**ΣΥΝΑΡΤΗΣΕΙΣ**

**Εισαγωγή**

*Ιδανικό του δομημένου προγραμματισμού:*

Να διαβάζουμε ένα πρόγραμμα από την αρχή ως το τέλος χωρίς να το βλέμμα μας να κάνει ούτε ένα άλμα προς τα εμπρός ή προς τα πίσω και στο τέλος να έχουμε καταλάβει τι κάνει και πώς.

Αυτό επιτυγχάνεται με τη βοήθεια των τριών στοιχειωδών δομών (επιλογή, επανάληψη και ακολουθία ή "συνένωση" αναθέσεων ή των άλλων δύο δομών). Όμως για μεγάλα προγράμματα χάνεται η "εποπτεία". Αυτή ανακτάται όταν αναφερόμαστε σε μια ομάδα εντολών με ένα όνομα που είναι το όνομα μιας συνάρτησης, ιδίως για ομάδες εντολών που επαναλαμβάνονται συχνά σε διάφορα σημεία του προγράμματος.

Έτσι, οδηγούμαστε στον "**αρθρωτό**" (modular) προγραμματισμό – σπάσιμο του μεγάλου προγράμματος σε μικρότερα υποπρογράμματα. Αυτή η διαδικασία μπορεί να επαναληφθεί αναδρομικά και για τα επιμέρους υποπρογράμματα όταν αυτά αναπτύσσονται και φτάνουν σε μεγάλο μέγεθος. Αυτό συνεπάγεται ένα ακόμη βήμα στην καθιέρωση της συστηματικής ανάπτυξης λογισμικού, την **ιεραρχική** σχεδίαση (δηλαδή σε πολλά επίπεδα ανάλυσης που σχηματίζουν μια ιεραρχία με το κύριο πρόγραμμα στην κορυφή να καλεί άλλα στο αμέσως κατώτερο επίπεδο κι αυτά με τη σειρά τους άλλα κλπ)

Ο κατακερματισμός ενός προγράμματος σε πολλά μικρότερα τμήματα

α) διευκολύνει τον εντοπισμό σφαλμάτων

β) επιτρέπει την επαναχρησιμοποίηση μερικών τμημάτων με γενικά χαρακτηριστικά και σε άλλα προγράμματα.

Ως συνέπεια των ανωτέρω, η ανάπτυξη μεγάλων προγραμμάτων διευκολύνεται και επιταχύνεται σημαντικά.

Τα παραπάνω μας οδηγούν στην εξής διαδικασία ανάπτυξης:

α) ανάλυση απαιτήσεων (είσοδος – έξοδος)

β) ανάλυση κύριου προγράμματος σε επιμέρους διαδικασίες

γ) καθορισμός της μεταξύ τους διασύνδεσης (λογικό διάγραμμα)

δ) επανάληψη του (β) και (γ) για τις επιμέρους διαδικασίες μέχρι να φτάσουμε σε ένα επίπεδο όπου έχουμε να αναπτύξουμε πολύ απλές στοιχειώδεις διαδικασίες

ε) ανάπτυξη κώδικα των στοιχειωδών διαδικασιών

στ) ανάπτυξη κάθε διαδικασίας των ανώτερων επιπέδων χρησιμοποιώντας τις συναρτήσεις των κατώτερων μέχρι να φτάσουμε στο κύριο πρόγραμμα.

Φυσικά, στην πορεία μπορεί να υπάρξει και αναθεώρηση του σχεδιασμού σε διάφορα επίπεδα αν ανακαλυφθούν στοιχεία που είχαν παραβλεφθεί αρχικά ή αν η διασύνδεση των υποπρογραμμάτων δεν είναι εφικτή με τη συγκεκριμένη γλώσσα προγραμματισμού με τον τρόπο που είχε σχεδιαστεί αρχικά.

*Ιδανικό μέγεθος συνάρτησης*: περίπου μισή με μία οθόνη.

Η ιδανική συνάρτηση κρύβει τις λειτουργικές της λεπτομέρειες από τα μέρη του προγράμματος που δε χρειάζεται να τις γνωρίζουν, δίνοντας πιο σαφές σύνολο και κάνοντας πιο εύκολες τις αλλαγές.

*Κύρια χαρακτηριστικά των συναρτήσεων*

Όπως έχουμε ξαναπεί μια συνάρτηση έχει

- τύπο
- όνομα
- λίστα παραμέτρων ή ορισμάτων

Ο τύπος μπορεί να είναι κάποιος από τους τύπους που παίρνουν και οι μεταβλητές ή void. **Προσοχή!** αν παραλειφθεί ο τύπος τότε είναι αυτομάτως int και όχι void!

Το όνομα υπακούει στους ίδιους κανόνες με τις μεταβλητές (το πολύ 31 χαρακτήρες, αρχίζει από γράμμα, συνδυασμός γραμμάτων, αριθμητικών ψηφίων και _ ).

Η λίστα παραμέτρων αποτελείται από παρενθέσεις αμέσως μετά από το όνομα οι οποίες επιτρέπουν τη διάκριση των συναρτήσεων από τις κοινές μεταβλητές. Μέσα στις παρενθέσεις γράφονται οι

παράμετροι αν υπάρχουν. Όταν δηλώνεται η συνάρτηση μέσα στις παρενθέσεις γράφεται ο τύπος και το όνομα κάθε παραμέτρου ενώ όταν καλείται μέσα στο πρόγραμμα γράφονται απλώς τα ονόματα των μεταβλητών ή οι σταθερές που παίζουν το ρόλο παραμέτρων. Πιο κάτω θα πούμε και για τα πρωτότυπα συναρτήσεων όπου μπορούμε να γράφουμε μόνο τους τύπους των και όχι απαραίτητα τα ονόματα των παραμέτρων.
*Προσοχή!* Αν δεν υπάρχουν παράμετροι πρέπει να γράφουμε τη λέξη void γιατί λόγοι συμβατότητας με παλιότερες εκδόσεις της C οδηγούν την κενή λίστα να εκλαμβάνεται ως δήλωση παλαιού τύπου και αναστέλλεται κάθε έλεγχος των ορισμάτων με αποτέλεσμα αυξημένη πιθανότητα λάθους κατά την ανάπτυξη του προγράμματος.
*Ένα συνηθισμένο λάθος*: αν ένα όνομα που δεν έχει δηλωθεί προηγουμένως ακολουθείται από άνοιγμα παρένθεσης, τότε δηλώνεται από το πλαίσιο αναφοράς (συμφραζόμενα) σαν όνομα συνάρτησης, και μάλιστα τύπου int και δε γίνεται καμία παραδοχή για τα ορίσματά της. Δηλαδή ο μεταγλωττιστής δεν το θεωρεί πρόβλημα και μετά ψάχνουμε να βρούμε το λάθος που προκύπτει κάπου αλλού με άλλο "πρόσωπο". Γι' αυτό προσέχουμε να μην αφήνουμε αδήλωτες συναρτήσεις.

