

10.

Υποπρογράμματα



### Εισαγωγή

Η επίλυση ενός προβλήματος διευκολύνεται με τη διαίρεση του σε μικρότερα υποπροβλήματα. Η επίλυση των υποπροβλημάτων αυτών οδηγεί στην επίλυση του αρχικού προβλήματος. Ο τμηματικός προγραμματισμός, η διαίρεση δηλαδή ενός προγράμματος σε υποπρογράμματα υλοποιεί αυτήν την ιδέα στον προγραμματισμό. Το κεφάλαιο αυτό ασχολείται με τις αρχές του τμηματικού προγραμματισμού, τα είδη των υποπρογραμμάτων που υποστηρίζει η **ΓΛΩΣΣΑ**, τις διαδικασίες και τις συναρτήσεις καθώς και τον τρόπο που τα υποπρογράμματα αυτά επικοινωνούν μεταξύ τους. Τέλος παρουσιάζεται και αναλύεται ο τρόπος υλοποίησης αναδρομικών αλγορίθμων με χρήση αναδρομικών υποπρογραμμάτων.



### Στόχοι

Να είναι σε θέση ο μαθητής :

- ⇒ Να αναλύει ένα σύνθετο πρόγραμμα σε απλά υποπρογράμματα.
- ⇒ Να διακρίνει τις συναρτήσεις από τις διαδικασίες και να επιλέγει τη χρήση διαδικασίας ή συνάρτησης για την υλοποίηση ενός υποπρογράμματος.
- ⇒ Να περιγράφει τη δομή των υποπρογραμμάτων και να χρησιμοποιεί παραμέτρους για την επικοινωνία τους.
- ⇒ Να καθορίζει τις περιοχές εμβέλειας των παραμέτρων.
- ⇒ Να συντάσσει αναδρομικά υποπρογράμματα και να συγκρίνει αναδρομικές και επαναληπτικές διαδικασίες.



### Προερωτήσεις

- ✓ Η δόμηση ενός προγράμματος με τη μορφή ενός συνόλου μικρότερων προγραμμάτων βοηθάει τον προγραμματιστή στην ανάπτυξη ενός σύνθετου προγράμματος;
- ✓ Νομίζετε ότι όλα τα είδη τμημάτων προγραμμάτων επιτελούν την ίδια εργασία;
- ✓ Πως μπορεί να οργανώνεται ένα πρόγραμμα σε μικρότερα προγράμματα;
- ✓ Χρειάζεται τα επιμέρους προγράμματα να επικοινωνούν μεταξύ τους;
- ✓ Γνωρίζεις από την άλγεβρα τους αναδρομικούς τύπους; Ποια τα πλεονεκτήματά τους;

## 10.1. Τμηματικός προγραμματισμός

Τα προβλήματα που αντιμετωπίστηκαν στα προηγούμενα κεφάλαια, ήταν αρκετά απλά, ώστε να μπορούν να αναπτυχθούν σωστά σε ένα και μόνο πρόγραμμα. Όπως αναφέρθηκε στο κεφάλαιο 6, ο καλύτερος τρόπος για να αντιμετωπισθούν σύνθετα προβλήματα και να γραφούν τα αντίστοιχα προγράμματα, είναι η ιεραρχική προσέγγιση, η ανάπτυξη του προγράμματος από επάνω προς τα κάτω (top-down). Κάθε πρόβλημα διαιρείται σε μικρότερα επιμέρους προβλήματα και κάθε ένα από αυτά τα προγράμματα διαιρείται σε ακόμα απλούστερα και μικρότερα. Στο τέλος τα επί μέρους υπο-προβλήματα είναι αρκετά απλά, ώστε οι αντίστοιχοι αλγόριθμοι και τα αντίστοιχα τμήματα προγράμματος να μπορούν να σχεδιασθούν και να γραφούν εύκολα. Ο τελικός αλγόριθμος του προβλήματος ανάγεται σε πολλούς απλούστερους επί μέρους αλγόριθμους και το τελικό πρόγραμμα σε πολλά απλούστερα τμήματα προγράμματος.

Η τεχνική του τμηματικού προγραμματισμού είναι ένα από τα βασικότερα συστατικά του δομημένου προγραμματισμού, ο οποίος εξασφαλίζει σε μεγάλο βαθμό την επιτυχή και εύκολη δημιουργία σωστών προγραμμάτων.

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

### Παράδειγμα 1

Ας μελετήσουμε κατ' αρχήν το πρόβλημα που μας απασχόλησε στο πρώτο κεφάλαιο του βιβλίου, την αξιολόγηση των αποτελεσμάτων των μαθητών Γ' Λυκείου στα μαθήματα ειδικότητας.

Το σύνθετο αυτό πρόβλημα για να αντιμετωπισθεί πιο εύκολα πρέπει να αναλυθεί σε επιμέρους μικρότερα προβλήματα.

Συγκεκριμένα τα τρία βασικά διαφορετικά τμήματα είναι:

- ⇒ Εισαγωγή δεδομένων
- ⇒ Επεξεργασία δεδομένων
- ⇒ Εκτύπωση αποτελεσμάτων

Τα τρία αυτά τμήματα μπορούν να αναλυθούν περισσότερο. Συγκεκριμένα :



**Εισαγωγή δεδομένων**

- ⇒ Καταχώριση δεδομένων
- ⇒ Έλεγχος δεδομένων

**Επεξεργασία δεδομένων**

- ⇒ Υπολογισμός μέσης τιμής
- ⇒ Υπολογισμός τυπικής απόκλισης
- ⇒ Κατανομή συχνοτήτων
- ⇒ Δημιουργία γραφικών παραστάσεων

**Εκτύπωση αποτελεσμάτων**

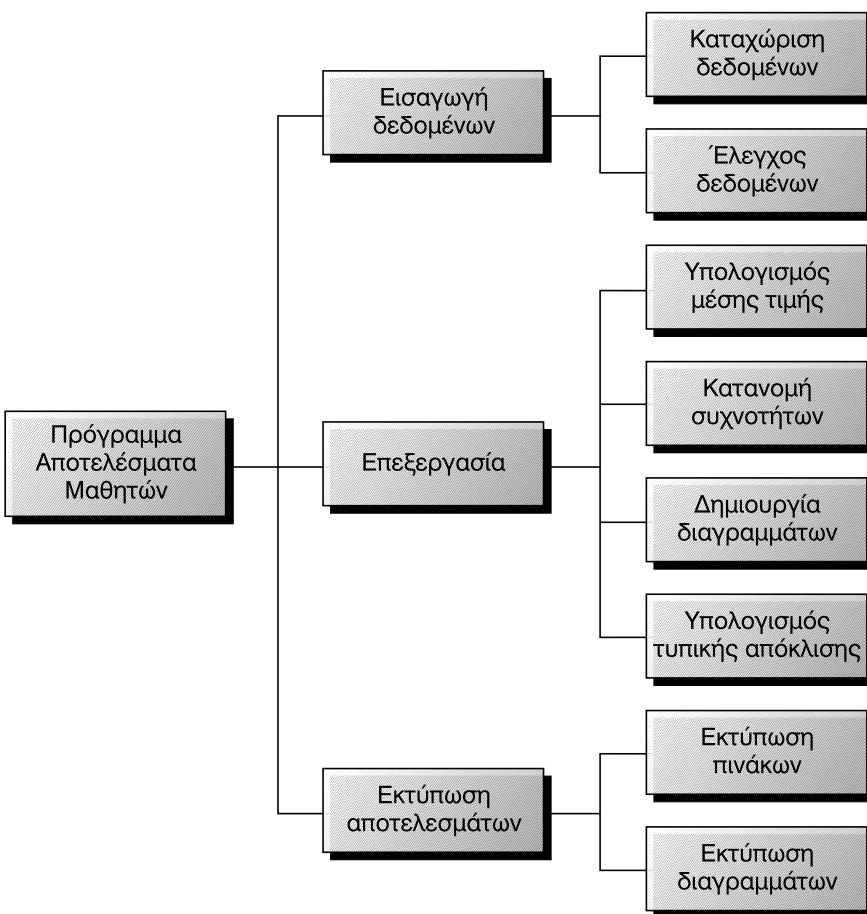
- ⇒ Εκτύπωση πινάκων συχνοτήτων
- ⇒ Εκτύπωση γραφικών παραστάσεων

