

Μωυσίδης Βασίλης

# Δομημένος Προγραμματισμός



Σημειώσεις για το Πανελλαδικά εξεταζόμενο μάθημα  
της Γ' Τάξης του ΕΠΑΛ

## Πρόλογος – εξήγηση

Αυτό που έχεις στα χέρια σου είναι ένα βοήθημα για να καταλάβεις καλύτερα το συγκεκριμένο μάθημα. Πιθανότατα να έχει λάθη και σίγουρα έχει παραλείψεις καθώς καλύπτει την ύλη 130 σελίδων του βιβλίου.

Το έγγραφο για να σε βοηθήσει να καταλάβεις πιο εύκολα κάποια πράγματα στο βιβλίο και σε καμιά περίπτωση για να το αντικαταστήσει, καθώς τίποτα δεν μπορεί να αντικαταστήσει το επίσημο βιβλίο του μαθήματος.

Για αυτό το λόγο ο μόνος τρόπος που προτείνω να το διαβάσεις είναι παράλληλα με το βιβλίο.

Φιλάκια.

## Κεφάλαιο 1 και 2

### Κατανόηση, Καθορισμός και Δομή προβλήματος - Καθορισμός απαιτήσεων

#### Κατανόηση προβλήματος

Η διαδικασία κατά την οποία με σαφήνεια και πληρότητα διακρίνουμε και συσχετίζουμε τα δεδομένα και τα ζητούμενα, προκειμένου να καταλάβουμε πλήρως το περιεχόμενο του προβλήματος, αποσαφηνίζοντας ακόμα και παρερμηνείες που προκύπτουν από τη διατύπωσή του.

#### Καθορισμός απαιτήσεων προβλήματος

Η διαδικασία κατά την οποία προσδιορίζονται τα δεδομένα και τα ζητούμενα και οι σχέσεις με τις οποίες αυτά συνδέονται. Πιο συγκεκριμένα, περιλαμβάνει τη σαφή καταγραφή των δεδομένων, την αναλυτική διατύπωση του προβλήματος, τη λεπτομερειακή περιγραφή των ζητούμενων και τη διερεύνηση των αποτελεσμάτων. Σημαντική διαδικασία στον καθορισμό των δεδομένων αποτελεί ο έλεγχος της ορθότητας και της πληρότητάς τους.

Ο έλεγχος αν η βαθμολογία που εισάγεται είναι μέσα στα **σωστά όρια**, (1 έως 20) αποτελεί **έλεγχο ορθότητας**.

Ο έλεγχος αν έχουν εισαχθεί οι βαθμολογίες **όλων** των μαθημάτων για τον υπολογισμό του Γενικού Μέσου Όρου αποτελεί **έλεγχο πληρότητας**.

#### Δομή προβλήματος

Τα απλούστερα προβλήματα από τα οποία αποτελείται ένα πρόβλημα, η σύνδεση των οποίων μας δίνει το ζητούμενο αποτέλεσμα.

Η διαδικασία ανάλυσης ενός προβλήματος σε απλούστερα προβλήματα καθώς και η περιγραφή του τρόπου σύνδεσης αυτών, έτσι ώστε η σύνθεσή τους να επιλύει το όλο πρόβλημα, καλείται **ανάλυση της δομής του προβλήματος**.

#### Δεδομένα

Το σύνολο εκείνων των στοιχείων που είναι απαραίτητα για την επίλυση του προβλήματος. Τα δεδομένα πρέπει να χαρακτηρίζονται από σαφήνεια, πληρότητα και ακρίβεια.

## Κεφάλαιο 3

### Αλγόριθμοι – Βασικές έννοιες

#### Αλγόριθμος

Η διαδοχική σειρά πεπερασμένου πλήθους λογικών βημάτων τα οποία εκτελούνται σε πεπερασμένο χρόνο και τα οποία απαιτούνται για την επίλυση ενός προβλήματος.

Ένας αλγόριθμος είναι καλός όταν:

- Περιγράφει με ακρίβεια τα δεδομένα που θα χρησιμοποιήσει
- Κάθε βήμα του είναι πλήρως καθορισμένο και σαφές για το τι ακριβώς κάνει
- Δίνει με ακρίβεια τα αποτελέσματα που παράγονται από κάθε βήμα του
- Ολοκληρώνει σωστά τη διαδικασία επίλυσης του προβλήματος με πεπερασμένο πλήθος βημάτων, κάθε ένα από τα οποία εκτελείται σε πεπερασμένο χρόνο

## Κεφάλαιο 4

### Ανάπτυξη αλγορίθμων

#### Ψευδοκώδικας

Η καταγραφή του αλγόριθμου με απλές προτάσεις και σύμβολα που υπακούουν σε κανόνες γραμματικής και συντακτικού, όπως με τις πιο διαδεδομένες γλώσσες προγραμματισμού. Ο ψευδοκώδικας μετατρέπεται σε πρόγραμμα προκειμένου ο αλγόριθμος να εκτελεσθεί σε υπολογιστή.

#### Διάγραμμα ροής

Η χρησιμοποίηση απλών σχημάτων υποστηριζόμενα από απλές λέξεις για την αναπαράσταση συγκεκριμένων λειτουργιών.

#### Βασικές αλγοριθμικές δομές

- **Ακολουθία:** Οι εντολές βρίσκονται σε διαδοχή και εκτελούνται με τη σειρά που είναι γραμμένες
- **Επιλογή:** Δυνατότητα εκτέλεσης μίας ή περισσότερων εντολών ανάλογα με το αποτέλεσμα ελέγχου μιας συνθήκης.
- **Επανάληψη:** Δυνατότητα επανάληψης εκτέλεσης μίας η περισσότερων εντολών, είτε για γνωστό αριθμό επαναλήψεων είτε σύμφωνα με το αποτέλεσμα ελέγχου μιας συνθήκης.

## Κεφάλαιο 5

### Δομημένος Προγραμματισμός

#### Δομημένος Προγραμματισμός

Ο **Δομημένος Προγραμματισμός** είναι μία τεχνική που ακολουθεί ένα συστηματικό τρόπο ανάπτυξης αλγορίθμου που βασίζεται στις τρεις βασικές αλγοριθμικές δομές, ακολουθίας, επιλογής και επανάληψης, έχοντας ως σκοπό τη δημιουργία ενός αλγορίθμου που να αποτελείται από ανεξάρτητα τμήματα.

Τα **πλεονεκτήματα** του Δομημένου Προγραμματισμού είναι:

- Διευκόλυνση στη ανάπτυξη του αλγόριθμου κατά τμήματα
- Ευκολία και ταχύτητα στην κωδικοποίηση
- Καλύτερη ποιότητα προγραμμάτων
- Ευκολία στις διορθώσεις και τη συντήρηση
- Τεκμηρίωση που περιέχεται σχεδόν εξ' ολοκλήρου στο ίδιο το πρόγραμμα

Οι τεχνικές σχεδίασης που χρησιμοποιούνται στο Δομημένο Προγραμματισμό είναι ο **Ιεραρχικός Προγραμματισμός** και ο **Τμηματικός Προγραμματισμός**.

Η σχεδίαση ενός προγράμματος σύμφωνα με τον **Ιεραρχικό Προγραμματισμό** ξεκινάει από πάνω προς τα κάτω. Στα υψηλότερα επίπεδα υπάρχει μία γενικότερη περιγραφή των επεξεργασιών του προγράμματος, κάθε μία από τις οποίες αναλύεται στα κατώτερα

επίπεδα. Η ανάλυση σταματά όταν έχει φτάσει σε τέτοιο βαθμό λεπτομέρειας ώστε να είναι δυνατή η κωδικοποίηση σε μία γλώσσα προγραμματισμού.

Στον **Τμηματικό Προγραμματισμό** ο αλγόριθμός μας αποτελείται από ανεξάρτητα λογικά τμήματα προγράμματος τα οποία είναι λογικές αυτοτελείς προγραμματικές ενότητες που έχουν μία είσοδο και μία έξοδο. Οι επεξεργασίες που αναλύονται στον Ιεραρχικό Προγραμματισμό είναι τέτοια τμήματα προγράμματος (υποπρογράμματα).

Η παράσταση των παραπάνω τεχνικών γίνεται με τα **διαγράμματα HIPO** (Hierarchical Input Processing Output), Ιεράρχηση Εισόδου – Επεξεργασίας – Εξόδου. Τα διαγράμματα αυτά αποτελούνται από τρία τμήματα. Είσοδο, Επεξεργασία και Αποτελέσματα.

## Κεφάλαιο 6 Έλεγχος αλγορίθμων

### Δοκιμή αλγορίθμου

Η διαδικασία κατά την οποία ελέγχουμε έναν αλγόριθμο ως προς την ορθότητα των αποτελεσμάτων του.

Για τον έλεγχο του αλγορίθμου ακολουθούμε τα παρακάτω βήματα:

- Καταγράφουμε εικονικά δεδομένα σύμφωνα με τις απαιτήσεις του προβλήματος
- Υπολογίζουμε χειρογραφικά τα αποτελέσματα σύμφωνα με τα βήματα του αλγορίθμου
- Κωδικοποιούμε τον αλγόριθμο σε μία γλώσσα προγραμματισμού
- Εκτελούμε το πρόγραμμα στον υπολογιστή με τα ίδια δεδομένα
- Συγκρίνουμε τα αποτελέσματα του προγράμματος με αυτά που υπολογίσαμε με χειρογραφικό τρόπο

### Εκσφαλμάτωση

Η διαδικασία ανεύρεσης των λογικών λαθών ενός προγράμματος.

Η εκσφαλμάτωση περιλαμβάνει:

- Διαπίστωση του είδους του λάθους
- Ανεύρεση του ανεξάρτητου τμήματος του αλγορίθμου με τη λανθασμένη λειτουργία
- Ανεύρεση του λάθους μέσα σε αυτό το τμήμα

### Έλεγχος Δεδομένων

Η επιλογή των κατάλληλων δεδομένων κατά τη διαδικασία ελέγχου ενός προγράμματος προκειμένου να καλυφθούν όλες οι δυνατές περιπτώσεις πραγματικών δεδομένων εκτέλεσής του.

### Αξιολόγηση αλγορίθμου

Η διαδικασία με την οποία ελέγχουμε (αξιολογούμε) το βαθμό στον οποίο ικανοποιεί κάποια κριτήρια ένας αλγόριθμος.

Τα κριτήρια με τα οποία αξιολογείται ένας αλγόριθμος είναι:

- Η δυνατότητα που παρέχει για την ελαχιστοποίηση του κόστους ελέγχου της ορθότητάς του
- Η δυνατότητα που παρέχει για εύκολη συντήρηση και ευκολία σε αλλαγές και επεκτάσεις
- Η δυνατότητα που παρέχει για εύκολη ανάπτυξη σε γλώσσα προγραμματισμού, ανάλογα με την υφή του

## Κεφάλαιο 7

### Είδη, Τεχνικές και Περιβάλλοντα προγραμματισμού

#### Τρόποι παράστασης αλγορίθμου

- Φραστικός
- Ψευδοκώδικας
- Λογικό διάγραμμα

#### Κύκλος Ανάπτυξης Προγράμματος

Οι τυποποιημένες διαδικασίες που χρησιμοποιούμε για την επίλυση ενός προβλήματος στον υπολογιστή:

- Σχεδίαση προγράμματος
  - κατανόηση του προβλήματος – Για να το λύσουμε είναι απαραίτητο να το καταλάβουμε
  - μέθοδοι επίλυσής του – Επιλέγουμε τον τρόπο που είναι πιο κατάλληλος και αναπτύσσουμε τον αλγόριθμο
- Λοιπές διαδικασίες
  - κωδικοποίηση – μετάφραση του προγράμματος – Γράφουμε τον κώδικα σε γλώσσα προγραμματισμού και χρησιμοποιούμε το κατάλληλο μεταφραστικό πρόγραμμα για να το μετατρέψει σε γλώσσα μηχανής
  - έλεγχος του προγράμματος
  - τεκμηρίωση του προγράμματος

#### Γλώσσες Προγραμματισμού

##### Γλώσσες Μηχανής

Χρησιμοποιούν εντολές που αποτελούνται από ακολουθίες 0 και 1 (π.χ. 00011001100) σταθερού ή μεταβλητού πλήθους. Τα προγράμματα είναι εξαρτημένα από τη δομή του υπολογιστή. Ο προγραμματισμός σε γλώσσα μηχανής είναι ιδιαίτερα δύσκολος.

##### Συμβολικές γλώσσες ή γλώσσες χαμηλού επιπέδου

Χρησιμοποιούνται μνημονικά ονόματα (ονόματα που θυμάσαι εύκολα) για τις εντολές, που είναι συντομογραφίες της αγγλικής γλώσσας (π.χ. MOV), όπως επίσης και για τις διευθύνσεις της μνήμης (μεταβλητές). Τα προγράμματα είναι εξαρτημένα από τη δομή και την αρχιτεκτονική του υπολογιστή γι' αυτό και οι γλώσσες αυτές ονομάστηκαν χαμηλού επιπέδου. Για την εκτέλεσή τους χρειάζονται ειδικά μεταφραστικά προγράμματα που τα μετατρέπουν σε γλώσσα μηχανής. Ο προγραμματισμός παραμένει αρκετά δύσκολος.

##### Γλώσσες 3ης γενιάς ή γλώσσες υψηλού επιπέδου

Χρησιμοποιούν εντολές που είναι λέξεις κλειδιά της Αγγλικής γλώσσας (π.χ. print, write, repeat) με τις οποίες δημιουργούνται προτάσεις εντολών προγραμματισμού. Ονομάζονται και γλώσσες υψηλού επιπέδου, καθώς είναι πιο κατανοητές για τον άνθρωπο και ο προγραμματισμός πλέον καθίσταται ευκολότερος. Για την εκτέλεσή τους χρειάζονται ειδικά μεταφραστικά προγράμματα που τα μετατρέπουν σε γλώσσα μηχανής.