*Σημείωση για την ορολογία*: όταν λέμε *παράμετρος* θα εννοούμε τη μεταβλητή που κατονομάζεται στην εντός παρενθέσεων λίστα στον ορισμό της συνάρτησης. Όταν λέμε *όρισμα* θα εννοούμε την τιμή που χρησιμοποιείται σε κάποια κλήση της συνάρτησης. Αντίστοιχα, λέμε και *τυπικό όρισμα* και *πραγματικό όρισμα*.
Μετά τό όνομα και τις παρενθέσεις ακολουθούν τα γνωστά άγκιστρα όπου μπορεί να υπάρχουν δηλώσεις μεταβλητών, αποδόσεις αρχικών τιμών και διάφορες εντολές κάθε είδους. Αυτό είναι το **σώμα** της συνάρτησης. Στο τέλος, πριν από το άγκιστρο που κλείνει το σώμα, μπορεί να υπάρχει μία εντολή της μορφής

```
return παράσταση;
```

ή

```
return (παράσταση);
```

(οι δυο μορφές είναι ισοδύναμες – οι παρενθέσεις τοποθετούνται για έμφαση).
Η παράσταση είναι η τιμή που επιστρέφει η συνάρτηση και ο τύπος της καλό είναι να συμφωνεί με τον τύπο της συνάρτησης – δεν είναι απαραίτητο, π.χ. αν η παράσταση είναι πραγματική και ο τύπος ακέραιος θα παραλειφθούν τα δεκαδικά ψηφία, γιατί ο υπολογιστής εκτελεί μετατροπή τύπου (casting).
Επειδή μερικοί μεταγλωττιστές εμφανίζουν προειδοποιητικά μηνύματα στην περίπτωση αλλαγής τύπου, καλό θα ήταν να υπάρχει ρητή μετατροπή τύπου ώστε να δείχνει ότι κάτι τέτοιο είναι σκόπιμο, π.χ. στην παρακάτω εντολή, το string μετατρέπεται σε τύπο float (με τη βοήθεια της συνάρτησης βιβλιοθήκης atof που δηλώνεται στο αρχείο stdlib.h) και μετά αυτό που προκύπτει μετατρέπεται σε int:

```
return (int) atof(string);
```

Μπορεί μία συνάρτηση να μην επιστρέφει τιμή (π.χ. απλώς τυπώνει μηνύματα) οπότε παραλείπεται η παράσταση ή και ολόκληρη η εντολή return. *Αυτές οι συναρτήσεις είναι τύπου* ***void***.
Αν υπάρχει εντολή return κάπου μέσα στο σώμα της συνάρτησης η εκτέλεσή της διακόπτεται εκεί!
Αν η συνάρτηση επιστρέφει τιμή αυτό δε σημαίνει ότι αυτή πρέπει να αποδοθεί κάπου. Η καλούσα συνάρτηση μπορεί να χρησιμοποιήσει την επιστρεφόμενη τιμή, αλλά μπορεί και όχι. Π.χ. μπορεί να επιστρέφουμε μια τιμή για να δείξουμε ότι ολοκληρώθηκε κανονικά μια διαδικασία, αλλά να μη μας ενδιαφέρει πάντα να ελέγξουμε κάτι τέτοιο.
Αν και δεν απαγορεύεται μία συνάρτηση να έχει στο εσωτερικό της μία διακλάδωση που σε μία περίπτωση επιστρέφει τιμή και σε άλλη όχι, αυτό μάλλον είναι κακός ή και λανθασμένος σχεδιασμός και θα ήταν καλό να τροποποιηθεί.

*Μια συνάρτηση μπορεί να καλεστεί*

- από τη main
- από άλλες συναρτήσεις
- από τον ίδιο της τον εαυτό (αναδρομή) – δηλαδή δημιουργείται ένα πιστό της αντίγραφο που καλείται από την αρχική συνάρτηση και ούτω καθ' εξής μέχρι να ικανοποιηθεί κάποια συνθήκη.

Όταν καλείται μία συνάρτηση ο έλεγχος μεταβιβάζεται στην πρώτη εντολή του σώματος, αμέσως μετά το εισαγωγικό άγκιστρο {. Όταν τελειώνει μία συνάρτηση, ο έλεγχος μεταβιβάζεται στην καλούσα συνάρτηση, στην πρώτη εντολή αμέσως μετά από την κλήση της καλούμενης.

**Εμβέλεια μεταβλητών**

Εμβέλεια (scope) ενός ονόματος είναι το μέρος του προγράμματος μέσα στο οποίο μπορεί να χρησιμοποιείται το όνομα.

Όταν μία συνάρτηση αναγνωρίζει μια μεταβλητή λέμε ότι η μεταβλητή είναι ορατή (visible) από τη συνάρτηση.

Οι μεταβλητές μπορούν, ανάλογα με την εμβέλειά τους να χωριστούν σε τοπικές και εξωτερικές. Όταν μία μεταβλητή ορίζεται μέσα σε μία συνάρτηση τότε είναι **τοπική** ή **ιδιωτική** και ορατή μόνο από αυτή τη συνάρτηση – δε φαίνεται και δε μπορεί να χρησιμοποιηθεί από άλλες. Και αντίστροφα, μια αυτόματη μεταβλητή δηλωμένη στην αρχή μιας συνάρτησης έχει ως εμβέλειά της αυτή τη συνάρτηση. Τοπικές μεταβλητές με το ίδιο όνομα σε διαφορετικές συναρτήσεις δε συνδεόνται μεταξύ τους.