Όπως φαίνεται το αρχικό πρόβλημα διασπάστηκε σε αρκετά απλούστερα υποπροβλήματα. Η δημιουργία λοιπόν του τελικού προγράμματος ανάγεται στη δημιουργία των επί μέρους τμημάτων προγραμμάτων ή ενοτήτων και τη σύνδεση αυτών μεταξύ τους. Μερικά από αυτά τα τμήματα, όπως ο υπολογισμός της μέσης τιμής ή της τυπικής απόκλισης, έχουν ήδη αντιμετωπιστεί στο προηγούμενο κεφάλαιο, που σημαίνει ότι μπορούμε να εκμεταλλευτούμε τα προγράμματα που ήδη έχουμε γράψει μειώνοντας έτσι την εργασία για την επίλυση του προβλήματος.

Η παράσταση της ανάλυσης του προβλήματος μπορεί να γίνει γραφικά με το διάγραμμα του σχήματος 10.1

Η έννοια του τμηματικού προγραμματισμού έχει ήδη αποτυπωθεί και σε προηγούμενα κεφάλαια. Για παράδειγμα στο πρόγραμμα υπολογισμού των θερμοκρασιών του προηγούμενου κεφαλαίου (παράδειγμα 3) το τμήμα της εισαγωγής δεδομένων έχει ξεχωρίσει από το τμήμα υπολογισμών σε αντίθεση με τα παραδείγματα 1 και 2 που είναι ενιαία.

Όταν ένα τμήμα προγράμματος επιτελεί ένα αυτόνομο έργο και έχει γραφεί χωριστά από το υπόλοιπο πρόγραμμα, τότε αναφερόμαστε σε **υποπρόγραμμα** (subprogram).



Σχ. 10.1 Διαγραμματική απεικόνιση της ανάλυσης προβλήματος

## 10.2. Χαρακτηριστικά των υποπρογραμμάτων

Ο χωρισμός ενός προγράμματος σε υποπρογράμματα προϋποθέτει την ανάλυση του αρχικού προβλήματος σε μικρότερα υποπροβλήματα, τα οποία να μπορούν να αντιμετωπισθούν ανεξάρτητα το ένα από το άλλο. Η ανάλυση όμως αυτή δεν είναι πάντα εύκολη και όπως ισχύει γενικά στον προγραμματισμό, δεν υπάρχουν συγκεκριμένοι κανόνες για την επιτυχή ανάλυση. Η δυσκολία δε αυξάνεται όσο πιο μεγάλο και πιο σύνθετο είναι το πρόβλημα. Η σωστή εφαρμογή του τμηματικού προγραμματισμού απαιτεί

μελέτη στην ανάλυση του προβλήματος, εμπειρία στον προγραμματισμό, ταλέντο και φυσικά γνώσεις.

Υπάρχουν πάντως τρεις ιδιότητες που πρέπει να διακρίνουν τα υποπρογράμματα:

- ⇒ **Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο.** Στην πραγματικότητα κάθε υποπρόγραμμα ενεργοποιείται με την είσοδο σε αυτό που γίνεται πάντοτε από την αρχή του, εκτελεί ορισμένες ενέργειες, και απενεργοποιείται με την έξοδο από αυτό που γίνεται πάντοτε από το τέλος του.
- ⇒ **Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα.** Αυτό σημαίνει ότι κάθε υποπρόγραμμα μπορεί να σχεδιαστεί, να αναπτυχθεί και να συντηρηθεί αυτόνομα χωρίς να επηρεαστούν άλλα υποπρογράμματα. Στην πράξη βέβαια η απόλυτη ανεξαρτησία είναι δύσκολο να επιτευχθεί.
- ⇒ **Κάθε υποπρόγραμμα πρέπει να μην είναι πολύ μεγάλο.** Η έννοια του μεγάλου προγράμματος είναι υποκειμενική, αλλά πρέπει κάθε υποπρόγραμμα να είναι τόσο, ώστε να είναι εύκολα κατανοητό για να μπορεί να ελέγχεται. Γενικά κάθε υποπρόγραμμα πρέπει να εκτελεί μόνο μία λειτουργία. Αν εκτελεί περισσότερες λειτουργίες, τότε συνήθως μπορεί και πρέπει να διασπαστεί σε ακόμη μικρότερα υποπρογράμματα.

### 10.3. Πλεονεκτήματα του τμηματικού προγραμματισμού

Η χρήση υποπρογραμμάτων σε ένα πρόγραμμα παρουσιάζει πολλά πλεονεκτήματα που αναφέρθηκαν συνοπτικά στο κεφάλαιο 6. Η σωστή χρήση του τμηματικού προγραμματισμού, δηλαδή ο σωστός χωρισμός ενός σύνθετου προγράμματος σε υποπρογράμματα εξασφαλίζει τέσσερα βασικά χαρακτηριστικά του σωστού προγραμματισμού:

**Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντιστοίχου προγράμματος.**

Επιτρέπει την εξέταση και την επίλυση απλών προβλημάτων και όχι στην αντιμετώπιση του συνολικού προβλήματος. Με τη σταδιακή επίλυση των υποπροβλημάτων και τη δημιουργία των αντιστοίχων υποπρογραμμάτων τελικά επιλύεται το συνολικό πρόβλημα.

**Διευκολύνει την κατανόηση και διόρθωση του προγράμματος.**

Ο χωρισμός του προγράμματος σε μικρότερα αυτοτελή τμήματα επιτρέπει τη γρήγορη διόρθωση ενός συγκεκριμένου τμήματος του χωρίς οι αλλαγές αυτές να επηρεάσουν όλο το υπόλοιπο πρόγραμμα.

Επίσης διευκολύνει οποιονδήποτε χρειαστεί να διαβάσει και να κατανοήσει τον τρόπο που λειτουργεί το πρόγραμμα. Όπως έχει πολλές φορές τονιστεί αυτό είναι πολύ σημαντικό χαρακτηριστικό του σωστού προγραμματισμού, αφού ένα μεγάλο πρόγραμμα στον κύκλο της ζωής του χρειάζεται να συντηρηθεί από διαφορετικούς προγραμματιστές.

**Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος.**

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

**Επεκτείνει τις δυνατότητες των γλώσσων προγραμματισμού.**

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

## 10.4. Παράμετροι

Τα υποπρογράμματα ενεργοποιούνται από κάποιο άλλο πρόγραμμα ή υποπρόγραμμα για να εκτελέσουν συγκεκριμένες λειτουργίες.

