ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

Η γλώσσα C σε βάθος

4η αναθεωρημένη έκδοση

Πλήρης οδηγός εκμάθησης της γλώσσας C με εκτενή αναφορά στις δομές δεδομένων

Υπόδειγματικές σελίδες

Απευθύνεται τόσο στο νέο σπουδαστή όσο και στον έμπειρο προγραμματιστή, με πλήθος από χαρακτηριστικά παραδείγματα, επεξηγηματικά σχήματα και ασκήσεις

Νίκος Μ. Χατζηγιαννάκης

Περιέχει δωρεάν CD-ROM

Προλογίζει ο τακτικός καθηγητής του τμήματος Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης
Δρ. Πάνος Τραχανιάς

ΚΛΕΙΔΑΡΙΘΜΟΣ

Ενδεικτικές σελίδες

Κεφάλαιο 1

Εισαγωγή

Ενδεικτικές σελίδες

Εισαγωγή

Οκτώβριος του 1983. Ως μεταπτυχιακός φοιτητής, γνώρισα μια δεκαοχτάχρονη αμερικανιδούλα που την έλεγαν BCPL. Η σχέση μας κράτησε ένα χρόνο περίπου. Ευέλικτη αλλά δύστροπη. Δύσκολο να την κατανοήσεις και δυσκολότερο να τη χειριστείς. Την επόμενη χρονιά μού γνώρισε τη μικρότερη αδελφή της, τη C. Από την πρώτη στιγμή με εντυπωσίασε, είχε πιο δομημένη σκέψη, ήταν τρομερά ευέλικτη και πολύ πιο φιλική. Η γνωριμία αυτή δεν άργησε να εξελιχθεί σε ένα μεγάλο έρωτα, ο οποίος κρατάει ακόμη και σήμερα. Από τότε συνάντησα και άλλες πολλές, πολύ πιο νέες και εμφανίσιμες, με πολλές δυνατότητες και προσόντα, έμεινα όμως πάντα πιστός στη μεσήλικα πια αγάπη μου, τη C. Ευτυχώς που είναι μόνο ... μια γλώσσα προγραμματισμού!

Και όμως, η πολύχρονη αυτή σχέση είχε και έναν καρπό, την παραγωγή πλούσιου εκπαιδευτικού υλικού που χρησιμοποιήθηκε όλα αυτά τα χρόνια για τη διδασκαλία της γλώσσας C και το οποίο αποτέλεσε τη βάση για τη συγγραφή αυτού του βιβλίου.

Το βιβλίο αυτό απευθύνεται τόσο στον αρχάριο όσο και στον έμπειρο προγραμματιστή που θέλει να γνωρίσει τις αρχές και τη φιλοσοφία του δομημένου προγραμματισμού, μέσα από μια ευέλικτη και χωρίς όρια γλώσσα όπως η C.

Με ιδιαίτερο τρόπο προσεγγίζονται όλα τα χαρακτηριστικά της γλώσσας και δίνεται έμφαση στην αναλυτική και σε βάθος επεξήγηση των "στρυφνών" της σημείων. Μεγάλο βάρος έχει δοθεί στη διδακτική σειρά αυτού του βιβλίου, ώστε η ανάγνωση και η κατανόηση ενός κεφαλαίου να προϋποθέτει **μόνο** τις γνώσεις που αποκτήθηκαν στα προηγούμενα κεφάλαια.

Πώς να διαβάσετε αυτό το βιβλίο

Αν είστε γνώστης μιας οποιασδήποτε άλλης γλώσσας προγραμματισμού μπορείτε να παραλείψετε το Κεφάλαιο 1 και να προχωρήσετε κατευθείαν στο επόμενο κεφάλαιο.

Το Κεφάλαιο 2 είναι μια μικρή "περιοδεία" στη γλώσσα. Γίνεται μια πρώτη γνωριμία, ώστε να αποκτήσετε μια σφαιρική εικόνα από τη δομή και τα χαρακτηριστικά της και να μπορέσετε να καταστρώσετε τα πρώτα σας απλά προ-

Ηλεκτρονικές Εκδόσεις

γράμματα. Τα επόμενα κεφάλαια αναλύουν όλα τα χαρακτηριστικά της γλώσσας, δίνοντας έμφαση στην παρουσίαση των βασικών, αλλά και των περισσότερο πολύπλοκων εννοιών, με απλό, εποπτικό, και κατανοητό τρόπο. Τα κεφάλαια 15, 16, 17, 18 και 19, είναι μια βαθιά "βουτιά" τόσο σε εξειδικευμένες τεχνικές προγραμματισμού, όσο και στα ιδιαίτερα χαρακτηριστικά της γλώσσας. Τέλος, το κεφάλαιο 20 σας προετοιμάζει για το επόμενο φυσικό σας βήμα: Μια πρώτη επαφή με τον αντικειμενοστρεφή προγραμματισμό και τη γλώσσα C++! Το Παράρτημα Α περιέχει μια αναλυτική αναφορά στις πιο συχνά χρησιμοποιούμενες συναρτήσεις της C. Στο Παράρτημα Β θα βρείτε λεπτομέρειες για το περιβάλλον ανάπτυξης του DEV C++. Το DEV C++ είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης για τις γλώσσες C και C++, το οποίο διατίθεται δωρεάν κάτω από τη γενική άδεια χρήσης της GNU[1]. Όλα τα προγράμματα που υπάρχουν σε αυτό το βιβλίο έχουν δοκιμαστεί στο περιβάλλον του DEV C++.

Ενδεικτικές σελίδες

Στο συνοδευτικό CD θα βρείτε το πρόγραμμα εγκατάστασης για το ολοκληρωμένο περιβάλλον του DEV C++, τον κώδικα των περισσοτέρων προγραμμάτων που βρίσκονται σε αυτό το βιβλίο, καθώς και τις απαντήσεις όλων των ασκήσεων. Επίσης, αποκλειστικά για το βιβλίο αυτό, έχει δημιουργηθεί ένας δικτυακός τόπος στη διεύθυνση **http://c.bytes.gr** με πλήθος επιπλέον αναφορών για τη γλώσσα C.

Σας συνιστώ να εγκαταστήσετε το περιβάλλον του DEV C++ ώστε να μπορείτε να δοκιμάζετε τα παραδείγματα του βιβλίου αλλά και τα δικά σας προγράμματα.

Το σύμβολο ⊙ δίπλα από τον τίτλο μιας παραγράφου σημαίνει ότι η παράγραφος παρέχει εξειδικευμένη γνώση και μπορεί να παραλειφθεί σε πρώτη ανάγνωση, ή όταν δεν επιθυμούμε ιδιαίτερη εμβάθυνση.

Στο μαύρο πλαίσιο που υπάρχει δίπλα από τα περισσότερα παραδείγματα αυτού του βιβλίου αναφέρεται το όνομα του αρχείου στο οποίο υπάρχει ο πηγαίος κώδικας του παραδείγματος. Αν έχετε ήδη εγκαταστήσει το DEV C++, ένα διπλό κλικ επάνω στο αρχείο του παραδείγματος είναι αρκετό για να ανοίξει το συγκεκριμένο αρχείο μέσα στο ολοκληρωμένο περιβάλλον του DEV C++. Η μεταγλώττιση και η εκτέλεση του προγράμματος γίνονται πλέον με ένα απλό κλικ σε ένα εικονίδιο του περιβάλλοντος (βλέπε Παράρτημα-Β).

arxeio.c

[1] http://www.gnu.org/philosophy

Γιατί να μάθω C;

Μια ερώτηση που ακούω συνέχεια από τους φοιτητές: Ποιός ο λόγος να μάθω C δεδομένου ότι υπάρχουν πιο σύγχρονες γλώσσες οι οποίες είναι περισσότερο φιλικές και έχουν περισσότερες δυνατότητες;

Πάντα απαντώ με ένα παράδειγμα. Αν θέλουμε να γίνουμε πιλότοι, θα αρχίσουμε την εκπαίδευση μας από ένα Airbus ή από ένα Τσέσνα; Το Airbus είναι προφανώς πιο φιλικό, πιο ασφαλές, έχει περισσότερες δυνατότητες, σχεδόν όλες οι λειτουργίες του είναι αυτοματοποιημένες, και σε τελική ανάλυση είναι αυτό που θα πιλοτάρουμε, ως επαγγελματίες πιλότοι, σε μια επιβατική πτήση. Γιατί λοιπόν να αρχίσουμε από ένα Τσέσνα;

Νομίζω ότι η απάντηση είναι προφανής. Με το Τσέσνα θα μάθουμε τη βασική φιλοσοφία της πτήσης χωρίς την πολυπλοκότητα σύνθετων συστημάτων. Θα έχουμε στη διάθεση μας τα βασικά και απολύτως απαραίτητα όργανα και θα γνωρίσουμε το αποτέλεσμα του κάθε χειρισμού μας. Σε ένα Airbus θα είμαστε περικυκλωμένοι από χιλιάδες διακόπτες και όργανα, ελάχιστα από τα οποία όμως έχουν να κάνουν με την ουσία της πτήσης. Μπορεί να ρυθμίζουν τη θερμοκρασία της καμπίνας, την ένταση των ηχείων ανακοινώσεων, το φωτισμό, κ.λπ. Οι αυτοματοποιημένες επίσης λειτουργίες δεν βοηθούν στο να κατανοήσουμε σε βάθος τους κανόνες και τις διαδικασίες της πτήσης.

Ας έλθουμε τώρα στον προγραμματισμό. Σίγουρα υπάρχουν γλώσσες πιο σύγχρονες και πιο παραγωγικές, οι οποίες μας βοηθούν να φτιάχνουμε ελκυστικά προγράμματα σε λιγότερο χρόνο. Γιατί λοιπόν να μάθουμε C;

Η C μας βοηθάει να κατανοήσουμε σε βάθος τους μηχανισμούς που οι περισσότερες "παραγωγικές" γλώσσες, για λόγους απλότητας και ευχρηστίας, κρύβουν από τον προγραμματιστή. Ο φοιτητής που σπουδάζει επιστήμες σχετικές με την πληροφορική πρέπει να έχει κατανοήσει σε βάθος όλους αυτούς τους μηχανισμούς. Είναι βέβαιο ότι στην παραγωγική διαδικασία θα χρησιμοποιήσει άλλες γλώσσες. Όμως η γνώση που απέκτησε μαθαίνοντας C θα τον βοηθήσει να επιλύει θέματα και να κατανοεί έννοιες που σε άλλες γλώσσες προγραμματισμού είναι πολύ πιο αφαιρετικές και γενικές.

Ας μην ξεχνάμε, επίσης, ότι οι πλέον σύγχρονες γλώσσες προγραμματισμού, όπως η C++, η Java, και η C#, βασίζονται στη C!

Ευδοκιμικές Σελίδες

Βασικές έννοιες του προγραμματισμού

Το τμήμα αυτό απευθύνεται στους αρχάριους στον προγραμματισμό. Αναφέρει και εξηγεί βασικές έννοιες του προγραμματισμού, απαραίτητες για τον αναγνώστη που έρχεται για πρώτη φορά σε επαφή με μια γλώσσα προγραμματισμού. Κάθε πρόγραμμα, αναπτυγμένο σε οποιαδήποτε γλώσσα προγραμματισμού, επεξεργάζεται δεδομένα και δίνει αποτελέσματα. Η επεξεργασία των δεδομένων γίνεται πάντα με έναν προκαθορισμένο τρόπο (αλγόριθμο). Τα συστατικά που συνθέτουν ένα απλό πρόγραμμα είναι οι μεταβλητές, οι σταθερές, οι παραστάσεις, και οι εντολές.

Ας μην ξεχνάμε ότι ένας Η/Υ είναι ένα μηχάνημα το οποίο επιτελεί μόνο τρείς λειτουργίες:

- **Πρόσθεση**. Στη πραγματικότητα ο Η/Υ εκτελεί μόνο την πράξη της πρόσθεσης. Όλες οι άλλες πράξεις ανάγονται σε προσθέσεις.
- **Σύγκριση**. Π.χ. να συγκρίνει δύο αριθμούς και να αποφασίσει ποιος είναι μεγαλύτερος.
- **Μεταφορά δεδομένων**. Ο Η/Υ μπορεί να αποθηκεύσει στη μνήμη του δεδομένα, τα οποία μπορεί να είναι σταθερές ή να προέρχονται από αποτελέσματα πράξεων.

Παρατηρούμε επομένως ότι ο Η/Υ εκτελεί πολύ απλές και βασικές λειτουργίες με χρήση όμως των οποίων επιτυγχάνει την εκτέλεση πολύπλοκων αλγορίθμων.

> Συνηθίζω να λέω στους φοιτητές ότι πρέπει να αντιμετωπίζουμε τον Η/Υ σαν έναν ασθενή που έχει τη νόσο **Αλτσχάιμερ** σε προχωρημένο στάδιο!

Όσοι νοσούν από Αλτσχάιμερ έχουν απώλεια μνήμης. Δεν θυμούνται τι έκαναν το προηγούμενο δευτερόλεπτο. Οι ασθενείς αυτοί καταφεύγουν σε διάφορα τεχνάσματα για να μπορούν να λειτουργήσουν κοινωνικά. Χρησιμοποιούν χαρτάκια ή κουτάκια για να σημειώνουν τι έκαναν ώστε αργότερα να μπορούν να το ελέγχουν. Π.χ., αν φάνε πρωινό βάζουν σε έναν κουτάκι τη φράση "έφαγα πρωινό". Έτσι όταν το δούνε αργότερα θα ξέρουν ότι έφαγαν πρωινό και δεν θα ξαναφάνε. Οτιδήποτε θέλουν να θυμούνται θα πρέπει να το καταχωρίσουν σε κάποιο κουτάκι!

Δείγματα σελίδες

Μεταβλητές

Ο Η/Υ – ασθενής έχει στη διάθεσή του δισεκατομμύρια τέτοια κουτάκια, που δεν είναι άλλο από τις θέσεις μνήμης που διαθέτει η μνήμη RAM του κάθε συστήματος. Τα δεδομένα, καθώς και τα αποτελέσματα των πράξεων, αποθηκεύονται στα κουτάκια της κεντρικής μνήμης (RAM) του Η/Υ. Σε διαφορετική περίπτωση, σαν ασθενής με Αλτσχάιμερ, θα τα είχε ξεχάσει.

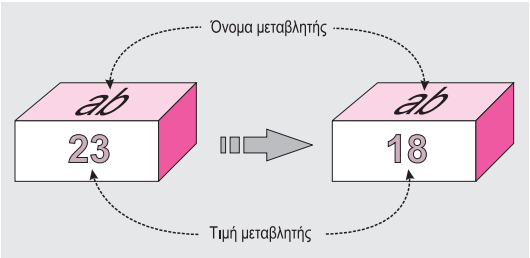
Η διαχείριση της μνήμης RAM είναι βασικό μέλημα μιας γλώσσας προγραμματισμού. Για να χρησιμοποιηθεί μια θέση μνήμης μέσα από μια γλώσσα προγραμματισμού, πρέπει να της δοθεί ένα όνομα.

Σε κάθε θέση μνήμης που χρησιμοποιείται σε ένα πρόγραμμα, ανατίθεται (αντιστοιχίζεται, αποδίδεται) ένα **όνομα**, το δε περιεχόμενο της θέσης μνήμης αποτελεί την **τιμή** της. Για να αναφερθεί το πρόγραμμα στη συγκεκριμένη θέση μνήμης, χρησιμοποιεί απλώς το όνομα της.

Ηλεκτρονικές Εκδόσεις

Φανταστείτε την κάθε θέση μνήμης που χρησιμοποιούμε σαν ένα κουτί, στο εξωτερικό του οποίου είναι γραμμένο το όνομα του. Το περιεχόμενο του κουτιού αποτελεί την **τιμή** αυτής της θέσης μνήμης.

Ας θεωρήσουμε τη θέση μνήμης **ab** η οποία περιέχει στην αρχή τον αριθμό 23, και έστω ότι με κατάλληλη εντολή αλλάζουμε το περιεχόμενό της σε 18.



Παρατηρούμε ότι το κουτί (θέση μνήμης) παραμένει το ίδιο (ab), αλλά το περιεχόμενό του μεταβάλλεται. Το όνομα, λοιπόν, που αντιστοιχεί σε μία θέση μνήμης μπορεί να πάρει διαφορετικές τιμές. Γι' αυτόν το λόγο κάθε όνομα θέσης μνήμης καλείται **μεταβλητή** (variable). Στο παραπάνω παράδειγμα η μεταβλητή ab είχε στην αρχή την τιμή 23 και μετά την τιμή 18.

Ένα πρόγραμμα χρησιμοποιεί μεταβλητές για να αποθηκεύει τα δεδομένα και τα αποτελέσματα τα οποία παράγει. Κατά την εκτέλεση του προγράμματος, οι τιμές των μεταβλητών μπορεί να αλλάζουν.

"μαθαίνουμε" τον τρόπο δουλειάς του οδοντίατρου, διακρίνοντας και κατανοώντας τα επιμέρους απλούστερα τμήματα από τα οποία αποτελείται η εργασία του, λέγεται **ανάλυση**.

Μετά το στάδιο της ανάλυσης ακολουθεί το στάδιο του **σχεδιασμού**. Στο στάδιο αυτό μπορούμε, αν αυτό κριθεί απαραίτητο, να προτείνουμε νέους καλύτερους τρόπους εργασίας ή ακόμη και νέες δυνατότητες. Η ανάλυση και ο σχεδιασμός δεν έχουν απαραιτήτως άμεση σχέση με τον προγραμματισμό, αλλά χρησιμοποιούνται ως αναγκαία εργαλεία για τη σωστή και επαγγελματική κατάστρωση προγραμμάτων.

Λογικό διάγραμμα

Στα στάδια της ανάλυσης και του σχεδιασμού ο αναλυτής χρησιμοποιεί, εκτός των άλλων «εργαλείων» του, διαγράμματα για να δώσει μια εικόνα της λειτουργίας του υπό μελέτη συστήματος. Η σχηματική αυτή απεικόνιση ενός συστήματος χρησιμοποιεί γεωμετρικά σχήματα και καλείται λογικό διάγραμμα ή διάγραμμα ροής (flow chart). Σε αυτό το βιβλίο χρησιμοποιείται συχνά το λογικό διάγραμμα για να επεξηγηθεί η λειτουργία μιας εντολής, μιας διαδικασίας, ή ενός προγράμματος.

Το διάγραμμα ροής αποτέλεσε την πρώτη μέθοδο αναπαράστασης αλγορίθμων. Για πρώτη φορά παρουσιάστηκε από τον Frank Gilbreth το 1921 και αργότερα κυριάρχησε ως μέθοδος αναπαράστασης αλγορίθμων για συστήματα Η/Υ. Ακόμη και οι σύγχρονες τεχνικές αναπαράστασης, όπως τα διαγράμματα ενέργειας της UML[3], αποτελούν μια επέκταση των κλασικών λογικών διαγραμμάτων.

