε. Υπολογίζουμε πρώτα το μέγιστο και στη συνέχεια το ελάχιστο. Δεν χρησιμοποιούμε ποτέ **Αλλιώς**. Π.χ.

```
Γράψε 'Δώσε 4 τιμές'
Διάβασε a, b, c, d

! Εύρεση μεγίστου
max <— a
Αν (b > max) τότε
      max <— b
Τέλος_αν
Αν (c > max) τότε
      max <— c
Τέλος_αν
Αν (d > max) τότε
      max <— d
Τέλος_αν
! Τέλος εύρεσης μεγίστου

! ----------------------------

! Εύρεση ελαχίστου
min <— a
Αν (b < min) τότε
      min <— b
Τέλος_αν
Αν (c < min) τότε
      min <— c
Τέλος_αν
Αν (d < min) τότε
      min <— d
Τέλος_αν
! Τέλος εύρεσης ελαχίστου

! Συνολικά αποτελέσματα
Γράψε 'Το μέγιστο είναι:', max, 'και το ελάχιστο είναι:', min
```

Επιπλέον στοιχεία που σχετίζονται με τη δομή της επιλογής:

- Ένας αριθμός Χ, είναι **άρτιος** όταν διαιρείται ακριβώς με το 2, δηλαδή όταν το υπόλοιπο της διαίρεσής του με το 2 είναι 0, δηλαδή: **Αν (Χ mod 2) = 0**
- Ένας αριθμός Χ, είναι **περιττός** όταν δεν διαιρείται ακριβώς με το 2, δηλαδή όταν το υπόλοιπο της διαίρεσής του με το 2 είναι διάφορο του 0 ή όταν είναι 1,
  δηλαδή: **Αν (Χ mod 2) <> 0,** ή **Αν (Χ mod 2) = 1**
- Ένας αριθμός Χ, **διαιρείται ακριβώς** και **είναι ακέραιο πολλαπλάσιο ενός αριθμού** Κ, όταν το υπόλοιπο της διαίρεσης του Χ με το Κ είναι 0, δηλαδή: **Αν (Χ mod Κ) = 0**
- Το **τελευταίο ψηφίο** ενός αριθμού Χ, προκύπτει, αν πάρουμε το **υπόλοιπο** της διαίρεσης του αριθμού αυτού με το 10, δηλαδή **τψ** <-- **Χ mod 10** (όπου τψ = τελευταίο ψηφίο). Παρομοίως, τα δύο τελευταία ψηφία του αριθμού προκύπτουν εάν πάρουμε το υπόλοιπο της διαίρεσης του αριθμού αυτού με το 100, τα τρία τελευταία με το 1000 κ.ο.κ.
- **Όλα τα ψηφία** ενός αριθμού Χ, **εκτός του τελευταίου**, προκύπτουν εάν πάρουμε το **ακέραιο μέρος** της διαίρεσης του αριθμού αυτού με το 10, δηλαδή **ψετ** <-- **Χ div 10** (όπου ψετ = όλα τα ψηφία εκτός του τελευταίου). Παρομοίως, όλα τα ψηφία εκτός των δύο τελευταίων προκύπτουν εάν πάρουμε το ακέραιο μέρος της διαίρεσης του αριθμού αυτού με το 100, κ.ο.κ.

Δομή επανάληψης
Ένας από τους κύριους λόγους που χρησιμοποιούμε τον Η/Υ για την επίλυση προβλημάτων είναι και η επαναληπτικότητα των διαδικασιών. Όταν μία ενέργεια πρέπει να επαναληφθεί 2 ή και παραπάνω φορές, τότε 2 λύσεις υπάρχουν:

1. Να γράψουμε εμείς την εντολή που πρέπει να επαναληφθεί, τόσες φορές όσες είναι και οι επαναλήψεις.
2. Να δώσουμε στον Η/Υ τις κατάλληλες οδηγίες – εντολές ώστε να επαναλάβει μόνος του τις εντολές, τόσες φορές όσες είναι και οι επαναλήψεις που επιθυμούμε. Τις εντολές αυτές εμείς θα τις γράψουμε μία (1) φορά, ο Η/Υ όμως θα τις εκτελέσει πολλές φορές.

Ας δούμε ένα απλό παράδειγμα ενός προγράμματος που πρέπει να εκτυπώσει στην έξοδο 10 φορές τη λέξη “Καλημέρα”. Αν ακολουθήσουμε την πρώτη μέθοδο, τότε θα πρέπει εμείς να επαναλάβουμε (να γράψουμε δηλαδή) τη συγκεκριμένη εντολή 10 φορές. Δηλαδή:

Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”
Γράψε “Καλημέρα”

Θα δουλέψει μια χαρά. Υπάρχουν όμως τρεις προφανείς περιορισμοί:
1. Δεν εκμεταλλευόμαστε την έμφυτη ικανότητα του Η/Υ να επαναλαμβάνει ταχύτατα υπολογισμούς.
2. Αν και το πιο πάνω πρόβλημα είναι εύκολο να το λύσουμε, αυτομάτως θα γινόταν πολύ δύσκολο, αν είχαμε για παράδειγμα να εκτυπώσουμε το ίδιο μήνυμα 1580 φορές :-)
3. Πρέπει εμείς οι ίδιοι να γνωρίζουμε από πριν τον αριθμό των επαναλήψεων, ώστε να επαναλάβουμε το γράψιμο της αντίστοιχης εντολής, τόσες φορές όσες και οι επαναλήψεις.

Η λύση θα ήταν να χρησιμοποιήσουμε τη δεύτερη μέθοδο, η οποία αποτελεί και την πρώτη μορφή εκτέλεσης επαναλήψεων στη γλωσσομάθεια και ονομάζεται δομή επανάληψης **Όσο…Επανάλαβε.**

1. **Δομή Όσο…Επανάλαβε**

Πώς δουλεύει όμως; Τα βήματα είναι τα ακόλουθα:

1. Χρησιμοποιούμε μία μεταβλητή – μετρητή (αποτελεί άγραφο κανόνα να χρησιμοποιούμε την **αγγλική λέξη i** από τον αγγλικό όρο **iterations** ή **iterator**, σαν όνομα για τη μεταβλητή μετρητή), για να δώσουμε στον υπολογιστή έναν αριθμό ο οποίος θα συμβολίζει την αρχή του μετρήματος των επαναλήψεων. Συνήθως εμείς οι άνθρωποι όταν ξεκινάμε μία αρίθμηση π.χ. το μέτρημα των μαθητών μιας τάξης, ξεκινάμε από τον αριθμό 1 και καταλήγουμε στο τελικό πλήθος. Άρα **i <-- 1**.
2. Στη συνέχεια θα πρέπει να πούμε στον υπολογιστή να επαναλαμβάνει την εκτέλεση μιας συγκεκριμένης εντολής (ή μιας ομάδας εντολών) όσο αυτός ο αριθμός – μετρητής δεν έχει ξεπεράσει το πλήθος των επαναλήψεων που επιθυμούμε. Π.χ. **Όσο (i <= 10) επανάλαβε**.
3. Αμέσως κάτω από τη δεσμευμένη λέξη **επανάλαβε** θα εισάγουμε τις προς εκτέλεση εντολές με μία ακόμη εντολή επιπλέον….
4. Την εντολή **i <-- i + 1**, ώστε μετά από κάθε μία εκτέλεση της ομάδας εντολών να καταγράφουμε την εκτέλεσή της, ώστε να γνωρίζει ο Η/Υ ότι πλησιάζουμε προς το τέλος των επαναλήψεων.
5. Στο τέλος δεν ξεχνάμε να εισάγουμε τη δεσμευμένη λέξη **Τέλος_Επαναλήψης**.

Συνοψίζοντας….

```
i <-- 1
Όσο (i <= 10) Επανάλαβε
      Γράψε "Καλημέρα"
      i <-- i + 1
Τέλος_Επαναλήψης
```

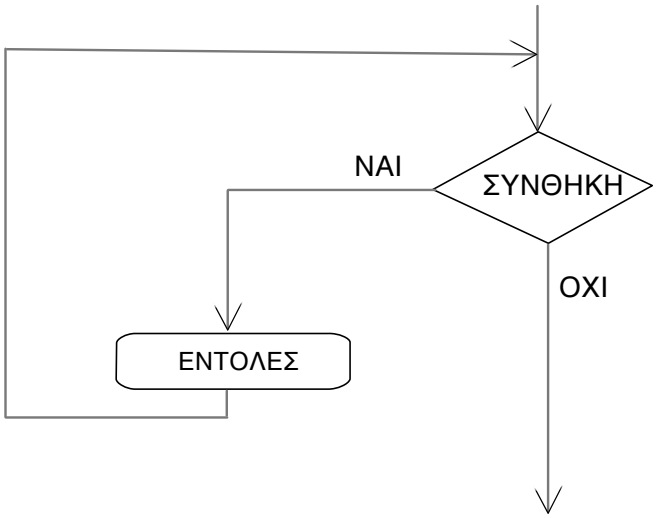
Με τον τρόπο αυτό εμείς έχουμε γράψει την εντολή **Γράψε "Καλημέρα"** μία φορά, αλλά ο Η/Υ την εκτελεί 10 φορές, αφού ξεκινάει από **i = 1**, και καταλήγει στην τιμή **i = ?**

Προσοχή: Η μεταβλητή – μετρητής **i** κατέχει διαδικαστικό ρόλο στην εκτέλεση του αλγορίθμου. Η εντολή που μας ενδιαφέρει στην ουσία είναι η **Γράψε "Καλημέρα"**. Παρόλα αυτά είναι καίριο να δώσουμε:

- σωστή τιμή εκκίνησης στο **i**, (**i <-- 1**)
- σωστή συνθήκη τερματισμού (**i <= 10)** και …
- σωστό τρόπο αναβάθμισης (βήμα) **i <-- i + 1**

Για τη συγκεκριμένη δομή επανάληψης να θυμάστε πάντα ότι η επανάληψη **διαρκεί όσο η συνθήκη είναι αληθής**. Την **πρώτη φορά που η συνθήκη θα γίνει ψευδής**, η επανάληψη τερματίζεται και συνεχίζουμε με τις υπόλοιπες εντολές του προγράμματος.