Όταν σε ένα αρχείο έχουμε περισσότερες από μία συναρτήσεις και έχουμε ορίσει κάποιες μεταβλητές έξω από τις συναρτήσεις, τότε αυτές λέγονται **εξωτερικές** και είναι ορατές από όλες τις συναρτήσεις από το σημείο δήλωσης της εξωτερικής μεταβλητής και μετά. Μία εξωτερική μεταβλητή μπορεί να χρησιμοποιηθεί από όλες τις συναρτήσεις στις οποίες είναι ορατή και έτσι αν υπολογιστεί η τιμή της σε μία συνάρτηση, τότε μπορεί αυτή η τιμή είναι γνωστή και μπορεί να χρησιμοποιηθεί σε μια άλλη συνάρτηση που “βλέπει” την εξωτερική μεταβλητή.

Η εμβέλεια μιας εξωτερικής μεταβλητής ή συνάρτησης εκτείνεται από το σημείο όπου δηλώθηκε μέχρι το τέλος του αρχείου που μεταγλωττίζεται.

Όταν ένα πρόγραμμα αποτελείται από πολλές συναρτήσεις μοιρασμένες σε πολλά αρχεία τότε μία εξωτερική μεταβλητή δηλώνεται ως **extern** σε όλα τα αρχεία όπου δηλώνεται εκτός από αυτό στο οποίο χρησιμοποιείται,καθώς και σε κάθε σημείο του αρχείου αυτού πριν από το σημείο ορισμού της. Ορισμός μιας εξωτερικής συνάρτησης είναι μια εντολή έξω από κάθε συνάρτηση όπου δηλώνεται ο τύπος της, το όνομά της και το μέγεθος αν είναι πίνακας και μπορεί προαιρετικά να αποδοθεί και αρχική τιμή. Κατά τον ορισμό δεσμεύεται και μνήμη. Ο ορισμός είναι και δήλωση για το υπόλοιπο αρχείο πηγαίου κώδικα. Δήλωση όμως δεν είναι ορισμός. Η δήλωση είναι επανάληψη του ορισμού (το μέγεθος προαιρετικά για πίνακες` πρέπει όμως να υπάρχουν [ ]) με τη λέξη extern. Η δήλωση δε δημιουργεί τις μεταβλητές ούτε και δεσμεύει μνήμη. Είναι απλώς ένα είδος υπόδειξης ή υπενθύμισης ότι η τάδε μεταβλητή είναι αυτού του τύπου για το υπόλοιπο αρχείο πηγαίου κώδικα. Γι' αυτό υπάρχει μόνο ένας ορισμός μιας εξωτερικής μεταβλητής για όλα τα αρχεία που αποτελούν τον πηγαίο κώδικα ενός προγράμματος.

Μια εξωτερική μεταβλητή πρέπιε να ορίζεται μία και μόνο μία φορά έξω από οποιαδήποτε συνάρτηση. Έτσι, δεσμεύεται για αυτή χώρος στη μνήμη. η μεταβλητή πρέπει επίσης να δηλώνεται σε κάθε συνάρτηση που θέλει να την προσπελάσει` έτσι δηλώνεται ο τύπος της μεταβλητής. Η δήλωση μπορεί να είναι μία ρητή εντολή extern ή να υπονοείται από τα συμφραζόμενα.

*Σημείωση 1*: Η μεταβίβαση τιμών μέσω εξωτερικών μεταβλητών μπορεί να είναι βολική αλλά ενέχει και κινδύνους. Αν υπάρχει μία τοπική μεταβλητή με το ίδιο όνομα τότε “υπερβαίνει” (overrrides) την εξωτερική μεταβλητή η οποία παύει να είναι ορατή. Έτσι, μπορεί να γίνουν λάθη στην ανάπτυξη μεγάλων προγραμμάτων με την έννοια ότι αποδίδουμε τιμές εκεί που δεν πρέπει ή δεν αποδίδουμε εκεί που πρέπει. Γι' αυτό προτιμάται η μεταβίβαση τιμών μέσω παραμέτρων.

*Σημείωση 2*: Μία μεταβλητή μπορεί να είναι τοπική (και να δηλωθεί για το σκοπό αυτό) όχι μόνο σε μια συνάρτηση αλλά ακόμη και μέσα στο σώμα μιας σύνθετης εντολής, δηλαδή αν μέσα στα άγκιστρα που ορίζουν το σώμα μιας if, for ή while ορίσουμε κάποιες μεταβλητές, η εμβέλειά τους είναι το σώμα αυτής της εντολής και δεν είναι ορατές. Βέβαια, αν ακολουθούμε τις υποδείξεις του δομημένου προγραμματισμού και φτιάχνουμε σχετικά μικρές συναρτήσεις, πολύ σπάνια θα χρειαστεί να καταφύγουμε σε μεταβλητές... “πιο τοπικές” από τις τοπικές.

*Σημείωση 3*: Οι ίδιες οι συναρτήσεις είναι εξωτερικές επειδή η C δεν επιτρέπει ορισμό συναρτήσεων μέσα σε συναρτήσεις.

**Κατηγορίες αποθήκευσης** (storage classes)
Τι συμβαίνει με τις τιμές των μεταβλητών καθώς εξελίσσεται η εκτέλεση ενός προγράμματος; Αυτό εξαρτάται από την "κατηγορία αποθήκευσης". Υπάρχουν οι εξής κατηγορίες αποθήκευσης:

- Αυτόματες
- Στατικές
- Αυτόματες μεταβλητές καταχωρητή

"Κανονικά", μία τοπική μεταβλητή διαγράφεται από τη μνήμη όταν τελειώσει η εκτέλεση της συνάρτησης όπου ανήκει και δημιουργούνται ξανά μετά την κλήση της. Αυτό σημαίνει ότι τα δεδομένα που αποθηκεύουν χάνονται μεταξύ των δύο κλήσεων. Έτσι, αυτές οι μεταβλητές λέγονται **αυτόματες** (automatic). Οι μεταβλητές μπορούν να δηλωθούν ως αυτόματες με τη λέξη auto αλλά αυτό δεν είναι απαραίτητο γιατί αυτό είναι το προεπιλεγμένο (default).
Οι αυτόματες μεταβλητές πρέπει να αρχικοποιούνται κατά τη δήλωσή τους μέσα στη συνάρτηση ή έστω δεν πρέπει να χρησιμοποιούνται κάπου αλλού πριν πάρουν τιμή με εντολή ανάθεσης γιατί λόγω της παροδικής φύσης τους πριν πάρουν τιμή έχουν "σκουπίδια".
Όταν δηλώσουμε μία μεταβλητή ως στατική (static) αυτό συνεπάγεται τα εξής:
Αν είναι εξωτερική μεταβλητή, τότε αυτή είναι ορατή μόνο από τις συναρτήσεις του αρχείου στο οποίο ανήκει. Έξω από αυτό είναι αόρατη.
Αν είναι τοπική μεταβλητή τότε συμπεριφέρεται σαν στατική εξωτερική αλλά μόνο για τη συνάρτηση στην οποία ανήκει – ούτε καν για τις υπόλοιπες του ίδιου αρχείου. Πρακτικά αυτό σημαίνει ότι είναι στην πραγματικότητα μια ακόμη τοπική μεταβλητή, αλλά, αντίθετα από τις αυτόματες, η στατική τοπική μεταβλητή δε χάνει την τιμή της μεταξύ δύο κλήσεων της συνάρτησης.
Επιπλέον, αν αποδώσουμε αρχική τιμή στη δήλωση μιας στατικής τοπικής μεταβλητής, τότε αυτή η τιμή ισχύει μόνο την πρώτη φορά που καλείται η συνάρτηση ενώ τις επόμενες αρχίζει με την τιμή που είχε την τελευταία φορά.
Αυτό είναι χρήσιμο για να διατηρούμε τιμές μέσα σε συναρτήσεις που θέλουμε μία μεταβλητή να "συσσωρεύει" τις μεταβολές της, π.χ. όταν θέλουμε να υπολογίσουμε το άθροισμα όρων που προστίθενται από ένας με κάθε νέα κλήση της συνάρτησης.
Τα παραπάνω για αυτόματες και στατικές μεταβλητές ισχύουν απαράλλακτα και για τοπικές μεταβλητές που η εμβέλειά τους εκτείνεται μόνο μέσα στο σώμα μιας σύνθετης εντολής. Αυτές αποκρύβουν τυχόν ομώνυμες τοπικές μεταβλητές μεγαλύτερης εμβέλειας (Κανόνας: η πιο τοπική αποκρύβει την πιο εξωτερική).
Οι μεταβλητές καταχωρητή (register) είναι μεταβλητές που θεωρούμε ότι θα χρησιμοποιηθούν πολύ συχνά και θα θέλαμε να επεξεργάζονται ταχύτερα. Αυτές καταχωρούνται στους καταχωρητές του ίδιου του μικροεπεξεργαστή για ταχύτερη πρόσβαση και επεξεργασία. Αλλά επειδή οι καταχωρητές του μικροεπεξεργαστή είναι συγκριτικά λίγες και γεμίζουν γρήγορα είναι πιθανό ότι μια τέτοια μεταβλητή δε θα αποθηκευτεί εκεί και το πρόγραμμα θα τη χειριστεί ως απλή αυτόματαη μεταβλητή. Γενικά, δε βλάπτει αν βάλουμε παραπανήσιες δηλώσεις register εκτός από το ότι υπάρχουν κάποιοι περιορισμοί και διαφοροποιήσεις. Π.χ. επειδή οι διευθύνσεις στον καταχωρητή δεν είναι σταθερές είναι αδύνατο να κάνουμε χρήση δεικτών. Επίσης, οι μεταβλητές καταχωρητή είναι καθολικές και ορατές από όλες τις συναρτήσεις του προγράμματος.

**Απόδοση αρχικών τιμών**
Αν δε γίνεται ρητή απόδοση αρχικής τιμής, οι εξωτερικές και οι στατικές μεταβλητές έχουν *εγγυημένα* αρχική τιμή μηδέν, ενώ οι αυτόματες και οι μεταβλητές καταχωρητή έχουν σκουπίδια.
Οι βαθμωτές μεταβλητές παίρνουν αρχική τιμή όταν ορίζονται αν μετά το όνομά τους μπει το ίσον και μια οποιαδήποτε παράσταση, όχι αναγκαστικά σταθερή, αρκεί να έχει προσδιοριστεί από πριν.
Οι εξωτερικές και στατικές μεταβλητές ως αρχική τιμή παίρνουν σταθερή παράσταση και μόνο μία φορά, θεωρητικά πριν αρχίσει η εκτέλεση του προγράμματος. Για αυτόματες και μεταβλητές καταχωρητή, η απόδοση αρχικής τιμή γίνεται κάθε φορά που γίνεται είσοδος στη συνάρτηση ή στο μπλοκ.
Για πίνακες ισχύουν όσα εκτέθηκαν στη σχετική διάλεξη.

**Μεταβίβαση παραμέτρων** (parameter passing)
Είναι ο μηχανισμός που μας επιτρέπει να δώσουμε και να πάρουμε πληροφορία από μία συνάρτηση. Γίνεται με δύο τρόπους:

- Μεταβίβαση μέσω τιμής (προεπιλεγμένος).
- Μεταβίβαση μέσω διεύθυνσης

Στην πρώτη περίπτωση δημιουργείται ένα τοπικό αντίγραφο της παραμέτρου μέσα στη συναρτήση και όταν επιστρέφει ο έλεγχος στην καλούσα η παράμετρος έχει μείνει άθικτη άσχετα από την επεξεργασία που έχει υποστεί το τοπικό αντίγραφο στην καλούσα. Σε αυτή την περίπτωση, μία συνάρτηση συμπεριφέρεται όπως μια μαθηματική συνάρτηση y = f(x), δηλαδή δεν πειράζει το x αλλά μας επιστρέφει μία τιμή y, ανάλογα και με τον τύπο της. Η συνάρτηση δε μπορεί να αλλάξει τη μεταβλητή της καλούσας συνάρτησης παρά μόνο το δικό της αντίγραφο.
Στη δεύτερη περίπτωση στην κλήση της συνάρτησης μεταβιβάζουμε τη διεύθυνση της μεταβλητής, &var, (όπως π.χ. κάναμε στη συνάρτηση scanf για την ανάγνωση δεδομένων) ενώ μέσα στον ορισμό της συνάρτησης δηλώνουμε παντού τη μεταβλητή με τον τελεστή περιεχομένου, *var, ώστε να αλλάξουμε την τιμή της. Τότε με την επιστροφή του ελέγχου η τιμή της var έχει όντως αλλάξει. Σε αυτή την περίπτωση η συνάρτηση συμπεριφέρεται όπως οι υπορουτίνες της Fortran ή οι διαδικασίες της Pascal, δηλαδή εμμέσως επιστρέφει περισσότερες από μία τιμές.
Ότι πρέπει να κάνουμε έναν τέτοιο ειδικό χειρισμό για να αλλάξουμε την τιμή μιας μεταβιβαζόμενης παραμέτρου έχει το πλεονέκτημα ότι δε θα αλλάξουμε κάποια δεδομένα κατά λάθος. Επίσης, με πέρασμα μέσω τιμής μπορούμε να χρησιμοποιούμε τις παραμέτρους ως τοπικές μεταβλητές κάνοντας πιο συμπαγή τα προγράμματα, όπως στον παρακάτω υπολογισμό του παραγοντικού ενός ακέραιου:

```
int fact(n) {
    int f;
    while(n-- > 1) f = f*n;
    return (f);
}
```