Το χτίσιμο ενός λογικού διαγράμματος βασίζεται στη χρήση συγκεκριμένων γεωμετρικών σχημάτων. Υπάρχουν δεκάδες διαφορετικά σχήματα που μπορούν να χρησιμοποιηθούν σε ένα διάγραμμα ροής, ανάλογα με τη πολυπλοκότητα και τη χρήση του. Όμως σε ένα απλό κλασικό λογικό διάγραμμα τα διαφορετικά σχήματα που χρησιμοποιούνται είναι πολύ περιορισμένα.

Ψηφιακές σελίδες

[3] Η UML (Unified Modeling Language) είναι μια σύγχρονη μέθοδος γραφικής αναπαράστασης αλγορίθμων, ιδιαίτερα χρήσιμη στον αντικειμενοστρεφή προγραμματισμό. Η ανάπτυξη της UML ξεκίνησε το 1994 και συνεχώς εξελίσσεται.

Σε ένα απλό λογικό διάγραμμα χρησιμοποιούνται καθορισμένα γεωμετρικά σχήματα που το κάθε ένα υποδεικνύει μια συγκεκριμένη λειτουργία. Κάθε σχήμα αντιστοιχεί σε μία κατηγορία διαδικασίας του "υπό μελέτη" συστήματος. Για παράδειγμα, ο ρόμβος δείχνει έλεγχο ενώ το ορθογώνιο παραλληλόγραμμο επεξεργασία. Στον πίνακα που ακολουθεί αναφέρονται τα σχήματα που συνήθως χρησιμοποιούνται σε ένα λογικό διάγραμμα.

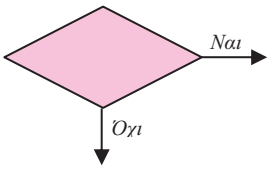
Γεωμετρικό σχήμα	Λειτουργία
	Αρχή/Τέλος διαδικασίας
	Επεξεργασία
	Έλεγχος
	Είσοδος/Έξοδος πληροφοριών
	Μεταφορά ελέγχου

Ενδεικτικές σελίδες

Σχήμα 1.1 *Τα σχήματα του λογικού διαγράμματος*

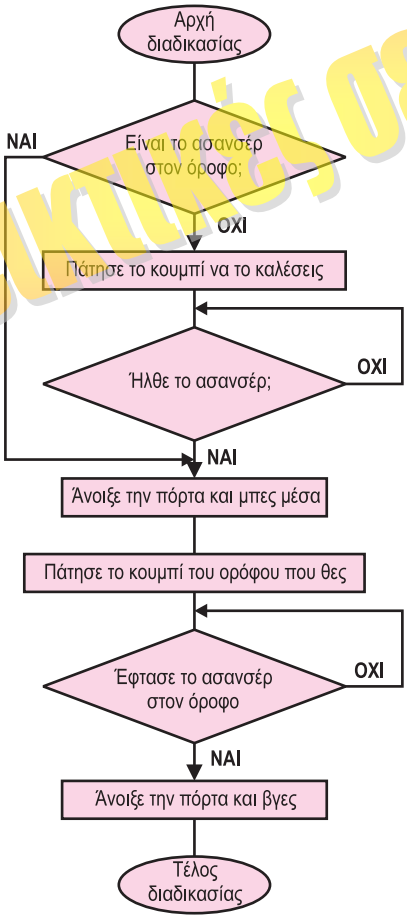
Μέσα σε κάθε γεωμετρικό σχήμα περιγράφεται με απλά λόγια η διαδικασία που εκτελείται.

Στη διαδικασία του ελέγχου είναι δυνατές μόνο δύο αποφάσεις: **Ναι** ή **Όχι**. Ανάλογα με το αποτέλεσμα του ελέγχου, το διάγραμμα ροής μάς οδηγεί σε διαφορετικό παρακλάδι του.



Το επόμενο λογικό διάγραμμα δείχνει τη διαδικασία που ακολουθείται όταν κάποιος καλεί το ασανσέρ μέχρι να πάει στον όροφο που τον ενδιαφέρει. Αναμφίβολα υπάρχουν πολλές εκδοχές για το σύστημα αυτό. Το λογικό διάγραμμα που ακολουθεί δείχνει εποπτικά τη λειτουργία της συγκεκριμένης διαδικασίας.

Ενδεικτικές σελίδες



Σχήμα 1.2 *Το λογικό διάγραμμα της διαδικασίας κλήσης ενός ασανσέρ*

Ενδεικτικές σελίδες

Κεφάλαιο

2

Μια πρώτη ματιά στη C

Ενδεικτικές σελίδες

Μια πρώτη ματιά στη C

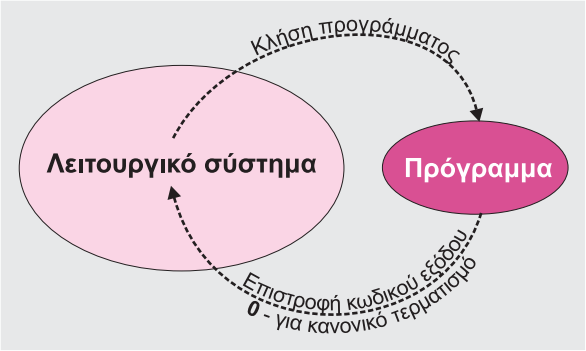
Στο κεφάλαιο αυτό ξεκινά μια μικρή "περιοδεία" στη γλώσσα C. Γίνεται μια πρώτη γνωριμία, ώστε να αποκτήσει ο αναγνώστης μια σφαιρική εικόνα της δομής και των χαρακτηριστικών της γλώσσας, αλλά και να μπορεί να φτιάχνει κάποια απλά προγράμματα.

Τα περισσότερα από όσα αναφέρονται στο παρόν κεφάλαιο εξετάζονται ξανά και πιο αναλυτικά στα επόμενα κεφάλαια.

Πριν ακόμα προχωρήσουμε σε αυτή τη πρώτη γνωριμία, ας δούμε τη διαδικασία με την οποία εκτελείται ένα οποιοδήποτε πρόγραμμα στον Η/Υ μας. Η διαδικασία αυτή θα μας βοηθήσει να κατανοήσουμε κάποιες έννοιες που θα συναντήσουμε αμέσως μετά.

Ενδεικτικές σελίδες

Γνωρίζουμε ότι οποιοδήποτε σύστημα Η/Υ βρίσκεται υπό τον έλεγχο του λειτουργικού του συστήματος (π.χ. Windows, Linux, Mac OS, κ.λπ.). Η εκτέλεση ενός προγράμματος συνήθως ξεκινάει από μια ενέργεια του χρήστη (π.χ. με ένα κλικ του ποντικιού), και μετά το λειτουργικό σύστημα "φορτώνει" το πρόγραμμα στη μνήμη του Η/Υ και μας βάζει στο περιβάλλον του. Με τον τερματισμό του προγράμματος επιστρέφουμε πίσω στο περιβάλλον του λειτουργικού συστήματος. Συνήθως, το πρόγραμμα μετά τον τερματισμό του επιστρέφει στο λειτουργικό σύστημα έναν κωδικό (κωδικός εξόδου), ο οποίος είναι απλώς ένας ακέραιος αριθμός. Ο **κωδικός εξόδου** δείχνει αν το πρόγραμμα τερματίστηκε κανονικά ή παρουσίασε κάποιο πρόβλημα.



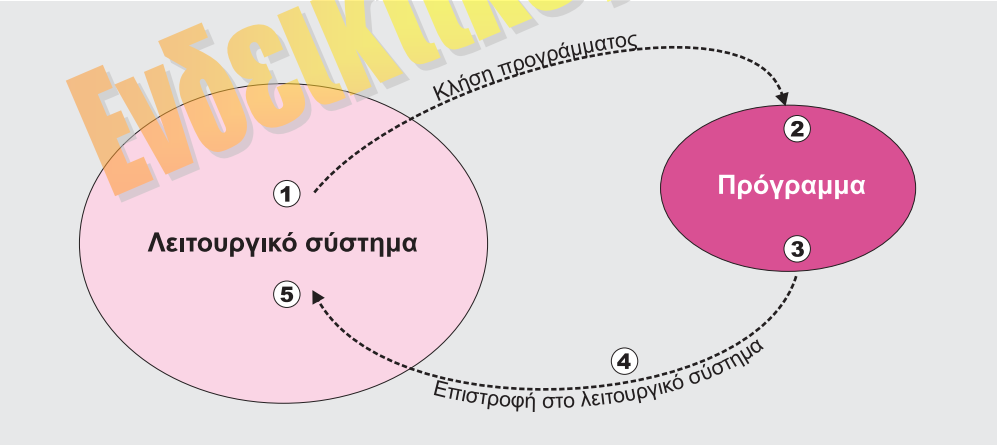
Κωδικός **0** σημαίνει ότι το πρόγραμμα τερματίστηκε κανονικά. Οποιοσδήποτε άλλος αριθμός υποδηλώνει πρόβλημα και ανορθόδοξο τερματισμό του προγράμματος! Αυτός ο κωδικός αργότερα ελέγχεται από το λειτουργικό σύστημα, το οποίο με αυτόν τον τρόπο γνωρίζει αν το πρόγραμμα εκτελέστηκε κανονικά. Στην περίπτωση προβλήματος (κωδικός εξόδου διάφορος του

μηδενός), το λειτουργικό σύστημα μπορεί να κάνει κάποια ενέργεια (π.χ. να εμφανίσει κάποιο μήνυμα).

Αναλυτικά, τα στάδια της παραπάνω διαδικασίας είναι τα ακόλουθα πέντε:

1. Κάποια ενέργεια του χρήστη υποδηλώνει την εκτέλεση ενός προγράμματος (π.χ. ένα διπλό κλικ επάνω σε κάποιο εικονίδιο).
2. Το λειτουργικό σύστημα εκτελεί το πρόγραμμα και μας βάζει στο περιβάλλον του.
3. Κάποια ενέργεια του χρήστη έχει αποτέλεσμα τον τερματισμό του προγράμματος (π.χ. από το μενού Αρχείο-έξοδος).
4. Το πρόγραμμα επιστρέφει στο λειτουργικό σύστημα τον **κωδικό εξόδου**, με τη λογική που αναφέρθηκε προηγουμένως.
5. Το λειτουργικό σύστημα ελέγχει τον κωδικό που επέστρεψε το πρόγραμμα και αντιδρά ανάλογα (π.χ. εμφανίζει ένα μήνυμα λάθους).

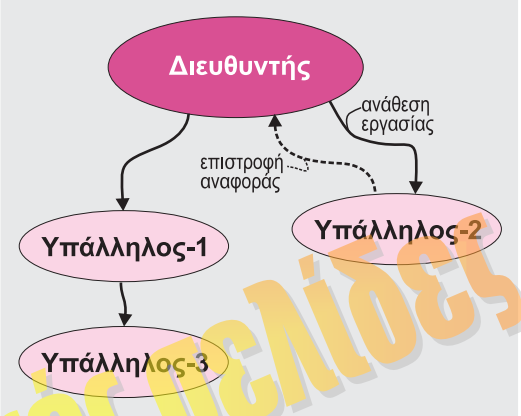
Ενδεικτικές σελίδες



Σχήμα 2.1 *Τα στάδια της κλήσης ενός προγράμματος από το λειτουργικό σύστημα*

Ο **κωδικός εξόδου** είναι ένας ακέραιος αριθμός τον οποίο επιστρέφει ένα πρόγραμμα, μετά από τον τερματισμό του, στο λειτουργικό σύστημα. Ο κωδικός εξόδου είναι στην ευθύνη του προγραμματιστή. Αυτός αποφασίζει τους κωδικούς εξόδου που θα επιστρέφει το πρόγραμμά του, έχοντας πάντα υπόψη ότι σε περίπτωση ομαλού τερματισμού το πρόγραμμα θα πρέπει να επιστρέφει κωδικό εξόδου 0.

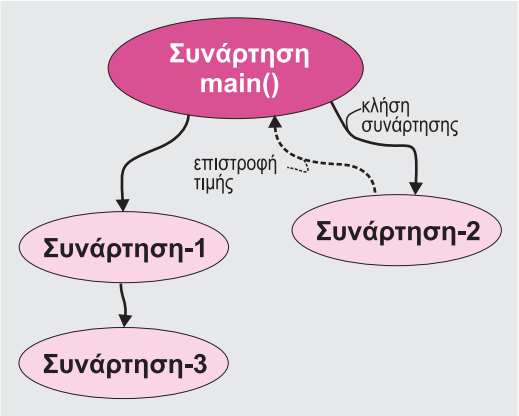
Η δομή ενός προγράμματος στη C



Σκεφτείτε τον τρόπο με τον οποίο λειτουργεί μια εταιρεία. Υπάρχει ο διευθυντής και οι υπάλληλοι που κάνουν συγκεκριμένες δουλειές. Όσο μεγαλώνει μια εταιρεία, τόσο περισσότερους υπαλλήλους χρειάζεται. Στη περίπτωση μιας πολύ μικρής εταιρείας, τότε ο μοναδικός που εργάζεται είναι ο διευθυντής ο οποίος κάνει όλες τις δουλειές (one man show). Για να κάνει όμως μια εργασία κάποιος υπάλληλος, θα πρέπει να του την αναθέσει ο διευθυντής ή κάποιος άλλος υπάλληλος (π.χ. ο προϊστάμενος του). Στο παράδειγμα του παραπάνω σχήματος, ο διευθυντής αναθέτει εργασία στον υπάλληλο-1 και στον υπάλληλο-2, ενώ ο υπάλληλος-1 αναθέτει εργασία στον υπάλληλο-3. Τα πάντα όμως ξεκινάνε από τις εντολές του διευθυντή!

Ενδεικτικές σελίδες

Ένας υπάλληλος μπορεί να κάνει τη δουλειά του χωρίς να επιστρέφει κάτι σε αυτόν που του την ανέθεσε, αλλά μπορεί να επιστρέψει πίσω και κάποια αναφορά. Π.χ., αν ανατεθεί σε έναν υπάλληλο να ταχυδρομήσει κάποιες επιστολές, δεν χρειάζεται να επιστρέψει κάποια αναφορά σε αυτόν που του ανέθεσε την εργασία. Αν όμως του ανατεθεί να ελέγξει τα διαθέσιμα χρήματα που υπάρχουν στο ταμείο, σίγουρα τότε θα πρέπει να επιστρέψει ως αναφορά το ποσό που εντόπισε.



Όπως έχει ήδη αναφερθεί, ένα πρόγραμμα στη C χωρίζεται συνήθως σε τμήματα (υπαλλήλους) κάθε ένα από τα οποία εκτελεί μια συγκεκριμένη εργασία. Τα τμήματα αυτά λέγονται **συναρτήσεις** και η κάθε

Ας ξαναγράψουμε το πρώτο μας πρόγραμμα

Ας θυμηθούμε λίγο την εικόνα του σχήματος 2.1 και τον τρόπο κλήσης ενός προγράμματος από το λειτουργικό σύστημα του Η/Υ μας. Έχουμε ήδη πει ότι το πρόγραμμα επιστρέφει στο λειτουργικό σύστημα έναν κωδικό εξόδου, ο οποίος είναι ένας ακέραιος αριθμός. Στην περίπτωση κανονικής λειτουργίας του προγράμματος, ο κωδικός αυτός πρέπει να είναι ο αριθμός 0. Η συνάρτηση η οποία περιέχει το βασικό κορμό ενός προγράμματος της C είναι η συνάρτηση main(). Όταν τερματιστεί η εκτέλεση της main(), τερματίζεται και το πρόγραμμά μας. Επομένως, την ευθύνη της επιστροφής του κωδικού εξόδου την έχει η συνάρτηση main().

> Συνοψίζοντας έχουμε τα εξής: α) Η συνάρτηση main() πρέπει να επιστρέφει έναν ακέραιο αριθμό (τον κωδικό εξόδου) και β) Στην περίπτωση κανονικής εκτέλεσης του προγράμματος, ο αριθμός αυτός πρέπει να είναι το 0.

Για να είμαστε λοιπόν και τυπικά σωστοί (σύμφωνα με το πρότυπο C99), θα πρέπει να ξαναγράψουμε το πρώτο μας πρόγραμμα λαμβάνοντας υπόψη τα παραπάνω. Έτσι το πρόγραμμα διαμορφώνεται ως εξής:

```
#include <stdio.h>

int main(void)
{
    printf("Hello world");
    return 0;
}
```

> Το πρόθεμα **int** πριν από το όνομα της συνάρτησης **main()**, υποδηλώνει ότι ο **τύπος** της συνάρτησης είναι **int** και επομένως επιστρέφει έναν ακέραιο αριθμό.

> Ο τύπος **void**, μέσα στις παρενθέσεις της main(), δηλώνει ρητά ότι η main() **δεν** έχει παραμέτρους.

> Με την πρόταση **return 0;** η συνάρτηση main() **επιστρέφει**, ως **κωδικό εξόδου**, το **0**.

Το πρότυπο C99 επιβάλλει η main() να είναι τύπου int, και επομένως να επιστρέφει (με return) έναν ακέραιο αριθμό. Όμως, αρκετοί μεταγλωττιστές θα μεταγλωττίσουν χωρίς πρόβλημα το πρόγραμμα μας, ακόμα και αν δεν δηλώσουμε ως int τη main() και παραλείψουμε την εντολή return. Αυτό γίνεται για λόγους συμβατότητας με τα αρχικά πρότυπα της γλώσσας, τα οποία δεν επέβαλλαν αυτούς τους κανόνες. Ο τύπος void, μέσα στις παρενθέσεις της main(), δηλώνει ρητά ότι η main() δεν έχει παραμέτρους.

> Από τώρα και στο εξής θα χρησιμοποιούμε την παραπάνω σύνταξη στη main().

Ενδεικτικές Σελίδες

Συχνά λάθη

- Ξεχνάμε να δηλώσουμε τη main() ως τύπου int και να βάλουμε την εντολή return 0. Αυτό είναι τυπικό (σύμφωνα με το πρότυπο C99) και όχι ουσιαστικό λάθος. Αρκετοί μεταγλωττιστές δεν θα έχουν πρόβλημα.
- Όλες οι δηλωτικές και εκτελέσιμες προτάσεις στη C τερματίζονται με ερωτηματικό (;). Καμιά φορά το ξεχνάμε!
- Κατά λάθος τερματίζουμε τις οδηγίες #include και #define με ερωτηματικό.
- Χρησιμοποιούμε μια μεταβλητή χωρίς προηγουμένως να την έχουμε δηλώσει.
- Κάνουμε διαίρεση με ακέραιους αριθμούς και περιμένουμε το αποτέλεσμα να είναι δεκαδικός. Για παράδειγμα, περιμένουμε ότι το αποτέλεσμα της παράστασης 5/2 θα είναι 2.5, ενώ είναι 2!
- Χρησιμοποιούμε συναρτήσεις βιβλιοθήκης χωρίς να έχουμε συμπεριλάβει (με #include) τα απαραίτητα αρχεία κεφαλίδας.