Τα χαρακτηριστικά τους είναι:

- Έχουν καθορισμένο σύνολο από λέξεις, σύμβολα και προτάσεις
- Οι εντολές τους μεταφράζονται σε πολλές αντίστοιχες εντολές γλώσσας μηχανής
- Έχουν γραμματικούς και συντακτικούς κανόνες που πρέπει να γνωρίζει ο προγραμματιστής
- Είναι συνήθως ανεξάρτητες από τη δομή και την αρχιτεκτονική του υπολογιστή
- Συνοδεύονται από μία βιβλιοθήκη υποπρογραμμάτων που μπορεί ο κάθε προγραμματιστής να χρησιμοποιήσει στα προγράμματά του αλλά και να εμπλουτίσει

#### **Εφαρμογές και γλώσσες προγραμματισμού:**

- Επιστημονικές εφαρμογές: FORTRAN, C, APL
- Εμπορικές εφαρμογές: COBOL, RPG
- Εκπαιδευτικές εφαρμογές: BASIC, LOGO
- Ειδικές εφαρμογές: LISP, PROLOG
- Επιστημονικές – εμπορικές εφαρμογές: PL-1, PASCAL

#### **Γλώσσες 4ης γενιάς**

Παρέχουν ένα δομημένο τρόπο ανάπτυξης πληροφοριακών συστημάτων. Είναι γλώσσες υψηλής παραγωγικότητας καθώς ο προγραμματισμός τους απαιτεί λιγότερο χρόνο από τις γλώσσες 3ης γενιάς και οι χρήστες τους δεν χρειάζεται να είναι προγραμματιστές. Είναι διαλογικές, καθοδηγούν το χρήστη μέσω μηνυμάτων στις διαδικασίες που θέλει να εκτελέσει. Χρησιμοποιούν δυναμικές εντολές για τη δημιουργία αρχείων και για σχεδίαση οθονών και εκτυπώσεις για τη διαχείριση και εκμετάλλευσή τους. Ο χρήστης καθορίζει το είδος της επεξεργασίας και όχι τον τρόπο υλοποίησής της. Οι γλώσσες αυτές αποτελούν μέρος ενός ολοκληρωμένου πακέτου Διοίκησης Βάσεων Δεδομένων (DBMS). Οι εντολές τους είναι πιο περιεκτικές και τα προγράμματα που δημιουργούνται είναι μικρότερα και κατ' επέκταση ευκολότερα στη συντήρησή τους.

#### **Είδη γλωσσών προγραμματισμού**

##### **Διαδικασιακός Προγραμματισμός**

Είναι ο προγραμματισμός ο οποίος γίνεται σε μία γλώσσα προγραμματισμού που είναι εξαρτημένη από τη διαδικασία, δηλαδή δίνει έμφαση στον προγραμματισμό υπολογιστικών και λογικών διαδικασιών που απαιτούνται για την επίλυση του προβλήματος.

##### **Δομημένος Προγραμματισμός**

Είναι ο προγραμματισμός που ακολουθεί ένα προκαθορισμένο σχέδιο για τη δομημένη σχεδίαση και έλεγχο ενός προγράμματος που αποτελείται από ανεξάρτητα τμήματα. Πέρα από τις βασικές αλγοριθμικές δομές, διαδοχή, απλή επιλογή και επανάληψη, χρησιμοποιεί και τις τεχνικές ανάπτυξης και σχεδίασης του ιεραρχικού και του τμηματικού προγραμματισμού.

##### **Παράλληλος Προγραμματισμός**

Οφείλει την καθιέρωσή του στην εξέλιξη του υλικού, καθώς η εμφάνιση της αρχιτεκτονικής των πολλών επεξεργαστών είχε ως συνέπεια την ανάπτυξη παράλληλων αλγορίθμων που επέβαλαν την καθιέρωση του παράλληλου προγραμματισμού. Πέρα από τις δομές του παραδοσιακού διαδικασιακού προγραμματισμού, ο Παράλληλος Προγραμματισμός διαθέτει δομές που επιτρέπουν την ταυτόχρονη εκτέλεση διαδικασιών από διαφορετικούς επεξεργαστές.

### Αντικειμενοστρεφής Προγραμματισμός

Ενσωματώνει τα δεδομένα και τον τρόπο χειρισμού τους σε αντικείμενα. Κάθε αντικείμενο αποτελείται από μία σειρά δεδομένων που αποτελούν και τα χαρακτηριστικά του, και από μία σειρά μεθόδων ή ενεργειών που σχετίζονται με την επεξεργασία των δεδομένων και καθορίζουν τη συμπεριφορά του αντικειμένου στο πρόγραμμα. Τα αντικείμενα μπορούν να κληρονομήσουν τα χαρακτηριστικά και τη συμπεριφορά τους από άλλα αντικείμενα, δημιουργώντας έτσι ιεραρχικές κλάσεις αντικειμένων με κοινά αλλά και επιπρόσθετα χαρακτηριστικά.

### Συναρτησιακός Προγραμματισμός

Οι εντολές και οι δομές ελέγχου είναι συναρτήσεις. Ως ορίσματα των συναρτήσεων μπορεί να είναι δεδομένα ή άλλες συναρτήσεις. Ο συναρτησιακός προγραμματισμός βασίζεται στην έννοια της συνάρτησης όπως τη γνωρίζουμε από τα Μαθηματικά. Μέσα από τις συναρτήσεις εφαρμόζονται τα δεδομένα και υλοποιούνται οι διάφορες δομές (διακλάδωση, επανάληψη κλπ).

### Λογικός Προγραμματισμός

Σε αντίθεση με το διαδικασιακό προγραμματισμό ο Λογικός Προγραμματισμός στηρίζεται στη γνώση. Η γνώση περιγράφεται με τη μορφή γεγονότων και κανόνων, ενώ ορίζεται μία πρόταση-στόχος προς απόδειξη. Το πρόγραμμα εφαρμόζει τους κανόνες στα γεγονότα προκειμένου να επιτευχθεί η απόδειξη της πρότασης-στόχου.

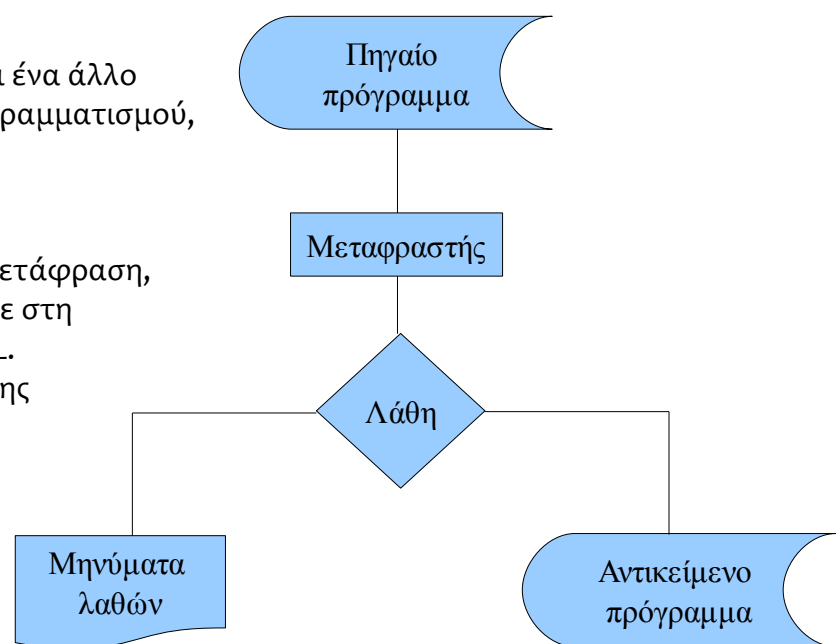
## Προγραμματιστικά Περιβάλλοντα

### Μεταγλωττιστής

Ειδικό πρόγραμμα που μεταφράζει ένα άλλο πρόγραμμα, από μία γλώσσα προγραμματισμού, σε γλώσσα μηχανής.

**Πηγαίο** είναι το πρόγραμμα προς μετάφραση, π.χ. Ένα πρόγραμμα που γράφουμε στη γλώσσα προγραμματισμού PASCAL.

**Αντικείμενο** είναι το αποτέλεσμα της μετάφρασης. (γλώσσα μηχανής)



Οι Μεταγλωττιστές διακρίνονται σε:

- Compiler (Μεταγλωττιστής γλώσσας υψηλού επιπέδου)
- Interpreter (Διερμηνέας, δηλαδή Μεταγλωττιστής με ταυτόχρονη εκτέλεση του προγράμματος)

### Compiler

Κάνει έλεγχο ορθογραφικών και συντακτικών λαθών, και αν δεν υπάρχουν μεταφράζει **ολόκληρο** το πρόγραμμα σε γλώσσα μηχανής.

**Ο Compiler:**

- Κάνει συντακτικό και γραμματικό έλεγχο του πηγαίου προγράμματος και εκτυπώνει τα λάθη
- Δεσμεύει την απαιτούμενη μνήμη στον υπολογιστή για την εκτέλεση του προγράμματος
- Αποθηκεύει το αντικείμενο πρόγραμμα στο δίσκο
- Ανάλογα με την επιλογή του χρήστη παράγει ένα πρόγραμμα σε γλώσσα μηχανής και ένα πρόγραμμα σε γλώσσα χαμηλού επιπέδου
- Παράγει πίνακες στη μνήμη που είναι απαραίτητοι για τη μετάφραση το προγράμματος

**Interpreter**

Κάνει έλεγχο ορθογραφικών και συντακτικών λαθών για κάθε μία εντολή τη φορά, και αν δεν υπάρχει λάθος τη μεταφράζει σε κώδικα μηχανής και την εκτελεί.

Ο interpreter διαφέρει από τον compiler στα εξής χαρακτηριστικά:

- Ελέγχει κάθε μία εντολή τη φορά και όχι ολόκληρο το πηγαίο πρόγραμμα
- Η διαδικασία μετάφρασης-ελέγχου επαναλαμβάνεται κάθε φορά που εκτελείται το πρόγραμμα (χρονοβόρο)
- Γίνεται ευκολότερα ο έλεγχος και η διόρθωση των λαθών καθώς υπάρχει άμεση πληροφόρηση του προγραμματιστή ο οποίος μπορεί να το διορθώσει και να συνεχίσει με την εκτέλεση του προγράμματος
- Πιο αργή εκτέλεση του προγράμματος

**Γραφή Εκτέλεση Προγράμματος**

Το πρόγραμμα γράφεται με τον εκδότη – διορθωτή (editor) και παράγεται το πηγαίο πρόγραμμα (source). Ο μεταγλωττιστής ελέγχει το πηγαίο πρόγραμμα και αν υπάρχουν λάθη εμφανίζει τα αντίστοιχα μηνύματα και η διαδικασία επιστρέφει στον εκδότη για να διορθωθούν. Αν δεν υπάρχουν λάθη, δημιουργεί το αντικείμενο πρόγραμμα που αν δεν χρησιμοποιεί άλλα υποπρογράμματα είναι εκτελέσιμο. Αν χρησιμοποιεί τότε ο συνδέτης (linker) τα συνδέει και παράγει το εκτελέσιμο πρόγραμμα. Η διαδικασία είναι:

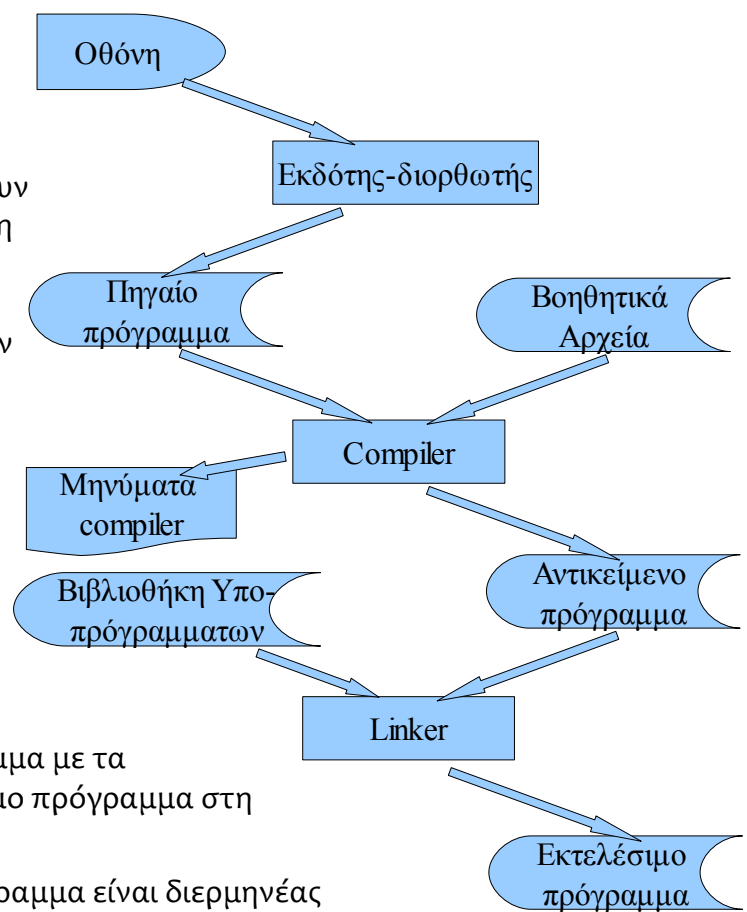
**γραφή – μεταγλώττιση – σύνδεση - εκτέλεση**

Αν χρησιμοποιείται φορτωτής (loader) αντί για τον συνδέτη η διαδικασία είναι:

**γραφή – μεταγλώττιση – φόρτωση**

Ο φορτωτής συνδέει το αντικείμενο πρόγραμμα με τα υποπρογράμματα και φορτώνει το εκτελέσιμο πρόγραμμα στη μνήμη για να εκτελεστεί.

Στην περίπτωση που το μεταφραστικό πρόγραμμα είναι διερμηνέας η μεταγλώττιση, η σύνδεση και η εκτέλεση γίνονται εντολή – εντολή.



## Κεφάλαιο 8

### Η γλώσσα PASCAL

Στα ιδιαίτερα χαρακτηριστικά της γλώσσας PASCAL συγκαταλέγονται οι δυνατότητες:

- Δημιουργίας από τον προγραμματιστή δικών του τύπων δεδομένων
- Χρήσης μεταβλητών τύπου δείκτη (pointer) και δυναμικής διαχείρισης της κεντρικής μνήμης
- Χρήσης σύνθετης εντολής (compound statement), δηλαδή η χρήση μιας σειράς εντολών ως μία εντολή (ανάμεσα σε begin ... end).

#### Γλώσσα προγραμματισμού

Ένα σύνολο κανόνων, συμβόλων και ειδικών λέξεων που χρησιμοποιούνται για τη δημιουργία ενός προγράμματος.

#### Συντακτικό (Syntax)

Ένα σύνολο τυπικών κανόνων οι οποίοι προσδιορίζουν πώς γράφονται έγκυρες εντολές σε μία γλώσσα προγραμματισμού.