*Σημείωση 1*: Είναι απολύτως θεμιτό να συνδυαστούν και οι δύο τρόποι μεταβίβασης στην ίδια συνάρτηση, δηλαδή άλλες παράμετροι να μεταβιβάζονται με την τιμή τους και άλλες με τη διεύθυνσή τους, κάπως έτσι: `float func(float x, float y, float &z);`
*Σημείωση 2*: έστω ότι σε μια συνάρτηση περνάει με διεύθυνση ένα π.χ. πραγματικό όρισμα x και μέσα στη συνάρτηση αυτή το όρισμα πρέπει να διαβαστεί μέσω της scanf. Πώς θα δοθεί το όρισμα στη scanf; Απάντηση: `scanf("%f", x)`, δηλαδή παραλείπουμε το & επειδή το x έχει μπει ήδη ως διεύθυνση και θα ήταν σα να γράφαμε `scanf("%f", &*x)` όπου οι τελεστές & και * αλληλοεξουδετερώνονται.

**Μεταβίβαση παραμέτρων στην περίπτωση των πινάκων**.
Στους πίνακες, το πέρασμά τους σε μια συνάρτηση είναι διαφορετική υπόθεση. Έχουμε πει ότι το όνομα ενός πίνακα είναι και συνώνυμο ενός δείκτη που δείχνει στην αρχή της περιοχής όπου βρίσκονται αποθηκευμένα τα στοιχεία του πίνακα, δηλαδή στο πρώτο στοιχείο του. Επομένως, η μεταβίβαση ενός πίνακα γίνεται μέσω διεύθυνσης και δεν έχουμε αντιγραφή των στοιχείων του. Έτσι, αν έχουμε π.χ. έναν ακέραιο πίνακα A[10] που περνάει ως παράμετρος σε μια συνάρτηση π.χ. void func( ), θα πρέπει να γράψουμε func(A); δηλαδή δε χρειάζεται να βάλουμε το & επειδή το A είναι ήδη δηλωτικό της διεύθυνσης. Προσοχή όμως! Στον ορισμό της συνάρτησης όπου δίνονται όλες οι εντολές στο σώμα της, το όρισμα πρέπει να δηλωθεί με αγκύλες [ και ] για να γνωρίζει ο υπολογιστής ότι πρόκειται να περάσει ένας πίνακας. Ο αριθμός των στοιχείων δεν είναι απαραίτητος για μονοδιάστατους πίνακες. Πρέπει να γράψουμε δηλαδή κάτι σαν

```
void func( int A[ ] ) { ... διάφορες εντολές ... }
```

Σε πολυδιάστατους πίνακες πάλι, όλες οι διαστάσεις εκτός από την πρώτη πρέπει να δηλώνονται ρητά, δηλαδή αν ο πίνακας A είχε δύο διαστάσεις, έστω A[3][3]θα γράφαμε κάτι σαν

```
void func( int A[ ][3] ) { ... διάφορες εντολές ... }
```

Αυτό εξηγείται αν θυμηθούμε ότι οι πολυδιάστατοι πίνακες αποθηκεύονται σε μία διάσταση, π.χ. ο

παραπάνω πίνακας Α με στοιχεία τους ακέραιους από 1 έως 9 με αύξουσα σειρά θα αποθηκευτεί ως εξής:

| ***Γραμμές*** | 0 | | | 1 | | | 2 | | |
|---|---|---|---|---|---|---|---|---|---|
| ***Στήλες*** | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| ***Τιμές*** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Αν υποθέσουμε ότι γίνεται μία επανάληψη της μορφής

```
for(i=0; i<2, i++)
    for(j=0; j<2; j++)
          printf("\n%d ",A[i][j]);
```

τότε τυπώνονται τα στοιχεία 1,2,4,5 και παραλείπονται τα υπόλοιπα. Αλλά για να πάμε από το 2 στο 4 πρέπει να ξέρουμε πού ακριβώς τελειώνει η πρώτη στήλη και πού αρχίζει η δεύτερη και αυτή την πληροφορία τη μεταβιβάζουμε αναγκαστικά δίνοντας το μέγεθος της δεύτερης διάστασης.