Ενδεικτικές σελίδες

Ανασκόπηση Κεφαλαίου 2

- Κάθε πρόγραμμα της C περιέχει υποχρεωτικά μία συνάρτηση main() η οποία είναι εκείνη που εκτελείται πρώτη. Κάθε πρόγραμμα της C μπορεί να περιέχει και άλλες συναρτήσεις, εκτός από τη main(), οι οποίες για να εκτελεστούν πρέπει να κληθούν.
- Κάθε μεταβλητή που χρησιμοποιείται πρέπει να δηλώνεται. Οι δηλωτικές προτάσεις των μεταβλητών συνήθως τοποθετούνται στην αρχή μιας συνάρτησης.
- Οι βασικοί τύποι μεταβλητών στη C είναι τέσσερις: ο τύπος char, ο τύπος int, ο τύπος float, και ο τύπος double.
- Οι αριθμητικές παραστάσεις είναι πράξεις μεταξύ αριθμητικών δεδομένων και έχουν αποτέλεσμα έναν αριθμό.
- Οι λογικές παραστάσεις χρησιμοποιούν συγκριτικούς και λογικούς τελεστές και έχουν αποτέλεσμα **αλήθεια** (τιμή 1) ή **ψέμα** (τιμή 0).
- Ο τελεστής = καταχωρίζει το αποτέλεσμα της παράστασης στα δεξιά του στη μεταβλητή που βρίσκεται αριστερά του, π.χ. a=b+3

➢ Μια σύνθετη πρόταση αποτελείται από απλές προτάσεις μέσα σε αριστερή και δεξιά αγκύλη. Η C αντιμετωπίζει μια σύνθετη πρόταση ως **μία** πρόταση.

Ασκήσεις Κεφαλαίου 2

2.1 Τι θα περιέχουν οι μεταβλητές a, b, και c μετά το τέλος του παρακάτω κώδικα: ★

```
int main(void)
{
    int a,b,c=3;
    a=b=2;
    a=c+b;
    return 0;
}
```

2.2 Τι θα περιέχουν οι μεταβλητές a, b, και c μετά το τέλος του παρακάτω κώδικα: ★

```
#define MM 23
int main(void)
{
    const int c=3;
    int a,b;
    a=4+(b=2);
    b=c+b+MM;
    return 0;
}
```

2.3 Εντοπίστε τα λάθη στον παρακάτω κώδικα: ★★

```
#include stdio.h
#define MM 23;
int main(void)
{
    const int c=3;
    int a,b;
    a=2;
    float d;
    d=4.3
```

ΕΥΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

```
    a=4+(b=2);
    MM=10;
    3=a;
    c=c+b+MM;
    return 0;
}
```

2.4 Τι θα περιέχουν οι μεταβλητές a, b, και c μετά το τέλος του παρακάτω κώδικα: ★★

```
#include <stdio.h>
int main(void)
{
    int a,b,c=3;
    a=b=2;
    a=c>b;
    b=b==1;
    c=printf("τέλος");
    return 0;
}
```

Ενδεικτικές σελίδες

2.5 Ποια από τα παρακάτω αληθεύουν: ★

- ❑ Δηλωτικές προτάσεις μπορούν να μπουν σε οποιοδήποτε σημείο του προγράμματος.
- ❑ Ένα πρόγραμμα της C μπορεί να χωριστεί σε περισσότερα τμήματα (συναρτήσεις).
- ❑ Μια λογική παράσταση έχει τιμή 1 ή 0.
- ❑ Μια μεταβλητή στη C, πριν της δοθεί τιμή, έχει τιμή 0.
- ❑ Η οδηγία #define χρησιμοποιείται για να ορίσει μία σταθερά του προγράμματός μας.
- ❑ Η παράσταση a==5 καταχωρίζει στη μεταβλητή a το 5.
- ❑ Η παράσταση a=5==5 καταχωρίζει στη μεταβλητή a το 1.
- ❑ Η παράσταση a=a!=a καταχωρίζει στη μεταβλητή a το 0.

2.6 Με δεδομένες τις τιμές των μεταβλητών a ,b, και c ίσες με 5, 10, και 15 αντίστοιχα, σημειώστε την τιμή (1 για αλήθεια, 0 για ψέμα) των παρακάτω λογικών παραστάσεων: ★★

Κεφάλαιο

3

Προετοιμασία για αργότερα

Ενδεικτικές σελίδες

Προετοιμασία για αργότερα

Η C και η μνήμη

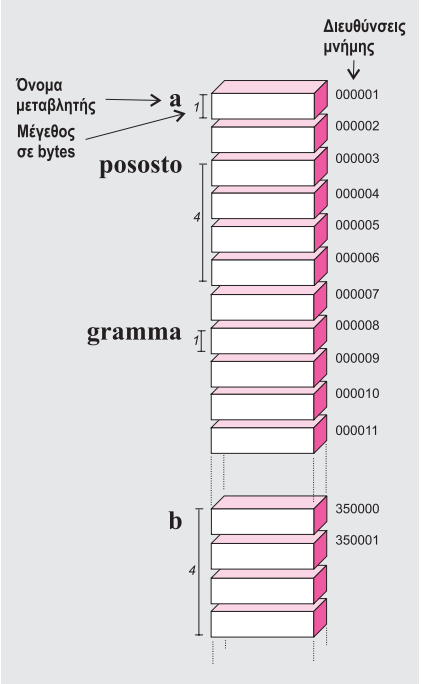
Όπως αναφέρθηκε και προηγουμένως, η C χειρίζεται τέσσερις βασικούς τύπους μεταβλητών: int, char, float, και double. Φανταζόμαστε την κάθε μεταβλητή σαν ένα κουτί με ένα όνομα και κάποιο περιεχόμενο. Οι μεταβλητές καταλαμβάνουν χώρο στη μνήμη RAM του Η/Υ, αλλά γνωρίζουμε ότι κάθε θέση μνήμης RAM έχει μέγεθος 1 byte.

Κάθε λοιπόν μεταβλητή του προγράμματος καταλαμβάνει ακριβώς τον ίδιο χώρο στη μνήμη του Η/Υ; ΟΧΙ βέβαια.

> Κάθε μεταβλητή καταλαμβάνει έναν αριθμό θέσεων μνήμης (byte), ο οποίος εξαρτάται από τον τύπο της μεταβλητής, από τον τύπο του συστήματος Η/Υ στο οποίο δουλεύουμε, αλλά και από το μεταγλωττιστή που χρησιμοποιούμε.

Ενδεικτικές σελίδες

Στα συστήματα προσωπικών Η/Υ που βασίζονται στους επεξεργαστές 32 bit της INTEL, οι μεταβλητές τύπου int καταλαμβάνουν τέσσερα byte, οι char ένα byte, οι float τέσσερα, και οι double 8. Αργότερα θα αναφερθούμε και σε άλλους τύπους μεταβλητών (που στηρίζονται πάντως στους τρεις βασικούς) με διαφορετικά μεγέθη σε byte.



Ας θεωρήσουμε τέσσερις μεταβλητές, οι οποίες δηλώνονται με τις παρακάτω προτάσεις:

```
char a,gramma;
float pososto;
int b;
```

Στο διπλανό σχήμα φαίνεται εποπτικά η μνήμη ενός Η/Υ σαν ένα σύνολο από κου-

τιά (θέσεις μνήμης) το ένα πάνω από το άλλο. Το κάθε κουτί έχει μέγεθος ενός byte και διαθέτει έναν αύξοντα αριθμό (ο αριθμός στα δεξιά), ο οποίος αποτελεί τη **'διεύθυνση'** του κουτιού μέσα στη μνήμη. Κάθε μεταβλητή που δηλώνεται, δεσμεύει κάποιες θέσεις μνήμης, ανάλογα με τον τύπο της. Έτσι η **a** και η **gramma** δεσμεύουν από μία θέση (την 1 και την 8 αντίστοιχα), η **pososto** τέσσερις (τις 3, 4, 5, και 6), και η **b** επίσης τέσσερις (τις 350000, 350001, 350002, και 350003).

Πέρα λοιπόν από το όνομα και το περιεχόμενο μιας μεταβλητής, έννοιες που είναι ήδη γνωστές, συναντάμε εδώ άλλες δύο νέες έννοιες: της **διεύθυνσης** και του **μεγέθους** μιας μεταβλητής.

Διεύθυνση μιας μεταβλητής είναι η διεύθυνση του πρώτου byte των θέσεων (ή της θέσης) μνήμης που δεσμεύει η μεταβλητή.

Έτσι, οι διευθύνσεις των παραπάνω μεταβλητών είναι:

a → `000001`
b → `350000`
pososto → `000003`
gramma → `000008`

Μέγεθος μιας μεταβλητής είναι ο αριθμός των byte που δεσμεύει αυτή η μεταβλητή, και καθορίζεται από τον τύπο της.

Η C διαθέτει τελεστές οι οποίοι επιστρέφουν τη διεύθυνση και το μέγεθος μιας μεταβλητής.

Ο τελεστής &

Ο τελεστής **&** επιστρέφει έναν ακέραιο αριθμό ο οποίος προσδιορίζει τη διεύθυνση μιας μεταβλητής. Έτσι, σύμφωνα με το προηγούμενο σχήμα, η παράσταση **&a** επιστρέφει τιμή 1 και η παράσταση **&gramma** τιμή 8.

Η διεύθυνση μιας θέσης μνήμης, στα συστήματα προσωπικών Η/Υ που βασίζονται στους επεξεργαστές 32 bit της INTEL, είναι πάντα ένας αριθμός μεγέθους 4 byte ο οποίος συνήθως αναπαριστάνεται σε δεκαεξαδική μορφή.

Ενδεικτικές σελίδες

η τελευταία printf() δεν θα εκτελεστεί ποτέ, επειδή με την κλήση της συνάρτησης exit() το πρόγραμμα θα τερματιστεί νωρίτερα. Η συνάρτηση exit() έχει υποχρεωτικά ως παράμετρο έναν ακέραιο αριθμό (τύπου int), π.χ. exit(0). Σε κάθε περίπτωση, το πρόγραμμα τερματίζεται και ταυτόχρονα επιστρέφει ως κωδικό εξόδου, στο λειτουργικό σύστημα, την τιμή της παραμέτρου.

☞ Οι διαφορετικοί κωδικοί εξόδου χρησιμοποιούνται από τους προγραμματιστές, για να δηλώσουν τις ενδεχόμενες περιπτώσεις τερματισμού ενός προγράμματος.

☞ Ο κωδικός εξόδου 0 προσδιορίζει κανονικό τερματισμό του προγράμματος, π.χ. exit(0).

Ολίγη if παρακαλώ

Η εντολή if χρησιμοποιείται για τον έλεγχο μιας λογικής παράστασης, και ανάλογα με την τιμή της (αληθής ή ψευδής) εκτελεί ή δεν εκτελεί κάποιες προτάσεις.

Για παράδειγμα, ο παρακάτω κώδικας:

```
if (bathmos>=5)
    printf("Πέρασες\n");
else
    printf("Δυστυχώς κόπηκες\n");
```

ελέγχει την τιμή της μεταβλητής bathmos και εμφανίζει ανάλογα τη φράση "Πέρασες" ή "Δυστυχώς κόπηκες".

Στην παρούσα ενότητα θα αναφέρουμε λίγες μόνο από τις δυνατότητες της if, και μάλιστα μέσα από παραδείγματα, δεδομένου ότι το πλήρες συντακτικό και η ανάλυση της εντολής θα γίνουν σε επόμενο κεφάλαιο.

Το συντακτικό της εντολής if είναι το εξής:

if (*λογική παράσταση*) *πρόταση-Α*; [else *πρόταση-Ψ*;]

Αν η ***λογική παράσταση*** έχει τιμή **αλήθεια** εκτελείται η ***πρόταση-Α***, ενώ αν έχει τιμή **ψέμα** εκτελείται η ***πρόταση-Ψ*** (αν υπάρχει). Το else και η ***πρόταση-Ψ*** είναι προαιρετικά. Οι αγκύλες [ ] **δεν** μετέχουν στο συντακτικό της εντολής, αλλά δείχνουν ότι το τμήμα αυτό της if είναι προαιρετικό.

Ενδεικτικές σελίδες

☞ Από τώρα και στο εξής θα χρησιμοποιούμε τις αγκύλες [ ] για να σηματοδοτούμε τα προαιρετικά τμήματα μιας εντολής. Οι αγκύλες δεν πρέπει να συγχέονται με τα άγκιστρα { } που σηματοδοτούν την αρχή και το τέλος μιας σύνθετης πρότασης.

Η επόμενη εντολή if ελέγχει την τιμή του x. Αν είναι μεγαλύτερη από 5 τότε εμφανίζει στην οθόνη "x>5"· διαφορετικά εμφανίζει "x<=5".

```
if (x>5) printf("x>5\n"); else printf("x<=5\n");
```

Η παραπάνω εντολή if θα μπορούσε να γραφτεί επίσης με τον ακόλουθο τρόπο, χωρίς να αλλάξει καθόλου το νόημά της:

```
if (x>5)
    printf("x>5\n");
else
    printf("x<=5\n");
```

Αυτός ο δεύτερος τρόπος κάνει το πρόγραμμα πιο ευανάγνωστο, και είναι αυτός που θα ακολουθήσουμε στη συνέχεια.

Παρατηρούμε ότι στο προηγούμενο παράδειγμα θα εκτελεστεί σε κάθε περίπτωση (αλήθεια ή ψέμα) μία απλή πρόταση. Τι θα γινόταν στην περίπτωση που θα θέλαμε να εκτελεστούν περισσότερες προτάσεις; Αναφερθήκαμε προηγουμένως στην έννοια της σύνθετης πρότασης (compound statement) και την ικανότητα της C να τη μεταχειρίζεται σαν οποιαδήποτε απλή πρόταση. Επομένως, μπορούμε ανά πάσα στιγμή να χρησιμοποιήσουμε στη θέση μιας απλής πρότασης μια σύνθετη πρόταση, η οποία μπορεί να συνίσταται από πολλές απλές προτάσεις. Το επόμενο παράδειγμα δείχνει τη χρήση δύο σύνθετων προτάσεων σε μια εντολή if.

```
if (x>5)
{
    printf("Το x είναι μεγαλύτερο του 5\n");
    x=x/2;
}
else
{
    printf("Το x είναι μικρότερο ή ίσο του 5\n");
    x=x*2;
}
```

Σύνθετη πρόταση. Θα εκτελεστεί στην περίπτωση που η λογική παράσταση **x>5** είναι **αληθής**.

Σύνθετη πρόταση. Θα εκτελεστεί στην περίπτωση που η λογική παράσταση **x>5** είναι **ψευδής**.

Ενδεικτικές σελίδες

Παραδείγματα

Π3.1 Το επόμενο πρόγραμμα ζητάει τρεις βαθμούς ενός μαθητή και υπολογίζει το μέσο όρο τους. Αν ο μέσος όρος είναι μεγαλύτερος ή ίσος με 10, εμφανίζει "Πέρασες με βαθμό ##", διαφορετικά εμφανίζει "Κόπηκες" (στη θέση του ## εμφανίζεται ο μέσος όρος).

3_p3_1.c

```
#include <stdio.h>
int main(void)
{
   int b1,b2,b3;
   float mo;
   printf("Δώσε τρεις βαθμούς:");
   scanf("%d %d %d",&b1,&b2,&b3);
   mo=(b1+b2+b3)/3.0;
   if (mo>=10)
      printf("Πέρασες με βαθμό %f\n",mo);
   else
      printf("Κόπηκες\n");
   return 0;
}
```

Η **scanf()** διαβάζει από το πληκτρολόγιο τρεις ακεραίους αριθμούς τους οποίους καταχωρίζει στις μεταβλητές **b1**, **b2**, και **b3**.

Υπολογισμός του μέσου όρου.

Έλεγχος του μέσου όρου και εμφάνιση του κατάλληλου μηνύματος.