Διάγραμμα ροής **Όσο…Επανάλαβε**



2. **Δομή Αρχή_Επανάληψης…Μέχρι_ότου**

Σε αντίθεση με τη δομή **Όσο…Επανάλαβε**, εδώ οι επαναλήψεις εκτελούνται μέχρι η συνθήκη να γίνει αληθής, δηλαδή όσο η συνθήκη είναι ψευδής. Έχει την τελείως ανάποδη λογική της **Όσο…Επανάλαβε**, οπότε για το ίδιο πρόβλημα η συνθήκη της δομής αυτής θα πρέπει να είναι τελείως αντίθετη από τη συνθήκη της **Όσο…Επανάλαβε**.

Για παράδειγμα, για να εκτυπώσουμε 10 φορές το μήνυμα "Καλημέρα", θα χρησιμοποιήσουμε τη δομή **Αρχή_Επανάληψης…Μέχρι_ότου** με τον ακόλουθο τρόπο:

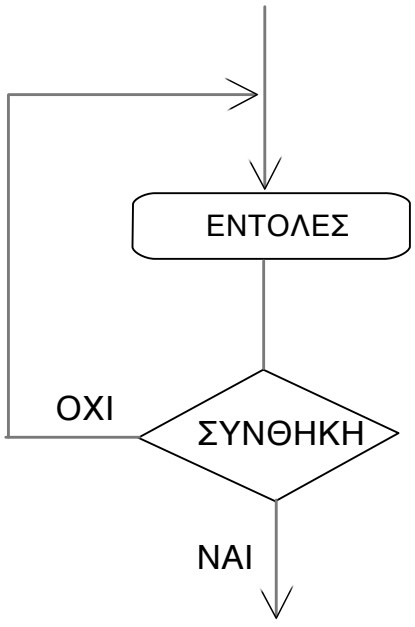
```
i <-- 1
Αρχή_Επανάληψης
      Γράψε "Καλημέρα"
      i <-- i + 1
Μέχρι_ότου (i > 10)
```

Θα πρέπει να συνηθίσει κανείς τη νέα δομή, σκεφτόμενος με τη λογική, ότι επαναλαμβάνω κάποιες ενέργειες μέχρι να συμβεί κάποιο γεγονός. Για παράδειγμα, αν με την **Όσο…Επανάλαβε** λέγαμε "επανάλαβε μία ενέργεια όσο είναι ημέρα", τότε με τη δομή **Αρχή_Επανάληψης…Μέχρι_ότου**, θα πρέπει να πούμε "επανάλαβε μία ενέργεια μέχρι να γίνει νύκτα".

Βασικές διαφορές μεταξύ των 2 δομών επανάληψης:

- Στην **Όσο…Επανάλαβε** οι επαναλήψεις εκτελούνται όσο η συνθήκη είναι αληθής, ενώ στην **Αρχή_Επανάληψης…Μέχρι_ότου** όσο η συνθήκη είναι ψευδής (ή μέχρι να γίνει αληθής).
- Στην **Όσο…Επανάλαβε** υπάρχει περίπτωση να μην εκτελεστεί καμία επανάληψη, εάν η συνθήκη είναι εξαρχής ψευδής, ενώ στην **Αρχή_Επανάληψης…Μέχρι_ότου** λόγω του τρόπου που λειτουργεί, θα εκτελεστεί τουλάχιστον μία επανάληψη.
- Στην **Όσο…Επανάλαβε** πρώτα ελέγχουμε και μετά εκτελούμε, ενώ στην **Αρχή_Επανάληψης… Μέχρι_ότου**, πρώτα εκτελούμε και μετά ελέγχουμε τη συνθήκη.

Διάγραμμα ροής **Αρχή_Επανάληψης…Μέχρι_ότου**



3. **Δομή Για…Από…Μέχρι**

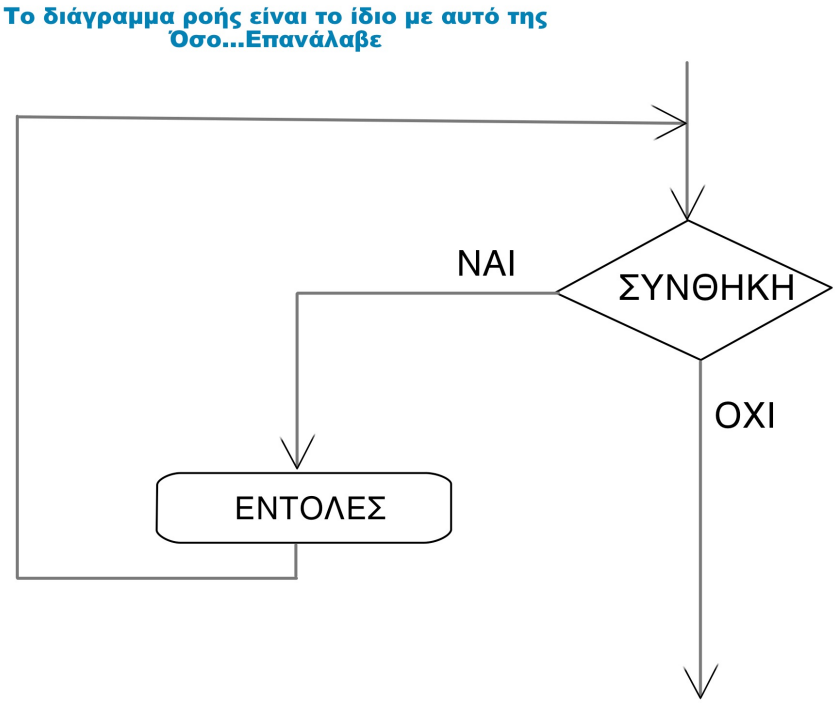
Πρόκειται για την τρίτη και τελευταία δομή επανάληψης. Χρησιμοποιείται μόνο όταν γνωρίζουμε από πριν το πλήθος των επαναλήψεων. Είναι πολύ βολική γιατί είναι μικρότερη σε μέγεθος. Επίσης είναι πιο δύσκολο να κάνει λάθος κανείς με τη δομή **Για…Από…Μέχρι**, απ' ότι με την Όσο…Επανάλαβε ή την Αρχή_Επανάληψης…Μέχρις_Ότου.

Συντάσσεται ως εξής:

```
Για i από 1 μέχρι 10 με βήμα 1
      Γράψε 'Καλημέρα'
Τέλος_Επανάληψης
```

- Στην ουσία πρόκειται για τη δομή Όσο…Επανάλαβε (έχει την ίδια λογική). Η δεσμευμένη λέξη **με βήμα κ** δηλώνει την αναβάθμιση του μετρητή κατά κ, δηλαδή, **i <-- i + κ**
- Όταν το **κ είναι ίσο με 1**, τότε μπορούμε να το παραλείψουμε χάριν συντομίας.
- Η αρχικοποίηση του μετρητή π.χ. **i <-- 1** δηλώνεται από τις λέξεις **Για i από 1**
- Η συνθήκη π.χ. (i <= 10), δηλώνεται από τις λέξεις **μέχρι 10**
- Σε αυτή τη δομή η συνθήκη εννοείται ότι περιέχει πάντα την ισότητα, δηλ είναι πάντα π.χ. (i <= 10) και ποτέ (i < 10)

Διάγραμμα ροής **Για…Από…Μέχρι**



Εύρεση Μεγίστου - Ελαχίστου (max - min) σε Επανάληψη

Όταν μας ζητείται να διαβάσουμε ένα μεγάλο πλήθος αριθμών και να βρούμε το μέγιστο ή το ελάχιστο, τότε θα πρέπει να προσαρμόσουμε τη μέθοδο της υπόθεσης και των διαδοχικών συγκρίσεων, ώστε να δουλεύει σε οποιαδήποτε από τις γνωστές δομές επανάληψης. Υπάρχουν 2 είδη ασκήσεων, ανάλογα με το αν έχουμε γνωστό ή άγνωστο πλήθος επαναλήψεων.

1. **Εύρεση max σε γνωστό πλήθος αριθμών** (π.χ. 100, που διαβάζονται από το πληκτρολόγιο)

   Μέθοδος 1η:
   Διαβάζουμε τον πρώτο αριθμό εκτός επανάληψης, τον χρησιμοποιούμε ως υπόθεση και στη συνέχεια διαβάζουμε όλους τους υπόλοιπους αριθμούς μέσα στην επανάληψη και τους χρησιμοποιούμε για τις απλές συγκρίσεις.

```
Γράψε 'Δώσε τον πρώτο αριθμό'
Διάβασε x

max <— x                        ! Υπόθεση, στον πρώτο αριθμό

i <-- 2
Όσο (i <= 100) Επανάλαβε
      Γράψε 'Δώσε τον αριθμό', i
      Διάβασε x

      Αν (x > max) τότε         ! Διαδοχικές απλές συγκρίσεις στους
            max <— x            ! υπόλοιπους αριθμούς
      Τέλος_αν

      i <-- i + 1
Τέλος_Επαναλήψης

Γράψε 'Το μέγιστο είναι:', max
```

   Μέθοδος 2η:
   Διαβάζουμε όλους τους αριθμούς μέσα στην επανάληψη. Χρησιμοποιούμε τον μετρητή i, για να διαπιστώσουμε αν είμαστε στην πρώτη επανάληψη (οπότε πρέπει να κάνουμε υπόθεση) ή σε όλες τις υπόλοιπες (οπότε πρέπει να κάνουμε απλές συγκρίσεις).

```
i <-- 1
Όσο (i <= 100) Επανάλαβε
      Γράψε 'Δώσε τον αριθμό', i
      Διάβασε x

      Αν (i = 1) τότε                 ! Είμαστε στην 1η επανάληψη, άρα
            max <— x                  ! κάνουμε υπόθεση
      Αλλιώς
            Αν x > max τότε           ! Είμαστε σε όλες τις υπόλοιπες
                  max <— x            ! επαναλήψεις, άρα κάνουμε σύγκριση
            Τέλος_αν
      Τέλος_αν

      i <-- i + 1
Τέλος_Επαναλήψης

Γράψε 'Το μέγιστο είναι:', max
```

2. **Εύρεση max σε άγνωστο πλήθος αριθμών** (π.χ. μέχρι να δοθεί ο αριθμός 999)