#### Σημασιολογία (Semantics)

Ένα σύνολο κανόνων οι οποίοι προσδιορίζουν τη σημασία των εντολών που γράφονται σε μία γλώσσα προγραμματισμού.

#### Τύπος Δεδομένων (Data Type)

Μία κατηγορία δεδομένων με ορισμένη απεικόνιση και ένα σύνολο λειτουργιών που μπορούν να εφαρμοστούν στο σύνολο των τιμών τους.

#### Βασικοί τύποι δεδομένων

Οι τύποι δεδομένων προσδιορίζουν τον τρόπο παράστασης και επεξεργασίας των δεδομένων. Τους τύπους δεδομένων τους χρησιμοποιούμε για να ορίσουμε τις μεταβλητές και τις συναρτήσεις. Όταν δηλώνουμε τις μεταβλητές και ορίζουμε μία συνάρτηση πρέπει να ορίσουμε τι τύπος δεδομένων είναι.

#### Απλοί ή στοιχειώδεις τύποι δεδομένων

- Ακέραιος τύπος (integer)
- Πραγματικός τύπος (real)
- Λογικός (boolean)
- Χαρακτήρας (char)

#### Σύνθετοι τύποι δεδομένων

- Αλφαριθμητικός (string)

#### Ακέραιος (integer)

Ο ακέραιος τύπος χρησιμοποιείται για τους ακέραιους θετικούς και αρνητικούς αριθμούς. Το εύρος των τιμών που μπορεί να πάρουν τα δεδομένα εξαρτάται από τον τύπο δεδομένων που έχει δηλωθεί στο πρόγραμμά μας:

- shortint για ακέραιους αριθμούς από -128 έως 127 (1 byte)
- integer για ακέραιους αριθμούς από -32768 έως 32767 (2 byte)
- longint για ακέραιους αριθμούς από -2148483648 έως 2147483647 (4 byte)
- byte για **θετικούς** ακέραιους αριθμούς από 0 έως 255 (1 byte)
- word για **θετικούς** ακέραιους αριθμούς από 0 έως 65535 (2 byte)

Οι επιτρεπτές πράξεις που μπορώ να κάνω στους ακέραιους είναι:

- + πρόσθεση
- - αφαίρεση
- \* πολλαπλασιασμός
- / διαίρεση
- div πηλίκο ακέραιας διαίρεσης  $7 \text{ div } 3 = 2$
- mod υπόλοιπο ακέραιας διαίρεσης  $7 \text{ mod } 3 = 1$

### Πραγματικός (real)

Οι πραγματικοί αριθμοί μπορούν να δηλωθούν ως:

- real (6 byte)
- single (4 byte)
- double (8 byte)
- extended (10 byte)
- comp (8 byte)

Οι επιτρεπτές πράξεις που μπορώ να κάνω στους πραγματικούς είναι:

- + πρόσθεση
- - αφαίρεση
- \* πολλαπλασιασμός
- / διαίρεση

### Προτεραιότητα αριθμητικών τελεστών

Στις αριθμητικές παραστάσεις με πολλούς διαφορετικούς τελεστές υπάρχει συγκεκριμένη σειρά με την οποία εκτελούνται οι πράξεις ανάλογα με την προτεραιότητα των τελεστών: Τελεστές με την υψηλότερη προτεραιότητα:

\*, /, DIV, MOD

Τελεστές με χαμηλότερη προτεραιότητα:

+, -

Δηλαδή πρώτα εκτελούνται οι πολλαπλασιασμοί και οι διαιρέσεις (\*, /, DIV, MOD) και μετά οι προσθέσεις και οι αφαιρέσεις (+, -):

$$8 * 3 + 6 / 3 = 24 + 2 = 26$$

$$3 + 6 \text{ DIV } 2 - 6 - 6 \text{ MOD } 4 = 3 + 3 - 6 - 2 = -2$$

Αν έχουμε τελεστές που έχουν την ίδια προτεραιότητα, τότε οι πράξεις εκτελούνται από αριστερά προς τα δεξιά:

$$3 * 8 / 2 \text{ DIV } 3 \text{ MOD } 3 = 1$$

Προκειμένου να αλλάξουμε τη σειρά εκτέλεσης των πράξεων χρησιμοποιούμε παρενθέσεις:

$$8 * (3 + 6) / 3 = 8 * 9 / 3 = 24$$

### Λογικός (boolean)

Ο λογικός τύπος δεδομένων μπορεί να πάρει μόνο δύο τιμές, **True** (αληθής) και **False** (ψευδής). Οι πράξεις που μπορούμε να κάνουμε είναι:

- `and`    Σύζευξη
- `or`     Διάζευξη
- `xor`    Αποκλειστική διάζευξη
- `not`    Άρνηση

### Πίνακας λογικών πράξεων

A	B	A and B	A or B	A xor B	Not A
True	True	True	True	False	False
True	False	False	True	True	False
False	True	False	True	True	True
False	False	False	False	False	True

Το αποτέλεσμα μιας συνθήκης είναι μία λογική τιμή, π.χ.

```
A > B
a = 5
a < 13
```

Κάθε μία από τις παραπάνω συγκρίσεις έχει ως αποτέλεσμα την τιμή True ή False, ανάλογα με τις τιμές που έχουν οι μεταβλητές, και ονομάζονται **Λογικές εκφράσεις**.

Ο παραπάνω πίνακας έχει εφαρμογή όταν κάνω πολλαπλές συγκρίσεις:

```
(a > 3) and (b > 10) or (c < 20)
(Hlikia >= 18) and (fylo = 'ΑΝΔΡΑΣ')
```

Και στους λογικούς τελεστές υπάρχει προτεραιότητα στις πράξεις:

```
Τελεστές με την υψηλότερη προτεραιότητα:
    and (τελεστής λογικού πολλαπλασιασμού)
Τελεστές με χαμηλότερη προτεραιότητα:
    or, xor (τελεστής λογικής πρόσθεσης)
```

Εφαρμόζοντας τα παραπάνω στην πρώτη συγκριτική παράσταση έχοντας π.χ. ως δεδομένες τις τιμές: a = 5, b= 8, c = 15:

```
(a > 3) and (b > 10) or (c < 20)
  True and False or True
  -----
      False or True
  -----
                True
```

Και σε αυτή την περίπτωση αν θέλω να αλλάξω τη σειρά των πράξεων μπορώ να χρησιμοποιήσω παρενθέσεις.

### Χαρακτήρας (char)

Τα δεδομένα αυτού του τύπου δεδομένων είναι όλα τα γράμματα, οι αριθμοί και τα σύμβολα που υπάρχουν στον κώδικα ASCII. Μία μεταβλητή αυτού του τύπου δεδομένων

μπορεί για παράδειγμα να πάρει μία από τις ακόλουθες τιμές:

'A', 'B', 'J', 'U', 'a', 'c', 'Θ', 'Ψ', 'π', 'ς', '1', '2', '9',  
'&', '/', ')', '%'.  
'

Τα εισαγωγικά (τα αυτάκια) είναι απαραίτητα όταν χρησιμοποιώ δεδομένα τύπου χαρακτήρα, καθώς εάν δεν τα βάλω τότε τα γράμματα θα εκληφθούν από το μεταγλωττιστή (compiler) ως ονόματα μεταβλητών ενώ τα νούμερα ως ακέραιοι αριθμοί.

### Αλφαριθμητικός τύπος (string)

Ο αλφαριθμητικός τύπος δεδομένων χρησιμοποιείται όταν θέλω να αποθηκεύσω και να διαχειριστώ περισσότερους από έναν χαρακτήρες όπως στην περίπτωση που έχω ένα όνομα, μία διεύθυνση ή έναν Ταχυδρομικό κώδικα:

'Βαγγέλης'  
'Καραϊσκάκη 55'  
'81400'

Εννοείται ότι και οι αλφαριθμητικές τιμές πρέπει να είναι μέσα σε εισαγωγικά. Όταν δηλώνω μία μεταβλητή ως αλφαριθμητική, μπορώ αν θέλω να ορίσω το μέγιστο αριθμό των χαρακτήρων που μπορεί να πάρει. Αν δεν το κάνω θεωρείται ως μέγιστο μήκος 255 χαρακτήρες.

### Δομή προγράμματος

Ένα πρόγραμμα στην PASCAL έχει την παρακάτω δομή:

Να γίνει πρόγραμμα που να διαβάζει την τιμή ενός προϊόντος και να εμφανίζει την τελική τιμή του με τον Φ.Π.Α.

```
program proion; {επικεφαλίδα}

uses wincrt;

        {δηλώσεις σταθερών}
const
  fpa=20;

        {δηλώσεις μεταβλητών}
var
  timi,teliki:real;

        {δηλώσεις υποπρογραμμάτων (διαδικασίες/συναρτήσεις) }
procedure diavasetimi(var t:real);
begin
  write('Δώσε την τιμή του προϊόντος: ');
  readln(t);
end;

        {κύριο πρόγραμμα}
begin
  diavasetimi(timi);
  teliki:=timi+timi*fpa/100;
  write('Η τελική τιμή του προϊόντος είναι: ',teliki);
end;
```

## Κεφάλαιο 9 – Βασικές εντολές

### Εντολές εισόδου – εξόδου, εντολή αντικατάστασης

#### Η εντολή write και writeln

Ας ξεκινήσουμε γράφοντας ένα πολύ απλό πρόγραμμα που απλά εμφανίζει ένα μήνυμα. Για να εμφανίσω ένα μήνυμα χρησιμοποιώ την εντολή `write`, π.χ:

```
write('Γεια');
```

Στην οθόνη εμφανίζεται ότι υπάρχει μέσα στην παρένθεση, είτε αυτό είναι σε αλφαριθμητική μορφή μέσα σε μονά εισαγωγικά ('αυτάκια') είτε ως περιεχόμενο μιας μεταβλητής.

Να γίνει πρόγραμμα που να εμφανίζει το μήνυμα: "Γεια σου κόσμε!"

```
program kosmos;           {Μέσα σε αγκύλες μπορώ να γράφω τα σχόλιά μου}
uses wincrt;             {Έτσι μπορώ να εξηγήσω τι κάνω στο πρόγραμμά μου}
begin
  write('Γεια σου κόσμε!'); {βάζω αυτάκια στο μήνυμα}
end.
```

```
Γεια σου κόσμε!
```

Το ίδιο πρόγραμμα θα μπορούσε να γίνει με τη βοήθεια μιας μεταβλητής. Μπορώ δηλαδή την αλφαριθμητική τιμή που θέλω να εμφανίσω να την εκχωρήσω (βάλω) μέσα σε μία μεταβλητή και μετά να εμφανίσω τη μεταβλητή. Αυτό βέβαια προϋποθέτει τη δήλωση της μεταβλητής. Τη μεταβλητή τη δηλώνω προκειμένου ο compiler να γνωρίζει τί τύπου δεδομένα θα αποθηκεύονται σε αυτήν έτσι ώστε να δεσμεύσει την ανάλογη μνήμη. Τις μεταβλητές τις δηλώνω ανάμεσα στη δεσμευμένη λέξη **var** και την εντολή έναρξης (**begin**) του προγράμματος. Το ίδιο πρόγραμμα με μεταβλητή θα γίνει ως εξής:

```
program kosmos;
uses wincrt;
var
  minima:string;        {εδώ δηλώνω τον τύπο δεδομένων της μεταβλητής}
begin
  minima:='Γεια σου κόσμε!';
  write(minima);        {δεν βάζω αυτάκια γιατί εμφανίζω το }
end.                    {περιεχόμενο της μεταβλητής } }
```

```
Γεια σου κόσμε!
```

Για να εκχωρήσω μία τιμή σε μία μεταβλητή χρησιμοποιώ την **εντολή αντικατάστασης :=** την οποία δεν πρέπει να συγχέουμε με το ίσον =

Με την εντολή αντικατάστασης μπορώ να αντικαταστήσω το περιεχόμενο μιας μεταβλητής με:

- μία σταθερή τιμή            `a := 2` ή        `a := 'hello'`
- την τιμή μιας μεταβλητής   `a := b`

- το αποτέλεσμα μιας αριθμητικής παράστασης  $a := (b + 2/3) + 50$

Ας δούμε τώρα πώς εμφανίζονται στην οθόνη μου τα μηνύματα αν προσθέσουμε ένα ακόμα μήνυμα στο πρόγραμμά μας.

Να γίνει πρόγραμμα που να εμφανίζει το μήνυμα: "Γεια σου κόσμε!" και μετά το μήνυμα "άρχισα να προγραμματίζω".

```
program kosmos;
uses wincrt;
begin
  write('Γεια σου κόσμε!');
  write('άρχισα να προγραμματίζω');
end.
```

Γεια σου κόσμε!άρχισα να προγραμματίζω

Βλέπουμε ότι τα μηνύματα εμφανίζονται το ένα κολλητά με το άλλο. Μία παραλλαγή της εντολής εμφάνισης είναι η **writeln**. Η συγκεκριμένη εντολή μετά την εμφάνιση του μηνύματος αλλάζει γραμμή. Εκτελώντας το παραπάνω πρόγραμμα με αυτή την εντολή το αποτέλεσμα θα είναι το εξής:

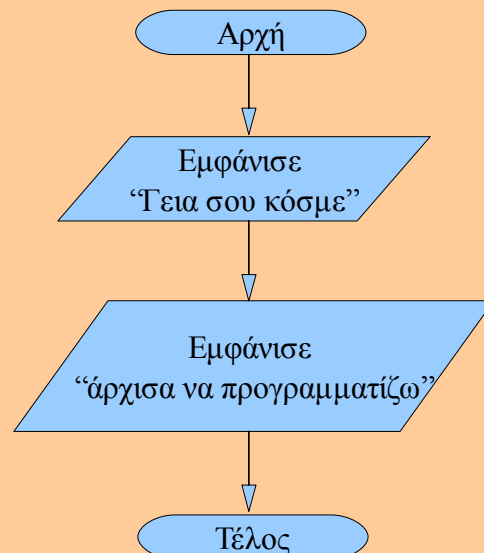
```
program kosmos;
uses wincrt;
begin
  writeln('Γεια σου κόσμε!'); {μετά την εκτέλεση αλλάζει γραμμή}
  write('άρχισα να προγραμματίζω');
end.
```

Γεια σου κόσμε!  
άρχισα να προγραμματίζω

**Το αντίστοιχο πρόγραμμα με ψευδοκώδικα:**

Αρχή  
Εμφάνισε "Γεια σου κόσμε"  
Εμφάνισε "άρχισα να προγραμματίζω"  
Τέλος

**Το αντίστοιχο πρόγραμμα με λογικό διάγραμμα:**



## Η εντολή read και readln

Στις περισσότερες περιπτώσεις τα προγράμματα που δημιουργούμε αλληλεπιδρούν με το χρήστη και δέχονται διάφορα δεδομένα. Με την εντολή read μπορώ να διαβάσω αυτό που πληκτρολογεί ο χρήστης στο πληκτρολόγιο και να το αποθηκεύσω μέσα σε μία μεταβλητή:

```
read(a);
```

Με την παραπάνω εντολή ότι γράφει ο χρήστης (μόλις πατήσει το enter), θα αποθηκευτεί στη μεταβλητή a.