Ενδεικτικές Λύσεις

Π3.2 Το επόμενο πρόγραμμα έχει λάθη:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   int a,b,c;
   float d;
   scanf("%d %d",a,b);
   int k;
   scanf("%c",&c);
   if (c=1)
      printf("NAI\n");
   else
      printf("OXI\n");
   printf(Τέλος);
   printf("\n");
   return 0;
}
```

Η **scanf()** χρειάζεται τις διευθύνσεις, και όχι τα ονόματα των μεταβλητών. Θα έπρεπε να ήταν **&a**,**&b**.

Εφόσον ζητείται ακέραιος αριθμός, το αλφαριθμητικό μορφοποίησης θα πρέπει να είναι **%d**.

Χρησιμοποιείται ο τελεστής ανάθεσης = και όχι ο τελεστής σύγκρισης ==.

Το αλφαριθμητικό **Τέλος** θα έπρεπε να είναι μέσα σε διπλά εισαγωγικά.

Κεφάλαιο

4

Ο τύπος δεδομένων int

Ενδεικτικές σελίδες

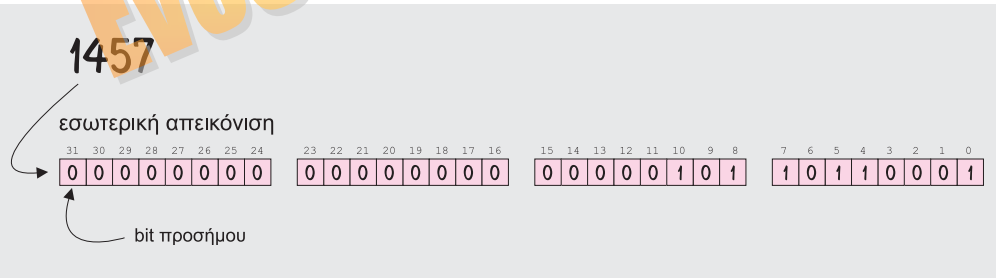
Ο τύπος δεδομένων int

Το κεφάλαιο αυτό εισάγει και αναλύει τον τρόπο χρήσης των ακεραίων (integers) στη γλώσσα C. Στο κεφάλαιο αυτό εξετάζεται η εσωτερική απεικόνιση των ακέραιων σταθερών και μεταβλητών, καθώς και οι τελεστές που εφαρμόζονται σε ακέραια μεγέθη. Πολλοί από αυτούς τους τελεστές εφαρμόζονται και σε άλλους τύπους δεδομένων.

Σταθερές τύπου int

Οι ακέραιες σταθερές δεν είναι τίποτε άλλο από απλοί ακέραιοι αριθμοί. Μια ακέραια σταθερά αποθηκεύεται στη μνήμη αφού μετατραπεί σε δυαδικό αριθμό. Μια σταθερά τύπου **int** μετατρέπεται σε ένα δυαδικό αριθμό μήκους 4 byte. Με 4 byte μπορεί να αναπαρασταθεί αριθμός από -2.147.483.648 μέχρι +2.147.483.647. Το bit υπ' αριθμόν 31 (τα 32 bit αριθμούνται από το 0 έως το 31) χρησιμοποιείται για το πρόσημο και τα υπόλοιπα 31 για την αναπαράσταση του αριθμού. Το παρακάτω σχήμα δείχνει την εσωτερική αναπαράσταση του αριθμού 1457.

Ενδεικτικές σελίδες



Μεταβλητές τύπου int

Η C διαθέτει, πέρα από τον τύπο int που έχουμε ήδη αναφέρει, άλλους δυο τύπους ακέραιων μεταβλητών: long int και short int.

Οι μεταβλητές τύπου int δηλώνονται ως εξής:

int *όνομα_μεταβλητής1*, *όνομα_μεταβλητής2*, ...;

Οι μεταβλητές τύπου long int δηλώνονται ως εξής:

long int *όνομα_μεταβλητής1*, *όνομα_μεταβλητής2*, ...; ή

Προτεραιότητα τελεστών

Στη C μπορεί μέσα σε μια παράσταση να συνυπάρχουν αριθμητικοί και λογικοί τελεστές, δεδομένου ότι τελικά όλοι αποδίδουν αριθμητικές τιμές.

Ο διπλανός πίνακας δείχνει την προτεραιότητα των τελεστών, αλλά και τη φορά εκτέλεσής τους στην περίπτωση που υπάρχουν πολλοί τελεστές με την ίδια προτεραιότητα. Οι τελεστές ανάμεσα σε δύο έντονες οριζόντιες γραμμές, με το ίδο χρώμα φόντου, έχουν την ίδια προτεραιότητα. Για παράδειγμα, ο πολλαπλασιασμός, η διαίρεση, και το υπόλοιπο (%) έχουν την ίδια προτεραιότητα.

Η παράσταση 10/2*5 θα έχει αποτέλεσμα 25. Οι δύο τελεστές (/ και *) έχουν την ίδια προτεραιότητα και, σύμφωνα με το διπλανό πίνακα, η φορά εκτέλεσης των δύο πράξεων θα είναι από τα αριστερά προς τα δεξιά (⇨). Πρώτα λοιπόν θα γίνει η διαίρεση (10/2) και μετά ο πολλαπλασιασμός (5*5).

Προτεραιότητα τελεστών		
Τελεστής	**Σημασία**	**Φορά**
!	Λογικό NOT	⇦
~	Συμπλήρωμα	⇦
++	Αύξηση	⇦
--	Μείωση	⇦
*	Πολ/σμός	⇨
/	Διαίρεση	⇨
%	Υπόλοιπο	⇨
+	Πρόσθεση	⇨
-	Αφαίρεση	⇨
<<	Ολίσθηση αριστερά	⇨
>>	Ολίσθηση δεξιά	⇨
<	Μικρότερο	⇨
<=	Μικρότερο ή ίσο	⇨
>	Μεγαλύτερο	⇨
>=	Μεγαλύτερο ή ίσο	⇨
==	Ίσο	⇨
!=	Όχι ίσο	⇨
&	Bitwise AND	⇨
^	Bitwise XOR	⇨
|	Bitwise OR	⇨
&&	Λογικό AND	⇨
||	Λογικό OR	⇨
?:	OR συνθήκης	⇦
=	Ανάθεση	⇦

☞ Για να αλλάξει η σειρά εκτέλεσης των πράξεων πρέπει να χρησιμοποιηθούν παρενθέσεις ( ). Η χρήση τους επιβάλλεται επίσης στα σημεία όπου δεν είμαστε σίγουροι για την προτεραιότητα των τελεστών. Έτσι η παράσταση 10/(2*5) θα έχει αποτέλεσμα 1, επειδή θα εκτελεστεί πρώτα η πράξη μέσα στις παρενθέσεις.

☞ Παρατηρούμε ότι ο τελεστής ανάθεσης, το ίσον (=), έχει τη μικρότερη προτεραιότητα. Προσέξτε επίσης ότι οι τελεστές σύγκρισης έχουν μικρότερη προτεραιότητα από τους αριθμητικούς τελεστές.

Ενδεικτικές υπόδεις

Στον πίνακα υπάρχουν και τελεστές στους οποίους θα αναφερθούμε αργότερα. Προς το παρόν θα χρησιμοποιήσουμε τους πιο απαραίτητους τελεστές.

Αναφέραμε προηγουμένως ότι στη C οι λογικές παραστάσεις επιστρέφουν αριθμητικές τιμές: 1 για αλήθεια και 0 για ψέμα. Επομένως δεν πρέπει να μας παραξενεύουν οι παραστάσεις της μορφής:

```
a=b>c;
```

Σύμφωνα με τον πίνακα προτεραιότητας, πρώτα θα υπολογιστεί η παράσταση b>c. Επομένως στην a θα καταχωριστεί το 1 αν η b είναι μεγαλύτερη από τη c, διαφορετικά θα καταχωριστεί το 0.

```
a=b==c;
```

Την a θα καταχωριστεί το 1 αν η b είναι ίση με τη c, διαφορετικά θα καταχωριστεί το 0.

Ενδεικτικές σελίδες

Παραδείγματα

Π4.1 Στις επόμενες παραστάσεις υποθέτουμε ότι το x και το y έχουν τιμή 100 και 4 αντίστοιχα πριν από την εκτέλεση **κάθε** παράστασης. Στις στήλες του πίνακα εμφανίζονται οι τιμές των μεταβλητών x και y, καθώς και η τιμή της παράστασης μετά από την εκτέλεσή της.

Παράσταση	Τιμή του x	Τιμή του y	Τιμή της παράστασης
`y * x++`	101	4	400 (4 * 100)
`++x * --y`	101	3	303 (101 * 3)
`x-- + ++y`	99	5	105 (100 + 5)
`--x * 2`	99	4	198 (99 * 2)
`x=x>y++`	1	5	1
`(x<y)||(y>5)`	100	4	0
`++x>100`	101	4	1
`x++>100`	101	4	0
`x=x<y`	0	4	0
`x==y++ + 96`	100	5	1
`x==++y + 96`	100	5	0
`x=y=10`	10	10	10

Π4.2 Υποθέτουμε ότι μια μετεωρολογική συσκευή μέτρησης, συνδεδεμένη με τον Η/Υ, επιστρέφει την κατάστασή της και τη μέτρηση σε ένα αριθμό ως εξής:

Ο.Κ	Είδος	Είδος	Τιμή	Τιμή	Τιμή	Τιμή	Τιμή
7	6	5	4	3	2	1	0

Το bit 7 του αριθμού έχει τιμή 1 αν η συσκευή λειτουργεί σωστά. Τα bit 6 και 5 αναφέρουν το είδος της μέτρησης (0-Θερμοκρασία, 1-Πίεση, 2-Υγρασία, 3-Ταχύτητα ανέμου). Τα υπόλοιπα bit (4, 3, 2, 1, και 0) αποτελούν την τιμή της μέτρησης. Ο επόμενος κώδικας αναλύει τα bit του αριθμού και εμφανίζει την κατάσταση της συσκευής, το είδος της μέτρησης, και την τιμή της:

4_p4_2.c

```
#include <stdio.h>
int main(void)
{
    int ar,ok,eidos,timi;
    printf("Δώσε αριθμό από τη συσκευή:");
    scanf("%d",&ar);
    ok=ar&128>>7;
    eidos=(ar&96)>>5;
    timi=ar&31;
    printf("OK=%d Είδος=%d Τιμή=%d\n",ok,eidos,timi);
    return 0;
}
```

Ενδεικτικές σελίδες

96							
0	1	1	0	0	0	0	0

Ο τελεστής & μαζί με κάποια σταθερή τιμή, χρησιμοποιείται για να απομονώσει συγκεκριμένα bit του αριθμού. Π.χ. η ar&96 απομονώνει τα bit 6 και 5 του ar. Η εφαρμογή του τελεστή >>5 ολισθαίνει τα 2 bit κατά 5 θέσεις δεξιά, ώστε να είναι στην αρχή του byte και να αποδώσουν τιμές από 0 μέχρι 3.

128							
1	0	0	0	0	0	0	0

Η παράσταση ar&128 απομονώνει το bit 7 του αριθμού, και η εφαρμογή του τελεστή >>7 ολισθαίνει το bit 7 θέσεις δεξιά ώστε να είναι στην αρχή του byte και να αποδώσει τιμή 0 ή 1. Με ανάλογο τρόπο απομονώνονται και τα υπόλοιπα bit που μας ενδιαφέρουν.

Ενδεικτικές σελίδες

Κεφάλαιο 5

Ο τύπος δεδομένων char

Ενδεικτικές σελίδες

Ο τύπος δεδομένων char

Το κεφάλαιο αυτό εισάγει και αναλύει τον τρόπο χρήσης των χαρακτήρων (characters) στη γλώσσα C. Εξετάζουμε τον τρόπο με τον οποίο η C χειρίζεται τις σταθερές και τις μεταβλητές τύπου char. Ακόμη, γίνεται αναφορά σε συναρτήσεις που χρησιμοποιούνται για το χειρισμό των χαρακτήρων.

Σταθερές τύπου char

Μια σταθερά τύπου char δεν είναι τίποτε άλλο από **ένας χαρακτήρας** που περικλείεται σε **μονά** εισαγωγικά. Για παράδειγμα, οι παρακάτω εκφράσεις:

'A'

'1'

'*'

είναι παραδείγματα σταθερών τύπου char.

ΕΝΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

Οι χαρακτήρες είναι αριθμοί!!!

Η εσωτερική απεικόνιση μιας σταθεράς (αλλά και μιας μεταβλητής) τύπου char είναι ένας ακέραιος αριθμός. Ο αριθμός αυτός είναι ο κωδικός ASCII[11] του χαρακτήρα. Η παρακάτω πρόταση αποδεικνύει αυτό το γεγονός:

```
printf("%c %d\n",'A','A');
```

θα εμφανίσει το A και το 65 (ο κωδικός ASCII του A). Δηλαδή θα εμφανίσει τον χαρακτήρα 'A' ως χαρακτήρα (%c) αλλά και ως ακέραιο αριθμό (%d).

Η C χειρίζεται τους χαρακτήρες ως αριθμούς. Αυτό σημαίνει ότι μπορούν να μετέχουν σε αριθμητικές παραστάσεις.

Επομένως, οι ακόλουθες παραστάσεις δεν είναι καθόλου παράξενες:

5 + '*' ➔ επιστρέφει την τιμή 47 (5 + 42 που είναι ο κωδικός ASCII του '*').

[11] Κάθε χαρακτήρας μετατρέπεται σε έναν ακέραιο αριθμό σύμφωνα με ένα καθορισμένο πρότυπο μετατροπής. Το πρότυπο αυτό ονομάζεται ASCII και ο ομώνυμος πίνακας μετατροπής υπάρχει στο τέλος του παρόντος κεφαλαίου.

printf("%d %c\n','A'+1,'A'+1); ➔ εμφανίζει στην οθόνη το 66 και το B.

Το 66 εμφανίζεται διότι είναι το αποτέλεσμα της παράστασης 'A'+1 και η παράμετρος %d "λέει" στην printf() να το εμφανίσει ως ακέραιο αριθμό. Το B εμφανίζεται διότι η δεύτερη παράμετρος %c "λέει" στην printf() να εμφανίσει το αποτέλεσμα της παράστασης 'A'+1 (το 66) ως χαρακτήρα. Έτσι εμφανίζεται το B, το οποίο έχει κωδικό ASCII 66.

Χαρακτήρες διαφυγής

Οι χαρακτήρες διαφυγής (escape characters) αναφέρονται και ως σειρές διαφυγής (escape sequences). Ήδη, σε προηγούμενο κεφάλαιο, έχουμε συναντήσει μερικούς από τους χαρακτήρες διαφυγής, καθώς και τον τρόπο χρήσης τους μέσω της συνάρτησης printf(). Οι χαρακτήρες αυτοί δεν εμφανίζουν τίποτα στην οθόνη, αλλά ελέγχουν ορισμένες από τις λειτουργίες της. Οι χαρακτήρες διαφυγής της C είναι οι ακόλουθοι:

Ενδεικτικές σελίδες

\n new line (νέα γραμμή)

\b backspace (μία θέση πίσω)

\f form feed (νέα σελίδα)

\r carriage return (αρχή γραμμής)

\t tab (επόμενη στηλοθετημένη στήλη))

\\ backslash (ο χαρακτήρας "ανάποδη κάθετος", \)

\' τα απλά εισαγωγικά (')

\" τα διπλά εισαγωγικά (")

\0 Χαρακτήρας τερματισμού (null character - byte 00000000)

Σύμφωνα με τα παραπάνω, η επόμενη κλήση της συνάρτησης printf() έχει το αποτέλεσμα που εμφανίζεται στο διπλανό πλαίσιο:

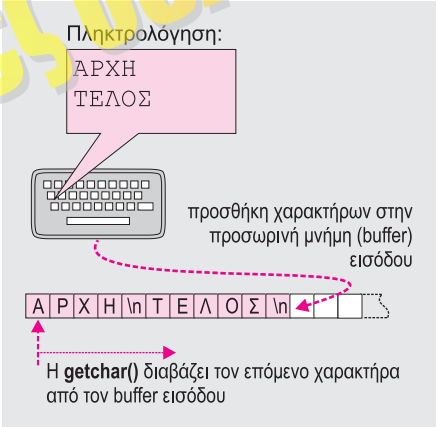
```
Οι
"χαρακτήρες διαφυγής"
είναι 9
```

printf("Οι\n\"χαρακτήρες διαφυγής\"\nείναι 9\n");

```
    printf("\nΠληκτρολόγησες το χαρακτήρα:");
    putchar(c);            // Εμφανίζει το χαρακτήρα στην οθόνη
    putchar('\n');         // Επιβάλλει αλλαγή γραμμής στην οθόνη
    printf("Πληκτρολόγησε ένα χαρακτήρα και πάτησε <enter>:");
    c = getchar();         // Περιμένει να πληκτρολογηθεί ένας χαρακτήρας ακολουθούμενος από <enter>
    printf("Πληκτρολόγησες το χαρακτήρα:%c\n",c);
    printf("Ο κωδικός ASCII του χαρακτήρα είναι:%d\n",c);
    printf("=========================================\n");
    return 0;
}
```

Μια πιο κοντική ματιά στη συνάρτηση getchar()

Για να κατανοήσουμε σε βάθος τη λειτουργία της συνάρτησης getchar(), πρέπει να αναλύσουμε τη διαδικασία με την οποία το πρόγραμμά μας διαβάζει χαρακτήρες από το πληκτρολόγιο.



Φανταστείτε ότι υπάρχει ένα χώρος προσωρινής αποθήκευσης (input buffer) στον οποίο καταχωρίζονται οι χαρακτήρες που πληκτρολογούνται. Οι χαρακτήρες προστίθενται στην προσωρινή αυτή μνήμη ανά γραμμή, δηλαδή μόλις πληκτρολογηθεί το πλήκτρο <Enter>. Στο παράδειγμα του διπλανού σχήματος, όταν ο χρήστης πληκτρολογήσει τη λέξη "ΑΡΧΗ" και πατήσει <Enter>, τότε οι χαρακτήρες της λέξης προστίθενται στον buffer εισόδου. Κατόπιν, όταν πληκτρολογήσει τη λέξη "ΤΕΛΟΣ" και <Enter>, τότε και οι χαρακτήρες αυτής της λέξης προστίθενται επίσης στον buffer εισόδου.

Η συνάρτηση getchar() διαβάζει έναν-έναν χαρακτήρες από την προσωρινή μνήμη εισόδου. Στην περίπτωση που η προσωρινή μνήμη εισόδου είναι κενή και η getchar() δεν βρίσκει κάποιον χαρακτήρα για να διαβάσει, τότε ενεργοποιείται η δυνατότητα πληκτρολόγησης και νέοι χαρακτήρες προστίθενται στην προσωρινή μνήμη, με την ίδια διαδικασία.

Ενδεικτικές σελίδες

Κάθε φορά που πληκτρολογείται ένα σύνολο χαρακτήρων ακολουθούμενο από το <Enter>, προστίθεται στην προσωρινή μνήμη εισόδου (input buffer). Ο χαρακτήρας <Enter> αντιστοιχεί στο χαρακτήρα νέας γραμμής '\n'.

Μελετήστε το παράδειγμα κώδικα που ακολουθεί:

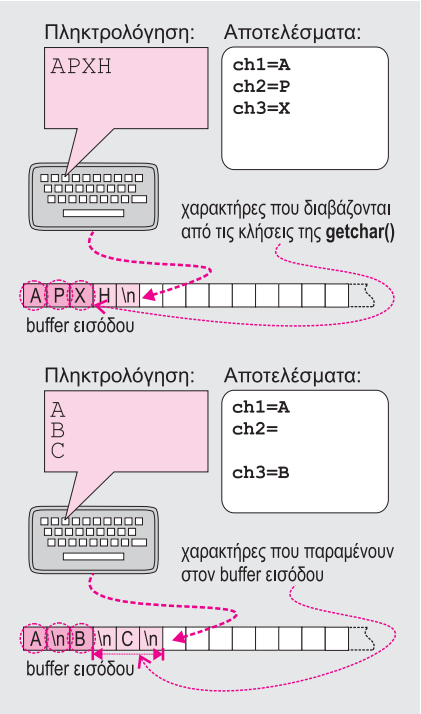
5_getchar1.c