Κάθε υποπρόγραμμα για να ενεργοποιηθεί καλείται, όπως λέγεται, από ένα άλλο υποπρόγραμμα ή το αρχικό πρόγραμμα, το οποίο ονομάζεται κύ-

ριο πρόγραμμα. Το υποπρόγραμμα είναι αυτόνομο και ανεξάρτητο τμήμα προγράμματος, αλλά συχνά πρέπει να επικοινωνεί με το υπόλοιπο πρόγραμμα. Συνήθως δέχεται τιμές από το τμήμα προγράμματος που το καλεί και μετά την εκτέλεση επιστρέφει σε αυτό νέες τιμές, αποτελέσματα.

Οι τιμές αυτές που περνούν από το ένα υποπρόγραμμα στο άλλο λέγονται *παράμετροι*.

Οι παράμετροι λοιπόν είναι σαν τις κοινές μεταβλητές ενός προγράμματος με μία ουσιώδη διαφορά, χρησιμοποιούνται για να περνούν τιμές στα υποπρογράμματα.



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

Παραδείγματα παραμέτρων, καθώς και ο τρόπος που περνούν οι τιμές προς και από το υποπρόγραμμα, θα δοθούν στη συνέχεια.

## 10.5. Διαδικασίες και συναρτήσεις

Υπάρχουν δύο ειδών υποπρογράμματα, οι **διαδικασίες** και οι **συναρτήσεις**. Το είδος κάθε υποπρογράμματος καθορίζεται από το είδος της λειτουργίας που καλείται να επιτελέσει.

Οι διαδικασίες μπορούν να εκτελέσουν οποιαδήποτε λειτουργία από αυτές που μπορεί να εκτελέσει ένα πρόγραμμα. Να εισάγουν δεδομένα, να εκτελέσουν υπολογισμούς, να μεταβάλλουν τις τιμές των μεταβλητών και να τυπώσουν αποτελέσματα. Με τη χρήση των παραμέτρων αυτές τις τιμές μπορούν να τις μεταφέρουν και στα άλλα υποπρογράμματα.

Αντίθετα η λειτουργία των συναρτήσεων είναι πιο περιορισμένη. Οι συναρτήσεις υπολογίζουν μόνο μία τιμή, αριθμητική, χαρακτήρα ή λογική και μόνο αυτήν επιστρέφουν στο υποπρόγραμμα που την κάλεσε. Οι συναρτήσεις μοιάζουν με τις συναρτήσεις των μαθηματικών και η χρήση τους είναι όμοια με τη χρήση των ενσωματωμένων συναρτήσεων που υποστηρίζει η γλώσσα προγραμματισμού.

Ο τρόπος κλήσης καθώς και ο τρόπος σύνταξης των δύο αυτών τύπων των υποπρογραμμάτων είναι διαφορετικός. Τόσο οι συναρτήσεις όσο και οι διαδικασίες τοποθετούνται μετά το τέλος του κυρίου προγράμματος.



Οι συναρτήσεις εκτελούνται απλά με την εμφάνιση του ονόματος τους σε οποιαδήποτε έκφραση, ενώ για να εκτελεστούν οι διαδικασίες χρησιμοποιείται η ειδική εντολή ΚΑΛΕΣΕ και το όνομα της διαδικασίας.

Η **συνάρτηση** είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή με το όνομά της (όπως οι μαθηματικές συναρτήσεις).

Η **διαδικασία** είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελεί όλες τις λειτουργίες ενός προγράμματος.



Ας δούμε ένα απλό παράδειγμα χρήσης διαδικασιών και συναρτήσεων.

### Παράδειγμα 2

**Να γραφεί πρόγραμμα, το οποίο υπολογίζει το εμβαδό του κύκλου από την ακτίνα του.**

Το πρόγραμμα εκτελεί τρεις συγκεκριμένες απλές λειτουργίες.

- α) Διαβάζει τα δεδομένα, την ακτίνα η οποία πρέπει να είναι θετικός αριθμός
- β) Υπολογίζει το εμβαδό ( $E = \pi r^2$ )
- γ) Τυπώνει το αποτέλεσμα, το εμβαδό, E

Αν και το πρόγραμμα είναι πολύ απλό και μπορεί κάλλιστα να γραφεί χωρίς τη χρήση υποπρογραμμάτων, ας το διασπάσουμε σε τρία υποπρογράμματα που εκτελούν τις τρεις παραπάνω λειτουργίες.

Τα πρώτο υποπρόγραμμα πρέπει να διαβάζει την ακτίνα και να την επιστρέφει στο κύριο πρόγραμμα. Αφού το υποπρόγραμμα πρέπει να διαβάζει δεδομένα, υλοποιείται με διαδικασία. Η διαδικασία αυτή που ονομάζεται Είσοδος\_δεδομένων, δέχεται από το πληκτρολόγιο την τιμή της ακτίνας που την καταχωρεί στη μεταβλητή Αριθμός και έχει ως εξής:

**ΔΙΑΔΙΚΑΣΙΑ** Είσοδος\_δεδομένων (Αριθμός)

**ΜΕΤΑΒΛΗΤΕΣ**

**ΠΡΑΓΜΑΤΙΚΕΣ :** Αριθμός

**ΑΡΧΗ**

**ΑΡΧΗ\_ΕΠΑΝΑΛΗΨΗΣ**

**ΓΡΑΨΕ** 'Δώσε την ακτίνα'

**ΔΙΑΒΑΣΕ** Αριθμός

**ΜΕΧΡΙΣ\_ΟΤΟΥ** Αριθμός > 0

**ΤΕΛΟΣ\_ΔΙΑΔΙΚΑΣΙΑΣ**

Το δεύτερο πρέπει να υπολογίζει το εμβαδό και να επιστρέφει την τιμή στο κύριο πρόγραμμα. Το υποπρόγραμμα αυτό παίρνει την τιμή της ακτίνας και επιστρέφει μόνο μία τιμή την τιμή του Εμβαδού. Μπορεί λοιπόν να υλοποιηθεί με μία συνάρτηση, η οποία επιστρέφει έναν πραγματικό αριθμό.

Η συνάρτηση Εμβαδό\_κύκλου(R) δέχεται έναν πραγματικό αριθμό και υπολογίζει το εμβαδό που επίσης είναι ένας πραγματικός αριθμός. Το είδος της συνάρτησης, δηλαδή η τιμή που επιστρέφει δηλώνεται στην αρχή της συνάρτησης.

```

ΣΥΝΑΡΤΗΣΗ Εμβαδό_κύκλου(R) : ΠΡΑΓΜΑΤΙΚΗ
ΕΤΑΘΕΡΕΣ
    Π=3.14
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ: R
ΑΡΧΗ
    Εμβαδό_κύκλου <- Π*R^2
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ

```

Το τρίτο υποπρόγραμμα τυπώνει το αποτέλεσμα. Εφόσον απαιτείται από αυτό η εκτέλεση της λειτουργίας της εκτύπωσης, πρέπει να υλοποιηθεί με διαδικασία. Η διαδικασία Εκτύπωση δέχεται από το κύριο πρόγραμμα μια τιμή στη μεταβλητή Αποτέλεσμα και την εκτυπώνει.

```

ΔΙΑΔΙΚΑΣΙΑ Εκτύπωση (Αποτέλεσμα)
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Αποτέλεσμα
ΑΡΧΗ
    ΓΡΑΨΕ 'Το εμβαδό του κύκλου είναι :', Αποτέλεσμα
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ

```

Το κύριο πρόγραμμα που καλεί όλα τα υποπρογράμματα έχει ως εξής:

```

ΠΡΟΓΡΑΜΜΑ Παράδειγμα_2
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : R, E
ΑΡΧΗ
    ΚΑΛΕΣΕ Είσοδος_δεδομένων(R)
    E <- Εμβαδό_κύκλου(R)
    ΚΑΛΕΣΕ Εκτύπωση(E)
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

```

Το πρόγραμμα πλέον έχει ολοκληρωθεί. Όταν εκτελεστεί, θα ζητήσει από το χρήστη να εισάγει μια τιμή για την ακτίνα και θα εμφανίσει το εμβα-

δό του κύκλου. Αν η εισαγόμενη τιμή για την ακτίνα είναι 10, τότε θα η οθόνη θα παρουσιάζει τα εξής:

Δώσε την ακτίνα  
10

Το εμβαδό του κύκλου είναι : 314

Ωστόσο υπάρχουν μερικά λεπτά σημεία που αφορούν στο πέρασμα τιμών, τα οποία θα διευκρινιστούν στη συνέχεια.