Μέθοδος**:**
Ξεκινάμε από το γεγονός ότι έχουμε άγνωστο πλήθος επαναλήψεων. Επομένως βρίσκουμε πρώτα τη συνθήκη και δημιουργούμε το σκελετό της επανάληψης. Επειδή πρέπει η υπόθεση να γίνει μόνο για τον πρώτο κατά σειρά αριθμό, χρησιμοποιούμε έναν μετρητή (αφού τώρα πια δεν έχουμε το i στη διάθεσή μας) τον οποίο αυξάνουμε κάθε φορά που διαβάζουμε έναν αποδεκτό αριθμό, δηλαδή διαφορετικό του 999. Τα υπόλοιπα είναι όμοια με τη μέθοδο 2 της προηγούμενης σελίδας.

```
πλ <— 0                          ! πλ <— 0, αφού δεν έχουμε ακόμη
                                 ! κανέναν αποδεκτό αριθμό

Γράψε 'Δώσε αριθμό (999 για τέλος)'
Διάβασε x

Όσο (x <> 999) Επανάλαβε         ! Εφόσον ο αριθμός είναι αποδεκτός
      πλ <— πλ + 1               ! μπαίνουμε στην Όσο και αυξάνουμε το πλ

      Αν (πλ = 1) τότε           ! Είμαστε στην 1η επανάληψη, άρα
            max <— x             ! κάνουμε υπόθεση
      Αλλιώς
            Αν x > max τότε      ! Είμαστε σε όλες τις υπόλοιπες
                  max <— x       ! επαναλήψεις, άρα κάνουμε σύγκριση
            Τέλος_αν
      Τέλος_αν

      Γράψε 'Δώσε αριθμό (999 για τέλος)'
      Διάβασε x
Τέλος_Επαναλήψης

Γράψε 'Το μέγιστο είναι:', max
```

Έλεγχος εισόδου:
Έλεγχο εισόδου εκτελούμε σε ένα πρόγραμμα:

- Όταν θέλουμε να υποχρεώσουμε το χρήστη του προγράμματος να εισάγει σωστά δεδομένα. Η διαδικασία που ακολουθούμε είναι ότι, εάν ο χρήστης δώσει στην είσοδο (πληκτρολόγιο) ακατάλληλα δεδομένα, τότε τον υποχρεώνουμε με μία **δομή επανάληψης** να ξαναδώσει είσοδο (και ενδεχομένως να ξαναδώσει και να ξαναδώσει κτλπ) μέχρι το πρόγραμμά μας να λάβει σωστά δεδομένα εισόδου οπότε και προχωράμε παρακάτω…

Παράδειγμα:
Να γραφεί πρόγραμμα το οποίο να διαβάζει την ηλικία ενός εργάτη και σε περίπτωση που αυτή είναι μεγαλύτερη ή ίση των 65, να εμφανίζει το μήνυμα «Συνταξιοδότηση», διαφορετικά να εμφανίζει τα χρόνια που υπολείπονται μέχρι της συνταξιοδότηση, π.χ. αν η ηλικία είναι 60, να εμφανίζει το μήνυμα «5 χρόνια μέχρι τη συνταξιοδότηση». Να γίνεται έλεγχος εισόδου, στην ηλικία ώστε αυτή να είναι μεγαλύτερη από 0 και μικρότερη από 100.

Λύση με Όσο…Επανάλαβε:

```
ΠΡΟΓΡΑΜΜΑ Έλεγχος_Εισόδου
ΜΕΤΑΒΛΗΤΕΣ
  ΠΡΑΓΜΑΤΙΚΕΣ: ηλικία
ΑΡΧΗ
  ΓΡΑΨΕ 'Δώσε την ηλικία του εργάτη (από 1 έως 100)'
  ΔΙΑΒΑΣΕ ηλικία
  ΟΣΟ (ηλικία < 1) Ή (ηλικία > 100) ΕΠΑΝΑΛΑΒΕ
    ΓΡΑΨΕ 'Η ηλικία του εργάτη πρέπει να είναι από 1 έως 100.'
    ΓΡΑΨΕ 'Παρακαλώ ξαναδώστε την ηλικία'
    ΔΙΑΒΑΣΕ ηλικία
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΑΝ (ηλικία >= 65) ΤΟΤΕ
    ΓΡΑΨΕ 'Συνταξιοδότηση'
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ (65-ηλικία):5:2, ' χρόνια μέχρι τη συνταξιοδότηση'
  ΤΕΛΟΣ_ΑΝ

ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Λύση με Αρχή_Επανάληψης…Μέχρις_Ότου:

```
ΠΡΟΓΡΑΜΜΑ Έλεγχος_Εισόδου
ΜΕΤΑΒΛΗΤΕΣ
  ΠΡΑΓΜΑΤΙΚΕΣ: ηλικία
ΑΡΧΗ
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Δώσε την ηλικία του εργάτη (από 1 έως 100)'
    ΔΙΑΒΑΣΕ ηλικία
  ΜΕΧΡΙΣ_ΟΤΟΥ (ηλικία >= 1 ) ΚΑΙ (ηλικία <= 100)

  ΑΝ (ηλικία >= 65) ΤΟΤΕ
    ΓΡΑΨΕ 'Συνταξιοδότηση'
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ (65 - ηλικία), ' χρόνια μέχρι τη συνταξιοδότηση'
  ΤΕΛΟΣ_ΑΝ

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Προσοχή:

Όταν έχουμε να κάνουμε έλεγχο εισόδου σε πάνω από μία μεταβλητές, ο έλεγχος να γίνεται **σε κάθε μεταβλητή μόνη της**.

ΟΧΙ ΟΛΕΣ ΜΑΖΙ !!!

Δημιουργία μενού επιλογών:
Σε ασκήσεις όπου ζητείται από τον χρήστη η δημιουργία **μενού επιλογών** το οποίο θα επαναλαμβάνεται συνεχώς μέχρι να πατηθεί συγκεκριμένη επιλογή, καλό είναι να χρησιμοποιείται η δομή **Αρχή_Επανάληψης … Μέχρις_Ότου**.

Παράδειγμα:
Να γραφεί αλγόριθμος ο οποίος θα εμφανίζει στο χρήστη το παρακάτω μενού

```
Πρόγραμμα Υπολογισμού Εμβαδού Γεωμετρικών Σχημάτων

Μενού Επιλογών:
   1. Εμβαδόν τριγώνου
   2. Εμβαδόν τετραγώνου
   3. Εμβαδόν κύκλου
   4. Εμβαδόν παραλληλογράμμου
   5. Έξοδος
Δώστε την επιλογή σας
```