```
#include <stdio.h>
int main(void)
{
    char ch1,ch2,ch3;
    ch1=getchar();
    ch2=getchar();
    ch3=getchar();
    printf("ch1=%c\nch2=%c\nch3=%c\n",ch1,ch2,ch3);
    return 0;
}
```

Ενδεικτικές σελίδες

Στο διπλανό σχήμα εμφανίζονται δύο περιπτώσεις πληκτρολόγησης, με τα αντίστοιχα αποτελέσματα. Ας υποθέσουμε πρώτα ότι ως είσοδο δίνουμε τη λέξη ΑΡΧΗ. Τότε οι τρεις κλήσεις της getchar() θα επιστρέψουν τους χαρακτήρες 'A', 'P' και 'X' οι οποίοι καταχωρίζονται στις μεταβλητές ch1, ch2, και ch3 αντίστοιχα. Η printf() εμφανίζει τα περιεχόμενα αυτών των μεταβλητών στην οθόνη. Οι χαρακτήρες 'H' και '\n' παραμένουν στον buffer εισόδου.

Τώρα ως είσοδο του προγράμματος ας πληκτρολογήσουμε τους χαρακτήρες 'A', 'B' και 'C' σε μία γραμμή τον καθένα . Οι χαρακτήρες που θα διαβάσει getchar() και θα καταχωριστούν στις μεταβλητές ch1, ch2, και ch3 είναι οι 'A', '\n', και 'B'. Οι χαρακτήρες '\n', 'C' και '\n' παραμένουν στον buffer εισόδου.



βολοσειράς είναι 35500 ενώ της άλλης 35508. Είναι φυσικό λοιπόν ότι η σύγκριση "ΝΙΚΟΣ"=="ΝΙΚΟΣ" (ουσιαστικά 35500==35508) θα είναι ψευδής[12].

☞ Για τη σύγκριση των συμβολοσειρών χρησιμοποιείται η συνάρτηση strcmp(), την οποία θα παρουσιάσουμε σε επόμενο κεφάλαιο, μαζί με άλλες συναρτήσεις που χειρίζονται συμβολοσειρές.

Πίνακας ASCII

Κάθε χαρακτήρας κωδικοποιείται σε έναν αντίστοιχο αριθμό. Η κωδικοποίηση αυτή ακολουθεί ορισμένα διεθνή πρότυπα (ASCII). Στον επόμενο πίνακα παρουσιάζεται το επεκταμένο σύνολο χαρακτήρων ASCII με τους αντίστοιχους κωδικούς τους.

Ενδεικτικές σελίδες

ASCII	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
128	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π
144	Ρ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	α	β	γ	δ	ε	ζ	η	θ
160	ι	κ	λ	μ	ν	ξ	ο	π	ρ	σ	ς	τ	υ	φ	χ	ψ
176	░	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
192	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
208	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█	▄	▌	▐	▀
224	ω	ά	έ	ή	ϊ	ί	ό	ύ	ϋ	ώ	Ά	Έ	Ή	Ί	Ό	Ύ
240	Ώ	±	≥	≤	Ϊ	Ϋ	÷	≈	°	·	·	√	n	2	■	

Ο κωδικός ενός χαρακτήρα προκύπτει από το άθροισμα του αριθμού της γραμμής και της στήλης που βρίσκεται. Έτσι, ο χαρακτήρας 'Γ' έχει κωδικό 130 (128+2). Η τυπική κωδικοποίηση ASCII αφορά στους 128 πρώτους χαρακτή-

[12] Ορισμένοι μεταγλωττιστές (μεταξύ των οποίων και ο GCC που περιλαμβάνεται στο DEV C++), όταν εντοπίσουν ίδιες συμβολοσειρές, τις αποθηκεύουν μόνο μια φορά στη μνήμη. Σε αυτούς τους μεταγλωττιστές το αποτέλεσμα θα είναι αληθές. Σε κάθε περίπτωση, όμως, η χρήση του τελεστή == συγκρίνει τις διευθύνσεις, και όχι τους χαρακτήρες των συμβολοσειρών.

ρες, με κωδικούς από το 0 έως το 127. Οι χαρακτήρες από το 128 μέχρι το 255 αποτελούν επέκταση του πρότυπου πίνακα χαρακτήρων, και διαφέρουν από σύστημα σε σύστημα.

Οι χαρακτήρες με κωδικούς από 0 μέχρι 31 δεν είναι εκτυπώσιμοι χαρακτήρες, και χρησιμοποιούνται ως χαρακτήρες ελέγχου. Χαρακτηριστικό παράδειγμα χαρακτήρων ελέγχου αποτελούν οι χαρακτήρες διαφυγής της C.

Παραδείγματα

Ενδεικτικές σελίδες

Π5.1 Το επόμενο πρόγραμμα διαβάζει και εμφανίζει ένα χαρακτήρα.

5_p5_1.c

```
#include <stdio.h>

int main(void)
{
    char c;
    printf("Πληκτρολογήστε ένα χαρακτήρα:");
    c = getch();
    putchar(c);        // Εμφανίζει το χαρακτήρα στην οθόνη
    putchar('\n');     // Επιβάλλει αλλαγή γραμμής στην οθόνη
    return 0;
}
```

Περιμένει να πληκτρολογηθεί ένας χαρακτήρας, τον οποίο καταχωρίζει στη μεταβλητή **c**.

Π5.2 Το επόμενο πρόγραμμα διαβάζει δύο χαρακτήρες και εμφανίζει στην οθόνη τον κωδικό ASCII τους.

5_p5_2.c

```
#include <stdio.h>

int main(void)
{
    char ch1,ch2;
    printf("Πληκτρολογήστε δύο χαρακτήρες και μετά <Enter>:");
    ch1 = getchar();
    ch2 = getchar();
    // Οι παρακάτω προτάσεις εμφανίζουν τον κάθε χαρακτήρα και τον αντίστοιχο
    // κωδικό ASCII του
    printf("ο ASCII του %c είναι %d\n",ch1,ch1);
    printf("ο ASCII του %c είναι %d\n",ch2,ch2);
    return 0;
}
```

Διαβάζει τον επόμενο χαρακτήρα από την προσωρινή μνήμη εισόδου (input buffer).

Π5.3 Το επόμενο πρόγραμμα περιμένει το πάτημα ενός χαρακτήρα και ελέγχει αν είναι λατινικός κεφαλαίος ή όχι.

5_p5_3.c

```
#include <stdio.h>

int main(void)
{
    char c;
    printf("Πληκτρολογήστε ένα χαρακτήρα:");
    c = getch();
    if ('A'<=c && c<='Z')
       printf("ο χαρακτήρας είναι κεφαλαίος\n");
    else
       printf("ο χαρακτήρας δεν είναι κεφαλαίος\n");
    return 0;
}
```

Ελέγχει αν ο χαρακτήρας **c** που πληκτρολογήθηκε βρίσκεται μεταξύ του 'A' και του 'Z'.

☞ Αν παρατηρήσουμε τον πίνακα των κωδικών ASCII, θα προσέξουμε ότι οι κεφαλαίοι και οι πεζοί χαρακτήρες είναι συνεχόμενοι. Αυτό όμως εξαρτάται από σύστημα σε σύστημα, και δεν είναι εξασφαλισμένο. Σε μια τέτοια περίπτωση, όμως, το παραπάνω πρόγραμμα δεν θα δούλευε σωστά. Η C διαθέτει συναρτήσεις που ελέγχουν αν ένας χαρακτήρας είναι πεζός ή κεφαλαίος, τις: islower() και isupper(), οι οποίες αναλύονται στο παράρτημα Α.

Ενδεικτικές σελίδες

Π5.4 Παρατηρήστε το αποτέλεσμα του επόμενου προγράμματος.

5_p5_4.c

```
#include <stdio.h>

int main(void)
{
    char ch,let;
    int a,b=4;
    ch='A';
    let='B';
    a=b+ch+let;
    ch++;
    printf("a=%d ch=%c\n",a,ch);
    return 0;
}
```

```
a=135 ch=B
```

Η μεταβλητή a θα πάρει την τιμή 135 (4+65+66), δεδομένου ότι ο κωδικός του 'A' (στη ch) είναι 65 και του 'B' (στη let) είναι 66. Με την παράσταση ch++ η τιμή του ch θα γίνει 66, οπότε στην printf() θα εμφανιστεί το 'B'.

Συχνά λάθη

- Το συχνότερο λάθος είναι ότι συγχέουμε χαρακτήρες με συμβολοσειρές. Για παράδειγμα, ο χαρακτήρας Α συμβολίζεται με 'Α' και όχι με "Α", το οποίο αποτελεί μια συμβολοσειρά.
- Χρησιμοποιούμε συμβολοσειρές εκεί όπου επιτρέπονται μόνο χαρακτήρες. Για παράδειγμα η πρόταση putchar("ΛΑΘΟΣ") είναι λάθος διότι ως παράμετρος της συνάρτησης putchar() επιτρέπεται χαρακτήρας ή αριθμός. Για παράδειγμα, putchar('Α') ή putchar(65) κ.λπ.

Ενδεικτικές σελίδες

Ανασκόπηση Κεφαλαίου 5

- Ο τύπος δεδομένων char αφορά χαρακτήρες. Οι μεταβλητές τύπου char δεσμεύουν 1 byte.
- Η εσωτερική απεικόνιση μιας σταθεράς (αλλά και μιας μεταβλητής) τύπου char είναι ένας ακέραιος αριθμός. Ο αριθμός αυτός είναι ο κωδικός ASCII του χαρακτήρα. Η C χειρίζεται τους χαρακτήρες ως **ακέραιους αριθμούς**. Μπορούν να μετέχουν σε αριθμητικές παραστάσεις, όπως και τα υπόλοιπα αριθμητικά μεγέθη.
- Εκτενώς χρησιμοποιούνται οι ειδικοί χαρακτήρες διαφυγής, όπως ο χαρακτήρας αλλαγής γραμμής '\n' και ο χαρακτήρας τερματισμού '\0'.
- Η getchar() διαβάζει τον επόμενο χαρακτήρα από την προσωρινή μνήμη εισόδου (input buffer). Η συνάρτηση getch() περιμένει να πληκτρολογηθεί ένας χαρακτήρας, αδειάζοντας προηγουμένως τον buffer εισόδου. Και οι δύο συναρτήσεις επιστρέφουν ως τιμή τον κωδικό του χαρακτήρα που διάβασαν.
- Η συνάρτηση putchar(ch) εμφανίζει το χαρακτήρα ch στην οθόνη.
- Οι συμβολοσειρές (ή *αλφαριθμητικά*) αποτελούνται από χαρακτήρες αποθηκευμένους σε συνεχόμενες θέσεις μνήμης, οι οποίες τερματίζουν με τον ειδικό χαρακτήρα '\0'. Η C "βλέπει" τις συμβολοσειρές ως **διευθύνσεις**! Η τιμή μιας συμβολοσειράς είναι η αρχική διεύθυνση της μνήμης όπου αποθηκεύθηκε το σύνολο χαρακτήρων, και όχι οι ίδιοι οι χαρακτήρες.
- Για το χειρισμό των συμβολοσειρών χρησιμοποιούνται ειδικές συναρτήσεις.

Ενδεικτικές σελίδες

Κεφάλαιο

6

Float, double, και λοιποί τύποι δεδομένων

Ενδεικτικές σελίδες

Οι τύποι δεδομένων float και double

Ο χειρισμός των δεδομένων κινητής υποδιαστολής (δηλαδή των δεκαδικών αριθμών) στη γλώσσα C γίνεται μέσω δύο βασικών τύπων της: του τύπου float και του τύπου double. Ο τύπος double χρησιμοποιείται όταν θέλουμε μεγαλύτερη ακρίβεια στα δεκαδικά ψηφία (double precision). Επίσης, όπως θα δούμε στη συνέχεια, η εσωτερική απεικόνιση των δύο τύπων είναι διαφορετική, με τον τύπο double να καταλαμβάνει διπλάσιο χώρο στη μνήμη του Η/Υ.

Σταθερές τύπου float και double

Εξ ορισμού, στη γλώσσα C μια σταθερά κινητής υποδιαστολής (δηλαδή ένας δεκαδικός αριθμός) μετατρέπεται σε double. Για να θεωρηθεί τύπου double, ένας αριθμός πρέπει απαραιτήτως να περιέχει το χαρακτήρα της υποδιαστολής (.). Έτσι οι παρακάτω αριθμοί:

123.45, 123.0, -123., 0.1

είναι παραδείγματα σταθερών τύπου double. Όμως, μια σταθερά κινητής υποδιαστολής, με την κατάλληλη κατάληξη (suffix), μπορεί να προσδιορίσει ρητά τον τύπο της. Για παράδειγμα, η σταθερά 123.4F αναφέρεται στο δεκαδικό αριθμό 123.4 τύπου float, και όχι double.

Ενδεικτικές σελίδες

Μεταβλητές τύπου float και double

Υπάρχουν δύο βασικοί τύποι μεταβλητών κινητής υποδιαστολής (floating point), ο τύπος **float** και ο τύπος **double**. Οι μεταβλητές τύπου float καταλαμβάνουν 4 byte, ενώ οι τύπου double 8. Οι μεταβλητές τύπου double έχουν μεγαλύτερη ακρίβεια στα δεκαδικά ψηφία.

Οι μεταβλητές τύπου **float** δηλώνονται με τον ακόλουθο τρόπο:

float *όνομα_μεταβλητής1*, *όνομα_μεταβλητής2*, ...;

double *όνομα_μεταβλητής1*, *όνομα_μεταβλητής2*, ...;

long float *όνομα_μεταβλητής1*, *όνομα_μεταβλητής2*, ...;

ο τύπος **long float** είναι ισοδύναμος με τον **double**.

Χρήση δεδομένων τύπου κινητής υποδιαστολής

Οι διαθέσιμοι τελεστές για πράξεις μεταξύ δεδομένων τύπου κινητής υποδιαστολής (floating point) είναι πολύ περιορισμένοι, σε σχέση με τούς τελεστές που εφαρμόζονται σε δεδομένα τύπου int και char. Οι τελεστές που εφαρμόζονται σε δεδομένα τύπου float (ή double) αναφέρονται στο διπλανό πίνακα.

Τελεστής	Λειτουργία
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
++	Μοναδιαία αύξηση
--	Μοναδιαία μείωση

☞ Ο αριθμητικός τελεστής %, δεν εφαρμόζεται σε δεδομένα κινητής υποδιαστολής, παρά μόνο σε ακέραια.

☞ Επίσης, στα αρχικά πρότυπα της γλώσσας, η εφαρμογή των τελεστών ++ και -- επιτρεπόταν μόνο σε μεταβλητές ακέραιου τύπου (int και char). Μερικοί μεταγλωττιστές της C έχουν ακόμα αυτόν τον περιορισμό.

Ενδεικτικές σελίδες

Η χρήση της printf() με δεδομένα κινητής υποδιαστολής

Ας θεωρήσουμε το επόμενο πρόγραμμα, του οποίου το αποτέλεσμα φαίνεται στο διπλανό πλαίσιο:

```
#include <stdio.h>
int main(void)
{
    float a,b;
    a=21.234;
    b=5467.1;
    printf("%f\n",a);
    printf("%f\n",b);
    return 0;
}
```

```
21.234
5467.1
```

Παρατηρούμε ότι, αν και τα αποτελέσματα εμφανίζονται σωστά, μπορεί να προτιμούσαμε τις χιλιάδες κάτω από τις χιλιάδες, τις εκατοντάδες κάτω από τις εκατοντάδες, κ.ο.κ. Για να επιτύχουμε μορφοποιημένη (φορμαρισμένη) έξοδο

```
        printf("Οι αριθμοί δόθηκαν με τη σειρά\n");
    else
        printf("Λάθος σειρά\n");
    return 0;
}
```

Όπως αναφέρθηκε, ο τύπος bool είναι ένας ακέραιος τύπος δεδομένων αλλά με μόνες επιτρεπτές τιμές το 1 (true) και το 0 (false). Τι γίνεται, όμως, στην περίπτωση που προσπαθήσουμε να καταχωρίσουμε σε μεταβλητές τύπου bool άλλες ακέραιες τιμές; Για παράδειγμα, τι θα περιέχουν οι μεταβλητές a και b μετά του τέλους του παρακάτω τμήματος κώδικα;

```
bool a,b;
a=10;
b=-20;
printf("a=%d b=%d\n",a,b);
```

```
a=1 b=1
```

Και στις δύο μεταβλητές θα καταχωριστεί το 1! Αυτό γίνεται επειδή τόσο η παράσταση 10 όσο και η -20 θεωρούνται αληθείς (true), δεδομένου ότι έχουν τιμή διάφορη του μηδενός.

ΕΥΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

Παραδείγματα

Π6.1 Το επόμενο πρόγραμμα ζητάει δύο δεκαδικούς διπλής ακρίβειας. Υπολογίζει και εμφανίζει το άθροισμά τους, και μετά εμφανίζει τους αριθμούς στην οθόνη, σε 10 θέσεις με 4 δεκαδικά ψηφία για τον καθένα.

6_p6_1.c

```
#include <stdio.h>
int main(void)
{
    double a,b,c;
    scanf("%lf %lf",&a,&b);
    c=a+b;
    printf("%10.4f\n",a);
    printf("%10.4f\n",b);
    printf("==========\n");
    printf("%10.4f\n",c);
    return 0;
}
```

```
   15.2143
  345.1234
==========
  360.3377
```

☞ Η μορφοποιημένη έξοδος στην οθόνη μπορεί να χρησιμοποιηθεί με ανάλογο τρόπο και σε δεδομένα ακέραιου τύπου. Για παράδειγμα, η πρόταση printf("%5d",10) θα εμφανίσει τον ακέραιο αριθμό 10 σε χώρο 5 θέσεων της οθόνης.

Π6.2 Το επόμενο πρόγραμμα ζητάει ένα δεκαδικό αριθμό. Υπολογίζει και εμφανίζει χωριστά το ακέραιο και το δεκαδικό του τμήμα.

6_p6_2.c

```
#include <stdio.h>
int main(void)
{
    double a,d;
    int i;
    scanf("%lf",&a);
    i=a;
    d=a-i;
    printf("%f\n",d);
    printf("%d\n",i);
    return 0;
}
```

Δεδομένου ότι η μεταβλητή **i** είναι τύπου **int**, με αυτή την πρόταση καταχωρίζεται στην **i** το ακέραιο μέρος της **a**.

Αν από την **a** αφαιρέσουμε το ακέραιο τμήμα της (την i), το αποτέλεσμα θα είναι το δεκαδικό τμήμα του αριθμού.

Ενδεικτικές Λύσεις

Π6.3 Ας μελετήσουμε τα αποτελέσματα του παρακάτω προγράμματος, και ας προσπαθήσουμε να τα εξηγήσουμε.

6_p6_3.c

```
#include <stdio.h>
int main(void)
{
    double a,b,c;
    a=50/100*10;
    b=50/100*10.0;
    c=50.0/100*10;
    printf("a=%f\nb=%f\nc=%f\n",a,b,c);
    return 0;
}
```