Να γίνει πρόγραμμα που να διαβάζει το όνομα του χρήστη και να εμφανίζει το μήνυμα “Γεια σου” με το όνομα που μόλις διάβασε.

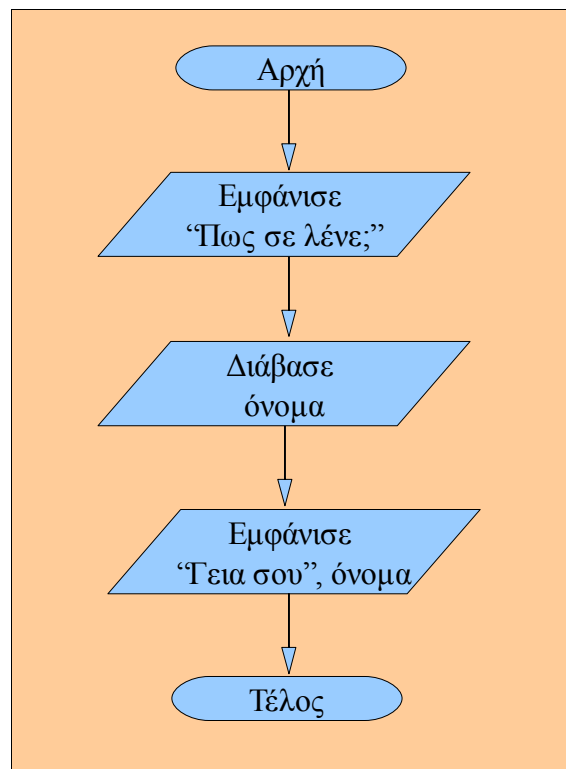
```
program geia_xara;
uses wincrt;
var
  onoma:string;
begin
  writeln('Πως σε λένε;');
  read(onoma);
  write('Γεια σου ',onoma);
end.
```

```
Πως σε λένε;
Γιάννη ↵
Γεια σου Γιάννη
```

Με το ↵ συμβολίζεται το πάτημα του κουμπιού 'Enter'

```
Αρχή
Εμφάνισε “Πως σε λένε;”
Διάβασε όνομα
Εμφάνισε “Γεια σου”, όνομα
Τέλος
```

Στο προηγούμενο παράδειγμα βλέπουμε ότι στην εντολή εμφάνισης του μηνύματος μέσα στην παρένθεση υπάρχουν μία σταθερή αλφαριθμητική τιμή 'Γεια σου ' και μία μεταβλητή onoma. Μπορώ να κάνω όποιους συνδυασμούς θέλω, αρκεί να τα χωρίζω μεταξύ τους με κόμμα. Καλό είναι να βάζω και τα απαραίτητα κενά έτσι ώστε να μην εμφανίζονται στο μήνυμά μου κολλητά το ένα με το άλλο.



Στο επόμενο πρόγραμμα έχουμε ένα ακόμα παράδειγμα τέτοιου μηνύματος.

Να γίνει πρόγραμμα που να διαβάζει το όνομα και την ηλικία του χρήστη και να εμφανίζει το μήνυμα “Είσαι ο (όνομα) και είσαι (ηλικία) ετών”. Σε παρένθεση είναι το όνομα και η ηλικία που διάβασε από το χρήστη.

```
program geia_xara;
uses wincrt;
var
  onoma:string;
  ilikia:integer;
begin
  writeln('Δώσε το όνομα και την ηλικία σου');
  read(onoma,ilikia);
  write('Είσαι ο ',onoma,' και είσαι ',ilikia,' ετών');
end.
```

```
Δώσε το όνομα και την ηλικία σου
Γιάννης↵
18↵
Είσαι ο Γιάννης και είσαι 18 ετών
```

Η εντολή read θα μπορούσε να αντικατασταθεί και από την readln.

Η διαφορά μεταξύ αυτών των δύο εντολών φαίνεται μόνο στην περίπτωση που διαβάζονται πολλές τιμές μαζί. Αν για παράδειγμα πρόκειται να διαβάσω ακέραιες τιμές σε πέντε μεταβλητές σύμφωνα με τις παρακάτω εντολές:

```
read(a,b,c);
read(d,e);
```

και κατά την πληκτρολόγηση δώσω τις ακόλουθες τιμές:

```
15 16 28 33 ↵
23 45 ↵
```

οι μεταβλητές θα πάρουν τις εξής τιμές:

```
a=15 b=16 c=28
d=33 e=23
```

Αν όμως χρησιμοποιήσω την εντολή readln:

```
readln(a,b,c);
readln(d,e);
```

οι μεταβλητές θα πάρουν τις εξής τιμές:

```
a=15 b=16 c=28
d=23 e=45
```

Στην readln όταν διαβάζει πολλές μεταβλητές ταυτόχρονα, αγνοεί οποιαδήποτε επιπλέον τιμή που δεν αντιστοιχεί σε μεταβλητή, ενώ στη read την κρατάει για το επόμενο διάβασμα. Με την readln όταν πατήσω 'Enter', όποια τιμή δεν αντιστοιχεί σε μεταβλητή και περισσεύει, χάνεται. Στη συγκεκριμένη περίπτωση η 4η τιμή που δεν αντιστοιχεί σε μεταβλητή (η 33) χάνεται, ενώ στην προηγούμενη περίπτωση με το read δεν χάθηκε αλλά πήγε στη μεταβλητή d στο αμέσως επόμενο διάβασμα.

## Κεφάλαιο 10 – Εντολές επιλογής και αποφάσεων

### Η εντολή if ... then – Δομή της επιλογής

Με την εντολή if – then υλοποιούμε τη δομή της επιλογής. Τη χρησιμοποιούμε όταν θέλουμε να εκτελεστεί κάποιο κομμάτι κώδικα υπό κάποια προϋπόθεση. Η συνθήκη που ακολουθεί την εντολή if είναι και η προϋπόθεση για να εκτελεστεί ο κώδικας μετά το then.

Να γίνει πρόγραμμα που να διαβάζει το όνομα και την ηλικία του χρήστη και σε περίπτωση που ο χρήστης είναι ενήλικος να εμφανίζει ανάλογο μήνυμα.

```

program enilikos;
uses wincrt;
var
  onoma:string;
  ilikia:integer;
begin
  writeln('Δώσε το όνομα και την ηλικία σου');
  read(onoma,ilikia);
  if ilikia >= 18 then
    write('Είσαι ο ',onoma,' και είσαι ενήλικος');
end.
    
```

```

Δώσε το όνομα και την ηλικία σου
Γιάννης ←
18 ←
Είσαι ο Γιάννης και είσαι ενήλικος
    
```

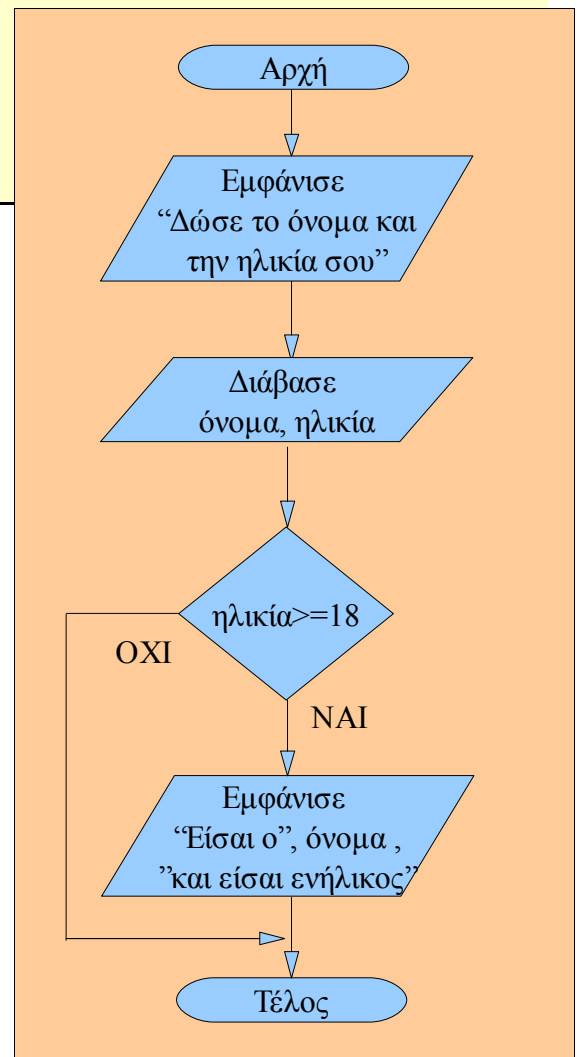
```

Αρχή
Εμφάνισε “Δώσε το όνομα και την ηλικία σου”
Διάβασε όνομα, ηλικία
Αν ηλικία >= 18
Τότε
Εμφάνισε “Είσαι ο”,όνομα,“και είσαι ενήλικος”
Τέλος
    
```

Το παραπάνω μήνυμα εμφανίζεται μόνο στην περίπτωση που ο χρήστης είναι ενήλικος. Σε αντίθετη περίπτωση δεν εμφανίζεται τίποτα.

```

Δώσε το όνομα και την ηλικία σου
Γιάννης ←
17 ←
    
```



Στην περίπτωση που θέλουμε να εμφανίζεται μήνυμα όταν ο χρήστης είναι ανήλικος, θα πρέπει να συμπληρώσουμε την παραπάνω εντολή με το **else**:

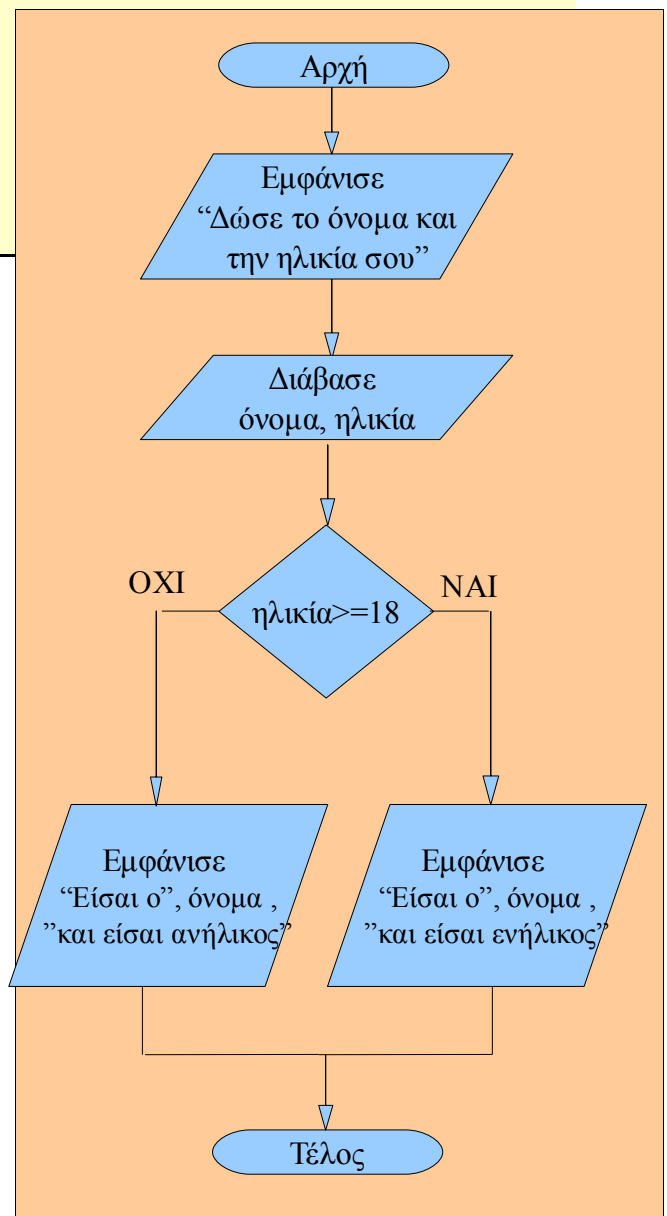
Να γίνει πρόγραμμα που να διαβάζει το όνομα και την ηλικία του χρήστη και να εμφανίζει μήνυμα που να λέει αν είναι ενήλικος ή ανήλικος

```

program enilikos;
uses wincrt;
var
  onoma:string;
  ilikia:integer;
begin
  writeln('Δώσε το όνομα και την ηλικία σου');
  read(onoma,ilikia);
  if ilikia >= 18 then
    write('Είσαι ο ',onoma,' και είσαι ενήλικος')
  else
    write('Είσαι ο ',onoma,' και είσαι ανήλικος')
end.
    
```

Δώσε το όνομα και την ηλικία σου  
 Γιάννης ↵  
 17 ↵  
 Είσαι ο Γιάννης και είσαι ανήλικος

Αρχή  
 Εμφάνισε “Δώσε όνομα και ηλικία”  
 Διάβασε όνομα, ηλικία  
 Αν ηλικία >= 18  
 Τότε  
 Εμφάνισε “Είσαι ο”,όνομα, “και είσαι ενήλικος”  
 Αλλιώς  
 Εμφάνισε “Είσαι ο”,όνομα, “και είσαι ανήλικος”  
 Τέλος



Υπάρχουν και περιπτώσεις όπου έχουμε περισσότερες από δύο επιλογές:

Να γίνει πρόγραμμα που να διαβάσει την ηλικία του χρήστη και να εμφανίζει μήνυμα που να λέει ότι :