Κάθε φορά που ο χρήστης θα δίνει ως είσοδο τους αριθμούς 1 έως 4, το πρόγραμμα θα ζητάει από το χρήστη τα ανάλογα δεδομένα (μήκη πλευρών, ακτίνα κύκλου κτλπ) και θα υπολογίζει και θα εμφανίζει το αντίστοιχο εμβαδόν. Στη συνέχεια θα εμφανίζει ξανά το μενού ώστε ο χρήστης να προχωρήσει σε επόμενο υπολογισμό. Η διαδικασία συνεχίζεται μέχρι ο χρήστης να δώσει ως είσοδο τον αριθμό 5.

```
ΠΡΟΓΡΑΜΜΑ Εμβαδόν
ΣΤΑΘΕΡΕΣ
  π = 3.14
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: επ                ! επ = η επιλογή του χρήστη
  ΠΡΑΓΜΑΤΙΚΕΣ: β, υ           ! β, υ = η βάση και το ύψος του τριγώνου
  ΠΡΑΓΜΑΤΙΚΕΣ: πλ1, πλ2       ! πλ1, πλ2 = οι 2 πλευρές του παραλληλογράμου
  ΠΡΑΓΜΑΤΙΚΕΣ: τ              ! τ = η πλευρά του τετραγώνου
  ΠΡΑΓΜΑΤΙΚΕΣ: ρ              ! ρ = η ακτίνα του κύκλου
  ΠΡΑΓΜΑΤΙΚΕΣ: Εμβ            ! Εμβ = το εμβαδόν

ΑΡΧΗ

  Γράψε ' Υπολογισμού Εμβαδού Γεωμετρικών Σχημάτων'

  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    Γράψε 'Μενού Επιλογών:'
    Γράψε '1.  Εμβαδόν τριγώνου'
    Γράψε '2.  Εμβαδόν τετραγώνου'
    Γράψε '3.  Εμβαδόν κύκλου'
    Γράψε '4.  Εμβαδόν παραλληλογράμμου'
    Γράψε '5.  Έξοδος'
    Γράψε 'Δώστε την επιλογή σας'
    Διάβασε επ

    ΑΝ (επ = 1) ΤΟΤΕ
      Γράψε 'Υπολογισμός εμβαδού τριγώνου'
      Γράψε 'Δώστε τη βάση και το ύψος'
      Διάβασε β, υ
      Εμβ <-- (β*υ)/2
      Γράψε 'Το εμβαδόν του τριγώνου είναι ', Εμβ
    ΑΛΛΙΩΣ_ΑΝ (επ = 2) ΤΟΤΕ
      Γράψε 'Υπολογισμός εμβαδού τετραγώνου'
      Γράψε 'Δώστε τη πλευρά'
      Διάβασε τ
      Εμβ <-- τ^2
      Γράψε 'Το εμβαδόν του τετραγώνου είναι ', Εμβ
    ΑΛΛΙΩΣ_ΑΝ (επ = 3) ΤΟΤΕ
      Γράψε 'Υπολογισμός εμβαδού κύκλου'
      Γράψε 'Δώστε τη ακτίνα'
      Διάβασε ρ
      Εμβ <-- π * ρ^2
      Γράψε 'Το εμβαδόν του κύκλου είναι ', Εμβ
    ΑΛΛΙΩΣ_ΑΝ (επ = 4) ΤΟΤΕ
      Γράψε 'Υπολογισμός εμβαδού παραλληλογράμμου'
      Γράψε 'Δώστε τις 2 πλευρες'
      Διάβασε πλ1, πλ2
      Εμβ <-- πλ1 * πλ2
      Γράψε 'Το εμβαδόν του κύκλου είναι ', Εμβ
    ΤΕΛΟΣ_ΑΝ

  ΜΕΧΡΙΣ_ΟΤΟΥ (επ = 5)

ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Κεφάλαιο 3
Δομές δεδομένων και αλγόριθμοι

Ορισμός Δομής Δεδομένων:
Δομή δεδομένων **ε**ίναι σύνολα αποθηκευμένων δεδομένων τα οποία υφίστανται επεξεργασία μέσω συγκεκριμένων λειτουργιών. Αποτελούνται από κόμβους.

Ποιες είναι οι βασικές λειτουργίες επί των δομών δεδομένων:
1. **Προσπέλαση**
2. **Εισαγωγή**
3. **Διαγραφή**
4. **Αναζήτηση**
5. **Ταξινόμηση**
6. **Αντιγραφή**
7. **Συγχώνευση**
8. **Διαχωρισμός**

Ποια η σχέση αλγορίθμου και δομών δεδομένων:
Υπάρχει μεγάλη εξάρτηση μεταξύ του αλγορίθμου και των δομών δεδομένων στις οποίες επενεργεί. Θεωρούνται αλληλένδετα. Επομένως:

ΑΛΓΟΡΙΘΜΟΣ + ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ = ΠΡΟΓΡΑΜΜΑ
(Εξίσωση Wirth. Ο Wirth, σχεδίασε και υλοποίησε τη γλώσσα Pascal το 1976)

Ποια είναι τα δύο είδη δομών δεδομένων και ποια τα χαρακτηριστικά τους:
1. **Στατικές δομές:** Είναι οι δομές που έχουν:
    a. συγκεκριμένο και σταθερό μέγεθος.
    b. το οποίο καθορίζεται (δηλώνεται) κατά τη δημιουργία τους (συγγραφή κώδικα).
    c. και τα στοιχεία τους αποθηκεύονται σε συνεχόμενες θέσεις μνήμης.
2. **Δυναμικές δομές:** Είναι οι δομές που:
    a. Δεν έχουν σταθερό μέγεθος.
    b. Ο αριθμός των στοιχείων τους (κόμβων) αυξομειώνεται με τις ανάγκες του προγράμματος.
    c. Και τα στοιχεία δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης.

Τι είναι πίνακας:
Πίνακας είναι ένα πεπερασμένο και διατεταγμένο σύνολο ομοειδών στοιχείων.

Τι είναι η αναζήτηση στοιχείου σε πίνακα:
Είναι η διαδικασία κατά την οποία αναζητούμε την ύπαρξη (ή όχι) ενός συγκεκριμένου στοιχείου μέσα σε έναν πίνακα. Υπάρχουν πολλές μέθοδοι αναζήτησης. Δύο από αυτές είναι:

1. **Σειριακή αναζήτηση (ή και γραμμική)**: Σύμφωνα με τη μέθοδο αυτή, εξετάζουμε ένα προς ένα τα στοιχεία του πίνακα και συγκρίνουμε τις τιμές τους με αυτή που ψάχνουμε, προκειμένου να εντοπίσουμε τη **θέση (ή τις θέσεις)** του προς αναζήτηση στοιχείου. Οι επαναλήψεις μπορεί να σταματούν **μόλις βρεθεί το στοιχείο**, ή και να συνεχίζονται **μέχρι το τέλος του πίνακα**, εάν αυτό που αναζητούμε είναι το **πλήθος** των εμφανίσεων του στοιχείου ή και **όλες οι εμφανίσεις** του (δηλαδή όλες τις θέσεις στις οποίες εντοπίζεται το στοιχείο). Η σειριακή μέθοδος αναζήτησης είναι η πιο απλή, αλλά και η λιγότερο αποτελεσματική. Χρησιμοποιείται σε όλους του πίνακες, αλλά η χρήση της προτείνεται κυρίως για περιπτώσεις όπου:
    a. Ο πίνακας είναι μη ταξινομημένος
    b. Ο πίνακας είναι μικρού μεγέθους
    c. Η αναζήτηση στον πίνακα γίνεται σπάνια
2. **Δυαδική αναζήτηση**: Είναι μέθοδος αναζήτησης που εφαρμόζεται μόνο σε ταξινομημένους πίνακες. Βασίζεται στην μέθοδο **διαίρει και βασίλευε** και είναι πολύ αποτελεσματική.

Τι είναι η ταξινόμηση πίνακα:
Είναι η διαδικασία κατά την οποία τα στοιχεία μίας δομής δεδομένων (επομένως και ενός στατικού πίνακα) διατάσσονται κατά αύξουσα ή φθίνουσα σειρά με βάση κάποιο κριτήριο. Συνήθως πρόκειται για αριθμητική ή αλφαριθμητική ταξινόμηση. Μία πολύ κοινή μέθοδος ταξινόμησης είναι η **ταξινόμηση ευθείας ανταλλαγής (φυσαλίδα)**.

Υπάρχουν άλλοι μέθοδοι ταξινόμησης εκτός της φυσαλίδας;
Για την ταξινόμηση δεδομένων, έχουν εκπονηθεί πάρα πολλοί αλγόριθμοι. Άλλοι σχετικά απλοί αλγόριθμοι είναι οι:

1. η ταξινόμηση με επιλογή
2. η ταξινόμηση με παρεμβολή
3. Η γρήγορη ταξινόμηση (quicksort)

Ο πιο γρήγορος αλγόριθμος ταξινόμησης είναι η “γρήγορη ταξινόμηση” (quicksort). Η ταξινόμηση φυσσαλίδας είναι ο πιο απλός και ταυτόχρονα ο πιο αργός αλγόριθμος ταξινόμησης.

Βασικά τμήματα αλγορίθμων σε πίνακες

Μονοδιάστατοι πίνακες

1) Εισαγωγή στοιχείων σε πίνακα

```
Για i από 1 μέχρι N
   Γράψε 'Δώσε το στοιχείο', i, 'του πίνακα'
   Διάβασε A[i]
Τέλος_Επανάληψης
```

2) Εμφάνιση στοιχείων του πίνακα

```
Για i από 1 μέχρι N
   Γράψε A[i]
Τέλος_Επανάληψης
```

3) Άθροισμα στοιχείων του πίνακα και μέσος όρος

```
sum <-- 0
Για i από 1 μέχρι N
   sum <-- sum + A[i]
Τέλος_Επανάληψης
mo <-- sum / N
Γράψε 'Το άθροισμα είναι', sum, 'και ο μέσος όρος', mo
```

4) Εύρεση μεγαλύτερου στοιχείου σε πίνακα και της θέσης στην οποία βρίσκεται

```
max <-- A[1]
θ <-- 1
Για i από 2 μέχρι N
   Αν (A[i] > max) τότε
        max <-- A[i]
        θ <-- i
   Τέλος_Αν
Τέλος_Επανάληψης
Γράψε 'Το μέγιστο στοιχείο είναι το', max, 'και βρίσκεται στη θέση', θ
```

5) Εύρεση μικρότερου στοιχείου σε πίνακα και της θέσης στην οποία βρίσκεται

```
min <-- A[1]
θ <-- 1
Για i από 2 μέχρι N
   Αν (A[i] < min) τότε
        min <-- A[i]
        θ <-- i
   Τέλος_Αν
Τέλος_Επανάληψης
Γράψε 'Το ελάχιστο στοιχείο είναι το', min, 'και βρίσκεται στη θέση', θ
```

6) Σειριακή αναζήτηση της πρώτης εμφάνισης του στοιχείου key, και έξοδος από την αναζήτηση

```
toBrika <-- Ψευδής
i <-- 1
Όσο (i <= N) και (toBrika = Ψευδής) επανάλαβε
    Αν (A[i] = key) τότε
         toBrika <-- Αληθής
         θ <-- i
    Αλλιώς
         i <- i + 1
    Τέλος_Αν
Τέλος_Επανάληψης
Αν (toBrika = Αληθής) τότε
    Γράψε 'Το στοιχείο', key, 'βρέθηκε στη θέση', θ
Αλλιώς
    Γράψε 'Το στοιχείο', key, 'δεν υπάρχει στον πίνακα'
Τέλος_Αν
```

7) Σειριακή αναζήτηση και εύρεση όλων των εμφανίσεων του στοιχείου key, της θέσης στην οποία βρίσκονται και του συνολικού πλήθους εμφανίσεων.

```
π <-- 0
Για i από 1 μέχρι N
    Αν (A[i] = key) τότε
         Γράψε 'Το στοιχείο', key, 'βρέθηκε στη θέση', i
         π <-- π + 1
    Τέλος_Αν
Τέλος_Επανάληψης
Αν (π = 0) τότε
    Γράψε 'Το στοιχείο', key, 'δεν υπάρχει στον πίνακα'
Αλλιώς
    Γράψε 'Ο συνολικός αριθμός εμφανίσεων του στοιχείου', key, 'είναι', π
Τέλος_Αν
```

8) Εύρεση όλων των εμφανίσεων του μεγαλύτερου στοιχείου πίνακα, και πλήθους αυτών.

```
max <-- A[1]
Για i από 2 μέχρι N
    Αν (A[i] > max) τότε
         max <-- A[i]
    Τέλος_Αν
Τέλος_Επανάληψης

Γράψε 'Το μέγιστο στοιχείο είναι το', max, 'και βρίσκεται στις θέσεις:'

π <-- 0
Για i από 1 μέχρι N
    Αν (A[i] = max) τότε
         Γράψε i
         π <-- π + 1
    Τέλος_Αν
Τέλος_Επανάληψης
Γράψε 'Ο συνολικός αριθμός εμφανίσεων του στοιχείου', max, 'είναι', π
```

(*Παρομοίως για εύρεση ελαχίστου*)

9) Ταξινόμηση πίνακα σε αύξουσα σειρά με τη μέθοδο της φυσαλίδας.