```
a=0.0000
b=0.0000
c=5.0000
```

☞ Και στις τρεις προτάσεις ανάθεσης στις μεταβλητές a,b, και c, οι τελεστές / και * έχουν την ίδια προτεραιότητα, επομένως οι πράξεις θα εκτελεστούν από τα αριστερά προς τα δεξιά.

☞ Στην πρώτη πρόταση, η πράξη 50/100 έχει αποτέλεσμα 0, επειδή και τα δύο μέλη είναι ακέραιοι αριθμοί οπότε και το αποτέλεσμα θα είναι ακέραιος αριθμός. Η επόμενη πράξη 0*10 θα έχει αποτέλεσμα επίσης 0.

Ενδεικτικές σελίδες

Κεφάλαιο

7

Εντολές συνθήκης

Ενδεικτικές σελίδες

Εντολές συνθήκης

Η C διαθέτει δύο εντολές με τις οποίες έχει τη δυνατότητα εκτέλεσης προτάσεων υπό συνθήκη ("υπό όρους"). Είναι η εντολή **if** και η εντολή **switch-case**.

Η εντολή if σε μια δεύτερη ματιά

Ήδη έχουμε γνωρίσει την εντολή if, η οποία ελέγχει μια παράσταση και ανάλογα με το αποτέλεσμα του ελέγχου (αληθές ή ψευδές) εκτελεί (ή δεν εκτελεί) κάποιες προτάσεις. Η εντολή if συντάσσεται με τρεις τρόπους:

- ως απλή πρόταση if
- ως συνδυασμός if - else
- ως συνδυασμός if - else if

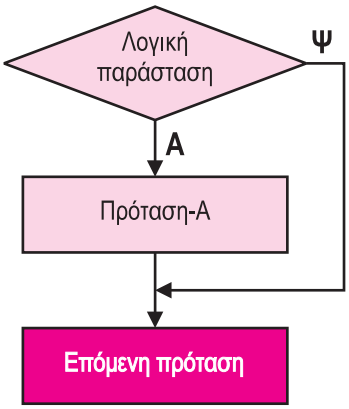
Ενδεικτικές σελίδες

Η απλή πρόταση if

Το συντακτικό της if στην πιο απλή της μορφή είναι:

if (*λογική παράσταση*) *πρόταση-Α*;

Σε αυτή τη μορφή η if ελέγχει τη λογική παράσταση και, αν είναι αληθής (**Α**), εκτελεί την ***πρόταση-Α*** που ακολουθεί. Αν η λογική παράσταση είναι ψευδής (**Ψ**), εκτελείται η πρόταση αμέσως μετά από την εντολή if.



Ας θυμηθούμε ότι μια λογική παράσταση στη C επιστρέφει μία αριθμητική τιμή: την τιμή 1 αν είναι αληθής, και την 0 αν είναι ψευδής. Στο λογικό διάγραμμα φαίνεται η λογική τής πρότασης if. Η ***πρόταση-Α*** μπορεί να είναι απλή ή σύνθετη πρόταση (compound statement).

Η C θεωρεί αληθή οποιαδήποτε παράσταση επιστρέφει τιμή διάφορη του μηδενός. Έτσι, στη θέση της λογικής παράστασης μπορεί να τοποθετηθεί **οποιαδήποτε παράσταση** που αποδίδει αριθμητική τιμή (δείτε το παράδειγμα Π7.3).

Παράδειγμα χρήσης της απλής πρότασης if:

```
if (a==5) printf("a=5\n");
```

η οποία μπορεί να γραφεί και

```
if (a==5)
    printf("a=5\n");
```

ή και

```
if (a==5)
{
    printf("a=5\n");
}
```

Όπως αναφέραμε, η ***πρόταση-Α*** μπορεί να είναι και μια σύνθετη πρόταση:

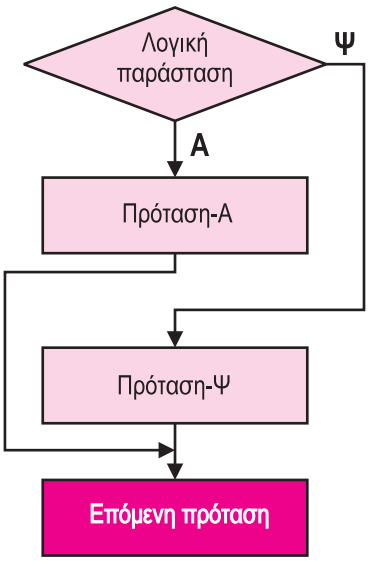
```
if (a==5)
{
    printf("a=5\n");
    printf("μία σύνθετη πρόταση\n");
}
```

Ενδεικτικές σελίδες

Η πρόταση if-else

if (*λογική παράσταση*) *Πρόταση-Α*; else *Πρόταση-Ψ*;

Σε αυτή τη μορφή, η if ελέγχει τη λογική παράσταση και, αν είναι αληθής (**Α**), εκτελεί την ***Πρόταση-Α*** που ακολουθεί. Αν η λογική παράσταση είναι ψευδής (**Ψ**), εκτελείται η ***Πρόταση-Ψ***. Μετά εκτελείται κανονικά η επόμενη από την if πρόταση του προγράμματος. Στο διπλανό λογικό διάγραμμα φαίνεται η λογική της πρότασης if-else.



Οι προτάσεις Α και Ψ μπορεί να είναι σύνθετες προτάσεις.

Παράδειγμα χρήσης της if-else:

```
if (a==5) printf("Ναι"); else printf("Όχι");
```

ή και

Π7.2 Το επόμενο πρόγραμμα ζητάει δύο αριθμούς και ένα χαρακτήρα. Αν ο χαρακτήρας είναι ο '+' υπολογίζει το άθροισμα των δύο αριθμών, αν είναι '-' ή '_' τη διαφορά τους, αν είναι '*', 'X' ή 'x' το γινόμενό τους, και αν είναι '/' ή '\' το πηλίκο τους.

Αν ο χαρακτήρας είναι οποιοσδήποτε άλλος, εμφανίζει το μήνυμα "Λάθος πράξη".

7_p7_2.c

```
#include <stdio.h>

int main(void)
{
   float a,b;
   char ch;
   printf("Δώσε δύο αριθμούς και ένα χαρακτήρα:");
   scanf("%f %f %c",&a,&b,&ch);
   switch (ch)
   {
      case '+':
         printf("Το άθροισμα είναι %f\n",a+b);
         break;
      case '-':
      case '_':
         printf("Η διαφορά είναι %f\n",a-b);
         break;
      case '*':
      case 'X':
      case 'x':
         printf("Το γινόμενο είναι %f\n",a*b);
         break;
      case '\':
      case '/':
         printf("Το πηλίκο είναι %f\n",a/b);
         break;
      default:
         printf("Λάθος πράξη\n");
   }
   return 0;
}
```

Πρόσθεση θα γίνει στην περίπτωση που ο χαρακτήρας **ch** είναι '+'.

Αφαίρεση θα γίνει στην περίπτωση που ο χαρακτήρας **ch** είναι '-' ή '_'.

Πολλαπλασιασμός θα γίνει στην περίπτωση που ο χαρακτήρας **ch** είναι '*', 'X', ή 'x'.

Διαίρεση θα γίνει στην περίπτωση που ο χαρακτήρας **ch** είναι '/' ή '\'.

Σε κάθε άλλη περίπτωση θα εμφανίσει μήνυμα λάθους.

☞ Οι συνεχόμενες προτάσεις case, έχουν αποτέλεσμα να εκτελεστεί ο κώδικας που ακολουθεί στην περίπτωση που επαληθεύεται οποιαδήποτε από αυτές.

Ενδεικτικές σελίδες

Π7.3 Στη C, στη θέση οποιασδήποτε λογικής παράστασης, μπορεί να χρησιμοποιηθεί οποιαδήποτε παράσταση που αποδίδει αριθμητική τιμή. Αν η παράσταση αποδίδει τιμή 0 θεωρείται ψευδής, διαφορετικά θεωρείται αληθής. Το επόμενο πρόγραμμα επιδεικνύει αυτή τη δυνατότητα.

7_p7_3.c

```
#include <stdio.h>

int main(void)
{
   int a=5,b=5,c;
   // Ελέγχεται η λογική παράσταση a==b
   if (a==b)
      printf("ίσα\n");
   else
      printf("άνισα\n");
   // Ελέγχεται η αριθμητική παράσταση a-b
   if (a-b)
      printf("άνισα\n");
   else
      printf("ίσα\n");
   // Ελέγχεται η αριθμητική παράσταση a=b
   if (c=a)
      printf("ΝΑΙ\n");
   else
      printf("ΟΧΙ\n");
   // Ελέγχεται η παράσταση printf("Μυτιλήνη\n");
   if (printf("Μυτιλήνη\n"))
      printf("ΝΑΙ\n");
   else
      printf("ΟΧΙ\n");
   // Ελέγχεται η παράσταση b;
   if (b)
      printf("Τέλος\n");
   else
      printf("Αρχή\n");
   return 0;
}
```

Κανονική λογική παράσταση. Το αποτέλεσμά της είναι αληθές (5==5), οπότε εμφανίζεται η λέξη "*ίσα*".

Αριθμητική παράσταση. Το αποτέλεσμα της είναι 0 (5-5) οπότε θεωρείται ψευδής. Επομένως εμφανίζεται πάλι η λέξη "*ίσα*".

Η παράσταση **c=a** καταχωρίζει το 5 στη μεταβλητή **c** και έχει αποτέλεσμα 5, οπότε θεωρείται αληθής. Εμφανίζεται η λέξη "*ΝΑΙ*".

Η παράσταση **printf("Μυτιλήνη\n")** εμφανίζει τη λέξη "*Μυτιλήνη*" και επιστρέφει την τιμή 8 (το πλήθος χαρακτήρων που εμφάνισε), οπότε θεωρείται αληθής. Επομένως εμφανίζεται η λέξη "*ΝΑΙ*".

Η παράσταση **b** θεωρείται αληθής διότι η τιμή της μεταβλητής **b** είναι διάφορη του μηδενός (5). Επομένως θα εμφανίσει τη λέξη "*Τέλος*".

```
ίσα
ίσα
ΝΑΙ
Μυτιλήνη
ΝΑΙ
Τέλος
```

☞ Στο διπλανό πλαίσιο εμφανίζονται τα αποτελέσματα του προγράμματος.

☞ Ο έλεγχος μη λογικών παραστάσεων είναι συνήθης πρακτική σε προγράμματα της C.

ΕΥΣΕΛΙΤΙΚΕΣ ΘΕΛΤΟΣ

Ενδεικτικές σελίδες

Κεφάλαιο

8

Εντολές αλλαγής ροής και επανάληψης

Ενδεικτικές σελίδες

Εντολές αλλαγής ροής και επανάληψης

Η "επικίνδυνη" εντολή goto

Στη C, όπως και στις περισσότερες γλώσσες προγραμματισμού, με την εντολή **goto** μεταφέρουμε τον έλεγχο του προγράμματος σε ένα διαφορετικό του σημείο. Δεδομένου ότι στη C οι προτάσεις ενός προγράμματος δεν αριθμούνται, χρησιμοποιούνται **ετικέτες** (labels) για να "σηματοδοτήσουν" τα διάφορα σημεία μέσα σε ένα πρόγραμμα.

Η **ετικέτα** (label) είναι ένα αναγνωριστικό ακολουθούμενο από μια άνω και κάτω τελεία (:), για παράδειγμα:

```
telos:
```

Η ετικέτα μπορεί να βρίσκεται μόνη της σε μια γραμμή του προγράμματος, ή να βρίσκεται μπροστά από μια πρόταση.

> Η εντολή **goto** μεταφέρει τον έλεγχο σε μια ετικέτα του προγράμματος (αλλά πάντα μέσα στην ίδια συνάρτηση).

Το συντακτικό της goto είναι:

goto *ετικέτα*;

όπου ***ετικέτα*** είναι η ετικέτα στην οποία θέλουμε να μεταφερθεί ο έλεγχος του προγράμματος. Το επόμενο πρόγραμμα εμφανίζει συνέχεια αριθμούς ξεκινώντας από το 1 και αυξάνοντάς τους κατά 1.

8_goto.c

```
#include <stdio.h>
int main(void)
{
    int a;
    a=0;
    pali:
    printf("%d\n",++a);   // Αυξάνει την τιμή της a κατά 1 και την εμφανίζει
    goto pali;            // Μεταφέρει τον έλεγχο του προγράμματος στη θέση της ετικέτας pali
    return 0;
}
```

Η μεταφορά ελέγχου προς τα πίσω έχει αποτέλεσμα να επαναλαμβάνονται συνεχώς οι προτάσεις μεταξύ της εντολής **goto** και της ετικέτας **pali**.

```
1
2
3
4
```

Ενδεικτικές σελίδες

☞ Παρατηρούμε ότι η ετικέτα pali δεν ακολουθείται από ερωτηματικό (;). Όταν η ετικέτα βρίσκεται πριν από μία πρόταση, η πρόταση τερματίζεται κανονικά με ερωτηματικό (;). Στη σύνταξη της goto, η ετικέτα δεν ακολουθείται από την άνω και κάτω τελεία (goto pali;)

Το επόμενο πρόγραμμα εκτελεί την ίδια λειτουργία με το προηγούμενο:

```
#include <stdio.h>
int main(void)
{
    int a=0;
    pali: printf("%d\n",++a);
    goto pali;
    return 0;
}
```

Η goto ως μέθοδος μεταφοράς ελέγχου του προγράμματος, ή ως μέθοδος επαναληπτικής λειτουργίας, είναι ο **χειρότερος** εχθρός του δομημένου προγραμματισμού. Αφού λοιπόν έγινε κατανοητή η χρήση της goto, το επόμενο στάδιο είναι να την **ξεχάσουμε**. Αν θέλουμε τα προγράμματά μας να είναι δομημένα και ευανάγνωστα, δεν πρέπει να τη χρησιμοποιούμε **ποτέ**. Η C διαθέτει εντολές ελέγχου και επαναληπτικές δομές που κάνουν περιττή τη χρήση της goto.

Ενδεικτικές σελίδες

Ο βρόχος while

Ο βρόχος (loop) **while** αποτελεί τη μία από τις τρεις δομές επανάληψης της C. Με τη χρήση της εντολής **while** είναι δυνατή η επαναληπτική εκτέλεση μίας ή παραπάνω προτάσεων. Η επανάληψη ή όχι των προτάσεων εξαρτάται από την τιμή μιας παράστασης (συνήθως λογικής). Για παράδειγμα, ο παρακάτω βρόχος **while** εκτελείται όσο η τιμή της μεταβλητής a είναι μεγαλύτερη από το 0.

```
a=5;
while (a>0)
{
    printf("a=%d\n",a);
    --a;
}
```

Ελέγχεται αν η τιμή της μεταβλητής **a** είναι μεγαλύτερη από 0. Αν είναι, συνεχίζεται η εκτέλεση των εντολών του βρόχου.

Ο βρόχος for

Στους βρόχους while και do-while δεν είναι καθορισμένο το πλήθος των επαναλήψεων. Ο βρόχος **for** είναι ο τρόπος με τον οποίο επιτυγχάνεται η επαναληπτική εκτέλεση προτάσεων για συγκεκριμένο αριθμό φορών.

Η κλασική χρήση του βρόχου **for** προϋποθέτει τη χρήση μιας μεταβλητής από την τιμή της οποίας εξαρτάται η επαναληπτική εκτέλεση του βρόχου. Η μεταβλητή αυτή παίρνει μια αρχική τιμή και μετά από κάθε εκτέλεση του βρόχου αυξάνεται ή μειώνεται (συνήθως κατά 1). Η επαναληπτική διαδικασία σταματάει όταν η τιμή της μεταβλητής ξεπεράσει τη μέγιστη (ή την ελάχιστη) τιμή που έχει οριστεί.

Στο επόμενα παράδειγμα χρησιμοποιείται η μεταβλητή i ως μεταβλητή του βρόχου for. Η πρόταση του βρόχου θα εκτελεστεί 8 φορές, έτσι θα εμφανιστούν στην οθόνη 8 γραμμές με το κείμενο "Η γλώσσα C σε βάθος".

```
int i;
for (i=1; i<=8 ; i=i+1)
{
    printf("Η γλώσσα C σε βάθος\n");
}
```

Αρχική τιμή μεταβλητής (1).

Μέγιστη τιμή μεταβλητής (8).

Η πρόταση θα εκτελεστεί 8 φορές, μία για κάθε τιμή της μεταβλητής **i**.

Ενδεικτικές σελίδες

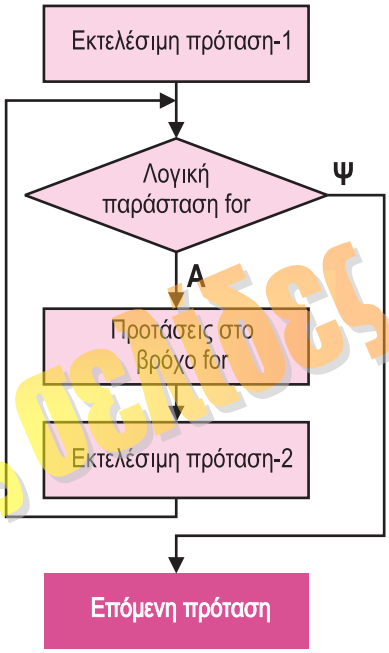
Η εντολή **for** είναι ίσως η εντολή που χρησιμοποιείται πιο συχνά για επαναληπτικές διαδικασίες. Ενώ σε άλλες γλώσσες προγραμματισμού η **for** έχει ένα "αυστηρό" συντακτικό, στη C το συντακτικό της, αν και σαφώς καθορισμένο, της προσφέρει μια μοναδική ευελιξία. Το συντακτικό της εντολής **for** είναι:

for (*εκτελέσιμη-πρόταση1*;*λογική_παράσταση*;*εκτελέσιμη-πρόταση2*)
** *πρόταση-βρόχου*;**

ή, για σύνθετη πρόταση βρόχου,

for (*εκτελέσιμη-πρόταση1*;*λογική_παράσταση*;*εκτελέσιμη-πρόταση2*)
{
** *πρόταση*;**
** *πρόταση*;**
** .....**
}

Η φιλοσοφία λειτουργίας της for γίνεται κατανοητή αν μελετήσουμε το διπλανό διάγραμμα ροής. Παρατηρούμε ότι η ***εκτελέσιμη πρόταση-1*** εκτελείται **μία μόνο** φορά, στην αρχή του βρόχου.



Μετά ελέγχεται η λογική παράσταση. Αν είναι αληθής, εκτελούνται μία φορά οι προτάσεις του βρόχου for και αμέσως μετά η ***εκτελέσιμη πρόταση-2***. Μετά ελέγχεται πάλι η λογική παράσταση, κ.ο.κ.

Η επαναληπτική εκτέλεση των προτάσεων του βρόχου θα σταματήσει όταν η λογική παράσταση γίνει ψευδής, οπότε ο έλεγχος του προγράμματος μεταφέρεται στην πρόταση του προγράμματος αμέσως μετά από τη for.

☞ Προσοχή πρέπει να δοθεί στο γεγονός ότι η εκτελέσιμη πρόταση-1, η λογική παράσταση, και η εκτελέσιμη πρόταση-2 βρίσκονται μέσα σε παρενθέσεις και χωρίζονται με ερωτηματικά (;)

Ενδεικτικές σελίδες

Το επόμενο πρόγραμμα εμφανίζει τους αριθμούς από το 1 μέχρι το 8:

8_for1.c

```
#include <stdio.h>
int main(void)
{
    int i;
    for (i=1; i<=8 ; i++)
       printf("i=%d\n",i);
    return 0;
}
```

```
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
```

- Στην αρχή, το i παίρνει τιμή 1 (i=1).
- Το πρόγραμμα ελέγχει αν i<=8.
- **Αν ναι** (είναι i<=8), εκτελεί τις εντολές του βρόχου μία φορά και μετά εκτελεί την πρόταση i++ αυξάνοντας το i κατά 1.

Ο βρόχος που ακολουθεί απλώς περιμένει να πατηθεί ο χαρακτήρας 'A':

```
for (ch='*';ch!='A';ch=getch());
```

☞ Ούτε εδώ υπάρχουν προτάσεις στο σώμα του βρόχου.

Ο επόμενος βρόχος θα εμφανίσει στην οθόνη τους αριθμούς από το 1 μέχρι το 1000:

```
for (a=1;a=<1000;printf("%d\n",a++));
```

☞ Και πάλι δεν υπάρχουν προτάσεις στο σώμα του βρόχου.

Μετά το τέλος ενός βρόχου for, η μεταβλητή του βρόχου έχει τιμή την τελευταία της τιμή, για την οποία **δεν εκτελέστηκε** ο βρόχος. Για παράδειγμα:

```
for (i=1;i<=20;i=i+6)
{
    προτάσεις βρόχου;
}
printf("i=%d\n",i);
```

Μόλις τελειώσει ο βρόχος, το **i** θα έχει την τιμή 25 που είναι η επόμενη τιμή του **i** μετά το 19 για την οποία **δεν** έγινε η επανάληψη του βρόχου. Ο βρόχος επαναλαμβάνεται για τις τιμές του i: 1, 7, 13, και 19.

Ενδεικτικές σελίδες

Χορεύοντας με τη for!

Για να κατανοήσουμε πλήρως τον τρόπο λειτουργίας της for, αλλά και για να τον θυμόμαστε πιο εύκολα, θα προσομοιάσουμε κάθε πρόταση της με τα βήματα ενός χορού. Στον παρακάτω βρόχο for έχουμε αριθμήσει τις προτάσεις του βρόχου από το 1 μέχρι το 4, με τη σειρά με την οποία εκτελούνται. Με 5 έχουμε αριθμήσει την αμέσως επόμενη πρόταση μετά από το βρόχο.