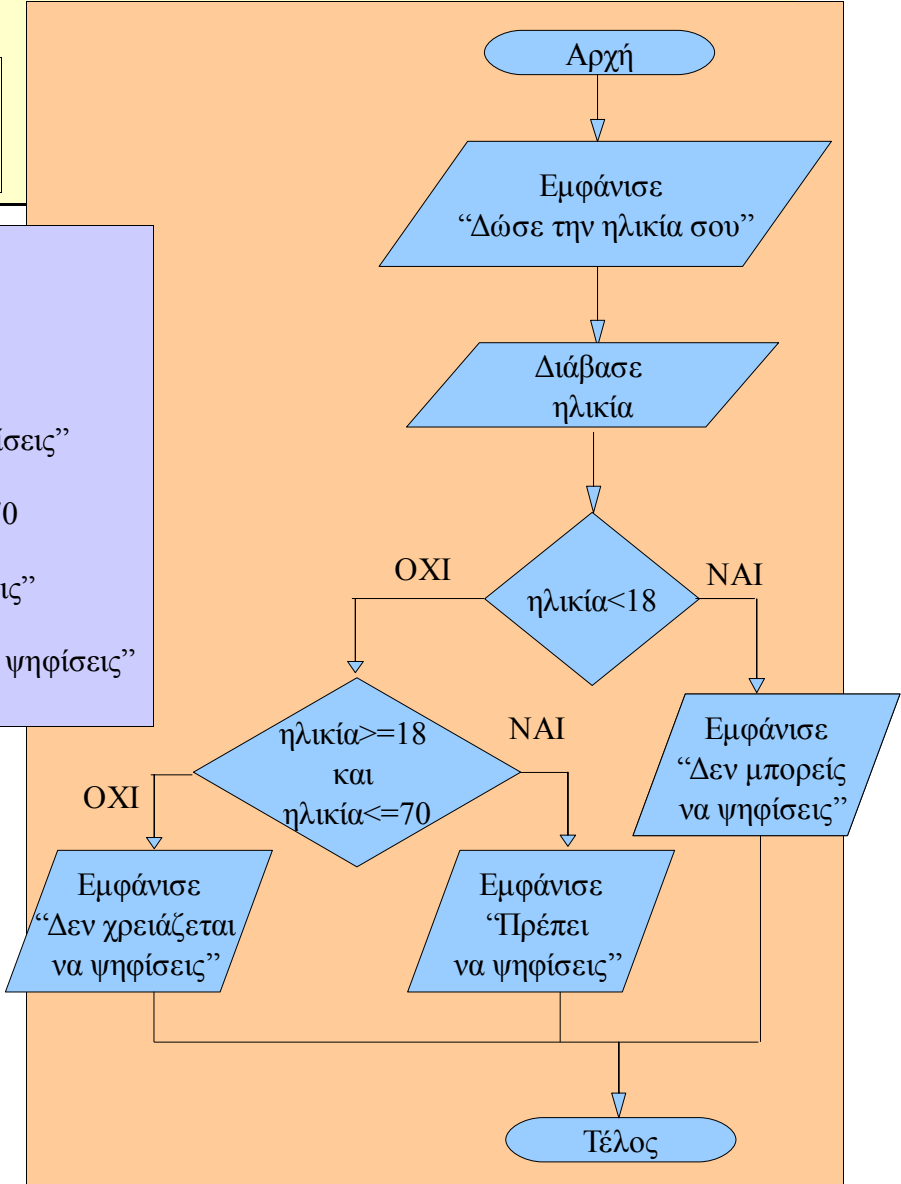
- δεν μπορεί να ψηφίσει αν είναι ανήλικος
- πρέπει να ψηφίσει αν είναι ενήλικος έως 70 ετών
- δεν χρειάζεται να ψηφίσει αν είναι ενήλικος άνω των 70 ετών

```

program ekloges;
uses wincrt;
var
  ilikia:integer;
begin
  writeln('Δώσε την ηλικία σου');
  read(ilikia);
  if ilikia < 18 then
    write('Δεν μπορείς να ψηφίσεις')
  else
    if (ilikia >= 18) and (ilikia <= 70) then
      write('Πρέπει να ψηφίσεις')
    else
      write('Δεν χρειάζεται να ψηφίσεις');
end.
    
```

Δώσε την ηλικία σου  
17 ←  
Δεν μπορείς να ψηφίσεις

Αρχή  
Εμφάνισε “Δώσε την ηλικία σου”  
Διάβασε ηλικία  
Αν ηλικία < 18  
Τότε  
Εμφάνισε “Δεν μπορείς να ψηφίσεις”  
Αλλιώς  
Αν ηλικία >= 18 και ηλικία <= 70  
Τότε  
Εμφάνισε “Πρέπει να ψηφίσεις”  
Αλλιώς  
Εμφάνισε “Δεν χρειάζεται να ψηφίσεις”  
Τέλος



Να γίνει πρόγραμμα που να διαβάζει δύο αριθμούς και τον κωδικό της πράξης (1 – πρόσθεση, 2 – αφαίρεση, 3 - πολλαπλασιασμός, 4 - διαίρεση) και να εκτελεί αυτή την πράξη σε αυτούς τους δύο αριθμούς.

```

program praxi;
uses wincrt;
var
  ar1,ar2,pr:integer;
  apot:real;
begin
  writeln('Δώσε τους δύο αριθμούς και την πράξη');
  read(ar1,ar2,pr);
  if pr=1 then
  begin
    apot:=ar1+ar2;
    write('Το άθροισμά τους είναι:',apot:5:0);
  end
  else
  if pr=2 then
  begin
    apot:=ar1-ar2;
    write('Η διαφορά τους είναι:',apot:5:0);
  end
  else
  if pr=3 then
  begin
    apot:=ar1*ar2;
    write('Το γινόμενο τους είναι:',apot:5:0);
  end
  else
  if pr=4 then
  begin
    if ar2 <> 0 then
    begin
      apot:=ar1/ar2;
      write('Το πηλίκο τους είναι:',apot:5:2);
    end
    else
      write('Δεν γίνεται διαίρεση με το μηδέν');
    end
  else
    write('Λάθος κωδικός πράξης');
  end.

```

Δώσε τους δύο αριθμούς και την πράξη 5 3 2 ← Η διαφορά τους είναι:      2
---

Τη μεταβλητή apot τη δηλώσαμε ως real (πραγματική) γιατί το αποτέλεσμα της διαίρεσης θα είναι πραγματικός αριθμός. Κατά την εμφάνιση της μεταβλητής χρειάστηκε να τη μορφοποιήσουμε:

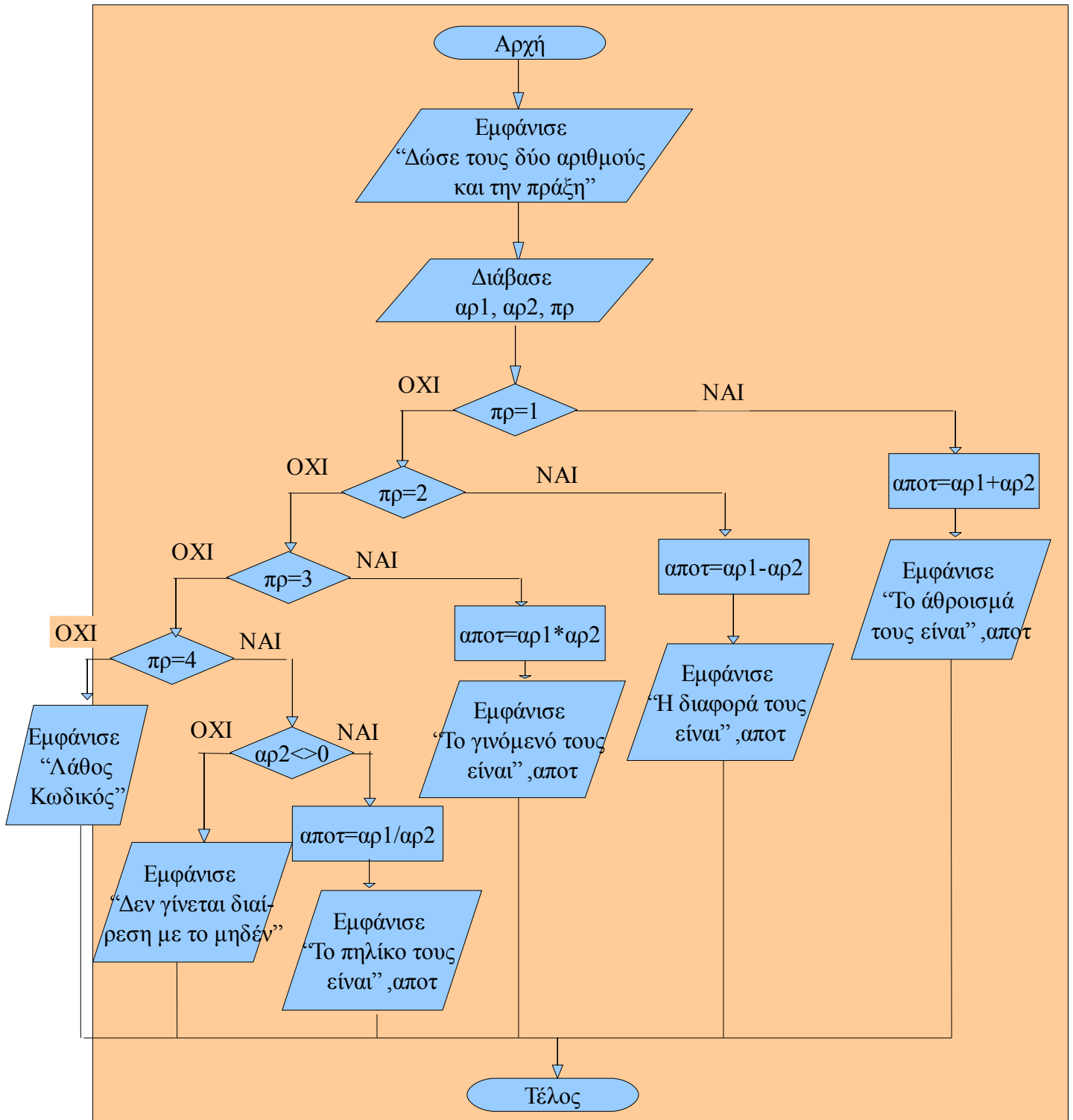
```
write (apot:5:2) ;
```

Στην παραπάνω εντολή εμφανίζω το περιεχόμενο της μεταβλητής `apot`, δεσμεύοντας 5 ψηφία για την εμφάνισή της και εμφανίζοντας δύο δεκαδικά ψηφία του πραγματικού αριθμού. Η υποδιαστολή και τα δύο δεκαδικά ψηφία καταλαμβάνουν τις 3 από τις 5 θέσεις που δεσμεύσαμε.

Βλέπουμε επίσης πως όταν οι εντολές που θα εκτελεστούν στη δομή της επιλογής είναι πάνω από μία, τις βάζουμε ανάμεσα σε `begin` και `end`, έχουμε δηλαδή μία σύνθετη εντολή (compound statement):

```
if pr=1 then
begin
  apot:=ar1+ar2;
  write('Το άθροισμά τους είναι:',apot:5:0);
end;
```

Αρχή  
 Εμφάνισε “Δώσε τους δύο αριθμούς και την πράξη”  
 Διάβασε `ar1`, `ar2`, `pr`  
 Αν `pr = 1`  
 Τότε  
   `apot = ar1 + ar2`  
 Εμφάνισε “Το άθροισμά τους είναι:”, `apot`  
 Αλλιώς  
 Αν `pr = 2`  
 Τότε  
   `apot = ar1 - ar2`  
 Εμφάνισε “Η διαφορά τους είναι:”, `apot`  
 Αλλιώς  
 Αν `pr = 3`  
 Τότε  
   `apot = ar1 * ar2`  
 Εμφάνισε “Το γινόμενο τους είναι:”, `apot`  
 Αλλιώς  
 Αν `pr = 4`  
 Τότε  
   Αν `ar2 <> 0`  
   Τότε  
     `apot = ar1/ar2`  
   Εμφάνισε “Το πηλίκο τους είναι:”, `apot`  
   Αλλιώς  
   Εμφάνισε “Δεν γίνεται διαίρεση με το μηδέν”  
 Αλλιώς  
 Εμφάνισε “Λάθος κωδικός”  
 Τέλος



Μία εντολή που μπορεί να μας διευκολύνει σε περιπτώσεις πολλαπλών επιλογών είναι η case. Ο παραπάνω κώδικας θα μπορούσε να γραφεί με την εντολή case ως εξής:

---

Να γίνει πρόγραμμα που να διαβάζει δύο αριθμούς και τον κωδικό της πράξης (1 – πρόσθεση, 2 – αφαίρεση, 3 πολλαπλασιασμός, 4 διαίρεση) και να εκτελεί αυτή την πράξη σε αυτούς τους δύο αριθμούς. Να λυθεί με την εντολή case.