### 10.5.1 Ορισμός και κλήση συναρτήσεων

Κάθε συνάρτηση έχει την ακόλουθη δομή.

**ΣΥΝΑΡΤΗΣΗ** όνομα (λίστα παραμέτρων):τύπος συνάρτησης

Τμήμα δηλώσεων

**ΑΡΧΗ**

....

όνομα <- έκφραση

...

**ΤΕΛΟΣ\_ΣΥΝΑΡΤΗΣΗΣ**

Το όνομα της συνάρτησης είναι οποιοδήποτε έγκυρο όνομα της ΓΛΩΣΣΑΣ. Η λίστα παραμέτρων είναι μια λίστα μεταβλητών, των οποίων οι τιμές μεταβιβάζονται στη συνάρτηση κατά την κλήση.

Οι συναρτήσεις μπορούν να επιστρέφουν τιμές όλων των τύπων δεδομένων που υποστηρίζει η γλώσσα. Μια συνάρτηση λοιπόν μπορεί να είναι ΠΡΑΓΜΑΤΙΚΗ, ΑΚΕΡΑΙΑ, ΧΑΡΑΚΤΗΡΑΣ, ΛΟΓΙΚΗ

Στις εντολές του σώματος της συνάρτησης πρέπει υποχρεωτικά να υπάρχει μία εντολή εκχώρησης τιμής στο όνομα της συνάρτησης, στο προηγούμενο παράδειγμα Εμβαδό\_κύκλου<-Π\*Ρ<sup>2</sup>.

Κάθε συνάρτηση εκτελείται, όπως ακριβώς εκτελούνται οι ενσωματωμένες συναρτήσεις της γλώσσας. Απλώς αναφέρεται το όνομα της σε μια έκφραση ή σε μία εντολή και επιστρέφεται η τιμή της. Στο παράδειγμα η συνάρτηση εκτελείται με την εντολή E<-Εμβαδό\_κύκλου(R).

Ο μηχανισμός που επιτυγχάνεται αυτό, είναι ο εξής: Το κύριο πρόγραμμα πριν την κλήση της συνάρτησης γνωρίζει την τιμή της μεταβλητής R. Κατά την κλήση μεταβιβάζεται αυτή η τιμή στην αντίστοιχη μεταβλητή R της συνάρτησης. Η συνάρτηση υπολογίζει το εμβαδό του κύκλου και το αποτέλεσμα αυτό εκχωρείται στο όνομα της συνάρτησης. Με το τέλος της συνάρτησης γίνεται επιστροφή στο κύριο πρόγραμμα, όπου η τιμή του εμβαδού εκχωρείται στη μεταβλητή E.

### 10.5.2 Ορισμός και κλήση διαδικασιών

Κάθε διαδικασία έχει την ακόλουθη δομή.

**ΔΙΑΔΙΚΑΣΙΑ** Όνομα (λίστα παραμέτρων)

Τμήμα δηλώσεων

**ΑΡΧΗ**

εντολές

**ΤΕΛΟΣ\_ΔΙΑΔΙΚΑΣΙΑΣ**

Το όνομα της διαδικασίας είναι οποιοδήποτε έγκυρο όνομα της **ΓΛΩΣΣΑΣ**. Η λίστα παραμέτρων είναι μια λίστα μεταβλητών, των οποίων οι τιμές μεταβιβάζονται προς τη διαδικασία κατά την κλήση ή/και επιστρέφονται στο κύριο πρόγραμμα μετά το τέλος της διαδικασίας. Στο σώμα της διαδικασίας μπορούν να υπάρχουν οποιεσδήποτε εντολές της γλώσσας.

Κάθε διαδικασία εκτελείται όταν καλείται από το κύριο πρόγραμμα ή άλλη διαδικασία. Η κλήση σε διαδικασία πραγματοποιείται με την εντολή **ΚΑΛΕΣΕ**, που ακολουθείται από το όνομα της διαδικασίας συνοδευόμενο μέσα σε παρενθέσεις με τη λίστα παραμέτρων.

Η γενική μορφή της εντολής **ΚΑΛΕΣΕ** είναι

#### Σύνταξη

**ΚΑΛΕΣΕ** όνομα-διαδικασίας (λίστα-παραμέτρων)

#### Παράδειγμα

**ΚΑΛΕΣΕ** Πράξεις (Α, Β, Διαφορά)

#### Λειτουργία

Η εκτέλεση του προγράμματος διακόπτεται και εκτελούνται οι εντολές της διαδικασίας που καλείται. Μετά το τέλος της διαδικασίας η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που ακολουθεί. Η λίστα των παραμέτρων ορίζει τις τιμές που περνούν στη διαδικασία και τις τιμές που αυτή επιστρέφει. Η λίστα παραμέτρων δεν είναι υποχρεωτική.

Στο προηγούμενο παράδειγμα η κλήση των δύο διαδικασιών έγινε με τις εντολές

**ΚΑΛΕΣΕ** Είσοδος\_δεδομένων (R)

**ΚΑΛΕΣΕ** Εκτύπωση (E)

Σε κάθε περίπτωση κλήσης διαδικασίας μπορεί να γίνεται πέρασμα τιμών μέσω της λίστας παραμέτρων. Πιο συγκεκριμένα, στην περίπτωση της διαδικασίας Είσοδος\_δεδομένων γίνεται επιστροφή στο κύριο πρόγραμμα της τιμής της ακτίνας, ενώ στη διαδικασία Εκτύπωση γίνεται μεταβίβαση της τιμής του εμβαδού από το κύριο πρόγραμμα στη διαδικασία. Δηλαδή, η Είσοδος\_δεδομένων δέχεται μια τιμή από πληκτρολόγιο, την εκχωρεί στη μεταβλητή Αριθμός και κατά την επιστροφή (μετά το τέλος της διαδικασίας) γίνεται μεταβίβαση αυτής της τιμής στη μεταβλητή R του κύριου προγράμματος. Αντίθετα στη διαδικασία Εκτύπωση κατά την κλήση της μεταβιβάζεται η τιμή της μεταβλητής E του κύριου προγράμματος στη μεταβλητή Αποτέλεσμα της διαδικασίας.

Στο συγκεκριμένο παράδειγμα κάθε διαδικασία έχει από μία παράμετρο. Στη γενική περίπτωση μπορούν να υπάρχουν καμία, μία ή περισσότερες παράμετροι. Όταν υπάρχουν πολλές παράμετροι, τότε άλλες χρησιμοποιούνται για να μεταβιβάσουν τιμές στη διαδικασία και άλλες για να επιστρέψουν τιμές στο κύριο πρόγραμμα.

Κάθε διαδικασία ή συνάρτηση μπορεί να καλείται από το κύριο πρόγραμμα ή άλλη διαδικασία. Σε κάθε περίπτωση μετά το τέλος της εκτέλεσης της διαδικασίας γίνεται επιστροφή ακριβώς μετά το σημείο απ' όπου κλήθηκε.