```
Για i από 2 μέχρι N
   Για j από N μέχρι i με_βήμα -1
        Αν (A[j] < A[j-1]) τότε
             tmp <-- A[j]
             A[j] <-- A[j-1]
             A[j-1] <-- tmp
        Τέλος_Αν
   Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

το μόνο που αλλάζει για φθίνουσα ταξινόμηση είναι η σύγκριση:

```
        Αν (A[j] < A[j-1]) τότε

        Η οποία πρέπει να γίνει:

        Αν (A[j] > A[j-1]) τότε
```

10) Παραλλαγή της φυσαλίδας που σταματάει τις σαρώσεις, όταν αντιληφθεί ότι ο πίνακας είναι πλέον ταξινομημένος.

```
Αρχή_Επανάληψης
   ! πλ = το πλήθος των αντιμεταθέσεων κάθε σάρωσης
   ! Εάν παραμείνει 0 μετά από κάποια σάρωση, σημαίνει
   ! ότι ο πίνακας είναι πλέον ταξινομημένος
   πλ <-- 0
   i <– 2
   Για j από N μέχρι i με_βήμα -1
        Αν (A[j] < A[j-1]) τότε
             πλ <-- πλ + 1

             tmp <-- A[j]
             A[j] <-- A[j-1]
             A[j-1] <-- tmp
        Τέλος_Αν
   Τέλος_Επανάληψης
   i <-- i + 1
Μέχρις_Ότου (i > N Ή πλ = 0)
```

11) Ταξινόμηση πίνακα σε αύξουσα σειρά με τη μέθοδο της επιλογής.

```
Για i από 1 μέχρι N
   min <– A[i]
   θmin <– i
   Για j από i+1 μέχρι N
        Αν (A[j] < min) τότε
             min <– A[j]
             θmin <– j
        Τέλος_ΑΝ
   Τέλος_Επανάληψης
   tmp <-- A[θmin]
   A[θmin] <-- A[i]
   A[i] <-- tmp
Τέλος_Επανάληψης
```

12) Ταξινόμηση πίνακα σε αύξουσα σειρά με τη μέθοδο της παρεμβολής.

```
Για i από 2 μέχρι N
    ! key = το στοιχείο που θέλουμε να παρεμβάλουμε
    ! στον ήδη ταξινομημένο πίνακα A[1 έως i-1]
    key <— A[i]

    stop <— Ψευδής
    j <— i - 1
    Όσο (j >= 1) και (stop = Ψευδής) επανάλαβε
         Αν (key < A[j]) τότε
              A[j+1] <— A[j]
              j <— j - 1
         Αλλιώς
              stop <— Αληθής
         Τέλος_Αν
    Τέλος_Επανάληψης

    A[j+1] <— key
Τέλος_Επανάληψης
```

13) Δυαδική αναζήτηση.

```
αρχή <— 1
τέλος <— N
toBrika <— Ψευδής
Όσο (αρχή <= τέλος) και (toBrika = Ψευδής) επανέλαβε
    μεσαίο <— (αρχή + τέλος) div 2
    Αν (A[μεσαίο] = key) τότε
         toBrika <— Αληθής
         θ <— μεσαίο
    Αλλιώς_Αν (A[μεσαίο] < key) τότε
         αρχή <— μεσαίο + 1
    Αλλιώς
         τέλος <— μεσαίο - 1
    Τέλος_Αν
Τέλος_Επανάληψης

Αν (toBrika = Αληθής) τότε
    Γράψε 'Το στοιχείο βρέθηκε στη θέση', θ
Αλλιώς
    Γράψε 'Το στοιχείο δεν βρέθηκε'
Τέλος_Αν
```

14) Συγχώνευση ταξινομημένων πινάκων χωρίς χρήση ταξινόμησης

```
i <– 1
j <– 1
k <– 1
Όσο (i <= M) και (j <= N) επανέλαβε
    Αν (A[i] < B[j]) τότε
         Γ[k] <– A[i]
         i <– i + 1
    Αλλιώς
         Γ[k] <– B[j]
         j <– j + 1
    Τέλος_Αν
    k <– k + 1
Τέλος_Επανάληψης

Αν (i > M) τότε
    Για z από j μέχρι N
         Γ[k] <– B[z]
         k <– k + 1
    Τέλος_Επανάληψης
Αλλιώς
    Για z από i μέχρι M
         Γ[k] <– A[z]
         k <– k + 1
    Τέλος_Επανάληψης
Τέλος_Αν
```

Δισδιάστατοι πίνακες

1) Εισαγωγή στοιχείων σε πίνακα

```
Για i από 1 μέχρι M
   Για j από 1 μέχρι N
        Γράψε 'Δώσε το στοιχείο', i, j, 'του πίνακα'
        Διάβασε A[i,j]
   Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

2) Εμφάνιση στοιχείων του πίνακα

```
Για i από 1 μέχρι M
   Για j από 1 μέχρι N
        Γράψε A[i,j]
   Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

3) Άθροισμα στοιχείων όλου του πίνακα και μέσος όρος

```
sum <-- 0
Για i από 1 μέχρι M
   Για j από 1 μέχρι N
        sum <-- sum + A[i,j]
   Τέλος_Επανάληψης
Τέλος_Επανάληψης
mo <-- sum / (M*N)
Γράψε 'Το άθροισμα είναι', sum, 'και ο μέσος όρος', mo
```

4) Άθροισμα στοιχείων του πίνακα και μέσος όρος κατά γραμμή

```
Για i από 1 μέχρι M
   sum <-- 0
   Για j από 1 μέχρι N
        sum <-- sum + A[i,j]
   Τέλος_Επανάληψης
   Γράψε 'Το άθροισμα είναι της γραμμής', i, 'είναι', sum
   Γράψε 'και ο μέσος όρος των στοιχείων της γραμμής είναι', sum / N
Τέλος_Επανάληψης
```

5) Άθροισμα στοιχείων του πίνακα και μέσος όρος κατά στήλη

```
Για j από 1 μέχρι N
   sum <-- 0
   Για i από 1 μέχρι M
        sum <-- sum + A[i,j]
   Τέλος_Επανάληψης
   Γράψε 'Το άθροισμα είναι της στήλης', j, 'είναι', sum
   Γράψε 'και ο μέσος όρος των στοιχείων της στήλης είναι', sum / M
Τέλος_Επανάληψης
```

6) Εύρεση μεγαλύτερου στοιχείου σε πίνακα και της θέσης στην οποία βρίσκεται

```
max <-- A[1,1]
θγ <-- 1
θσ <-- 1
Για i από 1 μέχρι M
   Για j από 1 μέχρι N
        Αν (A[i,j] > max) τότε
             max <-- A[i,j]
             θγ <-- i
             θσ <-- j
        Τέλος_Αν
   Τέλος_Επανάληψης
Τέλος_Επανάληψης
Γράψε 'Το μέγιστο στοιχείο είναι το', max
Γράψε 'και βρίσκεται στη γραμμή', θγ, 'και στη στήλη', θσ
```

7) Εύρεση μικρότερου στοιχείου σε πίνακα και της θέσης στην οποία βρίσκεται

```
min <-- A[1,1]
θγ <-- 1
θσ <-- 1
Για i από 1 μέχρι M
   Για j από 1 μέχρι N
        Αν (A[i,j] < min) τότε
             min <-- A[i,j]
             θγ <-- i
             θσ <-- j
        Τέλος_Αν
   Τέλος_Επανάληψης
Τέλος_Επανάληψης
Γράψε 'Το ελάχιστο στοιχείο είναι το', min
Γράψε 'και βρίσκεται στη γραμμή', θγ, 'και στη στήλη', θσ
```

8) Σειριακή αναζήτηση της πρώτης εμφάνισης του στοιχείου key, και έξοδος από την αναζήτηση

```
toBrika <-- Ψευδής
i <-- 1
Όσο (i <= M) και (toBrika = Ψευδής) επανάλαβε
   Όσο (j <= N) και (toBrika = Ψευδής) επανάλαβε
        Αν (A[i,j] = key) τότε
             toBrika <-- Αληθής
             θγ <-- i
             θσ <-- j
        Τέλος_Αν
        j <-- j + 1
   Τέλος_Επανάληψης
   i <-- i + 1
Τέλος_Επανάληψης
Αν (toBrika = Αληθής) τότε
   Γράψε 'Το στοιχείο', key, 'βρέθηκε στη γραμμή', θγ, 'και στη στήλη', θσ
Αλλιώς
   Γράψε 'Το στοιχείο', key, 'δεν υπάρχει στον πίνακα'
Τέλος_Αν
```

9) Σειριακή αναζήτηση και εύρεση όλων των εμφανίσεων του στοιχείου key, της θέσης στην οποία βρίσκονται και του συνολικού πλήθους εμφανίσεων.

```
π <-- 0
Για i από 1 μέχρι M
    Για j από 1 μέχρι N
         Αν (A[i,j] = key) τότε
              Γράψε 'Το στοιχείο', key, 'βρέθηκε στη γραμμή', i, 'και στήλη', j
              π <-- π + 1
         Τέλος_Αν
    Τέλος_Επανάληψης
Τέλος_Επανάληψης
Αν (π = 0) τότε
    Γράψε 'Το στοιχείο', key, 'δεν υπάρχει στον πίνακα'
Αλλιώς
    Γράψε 'Ο συνολικός αριθμός εμφανίσεων του στοιχείου', key, 'είναι', π
Τέλος_Αν
```

10) Εύρεση όλων των εμφανίσεων του μεγαλύτερου στοιχείου πίνακα, και πλήθους αυτών.

```
max <-- A[1,1]
Για i από 1 μέχρι M
    Για j από 1 μέχρι N
         Αν (A[i,j] > max) τότε
              max <-- A[i,j]
         Τέλος_Αν
    Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

Γράψε 'Το μέγιστο στοιχείο είναι το', max, 'και βρίσκεται στις θέσεις:'

```
π <-- 0
Για i από 1 μέχρι M
    Για j από 1 μέχρι N
         Αν (A[i] = max) τότε
              Γράψε i, j
              π <-- π + 1
         Τέλος_Αν
    Τέλος_Επανάληψης
Τέλος_Επανάληψης
Γράψε 'Ο συνολικός αριθμός εμφανίσεων του στοιχείου', max, 'είναι', π
```

(*Παρομοίως για εύρεση ελαχίστου*)

11) Ταξινόμηση δισδιάστατου πίνακα σε αύξουσα σειρά με τη μέθοδο της φυσαλίδας.

Έστω πίνακας A[M,N]

a. Μεταφορά όλων των στοιχείων του δισδιάστατου πίνακα A[M,N] σε μονοδιάστατο πίνακα Β, μεγέθους M*N