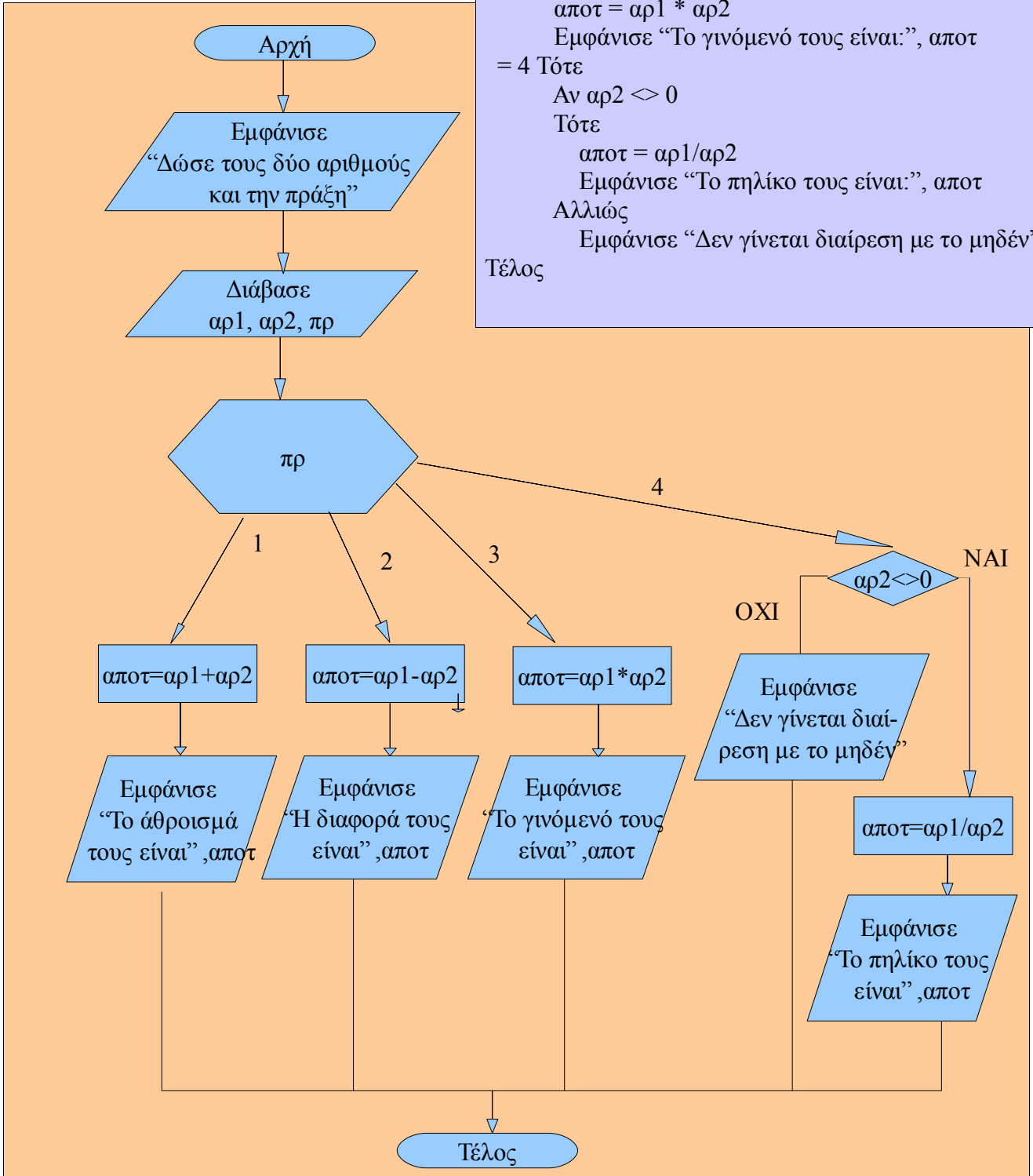
```
program praxi;
uses wincrt;
var
  ar1, ar2, pr: integer;
  apot: real;
begin
  writeln('Δώσε τους δύο αριθμούς και την πράξη');
  read(ar1, ar2, pr);
  case pr of
    1: begin
      apot:=ar1+ar2;
      write('Το άθροισμά τους είναι:', apot:5:0);
    end;
    2: begin
      apot:=ar1-ar2;
      write('Η διαφορά τους είναι:', apot:5:0);
    end;
    3: begin
      apot:=ar1*ar2;
      write('Το γινόμενό τους είναι:', apot:5:0);
    end;
    4: begin
      if ar2 <> 0 then
        begin
          apot:=ar1/ar2;
          write('Το άθροισμά τους είναι:', apot:5:2);
        end
      else
        write('Δεν γίνεται διαίρεση με το μηδέν');
      end;
  end; {το case κλείνει με end}
end.
```

---

Στα αντίστοιχα παραδείγματα με ψευδοκώδικα και λογικό διάγραμμα δεν υλοποιείται η else όπου εμφανίζεται το μήνυμα “Λάθος κωδικός πράξης”, καθώς αυτή η δυνατότητα υπάρχει μόνο στην case στην PASCAL.

```

Αρχή
Εμφάνισε “Δώσε τους δύο αριθμούς και την πράξη”
Διάβασε αρ1, αρ2, πρ
Αν πρ
= 1 Τότε
    αποτ = αρ1 + αρ2
    Εμφάνισε “Το άθροισμά τους είναι:”, αποτ
= 2 Τότε
    αποτ = αρ1 - αρ2
    Εμφάνισε “Η διαφορά τους είναι:”, αποτ
= 3 Τότε
    αποτ = αρ1 * αρ2
    Εμφάνισε “Το γινόμενο τους είναι:”, αποτ
= 4 Τότε
    Αν αρ2 <> 0
    Τότε
        αποτ = αρ1/αρ2
        Εμφάνισε “Το πηλίκο τους είναι:”, αποτ
    Αλλιώς
        Εμφάνισε “Δεν γίνεται διαίρεση με το μηδέν”
Τέλος
    
```



## Κεφάλαιο 11 – Εντολές επανάληψης

### Η δομή της επανάληψης

Η δομή της επανάληψης μπορεί να υλοποιηθεί με τρεις τρόπους:

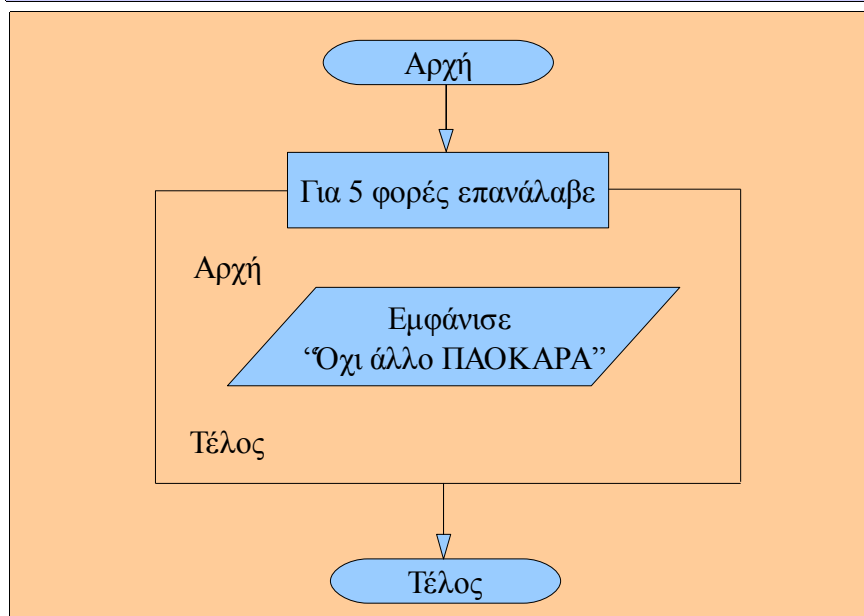
- for
- while
- repeat - until

Να γίνει πρόγραμμα που να εμφανίζει το μήνυμα “Όχι άλλο ΠΑΟΚΑΡΑ” πέντε φορές:

```
program paoknomore;
uses wincrt;
var
  i:integer;
begin
  for i:=1 to 5 do
  begin
    writeln('Όχι άλλο ΠΑΟΚΑΡΑ');
  end;
end.
```

```
Όχι άλλο ΠΑΟΚΑΡΑ
Όχι άλλο ΠΑΟΚΑΡΑ
Όχι άλλο ΠΑΟΚΑΡΑ
Όχι άλλο ΠΑΟΚΑΡΑ
Όχι άλλο ΠΑΟΚΑΡΑ
```

Αλγόριθμος  
 Για 5 φορές επανάλαβε  
 Αρχή  
 Εμφάνισε “Όχι άλλο ΠΑΟΚΑΡΑ”  
 Τέλος  
 Τέλος



Αυτή τη δομή επανάληψης τη χρησιμοποιούμε όταν γνωρίζουμε τον αριθμό των επαναλήψεων. Μία παραλλαγή αυτής της εντολή κάνει τις επαναλήψεις ανάποδα:

Να γίνει πρόγραμμα που να εμφανίζει τα νούμερα από το 5 μέχρι το 1 και τα τετράγωνά τους:

```

program paoknomore;
uses wincrt;
var
  i:integer;
begin
  for i:=5 downto 1 do      {χρησιμοποιώ το downto αντί για το to}
  begin
    writeln(i,' ',i*i);
  end;
end.
    
```

5	25
4	16
3	9
2	4
1	1

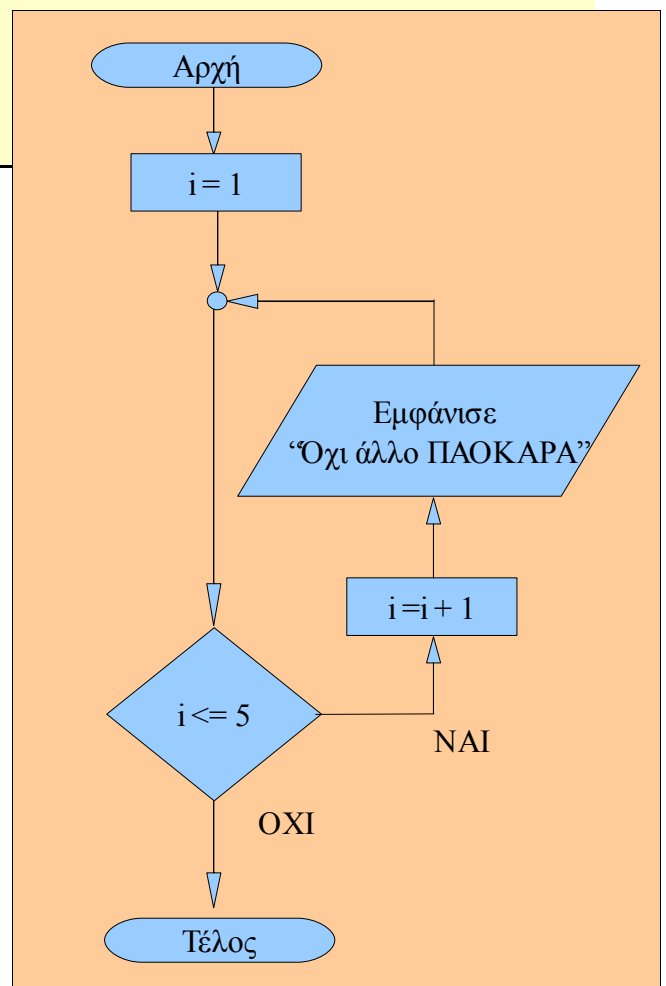
Ας επιστρέψουμε στην ΠΑΟΚΑΡΑ και το αντίστοιχο πρόγραμμα με την εντολή while:

```

program paoknomore;
uses wincrt;
var
  i:integer;
begin
  i:=1;          {δίνω αρχική τιμή στο μετρητή}
  while i <= 5 do {ελέγχω την τιμή του}
  begin
    writeln('Όχι άλλο ΠΑΟΚΑΡΑ');
    i:=i+1;      {τον αυξάνω κατά 1}
  end;
end.
    
```

Παρατηρούμε ότι αυτό που πριν γινόταν αυτόματα με την εντολή **for** όσον αφορά την αύξηση του μετρητή, θα πρέπει να φροντίσουμε να το κάνουμε εμείς τώρα. Εάν δεν γίνει η αύξηση του μετρητή τότε το πρόγραμμά μας θα εγκλωβιστεί σε έναν ατέρμων βρόχο, δηλαδή δεν θα βγει ποτέ από την επανάληψη.

Αλγόριθμος  
i = 1  
Εφόσον i <= 5 επανάλαβε  
  Αρχή  
  Εμφάνισε “Όχι άλλο ΠΑΟΚΑΡΑ”  
  i = i + 1  
  Τέλος  
Τέλος



Το ίδιο πρόγραμμα με repeat:

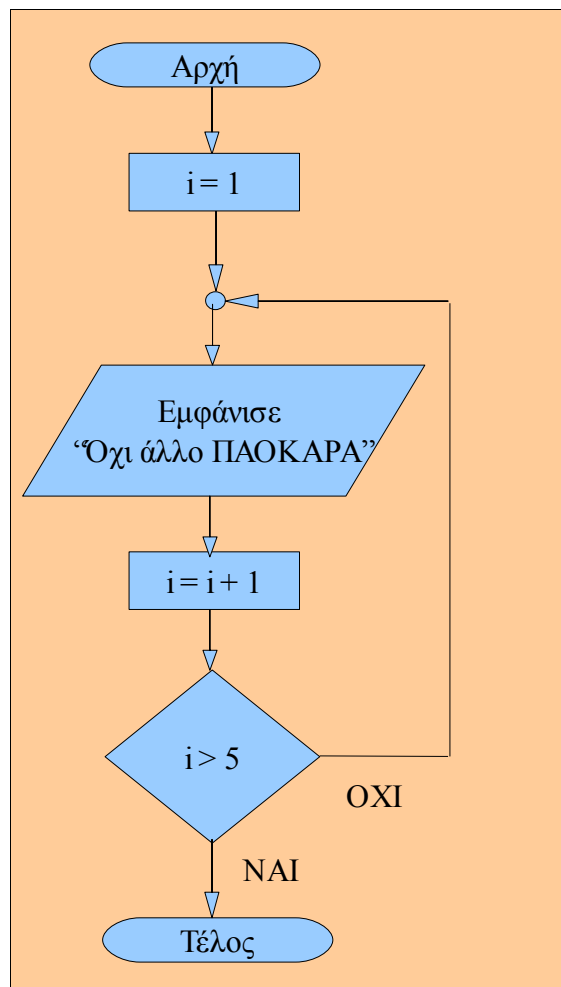
```

program paoknomore;
uses wincrt;
var
  i:integer;
begin
  i:=1;
  repeat
    writeln('Όχι άλλο ΠΑΟΚΑΡΑ');
    i:=i+1;
  until i > 5;
end.

```

Και εδώ ισχύει το ίδιο όσον αφορά το μετρητή. Η διαφορά αυτής της δομής επανάληψης από την προηγούμενη έχει να κάνει με το αν ο έλεγχος της συνθήκης για την επανάληψη γίνεται στην αρχή ή στο τέλος. Όταν γίνεται στην αρχή (while) υπάρχει το ενδεχόμενο να μην γίνει ούτε μία εκτέλεση της επανάληψής μου. Όταν γίνεται στο τέλος (repeat) η επανάληψή μου θα εκτελεστεί τουλάχιστον μία φορά.

Αλγόριθμος  
 $i = 1$   
 Επανάλαβε  
 Εμφάνισε “Όχι άλλο ΠΑΟΚΑΡΑ”  
 $i = i + 1$   
 Μέχρι  $i > 5$   
 Τέλος



Ας δούμε τώρα πως μπορούμε να χρησιμοποιήσουμε την επανάληψη για διάφορους υπολογισμούς. Στο επόμενο παράδειγμα βλέπουμε πως μπορώ να υπολογίσω ένα άθροισμα με τη βοήθεια της επανάληψης.