Στη συνέχεια παρουσιάζεται το παράδειγμα 2 υλοποιημένο στις γλώσσες Pascal και Basic.

### Προγραμματιστικό περιβάλλον Pascal

```
PROGRAM example2;
  VAR
    r,e:REAL;

  FUNCTION  area(r:REAL):REAL;
  BEGIN
    area:=pi*sqr(r)
  END;

  PROCEDURE input(var x:REAL);
  BEGIN
    REPEAT
      write ('Δώσε την ακτίνα:');
      readln(x)
    UNTIL x>0;
  END;
```

```

PROCEDURE output(result:REAL);
BEGIN
    writeln (To εμβαδό είναι :',result:6:2)
END;

BEGIN
    input(r);
    e:=area(r);
    output(e)
END.

```

### Προγραμματιστικό περιβάλλον Basic

```

\ Παράδειγμα 3
DECLARE SUB Eisodos (nb!)
DECLARE SUB Ektypwsh (res!)
DECLARE FUNCTION Emvado! (r!)
CLS
CALL Eisodos(r)
e = Emvado(r)
CALL Ektypwsh(e)
END

SUB Eisodos (nb)
DO
    INPUT "Δώσε την ακτίνα : ", nb
LOOP UNTIL nb > 0
END SUB

SUB Ektypwsh (res)
PRINT "Το εμβαδό του κύκλου είναι :"; res
END SUB

FUNCTION Emvado (r)
pi = 3.14
Emvado = pi * r ^ 2
END FUNCTION

```

### 10.5.3 Πραγματικές και τυπικές παράμετροι

Η κατανόηση του τρόπου που γίνεται η ανταλλαγή των τιμών ανάμεσα στις παραμέτρους είναι ιδιαίτερα σημαντική και γι αυτό ας παρακολουθήσουμε το επόμενο παράδειγμα.

## Παράδειγμα 3

Να γραφεί μια διαδικασία η οποία δέχεται στην είσοδο δύο τιμές και υπολογίζει και επιστρέφει το άθροισμα και τη διαφορά τους.

**ΠΡΟΓΡΑΜΜΑ** Παράδειγμα\_3

...  
A<-5  
B<-7

**ΚΑΛΕΣΕ** Πράξεις(A, B, Διαφ1, Αθρ1)

...  
A<-9  
B<-6

**ΚΑΛΕΣΕ** Πράξεις(A, B, Διαφ2, Αθρ2)

...

**ΔΙΑΔΙΚΑΣΙΑ** Πράξεις(X, Y, Διαφορά, Άθροισμα)

**ΜΕΤΑΒΛΗΤΕΣ**

**ΠΡΑΓΜΑΤΙΚΕΣ** : X, Y, Διαφορά, Άθροισμα

**ΑΡΧΗ**

Διαφορά <- X-Y

Άθροισμα <- X+Y

**ΤΕΛΟΣ\_ΔΙΑΔΙΚΑΣΙΑΣ**

Οι μεταβλητές A, B, Διαφ1, Αθρ1, A, B, Διαφ2, Αθρ2 είναι μεταβλητές του προγράμματος Παράδειγμα\_3 και αποτελούν τις **πραγματικές** παραμέτρους, ενώ οι μεταβλητές X, Y, Διαφορά, Άθροισμα είναι μεταβλητές της διαδικασίας Πράξεις, και ονομάζονται **τυπικές** παράμετροι.

Οι μεταβλητές X, Y, Διαφ1 καθώς και όλες οι μεταβλητές του προγράμματος Παράδειγμα\_3 δεν είναι γνωστές στη διαδικασία Πράξεις και αντίστοιχα όλες οι μεταβλητές της διαδικασίας Πράξεις είναι άγνωστες στο πρόγραμμα Παράδειγμα\_3. Τα ονόματα των τυπικών και των πραγματικών παραμέτρων μπορούν να είναι οποιαδήποτε. Αφού είναι ονόματα μεταβλητών σε διαφορετικά τμήματα προγράμματος, είναι υποχρεωτικά διαφορετικές μεταβλητές, άσχετα αν έχουν το ίδιο όνομα.

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



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.

Ας παρακολουθήσουμε πώς γίνεται η επικοινωνία ανάμεσα στο πρόγραμμα Παράδειγμα\_3 και τη διαδικασία Πράξεις.

Οι τιμές που υπάρχουν στις μεταβλητές του προγράμματος A,B, Διαφ1 και Αθρ1 δίνονται κατά την κλήση στις μεταβλητές της διαδικασίας X, Y, Διαφορά, Άθροισμα.

Έτσι η μεταβλητή X παίρνει την τιμή 5 κι η Y την τιμή 7. Οι μεταβλητές Διαφορά και Άθροισμα δεν παίρνουν καμία τιμή, αφού οι αντίστοιχες μεταβλητές Διαφ1 και Αθρ1 δεν έχουν συγκεκριμένη τιμή.



Μερικές γλώσσες προγραμματισμού ονομάζουν **ορίσματα** τις τυπικές παραμέτρους και απλά **παραμέτρους** τις πραγματικές παραμέτρους.

Μετά την εκτέλεση των εντολών της διαδικασίας, όταν εκτελεστεί η εντολή ΤΕΛΟΣ\_ΔΙΑΔΙΚΑΣΙΑΣ, οι μεταβλητές της διαδικασίας που αναφέρονται στη δήλωση της διαδικασίας δίνουν τις τιμές που περιέχουν στις αντίστοιχες μεταβλητές που περιλαμβάνονται στην κλήση της διαδικασίας Πράξεις. Έτσι η A παίρνει την τιμή της X (=5), η B την τιμή της Y (= 7), η Διαφ1 της Διαφορά (= -2) και η μεταβλητή Αθρ1 της Άθροισμα (=12). Με την επιστροφή στο κύριο πρόγραμμα όλες οι θέσεις μνήμης που είχαν δοθεί στη διαδικασία απελευθερώνονται.

Οι λίστες των παραμέτρων πρέπει να ακολουθούν τους εξής κανόνες:

A	5
B	7
Διαφ1	
Αθρ1	

(α)

A	5	→	5	X
B	7	→	7	Y
Διαφ1		→		Διαφορά
Αθρ1		→		Άθροισμα

(β)

5	X
7	Y
-2	Διαφορά
12	Άθροισμα

(γ)

A	5	←	5	X
B	7	←	7	Y
Διαφ1	-2	←	-2	Διαφορά
Αθρ1	12	←	12	Άθροισμα

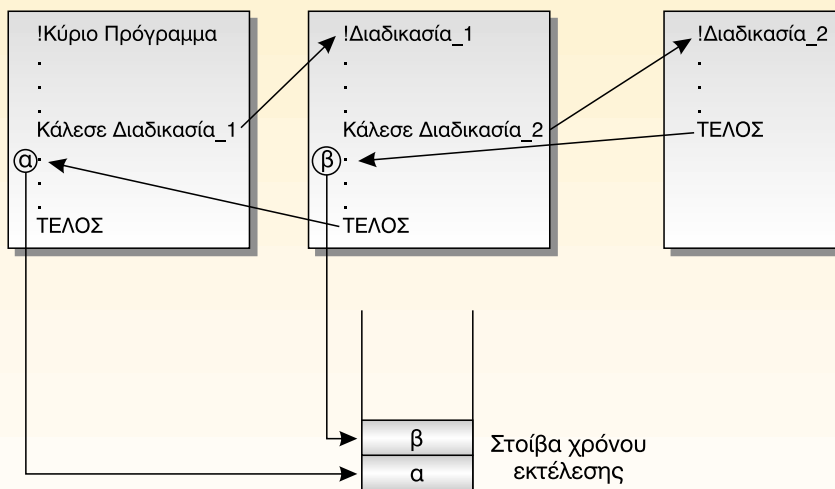
(δ)

**Σχ. 10.2.** Πέρασμα παραμέτρων κατά την κλήση διαδικασιών (α) Κατάσταση πριν την κλήση (β) Μεταβίβαση τιμών των μεταβλητών A και B στις X και Y αντίστοιχα (γ) Στη διαδικασία εκχωρούνται τιμές στις μεταβλητές Διαφορά και Άθροισμα (δ) Οι τιμές των τελευταίων επιστρέφονται στις Διαφ1 και Αθρ1 μετά το τέλος της διαδικασίας.