```
κ <-- 1
Για i από 1 μέχρι M
     Για j από 1 μέχρι N
       B[κ] <-- A[i,j]
       κ <-- κ + 1
     Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

b. Ταξινόμηση του μονοδιάστατου πίνακα Β (με μέγεθος M*N)

```
Για i από 2 μέχρι M*N
     Για j από M*N μέχρι i με_βήμα -1
          Αν (B[j] < B[j-1]) τότε
                tmp <-- B[j]
                B[j] <-- B[j-1]
                B[j-1] <-- tmp
          Τέλος_Αν
     Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

c. Μεταφορά όλων των στοιχείων του μονοδιάστατου πίνακα Β ξανά πίσω στο δισδιάστατο πίνακα A[M,N]

```
κ <-- 1
Για i από 1 μέχρι M
     Για j από 1 μέχρι N
       A[i,j] <-- B[κ]
       κ <-- κ + 1
     Τέλος_Επανάληψης
Τέλος_Επανάληψης
```

Κεφάλαιο 6
Εισαγωγή στον προγραμματισμό

Ποια είναι τα στοιχεία που προσδιορίζουν μία γλώσσα:

1. **Αλφάβητο**: Σύνολο των στοιχείων (συμβόλων) που χρησιμοποιούνται από τη γλώσσα
2. **Λεξιλόγιο**: Υποσύνολο των ακολουθιών που προκύπτουν από τα στοιχεία του αλφαβήτου και σχηματίζουν λέξεις αποδεκτές από τη γλώσσα.
3. **Γραμματική**: αποτελείται από:
    a. **Τυπικό**: Σύνολο κανόνων που καθορίζουν τις μορφές με τις οποίες μία λέξη είναι αποδεκτή
    b. **Συντακτικό**: Σύνολο κανόνων που καθορίζουν τη νομιμότητα της διάταξης και σύνδεσης των λέξεων, προκειμένου να δημιουργηθούν προτάσεις
4. **Σημασιολογία**: Σύνολο κανόνων που καθορίζουν την εννοιολογική σημασία των λέξεων και κατ' επέκταση των προτάσεων που χρησιμοποιούνται σε μια γλώσσα

Τα παραπάνω στοιχεία είναι κοινά μεταξύ φυσικών και τεχνητών γλωσσών.

Ποιες είναι οι διαφορές μεταξύ των φυσικών και τεχνητών γλωσσών:

1. Οι φυσικές γλώσσες εξελίσσονται συνεχώς ανάλογα με τα δεδομένα της εποχής και της κοινωνίας.
2. Οι τεχνητές είναι στάσιμες, γιατί ο σκοπός τους είναι συγκεκριμένος. Ωστόσο και αυτές βελτιώνονται και μεταβάλλονται με σκοπό τη διόρθωση των αδυναμιών τους και την επέκταση των δυνατοτήτων τους. Μεταβάλλονται σε επίπεδο **διαλέκτου** (λεξιλόγιο)
   και / ή σε επίπεδο **επέκτασης** (δυνατοτήτων).

Τι είναι η ιεραρχική σχεδίαση:
Είναι η τεχνική που βασίζεται στη συνεχή διαίρεση ενός προβλήματος σε υποπροβλήματα, τα οποία επιλύονται εύκολα και οδηγούν στην επίλυση του αρχικού προβλήματος. Ονομάζεται και **σχεδίαση από επάνω προς τα κάτω**.

Τι είναι ο τμηματικός προγραμματισμός:
Είναι η τεχνική σχεδίασης και ανάπτυξης των προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων. Αποτελεί την υλοποίηση της ιεραρχικής σχεδίασης. Μετά την ιεραρχική σχεδίαση, κάθε υποπρόβλημα αποτελεί ανεξάρτητη ενότητα και επιλύεται ξεχωριστά. Ο τελικός αλγόριθμος ανάγεται σε αυτά τα επιμέρους υποπρογράμματα. Αποτελεί την υλοποίηση από **κάτω προς τα πάνω**.

(+):
- Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντίστοιχου προγράμματος.
- Διευκολύνει την κατανόηση και διόρθωση του προγράμματος.
- Απαιτεί λιγότερο χρόνο και προσπάθεια τη συγγραφή του προγράμματος.
- Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού.

Τι είναι ο δομημένος προγραμματισμός:
Ο δομημένος προγραμματισμός είναι μια μεθοδολογία ανάπτυξης προγραμμάτων που σκοπό έχει να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων, να μειώσει τα λάθη, να εξασφαλίσει την εύκολη κατανόηση και να διευκολύνει τη διόρθωση και τη συντήρηση των προγραμμάτων.
Στηρίζεται στη χρήση τριών και μόνο λογικών δομών:
1. τη δομή της ακολουθίας
2. τη δομή της επιλογής
3. τη δομή της επανάληψης

καθώς και στους συνδυασμούς τους. Κάθε πρόγραμμα έχει μία είσοδο και μία έξοδο.

(+):
- Δημιουργία απλούστερων προγραμμάτων.
- Άμεση μεταφορά αλγορίθμου σε πρόγραμμα.
- Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα.
- Περιορισμός των λαθών κατά την ανάπτυξη.
- Διευκόλυνση της ανάγνωσης και κατανόησης του προγράμματος από τρίτους.
- Ευκολότερη διόρθωση και συντήρηση.

Ο δομημένος προγραμματισμός αναπτύχθηκε από την ανάγκη να υπάρχει μία κοινή μεθοδολογία στην ανάπτυξη των προγραμμάτων και τη μείωση των εντολών **GOTO** που χρησιμοποιούνται στο πρόγραμμα.

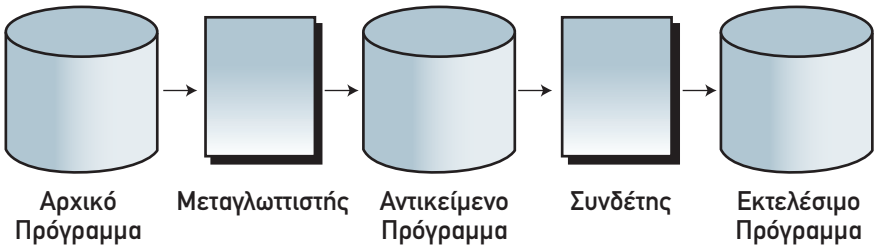
Τι γνωρίζετε για την εντολή GOTO;
Η εντολή GOTO χρησιμοποιείται για την αλλαγή της ροής ενός προγράμματος και τη μεταπήδηση σε μια άλλη εντολή μέσα το πρόγραμμα, προσπερνώντας την επόμενη ή τις επόμενες εντολές. Η χρήση της κάνει δυσκολότερη την κατανόηση του προγράμματος και την παρακολούθηση της ροής τους. Για αυτό το λόγο οι σύγχρονες γλώσσες προγραμματισμού διαθέτουν κατάλληλες εντολές που καθιστούν τη χρήση της GOTO περιττή. Κάποιες γλώσσες διαθέτουν ακόμη την εντολή GOTO για λόγους συμβατότητας.

Τι είναι το προγραμματιστικό περιβάλλον:
Προγραμματιστικό περιβάλλον ονομάζουμε το σύνολο των προγραμμάτων και εργαλείων που απαιτούνται και βοηθούν τη συγγραφή την εκτέλεση και κύρια τη διόρθωση ενός προγράμματος. Ένα σύγχρονο προγραμματιστικό περιβάλλον περιλαμβάνει τουλάχιστον:

- **Συντάκτη**
  Για τη συγγραφή και διόρθωση του προγγράματος
- **Μεταγλωττιστή**
  Για τη μετάφραση σε γλώσσα μηχανής
- **Συνδέτη**
  Για τη σύνδεση με τις βιβλιοθήκες και τη δημιουργία του τελικού εκτελέσιμου προγράμματος

Ποια είναι τα βήματα δημιουργίας και μεταγλώττισης ενός προγράμματος:
Η διαδικασία δημιουργίας και μεταγλώττισης ενός προγράμματος είναι η εξής:

1. Ο προγραμματιστής χρησιμοποιεί έναν **συντάκτη** (editor) για την αρχική συγγραφή του προγράμματος, που συνήθως είναι μια ακολουθία εντολών σε κάποια γλώσσα υψηλού επιπέδου. Το πρόγραμμα που δημιουργεί ο προγραμματιστής ονομάζεται **πηγαίο** (source) πρόγραμμα.
2. Εφόσον το πηγαίο πρόγραμμα δεν περιέχει **συντακτικά** λάθη, μετατρέπεται από το μεταγλωτιστή στο ισοδύναμό του πρόγραμμα σε γλώσσα μηχανής που λέγεται **αντικείμενο** (object) πρόγραμμα. Εάν υπάρχουν συντακτικά λάθη, ο μεταγλωττιστής τα ανιχνεύει, ο προγραμματιστής τα διορθώνει, και το διορθωμένο πρόγραμμα υποβάλλεται ξανά για μεταγλώττιση.
3. Αμέσως μετά, ο **συνδέτης - φορτωτής** (linker - loader) ολοκληρώνει τη σύνδεση του αντικείμενου προγράμματος και παράγει το **εκτελέσιμο** (executable) πρόγραμμα. Αυτό γιατί, το αντικείμενο πρόγραμμα αν και σε γλώσσα μηχανής, 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.



Ποιες είναι οι κατηγορίες μεταφραστικών προγραμμάτων:
Υπάρχουν δύο μεγάλες κατηγορίες τέτοιων προγραμμάτων:

- **Μεταγλωττιστής**
  Ο μεταγλωττιστής είναι ένα πρόγραμμα το οποίο δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου (**πηγαίο πρόγραμμα**) και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής (**αντικείμενο πρόγραμμα**). Το τελευταίο, μπορεί να αποθηκευτεί και να εκτελείται οποτεδήποτε από τον υπολογιστή και είναι τελείως ανεξάρτητο από το πηγαίο πρόγραμμα.
- **Διερμηνευτής**