Να γίνει πρόγραμμα που να διαβάζει τους μισθούς 5 υπαλλήλων και να εμφανίζει το μέσο μισθό

```
program mesos_misthos;
uses wincrt;
var
  i:integer;
  misthos,sum,mesos:real;
begin
  sum:=0;
  for i:=1 to 5 do
  begin
    write('Δώσε το μισθό του υπαλλήλου ', i, ': ');
    readln(misthos);
    sum:=sum+misthos;
  end;
  mesos:=sum/5;
  writeln('Ο μέσος μισθός είναι:',mesos:8:2);
end.
```

```
Δώσε το μισθό του υπαλλήλου 1: 1200↵
Δώσε το μισθό του υπαλλήλου 2: 1100↵
Δώσε το μισθό του υπαλλήλου 3: 980↵
Δώσε το μισθό του υπαλλήλου 4: 850↵
Δώσε το μισθό του υπαλλήλου 5: 1000↵
Ο μέσος μισθός είναι: 1026.00
```

Στο επόμενο παράδειγμα συνδυάζω μία εντολή επιλογής (if) μέσα σε μία επανάληψη.

Να γίνει πρόγραμμα που να διαβάζει τους μισθούς 10 υπαλλήλων και να εμφανίζει το πλήθος αυτών που είναι κάτω από 700 ευρώ

```
program mesos_misthos;
uses wincrt;
var
  i,kato:integer;
  misthos:real;
begin
  kato:=0;
  for i:=1 to 10 do
  begin
    write('Δώσε το μισθό του υπαλλήλου ', i, ': ');
    readln(misthos);
    if misthos < 700 then { δεν χρειάζεται begin/end }
      kato:=kato+1;      { είναι μία μόνο εντολή }
    end;
  writeln('Το πλήθος των μισθών κάτω από 700 είναι: ',kato);
end.
```

```

Δώσε το μισθό του υπαλλήλου 1: 1200↵
Δώσε το μισθό του υπαλλήλου 2: 1100↵
Δώσε το μισθό του υπαλλήλου 3: 680↵
Δώσε το μισθό του υπαλλήλου 4: 850↵
Δώσε το μισθό του υπαλλήλου 5: 700↵
Δώσε το μισθό του υπαλλήλου 6: 1200↵
Δώσε το μισθό του υπαλλήλου 7: 600↵
Δώσε το μισθό του υπαλλήλου 8: 980↵
Δώσε το μισθό του υπαλλήλου 9: 850↵
Δώσε το μισθό του υπαλλήλου 10: 650↵
Το πλήθος των μισθών κάτω από 700 είναι: 3

```

Όταν ο αριθμός των επαναλήψεων είναι γνωστός τότε ενδείκνυται η χρήση της εντολής **for** για την επανάληψη. Αν όμως ο αριθμός των επαναλήψεων καθορίζεται κατά την εκτέλεση του προγράμματος, τότε πρέπει να χρησιμοποιήσουμε μία από τις άλλες δύο εντολές επανάληψης.

Να γίνει πρόγραμμα που να διαβάζει τους βαθμούς άγνωστου πλήθους μαθημάτων και να εμφανίζει το συνολικό πλήθος των μαθημάτων και πόσοι από αυτούς είναι κάτω από τη βάση. Το διάβασμα θα σταματά όταν θα δίνεται η τιμή μηδέν.

```

program vathmoi;
uses wincrt;
var
  i,kato:integer;
  vathmos:real;
begin
  kato:=0;
  i:=0;
  write('Δώσε το βαθμό του μαθήματος: ');
  readln(vathmos);
  while vathmos <> 0 do
  begin
    i:=i+1;
    if vathmos < 10 then
      kato:=kato+1;
    write('Δώσε το βαθμό του μαθήματος: ');{Πρέπει να ξαναδιαβάσω}
    readln(vathmos); {το βαθμό του μαθήματος}
  end;
  writeln('Σύνολο ',i,' μαθήματα με ',kato,' κάτω από τη βάση');
end.

```

```

Δώσε το βαθμό του μαθήματος: 15↵
Δώσε το βαθμό του μαθήματος: 13↵
Δώσε το βαθμό του μαθήματος: 12↵
Δώσε το βαθμό του μαθήματος: 9↵
Δώσε το βαθμό του μαθήματος: 12↵
Δώσε το βαθμό του μαθήματος: 9↵
Δώσε το βαθμό του μαθήματος: 0
Σύνολο 6 μαθήματα με 2 κάτω από τη βάση

```

## Κεφάλαιο 12 – Υποπρογράμματα

Για να υλοποιήσουμε τον τμηματικό προγραμματισμό, χωρίζουμε το πρόγραμμά μας σε ανεξάρτητα τμήματα, τα υποπρογράμματα, τα οποία καλούμε μέσα από το κύριο πρόγραμμα. Το κύριο πρόγραμμα και τα υποπρογράμματα ανταλλάσσουν μεταξύ τους δεδομένα μέσω των παραμέτρων.

Υπάρχουν δύο ειδών υποπρογράμματα, οι διαδικασίες και οι συναρτήσεις. Βασική διαφορά τους είναι ότι με τη συνάρτηση (function) επιστρέφεται **πάντα** μία τιμή, ενώ με τη διαδικασία (procedure) επιστρέφονται καμία, μία ή και περισσότερες τιμές.

Οι παράμετροι με τις οποίες καλώ το υποπρόγραμμα μου από το κύριο πρόγραμμα λέγονται **πραγματικές παράμετροι**. Οι αντίστοιχες παράμετροι στο υποπρόγραμμα λέγονται **τυπικές παράμετροι**. Ο αριθμός των παραμέτρων με τις οποίες καλώ από το κύριο πρόγραμμα το υποπρόγραμμα, όπως επίσης και ο τύπος δεδομένων των πραγματικών παραμέτρων, πρέπει να συμφωνεί απόλυτα με τις αντίστοιχες τυπικές παραμέτρους στο ίδιο το υποπρόγραμμα.

Ανάλογα με τον τρόπο που δηλώνω τις τυπικές παραμέτρους στις διαδικασίες, καθορίζω αν οι τιμές που θα πάρουν κατά την εκτέλεση της διαδικασίας θα επηρεάσουν τις αντίστοιχες πραγματικές παραμέτρους στο κύριο πρόγραμμα με τις οποίες τις κάλεσαν. Στις διαδικασίες υπάρχουν περιπτώσεις που θέλουμε οι παράμετροι να επιστρέψουν τις τιμές τους στο κύριο πρόγραμμα, δηλαδή να αλλάξουν και να δώσουν τις τιμές τους στις αντίστοιχες τιμές των πραγματικών παραμέτρων του κύριου προγράμματος. Σε αυτήν την περίπτωση στον ορισμό της διαδικασίας δηλώνω την παράμετρο ως **παράμετρο μεταβλητής** ξεκινώντας τη δήλωση με τη δεσμευμένη λέξη **var**. Εάν δεν τη δηλώσω με αυτή τη λέξη τότε είναι μία **παράμετρος τιμής**, πράγμα που σημαίνει ότι όποια αλλαγή και αν γίνει στην τιμή της μέσα στη διαδικασία, αυτή δεν θα επηρεάσει την τιμή της αντίστοιχης πραγματικής παραμέτρου του κύριου προγράμματος. Καλό είναι τα ονόματα των μεταβλητών που χρησιμοποιώ στις πραγματικές και στις τυπικές παραμέτρους να είναι διαφορετικά για να μην συγχέονται οι παράμετροι.

Ο ορισμός του υποπρογράμματος ή των υποπρογραμμάτων γίνεται ανάμεσα στην περιοχή δήλωσης των μεταβλητών και της έναρξης του κύριου προγράμματος. Υπάρχει η δυνατότητα επίσης ένα υποπρόγραμμα να καλεί ένα άλλο υποπρόγραμμα. Σε αυτήν την περίπτωση, το υποπρόγραμμα που καλεί πρέπει να βρίσκεται κάτω από το υποπρόγραμμα που καλείται.

Μία ακόμη δυνατότητα που έχουν επίσης τα υποπρογράμματα είναι να καλούν τον εαυτό τους. Αυτό λέγεται αναδρομή.

Η διαδικασία και η συνάρτηση διαφέρουν και στον τρόπο δήλωσης και κλήσης. Στα επόμενα παραδείγματα θα δούμε πως γράφουμε μία διαδικασία και μία συνάρτηση και πως τις καλούμε.

Ας δούμε ένα παράδειγμα υποπρογράμματος με διαδικασία:

Να γίνει πρόγραμμα που να διαβάζει την πλευρά ενός τετραγώνου και να υπολογίζει το εμβαδόν του. Ο υπολογισμός του εμβαδού να γίνει με διαδικασία.

```

program tetragono;
uses wincrt;
var
    plevra, emvadon:integer;
                                {p παράμετρος τιμής}
procedure ypologismos(p:integer; var e:integer);{τυπικές παρ/τροι}
begin                                {e παράμετρος μεταβλητής}
    e:=p*p;                            {η τιμή της θα επιστραφεί στην }
end;                                    {πραγματική παράμετρο emvadon }

begin
    write('Δώσε την πλευρά του τετραγώνου: ');
    readln(plevra);
    ypologismos(plevra, emvadon); {πραγματικές παράμετροι }
    write('Το εμβαδόν του τετραγώνου είναι: ',emvadon);
end.

```

```

Δώσε την πλευρά του τετραγώνου: 5↵
Το εμβαδόν του τετραγώνου είναι: 25

```

Η μεταβλητή e στο υποπρόγραμμα δηλώθηκε ως παράμετρος μεταβλητής (var) που σημαίνει ότι η τιμή της θα επιστραφεί στο κύριο πρόγραμμα, θα αλλάξει δηλαδή την αντίστοιχη emvadon με την οποία κλήθηκε η διαδικασία.

Από τη στιγμή που η τιμή που επιστρέφεται είναι μία, το ίδιο υποπρόγραμμα μπορεί να υλοποιηθεί και με μία συνάρτηση

Να γίνει πρόγραμμα που να διαβάζει την πλευρά ενός τετραγώνου και να υπολογίζει το εμβαδόν του. Ο υπολογισμός του εμβαδού να γίνει με συνάρτηση.

```

program tetragono;
uses wincrt;
var
    plevra, emvadon:integer;

function ypologismos(p:integer):integer; {τύπος δεδομένων }
begin                                {της συνάρτησης }
    ypologismos:= p*p;
end;

begin
    write('Δώσε την πλευρά του τετραγώνου: ');
    readln(plevra);
    emvadon := ypologismos(plevra);
    write('Το εμβαδόν του τετραγώνου είναι: ',emvadon);
end.

```

Στη δήλωση της συνάρτησης πρέπει να δηλώσουμε τι τύπος δεδομένων θα επιστραφεί. Στο προηγούμενο παράδειγμα η συνάρτηση δηλώθηκε ως integer. Στη συνάρτηση πρέπει οπωσδήποτε να δώσω μία τιμή στο όνομά της ( $ypologismos := p * p$ ) καθώς αυτός είναι και ο τρόπος με τον οποίο επιστρέφεται η τιμή της στο κύριο πρόγραμμα.

Βλέπουμε επίσης ότι ο τρόπος κλήσης της συνάρτησης διαφέρει από αυτόν της διαδικασίας. Όταν καλώ μία διαδικασία από το κύριο πρόγραμμα, απλά γράφω το όνομά της:

```
ypologismos(plevra, emvadon);
```

Για να καλέσω μία συνάρτηση πρέπει να τη βάλω στη δεξιά πλευρά της εντολής αντικατάστασης:

```
emvadon := ypologismos(plevra);
```

Μία συνάρτηση μπορώ επίσης να την καλέσω και μέσα από μία εντολή write. Όταν θα αναλύσουμε την αναδρομή παρακάτω θα δούμε ένα τέτοιο παράδειγμα.

Ας δούμε πως μπορώ να υλοποιήσω το παραπάνω πρόγραμμα χρησιμοποιώντας μόνο υποπρογράμματα. Βέβαια σε αυτήν την περίπτωση είναι κάπως “τραβηγμένο” καθώς το κάθε υποπρόγραμμα έχει μόνο μία με δύο εντολές, αλλά είναι ένα χαρακτηριστικό παράδειγμα τμηματικού προγραμματισμού όπου από το κύριο πρόγραμμα βλέπω ξεκάθαρα τα μέρη από τα οποία αποτελείται.

*Να γίνει πρόγραμμα που να διαβάζει την πλευρά ενός τετραγώνου και να υπολογίζει το εμβαδόν του. Το διάβασμα να γίνει με μια διαδικασία, ο υπολογισμός του εμβαδού να γίνει με συνάρτηση και η εμφάνιση των αποτελεσμάτων με διαδικασία.*

```
program tetragono;
uses wincrt;
var
  a,e:integer;

procedure diavaseplevra(var plevra:integer); {τυπική παράμετρος}
begin
  {παράμετρος μεταβλητής}
  write('Δώσε την πλευρά του τετραγώνου: ');
  readln(plevra);
end;

function ypologismos(plevra:integer):integer;
begin
  ypologismos:=plevra*plevra;
end;

procedure emfanise(emvadon:integer); {τυπική παράμετρος}
begin
  {παράμετρος τιμής}
  write('Το εμβαδόν του τετραγώνου είναι: ',emvadon);
end;

begin
  diavaseplevra(a); {πραγματική παράμετρος}
  e:=ypologismos(a);
  emfanise(e);
end.
```

Ας δούμε ένα παράδειγμα κλήσης υποπρογράμματος μέσα από άλλο υποπρόγραμμα:

Να γίνει πρόγραμμα που να διαβάζει τους βαθμούς τριών τριμήνων και των γραπτών και να υπολογίζει τον τελικό βαθμό σύμφωνα με τον τύπο:

Τελικός βαθμός =  $(A' \text{ Τρίμηνο} + B' \text{ Τρίμηνο} + G' \text{ Τρίμηνο} + \text{Γραπτά} * 2) / 5$

```
program bathmos;

uses wincrt;

var
  t1,t2,t3,g,tv:real;

{ η διαδικασία που καλείται είναι πάνω από αυτήν που την καλεί }
procedure diavasededomena(var trim1,trim2,trim3,grapta:real);
begin
  writeln('Δώσε τους βαθμούς των τριών τριμήνων και των γραπτών: ');
  readln(trim1,trim2,trim3,grapta);
end;

procedure diavaseypologise(var telikos:real);
var
  trim1,trim2,trim3,grapta:real;
begin
  diavasededomena(trim1,trim2,trim3,grapta); {καλεί διαδικασία}
  telikos:=(trim1+trim2+trim3+2*grapta)/5;
end;

procedure emfanisededomena(telikos:real);
begin
  writeln('Ο τελικός βαθμός είναι:',telikos:5:2);
end;

begin
  diavaseypologise(tv);
  emfanisededomena(tv);
end.
```

```
Δώσε τους βαθμούς των τριών τριμήνων και των γραπτών:
15 16 13 12↵
Ο τελικός βαθμός είναι:
```

## Αναδρομή

Η αναδρομή είναι η δυνατότητα ενός υποπρογράμματος να καλεί τον ίδιο του τον εαυτό. Παρακάτω ακολουθεί ένα παράδειγμα υλοποίησης υποπρογράμματος με και χωρίς την αναδρομή.