## Η χρήση στοίβας στην κλήση διαδικασιών

Η έννοια της στοίβας είναι πολύ χρήσιμη στο ίδιο το λογισμικό των γλωσσών προγραμματισμού. Όταν μία διαδικασία ή συνάρτηση καλείται από το κύριο πρόγραμμα, τότε η αμέσως επόμενη διεύθυνση του κύριου προγράμματος, που ονομάζεται *διεύθυνση επιστροφής* (return address), αποθηκεύεται από το μεταφραστή σε μία στοίβα που ονομάζεται *στοίβα χρόνου εκτέλεσης* (execution time stack). Μετά την εκτέλεση της διαδικασίας ή της συνάρτησης η διεύθυνση επιστροφής απωθείται από τη στοίβα και έτσι ο έλεγχος του προγράμματος μεταφέρεται και πάλι στο κύριο πρόγραμμα. Η τεχνική αυτή εφαρμόζεται και γενικότερα, δηλαδή οποτεδήποτε μία διαδικασία ή συνάρτηση καλεί μία διαδικασία ή συνάρτηση. Για παράδειγμα, έστω ότι μία διαδικασία *a* καλεί τη διαδικασία *b*, που με τη σειρά της καλεί τη διαδικασία *c* κ.ο.κ. Στην περίπτωση αυτή οι διευθύνσεις επιστροφής εμφανίζονται στη στοίβα με σειρά *c*, *b*, *a*. Μετά την εκτέλεση κάθε διαδικασίας, η διεύθυνση επιστροφής απωθείται από τη στοίβα και ο έλεγχος μεταβιβάζεται στη διεύθυνση αυτή. Το παράδειγμα αυτό δείχνει μία από τις πολλές χρησιμότητες της LIFO ιδιότητας της στοίβας.



Σχ. 10.3. Χρήση στοίβας από το μεταφραστή για το χειρισμό κλήσεων διαδικασιών και επιστροφών από αυτές.

- ⇒ Ο αριθμός των πραγματικών και των τυπικών παραμέτρων πρέπει να είναι ίδιος.
- ⇒ Κάθε πραγματική παράμετρος αντιστοιχεί στην τυπική παράμετρο που βρίσκεται στην αντίστοιχη θέση. Για παράδειγμα η πρώτη της λίστας των τυπικών παραμέτρων στην πρώτη της λίστας των πραγματικών παραμέτρων κοκ.
- ⇒ Η τυπική παράμετρος και η αντίστοιχη της πραγματική πρέπει να είναι του ίδιου τύπου.

## 10.6. Εμβέλεια μεταβλητών-σταθερών.

Κάθε κύριο πρόγραμμα όπως και κάθε υποπρόγραμμα περιλαμβάνει τις δικές του μεταβλητές και σταθερές.

Οι μεταβλητές αυτές στη **ΓΛΩΣΣΑ** είναι γνωστές στο αντίστοιχο υποπρόγραμμα που δηλώνονται και μόνο σε αυτό. Όλες οι μεταβλητές (και οι σταθερές) είναι **τοπικές** στο συγκεκριμένο τμήμα προγράμματος.

Ο μόνος τρόπος για να περάσει μία τιμή από ένα υποπρόγραμμα σε ένα άλλο ή από το κυρίως πρόγραμμα σε ένα υποπρόγραμμα είναι δια μέσου των παραμέτρων κατά το στάδιο της κλήσης του υποπρογράμματος και μετά το τέλος της εκτέλεσης του υποπρογράμματος.

Ας δούμε τον παρακάτω σκελετό προγράμματος

```

ΠΡΟΓΡΑΜΜΑ Αρχικό
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Α, Β, Γ
ΑΡΧΗ
...
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
ΔΙΑΔΙΚΑΣΙΑ Πρώτη
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Δ, Ε, Ζ, Η
ΑΡΧΗ
...
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
ΔΙΑΔΙΚΑΣΙΑ Δεύτερη
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ : Γ, Θ, Ι
ΑΡΧΗ
...
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ

```

Οι μεταβλητές του προγράμματος Αρχικό με ονόματα Α, Β, Γ είναι γνωστές, ισχύουν μόνο για το πρόγραμμα. Έξω από το πρόγραμμα σε όλα τα υποπρογράμματα οι μεταβλητές αυτές δεν ισχύουν. Επίσης η διαδικασία Πρώτη έχει τις πραγματικές μεταβλητές Δ, Ε, Ζ, Η, οι οποίες ισχύουν μόνο για τη συγκεκριμένη διαδικασία και όχι για τα υπόλοιπα υποπρογράμματα ή το κύριο πρόγραμμα.

Η διαδικασία Δεύτερη έχει και αυτή τις δικές της μεταβλητές Γ, Θ, Ι. Οι μεταβλητές αυτές ισχύουν μόνο για τη διαδικασία Δεύτερη. Η μεταβλητή με το όνομα Γ δεν έχει καμία σχέση με τη μεταβλητή Γ του κύριου προγράμματος, η μία είναι τύπου Ακεραίου και η άλλη τύπου Πραγματικού. Αφού όλες οι μεταβλητές είναι τοπικές, το ίδιο όνομα μεταβλητής μπορεί να εμφανίζεται σε διαφορετικά τμήματα προγράμματος, χωρίς να αντιστοιχεί στην ίδια μεταβλητή. Το ίδιο έγινε και με τη μεταβλητή R της ακτίνας του κύκλου στο παράδειγμα 2.

Ότι ισχύει για τις μεταβλητές ισχύει και για τις σταθερές.

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 γλώσσας



Το τμήμα του προγράμματος που ισχύουν οι μεταβλητές λέγεται **εμβέλεια** (scope) μεταβλητών.

### Απεριόριστη εμβέλεια.

Σύμφωνα με αυτή την αρχή 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.

### Περιορισμένη εμβέλεια

Η περιορισμένη εμβέλεια υποχρεώνει όλες τις μεταβλητές που χρησιμοποιούνται σε ένα τμήμα προγράμματος, να δηλώνονται 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.

### Μερικώς περιορισμένη εμβέλεια.

Σύμφωνα με αυτή την αρχή 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.

## 10.7. Αναδρομή

Ενα υποπρόγραμμα καλείται από το κύριο πρόγραμμα ή άλλο υποπρόγραμμα. Υπάρχει όμως και η δυνατότητα ένα υποπρόγραμμα να καλεί τον εαυτό του. Η δυνατότητα αυτή αποκαλείται αναδρομή.

Η έννοια της αναδρομής παρουσιάστηκε στο κεφάλαιο 3, όπου δόθηκαν παραδείγματα και έγινε μία πρώτη αναφορά στα πλεονεκτήματα, αλλά και τα προβλήματα που παρουσιάζει η χρήση αναδρομικών αλγορίθμων.