**Αναδρομή**:
Ονομάζουμε αναδρομή (recursion) τη διαδικασία μέσω της οποίας μια συνάρτηση φαίνεται να καλεί επαναληπτικά τον εαυτό της. Αυτό επιτυγχάνεται με το να τοποθετήσουμε μία εντολή κλήσης της συνάρτησης μέσα στο ίδιο το σώμα των εντολών της. Αυτή η τεχνική μερικές φορές έχει το πλεονέκτημα ότι επιτρέπει τη διατύπωση ενός αλγόριθμου με πιο συνοπτική ή πυκνή μορφή και συνεπώς πιο κατανοητά, αλλά συνοδεύεται από το μειονέκτημα αυξημένων απαιτήσεων σε μνήμη και μειωμένης απόδοσης.
Η χρήση αναδρομής είναι αμφιλεγόμενη. Στην πραγματικότητα, όπως και η goto, δεν εμπίπτει στο Παράδειγμα του δομημένου προγραμματισμού με την αυστηρή έννοια – μάλιστα, το θεώρημα Jacopini στο οποίο βασίζεται η μεθοδολογία του δομημένου προγραμματισμού λέει ότι ακριβώς κάθε αναδρομική συνάρτηση μπορεί να υπολογιστεί με επαναλήψεις, επιλογές και ακολουθίες εντολών ανάθεσης.
Οι ιδιαίτερες απαιτήσεις των αναδρομικών συναρτήσεων σε υπολογιστικούς πόρους γίνονται κατανοητές αν χρησιμοποιήσουμε την έννοια της στοίβας σε συνδυασμό με ένα παράδειγμα. Στοίβα είναι μια περιοχή της μνήμης όπου ένα πρόγραμμα αποθηκεύει στοιχεία στη σειρά και τα παίρνει πίσω ξεκινώντας από αυτό που είναι στην "κορυφή" της στοίβας δηλαδή φεύγει πρώτο όποιο μπήκε τελευταίο. Οι αναδρομικές συναρτήσεις απαιτούν τοποθέτηση στοιχείων στη στοίβα μέχρι να πάψει να ισχύει η συνθήκη που επιτρέπει την αναδρομή, γιατί κάθε νέα κλήση της αναδρομικής συνάρτησης πρέπει να αποθηκεύσει κάπου τις τοπικές μεταβλητές. Σε μερικές περιπτώσεις η στοίβα μπορεί να γεμίσει.
Παράδειγμα: υπολογισμός παραγοντικού n με αναδρομική συνάρτηση.
$n! = n(n-1)(n-2) \ldots 3 \cdot 2 \cdot 1 = n \cdot (n-1)!$
Άρα fact(n) = n * fact(n-1).
Χρειάζεται όμως και μια συνθήκη τερματισμού αλλιώς θα συνεχιστεί μέχρι το -oo! Προφανώς μόνο για n > 1 έχει ουσιώδες νόημα ο υπολογισμός άρα αυτή είναι η συνθήκη μας. Επομένως θα ορίσουμε μια μεταβλητή x που θα επιστρέφεται και με αρχική τιμή x = 1 που θα υπεισέρχεται στην αναδρομική σχέση μόνο αν n > 1. Το αποτέλεσμα είναι το εξής:

```
#include <stdio.h>

int fact(int n) {
    int x = 1;
    if(n > 1) x *= n * fact(n-1);
    return (x);
}
```

```
int main() {
    int n;
    printf("\nNumber? ");
    scanf("%d",&n);
    printf("\nThe factorial of %d is %d\n", n, fact(n));
}
```

Στο παραπάνω παράδειγμα έστω n = 4. Θα έχουμε:

```
x = 1;
n = 4 > 1 => x = fact(4) = 4 * fact(3); /* 1ο αντίγραφο του x στη στοίβα` αρχικά = 1*/
x = 1;
n = 3 > 2 => x = fact(3) = 3 * fact(2); /* 2ο αντίγραφο του x στη στοίβα` αρχικά = 1*/
x = 1;
n = 2 > 1 => x = fact(2) = 2 * fact(1); /* 3ο αντίγραφο του x στη στοίβα` αρχικά = 1*/
x = 1; /* 4ο αντίγραφο του x στη στοίβα` αρχικά = 1*/
n = 1 => fact(1) = 1;
fact(2) = 2 * 1 = 2; /* πάρε το 3ο αντίγραφο */
fact(3) = 3 * 2 = 6; /* πάρε το 2ο αντίγραφο */
fact(4) = 4 * 3 = 24 /* πάρε το 1ο αντίγραφο */
```

Μία όχι καλά σχεδιασμένη αναδρομική συνάρτηση μπορεί να εξαντλήσει όλη τη στοίβα και γενικά η αποδοτικότητα σε όρους ταχύτητας και μνήμης δεν αναμένεται να είναι υψηλή. Γι' αυτό μπορεί ένας αλγόριθμος να σχεδιαστεί με τη βοήθεια αναδρομικών συναρτήσεων που διευκολύνουν τη σύλληψη του “βασικού σχεδίου” και μετά να μετατραπεί σε αλγόριθμο που χρησιμοποιεί αποκλειστικά αναθέσεις τιμών, επαναλήψεις και επιλογές. Εν συνεχεία, ο ισοδύναμος αλγόριθμος μπορεί να απλουστευτεί και ανασχεδιαστεί σε πιο αποτελεσματική βάση. Για την άρση της αναδρομικότητας από ένα πρόγραμμα υπάρχουν πολύ συγκεκριμένες διαδικασίες που οδηγούν σε απολύτως ισοδύναμους αλγορίθμους χωρίς αναδρομή. Αυτοί αρχικά έχουν μία ή περισσότερες εντολές goto, αλλά και αυτές μετά μπορούν να αντικατασταθούν από κατάλληλο συνδυασμό επιλογών και επαναλήψεων.

**Κατασκευή πρωτοτύπου συνάρτησης**.

Για να χρησιμοποιηθεί μία μεταβλητή ή συνάρτηση σε ένα σημείο του προγράμματος πρέπει να έχει οριστεί πιο πριν. Όσον αφορά τις συναρτήσεις δεν είναι απαραίτητο να υπάρχει ολόκληρος ο ορισμός του σώματος της συνάρτησης πριν από το σημείο όπου θα τη χρειαστούμε. Αρκεί να υπάρχει ένα είδος εντολής που λέγεται πρωτότυπο της συνάρτησης και είναι της μορφής

*Τύπος Όνομα* ( *τύπος1 παράμετρος1*, *τύπος2 παράμετρος2*, ... κλπ);

Προσοχή στο ότι βάζουμε και ; στο τέλος, δηλαδή είναι και αυτό ένα είδος εντολής. Κατά τα άλλα είναι το ίδιο με ο,τι γράφουμε όταν ορίζουμε μια συνάρτηση. Το πρωτότυπο της συνάρτησης λέει στον υπολογιστή ότι όταν δει το όνομα *Όνομα* πρόκειται για συνάρτηση με τύπο *Τύπος* και παραμέτρους όπως περιγράφονται στη λίστα ορισμάτων του πρωτοτύπου. Μάλιστα, τα ονόματα των μεταβλητών δεν πειράζει αν διαφέρουν στο πρωτότυπο και δεν είναι καν απαραίτητο να υπάρχουν στο πρωτότυπο – αρκεί να υπάρχουν οι τύποι και να είναι σε πλήρη αντιστοιχία ως προς τον αριθμό παραμέτρων, τη θέση και τον τύπο τόσο στον ορισμό όσο και στις κλήσεις της συνάρτησης μέσα στο πρόγραμμα. Αυτό γίνεται γιατί μας διευκολύνει στον εντοπισμό λαθών. Αν η αντιστοιχία με κάποιο από τα στοιχεία του πρωτοτύπου δεν είναι πλήρης ο μεταγλωττιστής θα σημειώσει το λάθος και θα σταματήσει τη μεταγλώττιση. Παλιότερα δεν υπήρχε η έννοια του