Ας δούμε την υλοποίηση του υπολογισμού του παραγοντικού αριθμού χωρίς αναδρομή:

Να γίνει πρόγραμμα που να διαβάζει έναν αριθμό και υπολογίζει και εμφανίζει το παραγοντικό του π.χ  $4! = 4 * 3 * 2 * 1$

```
program ypologismos_paragontikou;
uses wincrt;
var
  arithmos:integer;
  apotelesma:longint;

function paragontiko(n:integer):longint;
var
  i:integer;
  par:longint;
begin
  par:=1;
  for i:=2 to n do
    par:= par * i;
  paragontiko:=par;
end;

begin
  write('Δώσε έναν αριθμό: ');
  readln(arithmos);
  apotelesma:=paragontiko(arithmos);
  write('Το παραγοντικό του είναι :',apotelesma);
end.
```

```
Δώσε έναν αριθμό: 4↵
Το παραγοντικό του είναι : 24
```

Όπως αναφέρθηκε νωρίτερα η κλήση της συνάρτησης μπορεί να γίνει και μέσα από την εντολή write. Σε αυτήν την περίπτωση δεν θα χρειαστούμε τη μεταβλητή apotelesma και το κύριο πρόγραμμά μας θα έχει αυτή τη μορφή:

```
begin
  write('Δώσε έναν αριθμό: ');
  readln(arithmos);
  write('Το παραγοντικό του είναι :',paragontiko(arithmos));
end.
```

Ας δούμε τώρα πως μπορεί να λυθεί με αναδρομή:

Να γίνει πρόγραμμα που να διαβάζει έναν αριθμό και υπολογίζει και εμφανίζει το παραγοντικό του π.χ  $4! = 4 * 3 * 2 * 1$

```
program ypologismos_paragontikou;
uses wincrt;
var
  arithmos:integer;
  apotelesma:longint;

function paragontiko(n:integer):longint;
var
  i:integer;
begin
  if n = 0 then
    paragontiko:=1
  else
    paragontiko:=n*paragontiko(n-1)
end;

begin
  write('Δώσε έναν αριθμό: ');
  readln(arithmos);
  apotelesma:=paragontiko(arithmos);
  write('Το παραγοντικό του είναι :',apotelesma);
end.
```

```
Δώσε έναν αριθμό: 4↵
Το παραγοντικό του είναι : 24
```

Βλέπουμε ότι η συνάρτηση καλείται όχι μόνο από το κύριο πρόγραμμα αλλά και μέσα από την ίδια την συνάρτηση.

Για να καταλάβουμε ευκολότερα πώς λειτουργεί η αναδρομή ας εξετάσουμε την εκτέλεση του προγράμματος με συγκεκριμένα δεδομένα. Ας θεωρήσουμε δηλαδή ότι ο χρήστης πληκτρολογεί τον αριθμό 3 οπότε και η μεταβλητή `arithmos` έχει πάρει αυτήν την τιμή και η κλήση της συνάρτησης,

```
apotelesma:=paragontiko(arithmos);
```

έγινε με την τιμή 3.

Στο σχεδιάγραμμα που ακολουθεί μπορούμε να δούμε αναλυτικά την πορεία των κλήσεων της συνάρτησης ακολουθώντας τα βέλη προς τα κάτω και μετά την πορεία της επιστροφής των τιμών ακολουθώντας τα βέλη προς τα πάνω.

apotelesma:=paragontiko (arithmos),

Η συνάρτηση καλείται αρχικά με την τιμή 3

```
function paragontiko(n:integer):longint;  
var  
  i:integer;  
begin  
  if n = 0 then  
    paragontiko:=1  
  else  
    paragontiko:=n*paragontiko(n-1)  
  end;  
end;
```

**n = 3**      **3 \* 2 = 6**

ξανακαλεί τον εαυτό της με την τιμή 2

```
function paragontiko(n:integer):longint;  
var  
  i:integer;  
begin  
  if n = 0 then  
    paragontiko:=1  
  else  
    paragontiko:=n*paragontiko(n-1)  
  end;  
end;
```

**n = 2**      **2 \* 1 = 2**

ξανακαλεί τον εαυτό της με την τιμή 1

```
function paragontiko(n:integer):longint;  
var  
  i:integer;  
begin  
  if n = 0 then  
    paragontiko:=1  
  else  
    paragontiko:=n*paragontiko(n-1)  
  end;  
end;
```

**n = 1**      **1 \* 1 = 1**

ξανακαλεί τον εαυτό της με την τιμή 0

```
function paragontiko(n:integer):longint;  
var  
  i:integer;  
begin  
  if n = 0 then  
    paragontiko:=1  
  else  
    paragontiko:=n*paragontiko(n-1)  
  end;  
end;
```

**n = 0**      **1**

6

2

1

1

## Κεφάλαιο 13 – Τύποι δεδομένων

Οι τύποι δεδομένων είναι μία τυπική περιγραφή των τιμών που μπορεί να αποθηκευτεί σε μία σταθερή ή μεταβλητή καθώς και πράξεων που μπορούν να εκτελεσθούν με τις τιμές αυτές.

Οι τελεστές (αριθμητικοί και σχεσιακοί) που μπορώ να χρησιμοποιήσω στους τύπους δεδομένων:

- Ακέραιοι: + - \* / DIV MOD := < > <> =
- Πραγματικοί + - \* / := < > <> =
- Λογικοί and or xor not = <>
- Χαρακτήρας := < > <> =

Οι παραπάνω τύποι δεδομένων ονομάζονται **βαθμωτοί** ή **μονόμετροι** γιατί έχουν δύο ιδιότητες:

1. Αποτελούνται από **αδιαίρετα** ή **ατομικά** στοιχεία, δηλαδή δεν αποτελούνται από μέρη τα οποία μπορούν να προσπελαστούν ανεξάρτητα
2. Είναι διατεταγμένοι, δηλαδή συγκρίνοντας μία τιμή με μία άλλη μπορεί να είναι μεγαλύτερη (>), μικρότερη (<) ή ίση (=).

Οι ακέραιοι, οι λογικοί και οι χαρακτήρες είναι επιπλέον **τακτικοί** τύποι δεδομένων, δηλαδή για όλες τις τιμές (εκτός από την πρώτη και την τελευταία), υπάρχει μία προηγούμενη και μία επόμενη.

Οι παραπάνω τύποι δεδομένων λέγονται **απλοί**, καθώς οι τιμές δεδομένων που παίρνουν είναι **μοναδικές**. Οι απλοί τύποι δεδομένων μπορούν να συνδυαστούν για να γίνουν οι **σύνθετοι** τύποι δεδομένων. Ο αλφαριθμητικός τύπος δεδομένων είναι ένα παράδειγμα σύνθετου τύπου.

Οι σύνθετοι τύποι διακρίνονται σε τύπους:

- **με δόμηση**  
Τέτοιοι τύποι είναι οι **πίνακες** και τα **αρχεία**
- **χωρίς δόμηση**  
Τέτοιοι τύποι είναι οι **εγγραφές** και τα **σύνολα**

Οι δομές δεδομένων των σύνθετων τύπων μπορεί να είναι **στατικές** ή **δυναμικές**. Το μέγεθος των δυναμικών δομών δεδομένων μπορεί να αλλάξει κατά την εκτέλεση του προγράμματος. Αντιθέτως το μέγεθος των στατικών δομών δεδομένων, όπως είναι ο πίνακας, δεν μπορούν να αλλάξουν κατά τη διάρκεια της εκτέλεσης του προγράμματος.

**Οι συναρτήσεις τακτικών τύπων**

Οι συναρτήσεις που μπορώ να χρησιμοποιήσω στους τακτικούς τύπους είναι:

**chr(x)**

παίρνει μία ακέραια τιμή και επιστρέφει τον χαρακτήρα που αντιστοιχεί σε αυτή την τιμή στον κώδικα ASCII

**ord(x)**

παίρνει έναν χαρακτήρα και επιστρέφει την αντίστοιχη τιμή του στον κώδικα ASCII

**pred(x)**

εμφανίζει την επόμενη τιμή ενός δεδομένου τακτικού τύπου (ακέραιος αριθμός ή χαρακτήρας)

**succ(x)**

εμφανίζει την προηγούμενη τιμή ενός δεδομένου τακτικού τύπου (ακέραιος αριθμός ή χαρακτήρας)

Ας δούμε μερικά παραδείγματα:

Να γίνει πρόγραμμα που να εμφανίζει τους χαρακτήρες που αντιστοιχούν στις ακέραιες τιμές από 80 έως 85 στον κώδικα ASCII.

```
program characters;
uses wincrt;
var
  i:integer;

begin
  for i:= 80 to 85 do
    writeln('ο αριθμός ',i,' αντιστοιχεί στο γράμμα ',chr(i));
end.
```

ο αριθμός 80 αντιστοιχεί στο γράμμα P
ο αριθμός 81 αντιστοιχεί στο γράμμα Q
ο αριθμός 82 αντιστοιχεί στο γράμμα R
ο αριθμός 83 αντιστοιχεί στο γράμμα S
ο αριθμός 84 αντιστοιχεί στο γράμμα T
ο αριθμός 85 αντιστοιχεί στο γράμμα U

Στο επόμενο παράδειγμα το πρόγραμμα ζητάει από το χρήστη να δώσει έναν ακέραιο αριθμό και εμφανίζει το αντίστοιχο γράμμα στον κώδικα ASCII.

Επιπλέον γίνεται και έλεγχος ορθότητας των εισαγόμενων τιμών.

Να γίνει πρόγραμμα που να διαβάζει έναν ακέραιο αριθμό από το 0 μέχρι το 255 και να εμφανίζει τον χαρακτήρα που αντιστοιχεί στον κώδικα ASCII. Να γίνεται έλεγχος των εισαγόμενων τιμών όσον αφορά την ορθότητά τους (0-255).

```

program characters;
uses wincrt;
var
    timi,i:integer;

begin
    write('Δώσε μία τιμή από το 0 έως το 255:');
    readln(timi);
    while (timi < 0) or (timi > 255) do
    begin
        write('Λάθος τιμή, πρέπει να είναι μεταξύ 0 και 255: ');
        readln(timi);
    end;
    writeln('Αντιστοιχεί στο γράμμα: ',chr(timi));
end.

```

```

Δώσε μία τιμή από το 0 έως το 255: 380
Λάθος τιμή, πρέπει να είναι μεταξύ 0 και 255: 260
Λάθος τιμή, πρέπει να είναι μεταξύ 0 και 255: 88
Αντιστοιχεί στο γράμμα: X

```

Ας δούμε τώρα τη συνάρτηση που κάνει το αντίστροφο. Στη συγκεκριμένη άσκηση χρησιμοποιείται και η εντολή επανάληψης repeat-until για άγνωστο αριθμό επαναλήψεων.

Να γίνει πρόγραμμα που να διαβάζει ένα γράμμα και να εμφανίζει τον ακέραιο αριθμό που αντιστοιχεί στο γράμμα. Το πρόγραμμα να επαναλαμβάνει το διάβασμα μέχρι να δοθεί η τιμή μηδέν.

```

program characters;
uses wincrt;
var
    gramma:char;

begin
    repeat
        write('Δώσε το γράμμα: ');
        readln(gramma);
        writeln('ο αριθμός που αντιστοιχεί στον κώδικα ASCII:',ord(gramma));
    until gramma = '0';
end.

```

```

Δώσε το γράμμα: r↵
Ο αριθμός που αντιστοιχεί στον κώδικα ASCII: 114
Δώσε το γράμμα: γ↵
Ο αριθμός που αντιστοιχεί στον κώδικα ASCII: 227
Δώσε το γράμμα: ς↵
Ο αριθμός που αντιστοιχεί στον κώδικα ASCII: 242
Δώσε το γράμμα: 0↵
Ο αριθμός που αντιστοιχεί στον κώδικα ASCII: 48

```

Ας δούμε τώρα πως μπορούμε να χρησιμοποιήσουμε αυτές τις συναρτήσεις για κάτι πιο χρήσιμο. Στο συγκεκριμένο πρόγραμμα γίνεται μετατροπή ενός γράμματος από πεζό σε κεφαλαίο:

Να γίνει πρόγραμμα που να διαβάζει ένα ελληνικό πεζό γράμμα και να εμφανίζει το κεφαλαίο του. Να γίνεται έλεγχος ορθότητας ότι το γράμμα που διαβάζεται είναι ελληνικό πεζό.

```
program characters;
uses wincrt;
var
  gramma:char;
  kodikos:integer;

begin
  write('Δώσε ένα ελληνικό πεζό γράμμα: ');
  readln(gramma);
  while (gramma < 'α') or ( gramma > 'ω') do
  begin
    write('Λάθος γράμμα, να είναι πεζό ελληνικό είπαμε: ');
    readln(gramma);
  end;
  kodikos:=ord(gramma);      {τα κεφαλαία γράμματα είναι 32 θέσεις }
  kodikos:=kodikos-32;      {πιο πίσω από τα αντίστοιχα πεζά τους }
  writeln('Το κεφαλαίο του είναι το ',chr(kodikos));
end.
```

```
Δώσε ένα ελληνικό πεζό γράμμα: α↵
Λάθος γράμμα, να είναι πεζό ελληνικό είπαμε: Γ↵
Λάθος γράμμα, να είναι πεζό ελληνικό είπαμε: δ↵
Το κεφαλαίο του είναι το Δ
```

Ας δούμε και ένα παράδειγμα με τις άλλες δύο συναρτήσεις

Να γίνει πρόγραμμα που να διαβάζει ένα γράμμα και να εμφανίζει το επόμενο και το προηγούμενό του.

```
program characters;
uses wincrt;
var
  gramma:char;

begin
  write('Δώσε ένα γράμμα: ');
  readln(gramma);
  writeln('Το προηγούμενό του είναι το ',pred(gramma));
  writeln('Το επόμενό του είναι το ',succ(gramma));
end.
```

```
Δώσε ένα γράμμα: G↵
Το προηγούμενό του είναι το F
Το επόμενό του είναι το H
```

Καλή επιτυχία