```
for (❶;❷;❹)
{
    ❸;
}
❺
```

❶
❷
❸ ❹
❷
❸ ❹
.....
❷
❺

Εάν λοιπόν χορεύαμε στο ρυθμό της for, τα βήματα θα ήταν: μία φορά το ❶ και μετά συνεχώς θα επαναλαμβάναμε τα βήματα ❷-❸-❹ μέχρι στο βήμα ❷ η παράσταση να μην είναι αληθής. Τότε τελειώνει ο χορός και η επαναληπτική διαδικασία! Μετά το τέλος της επαναληπτικής διαδικασίας εκτελείται η αμέσως επόμενη πρόταση, μετά από το βρόχο, δηλαδή η πρόταση ❺.

Απλά παραδείγματα

Το επόμενο τμήμα κώδικα εμφανίζει τους αριθμούς από το 5 μέχρι το 0:

```
for (i=5;i>=0;i--)
    printf("%d\n",i);
```

Το επόμενο τμήμα κώδικα εμφανίζει τα πολλαπλάσια του 5 στους αριθμούς από το 5 μέχρι το 100:

```
for (i=5;i<=100;i=i+5)
    printf("%d\n",i);
```

Το επόμενο τμήμα κώδικα εμφανίζει όλους τους λατινικούς χαρακτήρες από το 'a' έως το 'z' και δίπλα από κάθε χαρακτήρα τον κωδικό ASCII του:

```
char ch;
for (ch='a';ch<='z';ch++)
{
    printf("%c %d\n",ch,ch);
    printf("-----\n");
}
```

```
a 97
-----
b 98
-----
c 99
-----
...
```

Το επόμενο τμήμα κώδικα εμφανίζει όλους τους αριθμούς από το 1 έως το 1000 και αμέσως μετά εμφανίζει το άθροισμά τους (1+2+3+...+1000):

```
int i, s=0;
for (i=1;i<=1000;i++)
{
    printf("%d\n",i);
    s=s+i;
}
printf("%d\n",s);
```

```
1
2
3
...
1000
500500
```

Το επόμενο τμήμα κώδικα εμφανίζει μόνο τη λέξη ΑΡΧΗ. Το σώμα του βρόχου δεν θα εκτελεστεί καμία φορά, επειδή η πρόταση a!=b είναι ψευδής από την αρχή. Θα γίνει πιο κατανοητό αν 'χορέψετε' μαζί της.

```
int a=5,b=5;
for (printf("ΑΡΧΗ");a!=b;a++)
    printf("ΤΕΛΟΣ\n");
```

```
ΑΡΧΗ
```

Ενδεικτικές σελίδες

Και εδώ αναφέρεται ένα ισοδύναμο παράδειγμα που χρησιμοποιεί την εντολή goto για την επανάληψη του βρόχου.

Καταμέτρηση και άθροιση σε επαναλαμβανόμενες διαδικασίες

Πολλές φορές στα προγράμματά μας θα χρειαστεί να μετρήσουμε πόσες φορές συνέβη ένα γεγονός (για παράδειγμα, το πλήθος των αριθμών που είναι μεγαλύτεροι από 10), αλλά και να αθροίσουμε τις διαφορετικές τιμές που λαμβάνει μια μεταβλητή. Για την καταμέτρηση ενός συμβάντος χρησιμοποιούμε πάντα μια μεταβλητή στην οποία δίνουμε αρχική τιμή 0, και κάθε φορά που δημιουργείται το συμβάν αυξάνουμε τη μεταβλητή κατά 1. Για το συνολικό άθροισμα διαφορετικών τιμών χρησιμοποιούμε πάλι μια μεταβλητή, στην οποία δίνουμε επίσης αρχική τιμή 0, και κάθε φορά που δημιουργείται μια νέα τιμή αυξάνουμε τη μεταβλητή κατά την τιμή αυτή.

Το επόμενο πρόγραμμα διαβάζει έναν-έναν δώδεκα ακέραιους αριθμούς που δίνονται από το πληκτρολόγιο και εμφανίζει το άθροισμά τους. Για την καταμέτρηση των αριθμών χρησιμοποιείται η μεταβλητή cnt, και για το άθροισμά τους χρησιμοποιείται η μεταβλητή sum. Στον κώδικα του προγράμματος χρησιμοποιείται η συνάρτηση scanf(), η οποία διαβάζει έναν αριθμό από το πληκτρολόγιο και τον καταχωρίζει στη μεταβλητή a.

8_sum.c

```
#include <stdio.h>
int main(void)
{
    int a,cnt=0,sum=0;
    while (cnt<12)
    {
        printf("Δώσε αριθμό:");
        scanf("%d",&a);
        cnt++;
        sum=sum+a;
    }
    printf("Αθροισμα=%d\n",sum);
    return 0;
}
```

- Θέτουμε αρχική τιμή στις μεταβλητές **cnt** και **sum** το 0.
- Όταν δοθούν 12 αριθμοί, η επαναληπτική διαδικασία σταματάει.
- Διαβάζει έναν αριθμό από το πληκτρολόγιο και τον καταχωρίζει στη μεταβλητή **a**.
- Κάθε φορά που δίνεται ένας αριθμός (a), η **cnt** αυξάνεται κατά 1. Επίσης ο αριθμός (a) προστίθεται στη **sum**.
- Εμφανίζει το συνολικό άθροισμα όλων των αριθμών που δόθηκαν.

Ενδεικτικές σελίδες

Ας υποθέσουμε ότι δίνουμε με τη σειρά τους αριθμούς 3, 4, 78, 9 ... όπως φαίνεται στην πρώτη στήλη του διπλανού πίνακα.

Κάθε φορά που εκτελείται η πρόταση **cnt++**, αυξάνεται η τιμή της μεταβλητής cnt κατά 1. Έτσι, την πρώτη φορά που θα διαβάσει τον πρώτο αριθμό η cnt θα γίνει 1, όταν διαβάσει το 4 θα γίνει 2, με το 78 θα γίνει 3, κ.ο.κ. Όταν διαβάσει τον τελευταίο αριθμό, η cnt θα γίνει 12.

a	cnt	sum
3	1	3
4	2	7
78	3	85
9	4	94
12	5	106
34	6	140
45	7	185
6	8	191
78	9	269
99	10	278
2	11	290
3	12	293

Με λίγα λόγια, χρησιμοποιήσαμε τη μεταβλητή cnt για να μετρήσουμε πόσους αριθμούς διαβάσαμε μέχρι στιγμής.

Με την πρόταση **sum=sum+a**, κάθε φορά που διαβάζει το πρόγραμμα έναν αριθμό, τον προσθέτει στο μέχρι στιγμής περιεχόμενο της sum. Την πρώτη φορά (όταν η a περιέχει το 3) η sum θα γίνει 3, όταν το πρόγραμμα διαβάσει το 4 η sum θα γίνει 7 (3 που είχε από πριν + 4), όταν διαβάσει το 78 η sum θα γίνει 85 (7+78), κ.ο.κ. Όταν θα έχει διαβάσει τον τελευταίο αριθμό, η sum θα είναι 293 το οποίο είναι το άθροισμα όλων των αριθμών που δώσαμε μέχρι στιγμής.

Επομένως, η μεταβλητή sum χρησιμοποιήθηκε για να μετρήσουμε το άθροισμα των αριθμών που δώσαμε μέχρι στιγμής.

Όταν θέλουμε να **μετρήσουμε** πόσες φορές έγινε ένα συμβάν, χρησιμοποιούμε μια μεταβλητή στην οποία δίνουμε αρχική τιμή 0 (π.χ. cnt=0) και κάθε φορά που γίνεται το συμβάν το οποίο θέλουμε να μετρήσουμε την αυξάνουμε κατά 1 (cnt++). Τη μεταβλητή αυτή την ονομάζουμε **μετρητή**.

Όταν θέλουμε να **αθροίσουμε** τις διαφορετικές τιμές που παίρνει μια μεταβλητή (π.χ. η a) χρησιμοποιούμε μια θέση μνήμης (την οποία καλούμε **αθροιστή**) στην οποία δίνουμε αρχική τιμή 0 (π.χ. sum=0), και κάθε φορά που η μεταβλητή παίρνει μια νέα τιμή αυξάνουμε τον αθροιστή κατά το περιεχόμενο της μεταβλητής (π.χ. sum=sum+a).

Ενδεικτικές σελίδες

Υπολογισμός μέγιστου και ελάχιστου σε ένα σύνολο αριθμών

Το παρακάτω πρόγραμμα διαβάζει δέκα ακέραιους αριθμούς που δίνονται από το πληκτρολόγιο και εμφανίζει το μέγιστο και τον ελάχιστο από αυτούς. Ο μέγιστος αριθμός φυλάσσεται στη θέση max και ο ελάχιστος στη θέση min.

Η τεχνική είναι η εξής: Διαβάζουμε τον πρώτο αριθμό και τον θεωρούμε ταυτόχρονα τον μέγιστο αλλά και τον ελάχιστο που έχουμε διαβάσει μέχρι στιγμής, καταχωρίζοντάς τον στις μεταβλητές max και min. Κατόπιν, με μια επαναληπτική διαδικασία, διαβάζουμε έναν-έναν τους υπόλοιπους αριθμούς και τους συγκρίνουμε με τα περιεχόμενα των μεταβλητών max και min. Αν ο αριθμός είναι μεγαλύτερος από τη θέση max τον καταχωρίζουμε στη max, και αν είναι μικρότερος από τη min τον καταχωρίζουμε στη min. Τελικά οι μεταβλητές max και min θα περιέχουν το μεγαλύτερο και το μικρότερο αριθμό αντίστοιχα.

Ενδεικτικές σελίδες

8_minmax1.c

```
#include <stdio.h>
int main(void)
{
    int i,ar,min,max;
    for (i=1;i<=10;i++)
    {
        printf("Δώσε αριθμό:");
        scanf("%d",&ar);
        if (i==1)
            max=min=ar;
        if (ar>max)
            max=ar;
        if (ar<min)
            min=ar;
    }
    printf("max=%d\nmin=%d\n",max, min);
    return 0;
}
```

Διαβάζει έναν-έναν τους 10 αριθμούς

Αν ο αριθμός είναι ο πρώτος που δόθηκε (i==1), τον καταχωρίζει ως αρχική τιμή τόσο στη μεταβλητή **max**, όσο και στη μεταβλητή **min**.

Συγκρίνει κάθε αριθμό με τις θέσεις **max** και **min**.

Το πρόγραμμα που ακολουθεί δεν γνωρίζει εκ των προτέρων το πλήθος των αριθμών, αλλά χρησιμοποιεί την τιμή 9999 ως τιμή-σήμα η οποία σηματοδοτεί

Κεφάλαιο

9

Συναρτήσεις

Ενδεικτικές σελίδες

Συναρτήσεις

Σε αυτό το κεφάλαιο θα αναφερθούμε στη χρήση των συναρτήσεων από τη C. Μέχρι στιγμής είδαμε τη χρήση της main(), μιας συνάρτησης απαραίτητης σε κάθε πρόγραμμα της C. Επίσης συναντήσαμε και ορισμένες συναρτήσεις βιβλιοθήκης, όπως οι: printf(), scanf(), getchar(), κ.ά.

Οι συναρτήσεις βοηθούν τον προγραμματιστή να "κομματιάσει" ένα μεγάλο πρόγραμμα σε μικρότερα τμήματα, κάθε ένα από τα οποία επιτελεί μια συγκεκριμένη λειτουργία.

Στις γλώσσες δομημένου προγραμματισμού, οι συναρτήσεις μπορούν να "κρύβουν" κομμάτια κώδικα από το υπόλοιπο πρόγραμμα με τρόπο ώστε οι μεταβλητές μιας συνάρτησης να μην επηρεάζουν το υπόλοιπο πρόγραμμα. Κάθε συνάρτηση έχει τις δικές της μεταβλητές οι οποίες καλούνται τοπικές μεταβλητές.

Όπως αναφέρθηκε στην εισαγωγή, ένα πρόγραμμα στη C έχει την ακόλουθη γενική μορφή:

```
main()
{
    προτάσεις της συνάρτησης main();
}

function-1()
{
    προτάσεις της συνάρτησης function-1();
}

function-2()
{
    προτάσεις της συνάρτησης function-2();
}

function-N()
{
    προτάσεις της συνάρτησης function-N();
}
```

Ενδεικτικές σελίδες

Οι συναρτήσεις εκτελούνται μόνο αν κληθούν από μια άλλη συνάρτηση. Η μόνη συνάρτηση που εκτελείται αυτόματα είναι η συνάρτηση main().

Στη C κάθε συνάρτηση επιστρέφει μια τιμή, εκτός αν έχει δηλωθεί με σαφήνεια ότι δεν επιστρέφει καμία τιμή. Σε κάθε περίπτωση, μια συνάρτηση έχει έναν τύπο (int, float κ.λπ.) ανάλογα με τον τύπο δεδομένων που επιστρέφει ως τιμή. Όπως θα γίνει κατανοητό στη συνέχεια του παρόντος κεφαλαίου, ακόμα και αν δεν επιστρέφει καμία τιμή, θα πρέπει να δηλωθεί με τον κατάλληλο τύπο (void).

Επίσης, μια συνάρτηση μπορεί να έχει παραμέτρους μέσω των οποίων ο κώδικας που καλεί τη συνάρτηση μπορεί να της "μεταβιβάσει" πληροφορίες.

Στη C κάθε συνάρτηση ανήκει σε κάποιον τύπο δεδομένων. Μια συνάρτηση μπορεί να έχει καμία, μία, ή περισσότερες παραμέτρους.

Ενδεικτικές σελίδες

Ορισμός μιας συνάρτησης

Μια συνάρτηση ορίζεται με τον ακόλουθο τρόπο:

τύπος_συνάρτησης *όνομα_συνάρτησης* (τύπος *παράμετρος1*, τύπος *παράμετρος2*, ...)

{

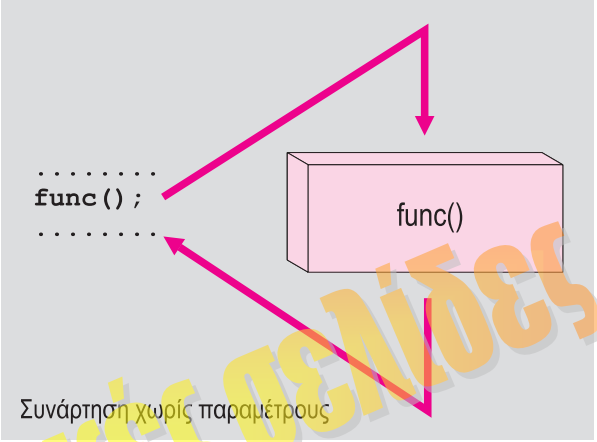
***προτάσεις συνάρτησης*;**

}

- Ο **τύπος της συνάρτησης** καθορίζει τον τύπο της τιμής που επιστρέφει η συνάρτηση.
- Στην περίπτωση που η συνάρτηση δεν επιστρέφει τιμή, δηλώνεται ως τύπου void.
- Οι μεταβλητές που δηλώνονται μέσα στο σώμα μιας συνάρτησης έχουν εμβέλεια μόνο μέσα στη συγκεκριμένη συνάρτηση και λέγονται **τοπικές** μεταβλητές.
- Οι **παράμετροι** της συνάρτησης (αν υπάρχουν) ορίζονται, μαζί με τον **τύπο** τους, μέσα στις παρενθέσεις που ακολουθούν το όνομα της συνάρτησης.
- Οι παράμετροι μιας συνάρτησης αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης.

Συναρτήσεις χωρίς παραμέτρους

Στην πιο απλή τους μορφή οι συναρτήσεις δεν έχουν παραμέτρους. Σε αυτή την περίπτωση το πρόγραμμα που τις καλεί **δεν** μπορεί να τους μεταβιβάσει πληροφορίες. Μια τέτοια συνάρτηση τη φανταζόμαστε σαν ένα ερμητικά κλειστό κουτί το οποίο επιτελεί κάποια λειτουργία, αλλά δεν έχουμε καμία δυνατότητα να τροποποιήσουμε τη λειτουργία του.



Συνάρτηση χωρίς παραμέτρους

Ενδεικτικές σελίδες

Για παράδειγμα, η παρακάτω συνάρτηση εμφανίζει τους αριθμούς από το 5 μέχρι το 10. Δεν επιστρέφει καμία τιμή και για το λόγο αυτόν δηλώνεται ως void. Δεν διαθέτει καμία παράμετρο και, κάθε φορά που καλείται, εκτελεί ακριβώς την ίδια λειτουργία.