  Ο διερμηνευτής είναι ένα πρόγραμμα το οποίο σε αντίθεση με το μεταγλωττιστή, δέχεται στην είσοδο μία προς μία τις εντολές του πηγαίου πρόγραμματος, και για κάθε μία εκτελεί αμέσως μια ισοδύναμη ακολυθία εντολών μηχανής.

Ποια τα μειονεκτήματα - πλεονεκτήματα μεταφραστή και διερμηνευτή:

Μεταφραστής:
(+):

- Το εκτελέσιμο πρόγραμμα που παράγει ο μεταγλωττιστής μπορεί να αποθηκευτεί και να εκτελεστεί οποτεδήποτε χωρίς να χρειάζεται πλέον το πηγαίο πρόγραμμα.
- Το εκτελέσιμο πρόγραμμα που παράγει ο μεταγλωττιστής είναι ταχύτερο από τη διαδικασία εκτέλεσης με το διερμηνευτή.

(-):

- Κατά τη φάση της ανάπτυξης του προγράμματος, εάν παρουσιαστούν λάθη, όλες οι εντολές του πηγαίου προγράμματος, πρέπει να ξαναπεράσουν επιτυχώς από τη διαδικασία της μεταγλώττισης και της σύνδεσης. Η διαδικασία επαναλαμβάνεται, μέχρι να διορθωθούν όλα τα λάθη (συντακτικά και λογικά).

Διερμηνευτής:
(+):

- Η χρήση διερμηνευτή έχει το πλεονέκτημα της άμεσης εκτέλεσης κάθε εντολής και συνεπώς και της άμεσης διόρθωσής της.

(-):

- Ο διερμηνευτής δεν παράγει εκτελέσιμο πρόγραμμα, ανεξάρτητο από το πηγαίο πρόγραμμα, όπως ο μεταγλωττιστής. Συνεπώς, για κάθε εκτέλεση του προγράμματος επαναλαμβάνεται η διαδικασία της μετάφρασης και της εκτέλεσης όλων των εντολών του πηγαίου προγράμματος μία προς μία.
- Η διαδικασία εκτέλεσης του προγράμματος με διερμηνευτή, καθίσταται πιο αργή από εκείνη του ισοδύναμου εκτελέσιμου προγράμματος που παράγει ο μεταγλωττιστής.

Τι συμβαίνει στα σύγχρονα προγραμματιστικά περιβάλλοντα:
Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρουσιάζονται συνήθως με μεικτές υλοποιήσεις, όπου χρησιμοποιείται διερμηνευτής κατά τη φάση δημιουργίας του προγράμματος και μεταγλωττιστής για την τελική έκδοση και εκμετάλλευση του προγράμματος.

Τι είναι ο συντάκτης:
Ο συντάκτης (editor) είναι ένα ειδικό πρόγραμμα που χρησιμοποιείται για την αρχική σύνταξη των προγραμμάτων και τη διόρθωσή τους. Ο συντάκτης είναι ουσιαστικά ένας μικρός επεξεργαστής κειμένου, με δυνατότητες όμως που διευκολύνουν τη γρήγορη γραφή των εντολών των προγραμμάτων.

Ποια είδη λαθών εμφανίζονται κατά τη δημιουργία και εκτέλεση των προγραμμάτων:
Υπάρχουν δύο είδη λαθών:

1. **Συντακτικά**:
   Οφείλονται συνήθως σε αναγραμματισμούς εντολών, παραλείψεις δηλώσεων μεταβλητών κτλπ. Εμφανίζονται στο στάδιο της μεταγλώττισης. Είναι αρκετά εύκολο να διορθωθούν. Τα σύγχρονα προγραμματιστικά περιβάλλοντα, έχουν τη δυνατότητα να τα ανιχνεύουν με ακρίβεια, αλλά και να υποδεικνύουν πως να διορθωθούν στον προγραμματιστή.
2. **Λογικά**:
   Οφείλονται σε σφάλματα στην υλοποίηση του αλγορίθμου. Εμφανίζονται μόνο στη φάση της εκτέλεσης. Είναι πολύ πιο δύσκολο να διορθωθούν. Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρέχουν στον προγραμματιστή και εργαλεία εκσφαλμάτωσης (debugger).

Κεφάλαιο 9
Πίνακες

Πότε πρέπει να χρησιμοποιούνται οι πίνακες:

(+):
- Η χρήση πινάκων είναι ένας βολικός τρόπος για τη διαχείριση πολλών δεδομένων ιδίου τύπου.

(-):
- Οι πίνακες **απαιτούν μνήμη**. Κάθε πίνακας δεσμεύει από την αρχή του προγράμματος πολλές θέσεις μνήμης (στατικοί πίνακες). Η άσκοπη χρήση τους μπορεί να δεσμεύσει μεγάλο τμήμα της κύριας μνήμης (RAM) του υπολογιστή και να οδηγήσει σε αδυναμία εκτέλεσης του προγράμματος ή εκκίνησης άλλων προγραμμάτων.
- Οι πίνακες **περιορίζουν (στατικοί πίνακες) τις δυνατότητες** του προγράμματος. Αυτό γιατί εμείς εξετάζουμε μόνο τους στατικούς πίνακες, όπου το μέγεθος τους πρέπει να είναι δηλώνεται στην αρχή του προγράμματος και δεν προσαρμόζεται κατά τη διάρκεια εκτέλεσής του. Αυτό βέβαια δεν ισχύει ως μειονέκτημα στην περίπτωση των δυναμικών πινάκων.

Ποιες είναι οι τυπικές επεξεργασίες πινάκων:
1. Υπολογισμός αθροισμάτων στοιχείων του πίνακα.
2. Εύρεση μεγίστου ή ελαχίστου στοιχείου.
3. Ταξινόμηση των στοιχείων του πίνακα.
4. Αναζήτηση ενός στοιχείου του πίνακα.
5. Συγχώνευση πινάκων.

Στη συγχώνευση 2 ή περισσοτέρων πινάκων, ο νέος πίνακας που θα προκύψει πρέπει να διατηρεί τις ιδιότητες των πινάκων από τους οποίους προήλθε, π.χ. αν αυτοί ήταν ταξινομημένοι, πρέπει και ο νέος πίνακας να είναι ταξινομημένος.

Κεφάλαιο 10
Υποπρογράμματα

Τι είναι το υποπρόγραμμα:
Είναι ένα τμήμα προγράμματος που επιτελεί αυτόνομο έργο και έχει γραφεί χωριστά από το υπόλοιπο πρόγραμμα.

Ποια είναι τα χαρακτηριστικά των υποπρογραμμάτων:
1. Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο (εδώ δεν εννοούμε τον αριθμό των παραμέτρων στην είσοδο ή τον αριθμό των αποτελεσμάτων στην έξοδο του υποπρογράμματος, αλλά ότι ενεργοποιείται μόνον με έναν τρόπο, δηλ. κλήση και είσοδος παραμέτρων και απενεργοποιείται με έναν μόνο τρόπο, δηλ. τέλος του υποπρογράμματος και έξοδος αποτελεσμάτων).
2. Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα. (Αν και πρακτικά αυτό είναι δύσκολο να επιτευχθεί, ουσιαστικά συνίσταται στην αυτόνομη σχεδίαση, ανάπτυξη και συντήρηση κάθε υποπρογράμματος)
3. Κάθε υποπρόγραμμα δεν πρέπει να είναι πολύ μεγάλο. (ώστε να είναι εύκολα κατανοητό και να μπορεί να ελέγχεται. Η μεθοδολογία για την επίτευξη αυτού είναι κάθε πρόγραμμα να επιτελεί μόνο μια λειτουργία διαφορετικά πρέπει να διασπάται περαιτέρω.)

Ποια τα πλεονεκτήματα του τμηματικού προγραμματισμού
Αναφέρονται στην σελίδα 30, του παρόντος εγγράφου.

Τι είναι η συνάρτηση
Η **συνάρτηση** είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή (αριθμητική, χαρακτήρα ή λογική) η οποία αποθηκεύεται σε μία μεταβλητή με το όνομα της συνάρτησης (όπως οι μαθηματικές συναρτήσεις και οι έτοιμες συναρτήσεις της γλώσσας).

Τι είναι η διαδικασία
Η **διαδικασία** είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελεί όλες τις λειτουργίες ενός προγράμματος (εισάγει δεδομένα, κάνει υπολογισμούς και εμφανίζει ή εκτυπώνει αποτελέσματα).

Ποιες οι διαφορές Διαδικασιών – Συναρτήσεων:
1. Τρόπος κλήσης: Οι διαδικασίες καλούνται με τη χρήση της δεσμευμένης λέξης ΚΑΛΕΣΕ, οι συναρτήσεις με χρήση του ονόματός τους.
2. Πλήθος επιστρεφόμενων τιμών: Οι διαδικασίες μπορούν να επιστρέφουν καμία, μία ή περισσότερες τιμές, οι συναρτήσεις επιστρέφουν ακριβώς μία τιμή.
3. Τρόπος επιστροφής τιμών: Οι διαδικασίες επιστρέφουν τιμές (εφόσον επιστρέφουν) μέσω των παραμέτρων τους, οι συναρτήσεις επιστρέφουν μία τιμή μέσω του ονόματος τους. Άρα το όνομα μιας συνάρτησης είναι ουσιαστικά η παράμετρος εξόδου.
4. Δυνατότητες: Οι διαδικασίες μπορούν να εκτελέσουν όλες τις λειτουργίες που μπορεί να εκτελέσει ένα πρόγραμμα, ενώ οι συναρτήσεις υπολογίζουν και επιστρέφουν μόνο μία τιμή.

Προσοχή: Μία συνάρτηση είναι δυνατόν να εκφραστεί μέσω μιας διαδικασίας μια και η τελευταία είναι ένα πιο γενικό υποπρόγραμμα ενώ το αντίστροφο δεν είναι πάντοτε δυνατό.

Παράμετροι

Τι ονομάζουμε παράμετρο;
Μία παράμετρος είναι μία μεταβλητή που επιτρέπει το πέρασμα της τιμής της από ένα τμήμα προγράμματος σε ένα άλλο.