Η αναδρομή υλοποιείται στον προγραμματισμό με χρήση αναδρομικών υποπρογραμμάτων και επειδή τόσο η αναδρομή ως έννοια όσο και η λειτουργία της είναι αρκετά πολύπλοκες, θα παρουσιαστούν αναλυτικά στη συνέχεια.

Ο ορισμός κάθε αναδρομικού υποπρογράμματος έχει δύο τμήματα, την αναδρομική σχέση και την τιμή βάσης ή συνθήκη τερματισμού

Ας εξετάσουμε το παράδειγμα υπολογισμού του παραγοντικού:

Το παραγοντικό ορίζεται αναδρομικά ως εξής:

$$n! = \begin{cases} n \cdot (n-1)! & \text{αν } n > 0 \\ 1 & \text{αν } n = 0 \end{cases}$$

Τιμή βάσης ή συνθήκη τερματισμού είναι η τιμή που ορίζεται για μία συγκεκριμένη τιμή της παραμέτρου της συνάρτησης, στο παράδειγμα η τιμή είναι 1 για  $n=0$ .



Αναδρομή ονομάζεται η δυνατότητα ενός υποπρογράμματος να καλεί τον εαυτό του.

Αναδρομική σχέση είναι η  $n*(n-1)!$ , όπου η τιμή της συνάρτησης υπολογίζεται με βάση τις προηγούμενες τιμές της συνάρτησης, οι οποίες πρέπει να υπολογιστούν.

Οι συναρτήσεις στη γλώσσα που υλοποιούν τον υπολογισμό του παραγοντικού τόσο αναδρομικά όσο και επαναληπτικά είναι οι εξής.

**ΣΥΝΑΡΤΗΣΗ** Παραγοντικο(N) : **ΑΚΕΡΑΙΑ**

**!Υπολογισμός του παραγοντικού με αναδρομική διαδικασία**  
**ΜΕΤΑΒΛΗΤΕΣ**

**ΑΚΕΡΑΙΕΣ:** N

**ΑΡΧΗ**

**ΑΝ** N=0 **ΤΟΤΕ**

Παραγοντικο  $\leftarrow$  1

**ΑΛΛΙΩΣ**

Παραγοντικο  $\leftarrow$  N\*Παραγοντικο(N-1)

**ΤΕΛΟΣ\_ΑΝ**

**ΤΕΛΟΣ\_ΣΥΝΑΡΤΗΣΗΣ**

**ΣΥΝΑΡΤΗΣΗ** Παραγοντικο(N) : **ΑΚΕΡΑΙΑ**

**!Υπολογισμός του παραγοντικού με επαναληπτική διαδικασία**  
**ΜΕΤΑΒΛΗΤΕΣ**

**ΑΚΕΡΑΙΕΣ:** i, N

**ΑΡΧΗ**

Fact  $\leftarrow$  1

**ΓΙΑ** i **ΑΠΟ** 2 **ΜΕΧΡΙ** N

Fact  $\leftarrow$  Fact\*i

**ΤΕΛΟΣ\_ΕΠΑΝΑΛΗΨΗΣ**

**ΤΕΛΟΣ\_ΣΥΝΑΡΤΗΣΗΣ**

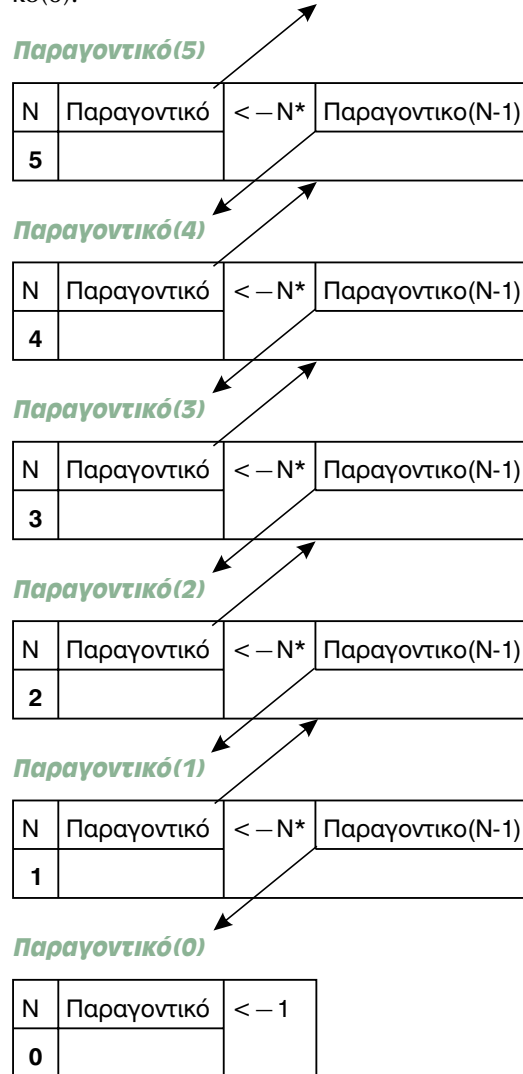
Και οι δύο αυτές συναρτήσεις, η πρώτη αναδρομική και η δεύτερη επαναληπτική έχουν το ίδιο αποτέλεσμα. Το αναδρομικό υποπρόγραμμα επειδή εκφράζει ακριβώς την αλγεβρική συνάρτηση υπολογισμού του παραγοντικού είναι πιο απλό στη διατύπωση, αλλά είναι πιο δύσκολο στην κατανόηση του τρόπου που λειτουργεί.

Γενικά η έννοια της αναδρομής καθώς και ο τρόπος με τον οποίο εκτελείται μία αναδρομική διαδικασία από τον υπολογιστή παρουσιάζει στην αρχή δυσκολία στην κατανόηση της.

Ας παρακολουθήσουμε τον τρόπο με τον οποίο υπολογίζεται το 5! με τη χρήση της αναδρομικής συνάρτησης Παραγοντικο().

Εφόσον η παράμετρος της συνάρτησης είναι αρχικά 5, διάφορη δηλαδή του 0, καλείται η συνάρτηση Παραγοντικό(4). Στην συνέχεια επειδή η πα-

ράμετρος είναι διάφορη του μηδενός, καλείται το Παραγοντικό(3) και η διαδικασία συνεχίζεται όπως δείχνει το σχήμα μέχρι την κλήση του Παραγοντικού(0).



Για τη τιμή της παραμέτρου  $N=0$  το Παραγοντικό(0) έχει τιμή 1, την τιμή βάσης. Έτσι αρχίζει η αντίστροφη διαδικασία υπολογισμού διαδοχικά του υπολογισμού των συναρτήσεων  $\text{Παραγοντικό}(2)=1*2$ ,  $\text{Παραγοντικό}(3)=2*3$ ,  $\text{Παραγοντικό}(4)=6*4$  όπως δείχνει το παρακάτω σχήμα.

Το τελευταίο βήμα είναι ο υπολογισμός του  $\text{Παραγοντικό}(5)=24*5$ . Η τιμή αυτή δηλαδή 120 είναι η τιμή που η συνάρτηση τελικά επιστρέφει.