πρωτοτύπου και τα λάθη ήταν συχνότερα. Σήμερα είναι λάθος αν το πρωτότυπο δε συμφωνεί με τον ορισμό και τις χρήσεις γενικότερα, της συνάρτησης.
Σημείωση: επαναλαμβάνουμε ότι αν μία λίστα παραμέτρων είναι άδεια, τότε πρέπει να δηλώνεται με τη λέξη void γιατί η κενή λίστα εκλαμβάνεται ως σύνταξη παλαιού τύπου και αναστέλλεται ο έλεγχος λάθους με αποτέλεσμα τον αυξημένο κίνδυνο λάθους.
Επίσης, αν δεν υπάρχει επιστρεφόμενη τιμή, ο τύπος της συνάρτησης πρέπει να δηλώνεται void επειδή απουσία δηλωμένου τύπου εκλαμβάνεται ως int.
Παράδειγμα: το πιο πάνω πρόγραμμα της αναδρομικής συνάρτησης μπορεί να ξαναγραφεί έτσι:

```
#include <stdio.h>
int main() {
    int n;
    printf("\nNumber? ");
    scanf("%d",&n);
    printf("\nThe factorial of %d is %d\n", n, fact(n-1,n));
}

int fact(int n) {
    int x = 1;
    for( ;n>1;n--) x *= n;
    return (x);
}
```

Εδώ έχουμε βάλει επίτηδες ένα λάθος στο κύριο πρόγραμμα όσον αφορά τις παραμέτρους της συνάρτησης fact (έντονα γράμματα). Ο υπολογιστής βλέπει δύο ορίσματα και αγνοεί το πρώτο χρησιμοποιώντας τη συνάρτηση όπως είναι ορισμένη, δίνοντας λάθος αποτελέσματα (ανάλογα με την υλοποίηση τα αποτελέσματα διαφέρουν` άλλοι μεταγλωττιστές μπορεί να οδηγήσουν σε σφάλμα εκτέλεσης).
Τώρα προσθέτουμε το πρωτότυπο της συνάρτησης (έντονα στοιχεία):

```
#include <stdio.h>
int fac(int );
int main() {
    int n;
    printf("\nNumber? ");
    scanf("%d",&n);
    printf("\nThe factorial of %d is %d\n", n, fac(n-1,n));
}

int fac(int n) {
    int x = 1;
    for(;n>1;n--) x *= n;
    return (x);
}
```

Τώρα κατά τη μεταγλώττιση παίρνουμε το μήνυμα:

```
>cc prototype.c
prototype.c: In function ‘main’:
prototype.c:9: error: too many arguments to function ‘fac’
```

Διορθώνουμε το λάθος και το πρόγραμμα μεταγλωττίζεται και δίνει σωστά αποτελέσματα.

Αν μία μεταβλητή περνιέται μέσω διεύθυνσης θα δηλωθεί ως δείκτης δηλαδή στον τύπο θα προστεθεί ο τελεστής περιεχομένου *, π.χ.

```
int example (int * , float , float);
```

Είναι ευνόητο ότι αν έχουμε διάφορες χρήσιμες και συχνά καλούμενες συναρτήσεις οι οποίες υλοποιούνται και σε κάποια άλλα αρχεία υλοποιούνται άλλα τμήματα ενός πολύ μεγάλου προγράμματος, δε θα θέλαμε να επαναλαμβάνουμε όλα τα πρωτότυπα των συναρτήσεων σε κάθε άλλο αρχείο όπου χρειάζονται. Π.χ. αν κάτι άλλαζε σε μια συνάρτηση δε θα θέλαμε να διορθώνουμε ένα-ένα όλα τα αρχεία όπου υπάρχει αυτό το πρωτότυπο. Επίσης, θα θέλαμε να εξασφαλίσουμε ότι δε θα γίνει κανένα λάθος αλλά όλα τα αρχεία πηγαίου κώδικα θα διαθέτουν τους ίδια πρωτότυπα. Ένα βήμα προς το να αποφύγουμε αυτή την επανάληψη είναι το να περιλάβουμε όλες τις δηλώσεις πρωτοτύπων σε ένα "αρχείο επικεφαλίδας" με κατάληξη *.h, π.χ. myprototypes.h, ενώ σε άλλα αρχεία με κατάληξη *.c υλοποιούνται οι χρησιμοποιούνται συναρτήσεις μας και όπου χρησιμοποιούνται αυτές εμείς δεν έχουμε παρά να βάλουμε την παρακάτω οδηγία προς τον προεπεξεργαστή:

```
#include "myprototypes.h"
```

Αυτή είναι πολύ όμοια με τις οδηγίες που βάζαμε μέχρι τώρα για να ενσωματώσουμε τις πρότυπες βιβλιοθήκες εισόδου-εξόδου κλπ στο πρόγραμμά μας. Αυτό είναι απόλυτα φυσικό γιατί και εκείνες οι οδηγίες κάνουν ακριβώς το ίδιο πράγμα, δηλαδή ενσωματώνουν τα πρωτότυπα των κατάλληλων συναρτήσεων (καθώς και διάφορες άλλες βοηθητικές δηλώσεις). Η μόνη διαφορά είναι τα " " αντί για τα γωνιακά άγκιστρα < και >. Το νόημα των " " είναι ότι ο υπολογιστής θα καταλάβει ότι πρόκειται για δικό μας αρχείο και θα ψάξει για να το βρει πρώτα στον τρέχοντα φάκελλο όπου πιθανότατα βρίσκονται και τα αρχεία πηγαίου κώδικα. Αν δεν το βρει εκεί τότε θα το αναζητήσει σε άλλες προεπιλεγμένες θέσεις που εξαρτώνται από το σύστημά μας. Τα αρχεία αυτά μπορούν να έχουν όχι μόνο πρωτότυπα συναρτήσεων αλλά και ορισμούς συμβολικών σταθερών και δηλώσεις μεταβλητών που εξασφαλίζεται ότι είναι ίδια για όποια αρχεία πηγαίου κώδικα περιλαμβάνουν αυτό το αρχείο επικεφαλίδας. Όταν αλλάζει ένα αρχείο επικεφαλίδας πρέπει να μεταγλωττίσουμε πάλι όλα τα αρχεία που το περιλαμβάνουν.