Η λίστα παραμέτρων σε ένα υποπρόγραμμα είναι υποχρεωτική;
Η λίστα παραμέτρων δεν είναι υποχρεωτική. Μία διαδικασία θα μπορούσε να διαβάζει τα δεδομένα της, ενώ μία συνάρτηση θα μπορούσε να δημιουργεί δεδομένα μόνη της π.χ. με τη χρήση γεννήτριας τυχαίων αριθμών.

Ποια η χρήση των πραγματικών και τυπικών παραμέτρων;
Η λίστα των τυπικών παραμέτρων (formal parameter list) καθορίζει τις παραμέτρους στη δήλωση του υποπρογράμματος.
Η λίστα των πραγματικών παραμέτρων (actual parameter list) καθορίζει τις παραμέτρους στην κλήση του υποπρογράμματος.

Ορίσματα (!):
Ναι, μερικές γλώσσες προγραμματισμού ονομάζουν ορίσματα τις τυπικές παραμέτρους και απλά παραμέτρους τις πραγματικές παραμέτρους.

Ποιους κανόνες πρέπει να ακολουθούν οι λίστες παραμέτρων;
Οι λίστες των παραμέτρων πρέπει να ακολουθούν τους εξής κανόνες:

1. Ο αριθμός των πραγματικών και τυπικών παραμέτρων πρέπει να είναι ο 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 ίστας των πραγματικών παραμέτρων κοκ.
3. Η τυπική παράμετρος και η αντίστοιχη της πραγματική πρέπει να είναι του ιδίου τύπου.

Εμβέλεια μεταβλητών - σταθερών

Τι ονομάζεται εμβέλεια (scope);
Εμβέλεια ονομάζεται το τμήμα του προγράμματος που ισχύουν οι μεταβλητές

Ποια η εμβέλεια (ισχύς) των μεταβλητών στα προγράμματά μας σε ΓΛΩΣΣΑ:
Όλες οι μεταβλητές ενός υποπρογράμματος ή προγράμματος έχουν εμβέλεια μόνο εντός του προγράμματος ή υποπρογράμματος ή αλλιώς, όπως λέγεται, είναι **τοπικές** μεταβλητές. Γι' αυτό και το υποπρόγραμμα έχει ξεχωριστό τμήμα δηλώσεων.

Αυτό πρακτικά σημαίνει ότι η τιμή μιας μεταβλητής του υποπρογράμματος (ακόμη κι αν υπάρχει μεταβλητή με το ίδιο όνομα στο κύριο πρόγραμμα) είναι γνωστή μόνο μέσα στο υποπρόγραμμα.

Περί εμβέλειας στις γλώσσες προγραμματισμού
Πολλές γλώσσες προγραμματισμού επιτρέπουν τη χρήση των μεταβλητών και των σταθερών, όχι μόνο στο τμήμα προγράμματος που δηλώνονται, αλλά και σε άλλα ή ακόμη και σε όλα τα υπόλοιπα υποπρογράμματα. Αυτό που καθορίζει την περιοχή που ισχύουν οι μεταβλητές και οι σταθερές είναι η εμβέλεια των μεταβλητών της συγκεκριμένης γλώσσας. Στις διάφορες γλώσσες προγραμματισμού συναντάμε τα εξής είδη εμβέλειας:

1. **Απεριόριστη εμβέλεια.**
Όλες οι μεταβλητές και όλες οι σταθερές είναι γνωστές και μπορούν να χρησιμοποιούνται σε οποιοδήποτε τμήμα του προγράμματος, άσχετα που δηλώθηκαν. Όλες οι μεταβλητές είναι **καθολικές**.

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

2. **Περιορισμένη εμβέλεια**
Όλες τις μεταβλητές που χρησιμοποιούνται σε ένα τμήμα προγράμματος, πρέπει να έχουν δηλωθεί σε αυτό το τμήμα. Όλες οι μεταβλητές είναι τοπικές, ισχύουν δηλαδή για το υποπρόγραμμα στο οποίο δηλώθηκαν. Στη ΓΛΩΣΣΑ έχουμε περιορισμένη εμβέλεια.

Τα πλεονεκτήματα της περιορισμένης εμβέλειας είναι η απόλυτη αυτονομία όλων των υποπρογραμμάτων και η δυνατότητα να χρησιμοποιείται οποιοδήποτε όνομα, χωρίς να ενδιαφέρει αν το ίδιο χρησιμοποιείται σε άλλο υποπρόγραμμα.

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

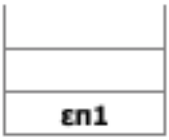
Η μερικώς περιορισμένη εμβέλεια προσφέρει μερικά πλεονεκτήματα στον πεπειραμένο προγραμματιστή, αλλά για τον αρχάριο περιπλέκει το πρόγραμμα δυσκολεύοντας την ανάπτυξή του.

Ποια η χρήση της στοίβας στην κλήση διαδικασιών:
Η έννοια της στοίβας είναι πολύ χρήσιμη στο λογισμικό των γλωσσών προγραμματισμού.
Όταν στο κυρίως πρόγραμμα μας έχουμε κλήση μιας διαδικασίας ή συνάρτησης, ο έλεγχος όπως λέμε περνάει στο υποπρόγραμμα. Εκείνη τη στιγμή και για όσο χρονικό διάστημα το υποπρόγραμμα λειτουργεί (τρέχει), το κυρίως πρόγραμμα δεν εκτελείται. Όταν όμως το υποπρόγραμμα τερματίσει, τότε ο έλεγχος περνάει ξανά στο κυρίως πρόγραμμα (εκτός αν εν τω μεταξύ έχουν προηγηθεί κλήσεις άλλων υποπρογραμμάτων), και η εκτέλεση του κυρίως προγράμματος πρέπει να συνεχίσει από την αμέσως επόμενη εντολή από αυτήν που περιείχε την κλήση στο υποπρόγραμμα.
Αυτή η εντολή (ή η διεύθυνσή της στη μνήμη καλύτερα) ονομάζεται διεύθυνση επιστροφής και αποθηκεύεται (**ωθείται**) από το μεταφραστή σε μια στοίβα που ονομάζεται **στοίβα χρόνου εκτέλεσης** τη στιγμή που γίνεται η κλήση στο υποπρόγραμμα.
Όταν τελειώσει η εκτέλεση του υποπρογράμματος, η διεύθυνση επιστροφής λαμβάνεται (**απωθείται**) από τη στοίβα και έτσι ο έλεγχος μεταφέρεται πάλι στο κυρίως πρόγραμμα.
Αυτή η τεχνική εφαρμόζεται πάντα όταν το κυρίως πρόγραμμα καλεί ένα υποπρόγραμμα, ή όταν ένα υποπρόγραμμα καλεί άλλο υποπρόγραμμα κοκ.

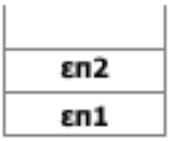
Παράδειγμα:
Το κυρίως πρόγραμμα καλεί μια διαδικασία Α, η οποία με τη σειρά της καλεί μία άλλη διαδικασία Β.

	Πρόγραμμα Τεστ		Διαδικασία Διαδ_Α	Διαδικασία Διαδ_Β
	.....		.....	.....
	ΚΑΛΕΣΕ Διαδ_Α		.....	.....
επ1	.....		ΚΑΛΕΣΕ Διαδ_Β	.....
	.....	επ2	.....	.....
	Τέλος_Προγράμματος		Τέλος_Διαδικασίας	Τέλος_Διαδικασίας

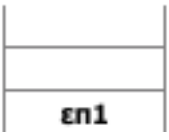
Όταν το κυρίως πρόγραμμα καλέσει τη Διαδ_Α, τότε η διεύθυνση της επόμενης εντολής επ1, ωθείται στη στοίβα χρόνου εκτέλεσης η οποία θα έχει τη μορφή:



Καθώς εκτελείται η Διαδ_Α και τη στιγμή που θα καλέσει τη Διαδ_Β, η διεύθυνση της επόμενης εντολής της, δηλ. επ2 ωθείται και αυτή στη στοίβα χρόνου εκτέλεσης η οποία έχει πλέον τη μορφή:



Μετά το τέλος εκτέλεσης της Διαδ_Β, από τη στοίβα απωθείται πρώτο το επ2, οπότε ο έλεγχος επιστρέφει στη Διαδ_Α και η εκτέλεση της συνεχίζεται. Η μορφή που έχει η στοίβα χρόνου εκτέλεσης θα είναι:



Αφού τελειώσει και η Διαδ_Α, απωθείται από τη στοίβα το επ1, οπότε ο έλεγχος επιστρέφει στο κυρίως πρόγραμμα και παραμένει εκεί μέχρι να τελειώσει η εκτέλεσή του. Η στοίβα πλέον είναι άδεια.

Μεθοδολογία (για δημιουργία υποπρογραμμάτων)
Χρησιμοποιούμε διαδικασίες και συναρτήσεις όταν:

1. η εκφώνηση της άσκησης καθορίζει κάτι τέτοιο και
2. μια σειρά από εντολές θα πρέπει να εκτελεστεί πολλές φορές με διαφορετικές τιμές εισόδου κάθε φορά (γι' αυτό και δεν μας καλύπτει η δομή επανάληψης)

Γενικά είναι ασφαλέστερο να ακολουθείτε τα εξής:

1. Επειδή το κύριο πρόγραμμα πρέπει να γραφεί πριν από διαδικασίες (ή συναρτήσεις), κάτι που εν γένει είναι δύσκολο, σχηματίστε στο πρόχειρο διαδικασίες και συναρτήσεις και αφού γράψετε το κύριο πρόγραμμα καθαρογράψτε τις διαδικασίαες (ή συναρτήσεις).
2. Αφήστε τη συμπλήρωση των παραμέτρων για το τέλος εάν αυτές δεν είναι προφανείς.
3. Αν η εκφώνηση δεν διευκρινίζει ποιο είδος υποπρογράμματος να χρησιμοποιήσετε, προτιμήστε τη διαδικασία ώστε να είστε καλυμμένοι.
4. Σε έτοιμους αλγόριθμους όπου το κύριο πρόγραμμα καλεί ένα υποπρόγραμμα, εκτελούμε κανονικά τις πράξεις στο κύριο πρόγραμμα και στο σημείο που καλείται το υποπρόγραμμα μεταφερόμαστε εκεί, κάνουμε τις πράξεις και επιστρέφουμε την τιμή ξανά στο κύριο πρόγραμμα.