**Παραγοντικό(5)**

N	Παραγοντικό	< - N*	Παραγοντικο(N-1)
5	120		

**Παραγοντικό(4)**

N	Παραγοντικό	< - N*	Παραγοντικο(N-1)
4	24		

**Παραγοντικό(3)**

N	Παραγοντικό	< - N*	Παραγοντικο(N-1)
3	6		

**Παραγοντικό(2)**

N	Παραγοντικό	< - N*	Παραγοντικο(N-1)
2	2		

**Παραγοντικό(1)**

N	Παραγοντικό	< - N*	Παραγοντικο(N-1)
1	1		

**Παραγοντικό(0)**

N	Παραγοντικό	< - 1
0	1	



Ο μηχανισμός διαχείρισης των αναδρομικών διαδικασιών από το μεταφραστή κάνει χρήση μιας στοιβάς. Κατά την κλήση μιας αναδρομικής διαδικασίας από την ίδια φυλάσσονται σε μία στοιβά ό-λες οι τιμές των μεταβλητών, που έχουν πριν την κλήση (λειτουργία ώθησης). Κατά την επιστροφή ανασύρονται οι τι-μές αυτές από τη στοιβά (λειτουργία απώθησης) και έ-τσι η διαδικασία μπορεί να συνεχίσει τη λειτουργία της με τις προηγούμενες τιμές. Προφανώς, αν υπάρχουν δια-δοχικές κλήσεις της διαδικα-σίας, λόγω της ιδιότητας LIFO με την οποία λειτουργεί η στοιβά, εξασφαλίζεται ότι ε-ξάγονται από αυτή οι πλέον πρόσφατες τιμές των μετα-βλητών, δηλαδή αυτές που α-ποθηκεύτηκαν κατά την τε-λευταία κλήση.

Πολύ συχνά προβλήματα τα οποία μπορούν να λυθούν με χρήση αναδρομής λύνονται και με απλούς επαναληπτικούς αλγόριθμους. Παραδείγ-ματα παρουσιάστηκαν στο κεφάλαιο 3, όπου έγινε και σύγκριση μεταξύ των δύο μεθόδων. Εδώ επαναλαμβάνουμε ότι αν και η αναδρομή φαίνεται ως πιο φυσικός τρόπος προγραμματισμού, πρέπει να χρησιμοποιείται με μέτρο, γιατί το κέρδος σε χρόνο προγραμματισμού δημιουργεί συχνά απώ-λεια σε χρόνο εκτέλεσης.

### Προγραμματιστικό περιβάλλον Pascal

```
FUNCTION factorial(n:INTEGER) : INTEGER;
BEGIN
    IF n=0 THEN
        factorial:=1
    ELSE
        factorial:=n*factorial(n-1);
    END;

FUNCTION factorial(n:INTEGER) : INTEGER;
VAR
    i, fact:integer;
BEGIN
    fact:=1;
    FOR i:=2 TO n DO
        fact:=i*fact;
        factorial:=fact
    END;
```

### Προγραμματιστικό περιβάλλον Basic

```
FUNCTION Factorial (n)
IF n = 0 THEN
    Factorial = 1
ELSE
    Factorial = n * Factorial(n - 1)
END IF
END FUNCTION

FUNCTION Factorial (n)
fact = 1
FOR i = 2 TO n
    fact = fact * i
NEXT i
Factorial = fact
END FUNCTION
```



## Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάστηκαν οι αρχές και τα χαρακτηριστικά του τμηματικού προγραμματισμού και ο τρόπος που χειρίζεται τα υποπρογράμματα η ΓΛΩΣΣΑ.

Η χρήση υποπρογραμμάτων σε ένα πρόγραμμα παρουσιάζει πολλά πλεονεκτήματα, αφού διευκολύνει την ανάπτυξη των αλγόριθμων και την υλοποίηση του προγράμματος σε λιγότερο μάλιστα χρόνο και διευκολύνει την κατανόηση και τη διόρθωσή του. Υπάρχουν δύο ειδών υποπρογράμματα, οι συναρτήσεις και οι διαδικασίες. Η συνάρτηση παράγει ένα αποτέλεσμα και επιστρέφει μόνο μία τιμή, ενώ οι διαδικασίες μπορούν να εκτελούν όλες τις ενέργειες ενός προγράμματος. Τα υποπρογράμματα επικοινωνούν μεταξύ τους καθώς και με το κύριο πρόγραμμα με τη βοήθεια των παραμέτρων. Στη δήλωση του υποπρογράμματος βρίσκονται οι τυπικές παράμετροι, ενώ οι πραγματικές παράμετροι βρίσκονται στην εντολή κλήσης του υποπρογράμματος. Οι διαδικασίες ενεργοποιούνται στη ΓΛΩΣΣΑ με την εντολή ΚΑΛΕΣΕ, ενώ οι συναρτήσεις μόνο με την αναγραφή του ονόματός τους μέσα σε μία έκφραση.

Η δυνατότητα ενός υποπρογράμματος να καλεί τον εαυτό του, ονομάζεται αναδρομή. Πολύ συχνά τα προβλήματα που λύνονται με αναδρομή μπορούν να λυθούν με χρήση απλής επανάληψης.



## Λέξεις κλειδιά

Τμήμα προγράμματος, διαδικασία, συνάρτηση, παράμετρος, εμβέλεια παραμέτρων, αναδρομή.



## Ερωτήσεις - Θέματα για συζήτηση

1. Πως ορίζεται ο τμηματικός προγραμματισμός;
2. Ποια τα βασικά χαρακτηριστικά των υποπρογραμμάτων;
3. Ποια η διαφορά παραμέτρων και απλών μεταβλητών;
4. Ποια η βασική διαφορά διαδικασιών και συναρτήσεων;
5. Πώς εκτελείται μία συνάρτηση;
6. Πώς καλείται μία διαδικασία;
7. Ποια η διαφορά τυπικών και πραγματικών παραμέτρων;



8. Πώς γίνεται η ανταλλαγή των τιμών ανάμεσα στις τυπικές και πραγματικές παραμέτρους; Δώσε ένα παράδειγμα ανταλλαγής παραμέτρων.
9. Τι ονομάζεται εμβέλεια μεταβλητών;
10. Τι είναι η αναδρομή;
11. Ποια είναι τα τμήματα που περιλαμβάνει κάθε αναδρομικός ορισμός; Δώσε ένα παράδειγμα.

### Βιβλιογραφία

1. Θ. Αλεβίζος, Α. Καμπουρέλης, *Εισαγωγή με τη γλώσσα Pascal*, Αθήνα, 1984.
2. Γ. Βουτυράς, *Basic: Αλγόριθμοι και εφαρμογές*, Κλεψύδρα, Αθήνα, 1991
3. Χρ. Κοίλιας, *Η QuickBasic και οι εφαρμογές της*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.
4. R. Shackelford, *Introduction to Computing and Algorithms*, Addison-Wesley, USA, 1998.
5. S. Leestma-L.Nyhoff, *Turbo Pascal, Programming and Solving*, McMillan, New York, 1990.
6. N. Wirth, *Systematic Programming: An introduction*, Prentice Hall, 1973.

### Διευθύνσεις Διαδικτύου

- ⇒ <http://www.swcp.com/~dodril/>
- ⇒ <http://www.progsources.com>
- ⇒ [www.cit.ac.nz/smac/pascal/default.htm](http://www.cit.ac.nz/smac/pascal/default.htm)
- ⇒ <http://www.cs.vu.nl/~jprins/tp.html>
- ⇒ <http://qbasic.com/>
- ⇒ [www.basicguru.com](http://www.basicguru.com)

Επίσης στο διαδίκτυο παρουσιάζουν ενδιαφέρον οι ακόλουθες ομάδες νέων (Usenet):

[comp.lang.pascal](#) και [comp.lang.pascal.misc](#) σχετικές με τη γλώσσα Pascal.  
[alt.lang.basic](#) και [comp.lang.basic.misc](#) σχετικές με τη γλώσσα Basic.