Όταν αναπτύσσουμε ένα μικρού ή μεσαίου μεγέθους πρόγραμμα μπορούμε να περιοριστούμε σε ένα μόνο αρχείο επικεφαλίδας για τις δηλώσεις πρωτοτύπου συναρτήσεων. Καθώς μεγαλώνει το πρόγραμμα μπορεί να χρειαστεί να μοιράσουμε τις δηλώσεις σε περισσότερα του ενός αρχεία κατά κατηγορίες ώστε να περιλάβουμε μόνο τις απαραίτητες κάθε φορά συναρτήσεις.
Σε ένα αρχείο επικεφαλίδας μπορούμε να βάλουμε και άλλα στοιχεία όπως για παράδειγμα δηλώσεις συμβολικών σταθερών ή ακόμη και οδηγίες συμπερίληψης (include) άλλων αρχείων επικεφαλίδας.
Στο σημείο αυτό αξίζει να πούμε άλλο ένα μεθοδολογικό ζήτημα: αν ένα αρχείο επικεφαλίδας έχει ήδη συμπεριληφθεί μέσω ενός άλλου και πάμε να το ξαναβάλουμε είτε έμμεσα επειδή περιλαμβάνεται μέσω ενός ακόμη τρίτου αρχείου επικεφαλίδας είτε απευθείας, θα είχαμε φαινόμενα όπως διπλή δήλωση πρωτοτύπου και γενικά έναν ογκώδη πηγαίο κώδικα. Για να το αποφύγουμε εφαρμόζουμε την πρακτική του "περιτυλίγματος" για κάθε αρχείο επικεφαλίδας. Έστω το αρχείο myprototypes.h. Ο,τι γράψουμε σε αυτό θα βρίσκεται μεταξύ των παρακάτων προεπεξεργαστικών οδηγιών:

```
#ifndef MYPROTOTYPES_H
#define MYPROTOTYPES_H
.......
#endif
```

Οι παραπάνω οδηγίες με απλά λόγια λένε το εξής: Αν δεν είναι ορισμένη η συμβολική σταθερά MYPROTOTYPES_H όρισέ την, ..., τέλος_Αν. Η αντιστοιχία μεταξύ ονόματος αρχείου και ονόματος συμβολικής σταθεράς είναι προφανής. Δεν είναι υποχρεωτική αλλά βοηθά στο να μην υπάρχει σύγχυση. Αν το αρχείο έχει ήδη συμπεριληφθεί τότε η σταθερά MYPROTOTYPES_H είναι ήδη ορισμένη και ο,τι υπάρχει μέσα στο #ifndef ... #endif δε θα συμπεριληφθεί` αν όμως δεν έχει συμπεριληφθεί ακόμη, τότε δεν υπάρχει αυτή η σταθερά η οποία ορίζεται και επιπλέον

συμπεριλαμβάνεται ο,τι ακολουθεί.

**Δείκτες σε συναρτήσεις**.
Μπορούμε να ορίσουμε δείκτες σε συναρτήσεις όπως ακριβώς ορίζουμε δείκτες και σε μεταβλητές. Ο ορισμός γίνεται ως εξής:

```
Τύπος (*όνομα_δείκτη) ( );
```

Ο δείκτης ορίζει τη διεύθυνση της μνήμης όπου αρχίζει η αποθηκευμένη συνάρτηση. Η διεύθυνση αυτή είναι διαθέσιμη με βάση το όνομα της συνάρτησης. Δηλαδή, όπως ακριβώς και με τους πίνακες, *το όνομα της συνάρτησης είναι συνώνυμο με δείκτη που δείχνει στη διεύθυνση όπου αρχίζει η συνάρτηση*. Οι παρενθέσεις δεξιά χρειάζονται για να δείξουμε ότι πρόκειται για δείκτη σε συνάρτηση και οι συναρτήσεις γύρω από το όνομα του δείκτη χρειάζονται γιατί αλλοιώς θα πάρουμε ορισμό συνάρτησης που επιστρέφει δείκτη σε τύπο *Τύπος*, που είναι κάτι άλλο!
Οι δείκτες σε συναρτήσεις χρησιμοποιούνται κυρίως όπου έχουμε περισσότερες από μία δυνατότητες για την επεξεργασία των στοιχείων και επιλέγει ο χρήστης.
Οι δείκτες σε συναρτήσεις μπορούν να μεταβιβαστούν ως παράμετροι μεταξύ των συναρτήσεων του προγράμματος όπως ακριβώς και οι κανονικοί δείκτες μεταβλητών. Μπορούμε π.χ. να γράψουμε

```
intFc(arg1, arg2, someFunc);
intFc(arg3, arg4, someOtherFunc);
```

όπου someFunc και someOtherFunc είναι ονόματα συναρτήσεων, ενώ η intFc ορίζεται ως εξής:

```
intFc(int a, int b, int (*aFunc)( ));
```

Μπορούν επίσης να οριστούν και πίνακες δεικτών σε συναρτήσεις, π.χ.

```
void (*fptr[])( ) = {func1, func2, func3};
```

όπου func1, func2 και func3 είναι ονόματα συναρτήσεων των οποίων τουλάχιστον τα πρωτότυπα έχουν οριστεί πιο πριν. Έτσι, μπορούμε να ορίσουμε ένα "μενού" με αριθμητικές επιλογές και ανάλογα με τον αριθμό n που εισάγει ο χρήστης καλείται η αντίστοιχη συνάρτηση ως το στοιχείο (*fptr[n-1])( ) του πίνακα δεικτών σε συναρτήσεις.
Θα μπορούσαμε επίσης, τουλάχιστον στη φάση ανάπτυξης ενός προγράμματος, να ορίσουμε ένα πίνακα δεικτών σε συναρτήσεις που είναι εναλλακτικές εκδοχές μιας συγκεκριμένης συνάρτησης που αναπτύσσουμε. Εμείς καλούμε το πρώτο στοιχείο του πίνακα (*fptr[0])( ) και όταν θέλουμε να δοκιμάσουμε μια άλλη εκδοχή της δεν έχουμε παρά να αλλάξουμε τις θέσεις των στοιχείων του πίνακα και να βάλουμε στη θέση 0 εκείνη που μας ενδιαφέρει.