```
void display_numbers()
{
    int i;
    for (i=5;i<=10;i++) printf("%d\n",i);
}
int main(void)
{
    printf("Πρώτη κλήση της συνάρτησης\n");
    display_numbers();
    printf("Δεύτερη κλήση της συνάρτησης\n");
    display_numbers();
    return 0;
}
```

Η συνάρτηση εμφανίζει τους αριθμούς 5, 6, 7, 8, 9, και 10.

Κλήση της συνάρτησης

Κλήση της συνάρτησης

Η συνάρτηση main() καλεί τη συνάρτηση display_numbers() δύο φορές. Κάθε φορά η συνάρτηση εμφανίζει τους ίδιους αριθμούς. Από τη στιγμή που η

συνάρτηση δεν έχει παραμέτρους, δεν υπάρχει τρόπος να επηρεάσουμε τη λειτουργία της.

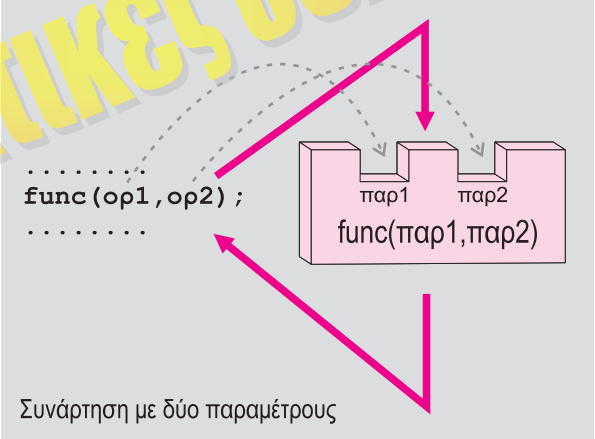
Συναρτήσεις με παραμέτρους

Οι παράμετροι είναι ένας τρόπος με τον οποίο ένα τμήμα προγράμματος μεταβιβάζει πληροφορίες σε μια συνάρτηση.

Μια τέτοια συνάρτηση τη φανταζόμαστε σαν ένα κουτί το οποίο διαθέτει εγκοπές στις οποίες μπορούμε να τοποθετήσουμε πληροφορίες. Ο κώδικας της συνάρτησης χρησιμοποιεί αυτές τις πληροφορίες και μπορεί ανάλογα να τροποποιεί τη λειτουργία του. Όταν καλούμε μια συνάρτηση με παραμέτρους, την καλούμε με ισάριθμα ορίσματα.

ΕΛΛΕΙΠΤΙΚΕΣ ΣΕΛΙΔΕΣ

Το διπλανό σχήμα δείχνει μια συνάρτηση με δύο παραμέτρους, func(παρ1,παρ2), σαν ένα κουτί με δύο εγκοπές. Όταν καλούμε τη συνάρτηση func(), πρέπει να χρησιμοποιήσουμε ισάριθμα ορίσματα func(ορ1,ορ2). Η τιμή του πρώτου ορίσματος μεταβιβάζεται στην πρώτη παράμετρο (εγκοπή), και του δεύτερου ορίσματος στη δεύτερη. Στο παρακάτω πρόγραμμα η συνάρτηση display_numbers() είναι μια παραλλαγή της συνάρτησης που συναντήσαμε στη προηγούμενη ενότητα, η οποία τώρα διαθέτει δύο παραμέτρους, ώστε ο κώδικας που την καλεί να μπορεί να καθορίζει το εύρος των αριθμών που εμφανίζει.



Συνάρτηση με δύο παραμέτρους

9_numbers.c

```
#include <stdio.h>

void display_numbers(int apo, int eos)
{
    int i;
    for (i=apo;i<=eos;i++) printf("%d\n",i);
}
```

Εμφανίζει τους αριθμούς από την τιμή της πρώτης παραμέτρου **apo** μέχρι την τιμή της δεύτερης **eos**.

```
    float c;
    scanf("%d %d",&a,&b);
    c=mo(a,b);
    printf("Ο μέσος όρος είναι %f\n",c);
    return 0;
}
```

Στο σημείο αυτό ο μεταγλωττιστής γνωρίζει τον τύπο της **mo()**, δεδομένου ότι έχει οριστεί πριν από τη **main()**. Στην περίπτωση αυτή δεν χρειάζεται "πρόσθια" δήλωση.

Παράμετροι συνάρτησης

Όπως αναφέρθηκε προηγουμένως, αν μια συνάρτηση έχει παραμέτρους, αυτές δηλώνονται μαζί με το όνομα της συνάρτησης μέσα στις παρενθέσεις που ακολουθούν.

```
int add(int x, int y)
{
    ............
}
```

Τύπος και όνομα συνάρτησης

Δήλωση παραμέτρων

Σώμα συνάρτησης

Ενδεικτικές σελίδες

Ορίσματα και μεταβίβαση παραμέτρων

Οι παράμετροι μιας συνάρτησης χρησιμοποιούνται για τη μεταβίβαση δεδομένων από το πρόγραμμα προς τη συνάρτηση την οποία καλεί.

Οι τιμές με τις οποίες καλείται μια συνάρτηση λέγονται **ορίσματα** της συνάρτησης.

Ας θεωρήσουμε το επόμενο παράδειγμα κλήσης της συνάρτησης add() από τη συνάρτηση main().

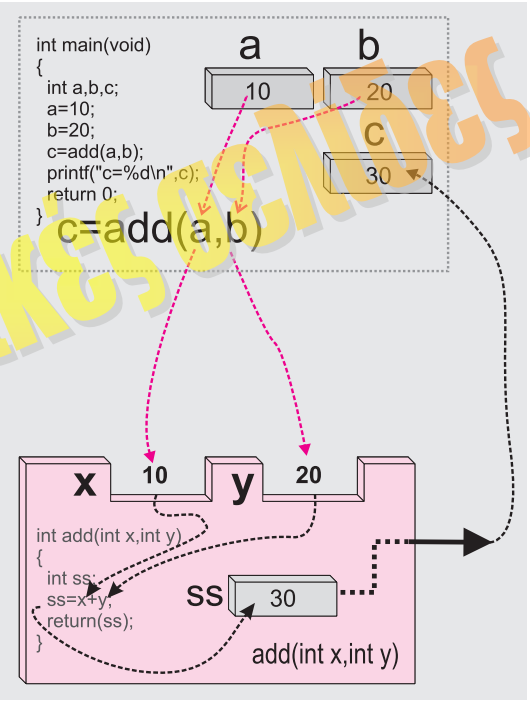
```
int main(void)
{
    int a,b,c;
    a=10;
    b=20;
    c=add(a,b);
    printf("%d\n",c);
    return 0;
}
```

Η συνάρτηση **add()** καλείται με ορίσματα τις μεταβλητές **a** και **b**. Τα ορίσματα κλήσης μιας συνάρτησης λέγονται και **πραγματικές παράμετροι**.

```
int add(int x, int y)
{
    int ss;
    ss=x+y;
    return(ss);
}
```

Οι παράμετροι μιας συνάρτησης. λέγονται και **τυπικές παράμετροι**.

Η συνάρτηση add() έχει οριστεί με δύο παραμέτρους x και y. Όταν καλείται η συνάρτηση, οι τιμές των ορισμάτων της (το 10 και το 20) αντιγράφονται στις παραμέτρους x και y αντίστοιχα. Τα ορίσματα με τα οποία καλείται μια συνάρτηση καλούνται **πραγματικές παράμετροι**.



Αυτός ο τρόπος μεταβίβασης των ορισμάτων λέγεται **κλήση** ή **μεταβίβαση κατ' αξία** (*call by value*), κατά τον οποίο αντιγράφονται οι τιμές των ορισμάτων στις αντίστοιχες παραμέτρους της συνάρτησης. Με αυτόν τον τρόπο η συνάρτηση δεν έχει δυνατότητα να τροποποιήσει τις τιμές των ορισμάτων με τα οποία κλήθηκε. Με άλλα λόγια, η συνάρτηση add() δεν μπορεί να μεταβάλει τις τιμές των ορισμάτων a και b.

Οι μεταβλητές των παραμέτρων που ορίζονται σε μια συνάρτηση (στην περίπτωση μας οι x και y) λέγονται **τυπικές παράμετροι** (*formal parameters*) της συνάρτησης.

☞ Όταν καλείται μια συνάρτηση, το πλήθος και ο τύπος των ορισμάτων πρέπει να είναι αντίστοιχα με τις τυπικές παραμέτρους της, όπως αυτές έχουν δηλωθεί στον ορισμό της συνάρτησης.

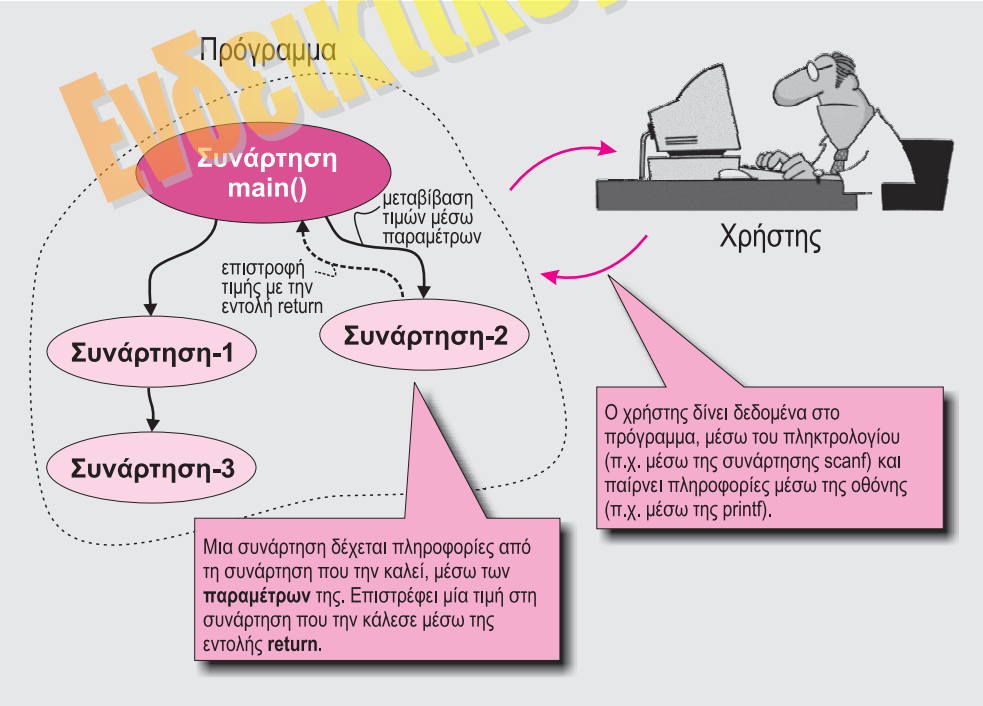
Ελληνικές Σελίδες

Ένα ξεκαθάρισμα ...

Αμφιταλαντευόμουν αν θα έπρεπε να βάλω αυτή την ενότητα στην αρχή ή στο τέλος του κεφαλαίου. Παρά το γεγονός ότι φαίνεται να είναι "εισαγωγική", προτίμησα να τη βάλω στο τέλος ώστε να ξεκαθαρίσει κάποια πράγματα, τα οποία έχω διαπιστώσει ότι προκαλούν σύγχυση στους σπουδαστές, και αφορούν στη μεταβίβαση παραμέτρων.

Ας μην ξεχνάμε ότι σε αυτό το παιχνίδι παίζουν δύο: Ο χρήστης και το πρόγραμμα! Ο χρήστης βλέπει το πρόγραμμα σαν ένα μαγικό πράγμα που απλώς του κάνει τη δουλειά του. Δίνει δεδομένα στο πρόγραμμα και παίρνει πληροφορίες από αυτό, χωρίς να τον απασχολεί αν τη δουλειά την έκανε ή συνάρτηση main() ή οποιαδήποτε άλλη συνάρτηση του προγράμματος. Ο χρήστης αγνοεί την ύπαρξη των συναρτήσεων και βλέπει το πρόγραμμα ως ενιαία οντότητα.

Ενδεικτικές σελίδες



Υπάρχει σε αρκετούς σπουδαστές η λανθασμένη εντύπωση ότι τις τιμές των παραμέτρων μιας συνάρτησης τις δίνει ο χρήστης και ότι η τιμή που επιστρέφει η συνάρτηση εμφανίζεται στην οθόνη. **Η αλήθεια είναι ότι τις τιμές στις παραμέτρους μιας συνάρτησης τις μεταβιβάζει η συνάρτηση που την καλεί, και η τιμή που επιστρέφει μια συνάρτηση επιστρέφεται στη συνάρτηση που την κάλεσε.** Φυσικά, μια συνάρτηση μπορεί να παίρνει δεδομένα από το χρήστη και να εμφανίζει πληροφορίες στην οθόνη, αλλά αυτό δεν έχει καμία σχέση με τις παραμέτρους της και με την τιμή που επιστρέφει.

Ο χρήστης επικοινωνεί με το πρόγραμμα μέσω συναρτήσεων εισόδου και εξόδου (π.χ. scanf, printf κ.λ.π.). Οι συναρτήσεις δέχονται πληροφορίες μέσω των παραμέτρων τους και επιστρέφουν τιμή μέσω της εντολής **return**.

Ενδεικτικές Σελίδες

Μελετήστε προσεκτικά τα παρακάτω παραδείγματα συναρτήσεων, οι οποίες υλοποιούν τις απαιτήσεις που εμφανίζονται στα πλαίσια αριστερά.

Θα ζητάς από το χρήστη δύο αριθμούς και θα εμφανίζεις το άθροισμά τους στην οθόνη.

```
.....
add();
```

```
void add()
{
  int ss,x,y;
  scanf("%d %d",&x,&y);
  ss=x+y;
  printf("%d\n",ss);
}
```

Η συνάρτηση δεν έχει παραμέτρους γιατί ο κώδικας που την καλεί δεν της μεταβιβάζει κάτι (τις τιμές τις δίνει ο χρήστης με τη scanf). Η συνάρτηση δεν επιστρέφει τιμή στον κώδικα που την κάλεσε αλλά εμφανίζει τα αποτελέσματα στην οθόνη. Για το λόγο αυτό δηλώνεται ως void.

Θα σου δίνω εγώ, η συνάρτηση που σε καλεί, δύο αριθμούς και θα εμφανίζεις το άθροισμά τους στην οθόνη.

```
....
int a=3, b=10;
add(a,b);
```

```
        x        y
void add(int x,int y)
{
  int ss;
  ss=x+y;
  printf("%d\n",ss);
}
```

Η συνάρτηση έχει δύο παραμέτρους μέσω των οποίων ο κώδικας που την καλεί της μεταβιβάζει δύο αριθμούς.
Η συνάρτηση δεν επιστρέφει τιμή στον κώδικα που την κάλεσε αλλά εμφανίζει τα αποτελέσματα στην οθόνη.
Για το λόγο αυτό πάλι δηλώνεται ως void.

Θα σου δίνω εγώ, η συνάρτηση που σε καλεί, δύο αριθμούς και θα επιστρέφεις το άθροισμά τους πίσω σε εμένα.

```
....
int a=3, b=10,sum;
sum=add(a,b);
printf("%d\n",sum);
```

```
        x        y
int add(int x,int y)
{
  int ss;
  ss=x+y;
  return ss;
}
```

Η συνάρτηση έχει δύο παραμέτρους μέσω των οποίων ο κώδικας που την καλεί της μεταβιβάζει δύο αριθμούς.
Η συνάρτηση επιστρέφει ως τιμή το άθροισμα των αριθμών. Η συνάρτηση δεν εμφανίζει τίποτα στην οθόνη αλλά αυτό γίνεται από τον κώδικα που την κάλεσε.
Η συνάρτηση επιστρέφει ως τιμή έναν ακέραιο για το λόγο αυτό δηλώνεται ως int.

Ενδεικτικές σελίδες

Κεφάλαιο

10

Εμβέλεια μεταβλητών

Ενδεικτικές σελίδες

Εμβέλεια μεταβλητών

Σε ένα πρόγραμμα της C μπορούν να δηλωθούν μεταβλητές σε διαφορετικά σημεία του προγράμματος. Η θέση και το είδος της δήλωσης μιας μεταβλητής καθορίζει την εμβέλειά της.

Εμβέλεια μιας μεταβλητής (variable scope) είναι ο χώρος του προγράμματος μέσα στον οποίο η μεταβλητή είναι γνωστή.

Ανάλογα με τη θέση και τον τρόπο δήλωσής τους, οι μεταβλητές χωρίζονται σε **τοπικές** (local), **καθολικές** (global), και **στατικές** (static).

Τοπικές μεταβλητές (local variables)

Όπως έχουμε δει μέχρι τώρα, οι μεταβλητές μιας συνάρτησης δηλώνονται σε οποιοδήποτε σημείο της, αρκεί να έχουν δηλωθεί πριν να χρησιμοποιηθούν. Στην περίπτωση που η συνάρτηση έχει τυπικές παραμέτρους, αυτές δηλώνονται μέσα στις παρενθέσεις. Οι τυπικές παράμετροι είναι και αυτές μεταβλητές της συνάρτησης.

Οι μεταβλητές μιας συνάρτησης, καθώς και οι τυπικές της παράμετροι (αν υπάρχουν), καλούνται **τοπικές μεταβλητές** και έχουν ισχύ μόνο μέσα στη συνάρτηση που δηλώθηκαν, παραμένοντας άγνωστες στο υπόλοιπο πρόγραμμα.

Στο παράδειγμα που ακολουθεί γίνεται κατανοητή η χρήση των τυπικών παραμέτρων και των τοπικών μεταβλητών.

```
#include <stdio.h>
int add(int x, int y);
int main(void)
{
    int a,b,sum;
    printf("Δώσε δύο αριθμούς:");
    scanf("%d %d",&a,&b);
    sum=add(a,b);
    printf("Άθροισμα=%d\n",sum);
    return 0;
}
```

Πρόσθια δήλωση της συνάρτησης **add()**.

Τοπικές μεταβλητές της συνάρτησης **main()**.

ΕΝΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

Στην επόμενη συνάρτηση add(), η μεταβλητή ss είναι τοπική μεταβλητή της συνάρτησης και έχει ισχύ μόνο μέσα στη συνάρτηση add().

Οι τυπικές παράμετροι x και y αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης, και έχουν ισχύ μόνο μέσα στη συνάρτηση.

```
int add(int x, int y)
{
    int ss;
    ss=x+y;
    return ss;
}
```

Οι τυπικές παράμετροι παίζουν επίσης το ρόλο τοπικών μεταβλητών.

Δήλωση της τοπικής μεταβλητής **ss**.

Η συνάρτηση **add()** επιστρέφει ως τιμή το περιεχόμενο της μεταβλητής **ss**.

Οι τυπικές παράμετροι παίζουν και αυτές το ρόλο των τοπικών μεταβλητών, με μόνη διαφορά ότι σε αυτές αντιγράφονται οι τιμές των ορισμάτων με τα οποία καλείται η συνάρτηση.

Αν δεν γίνει κατανοητή η χρήση και η "εμβέλεια" των τυπικών παραμέτρων και των τοπικών μεταβλητών, είναι πολύ εύκολο να οδηγηθούμε σε λανθασμένα προγράμματα.

Στη συνέχεια βλέπετε το προηγούμενο πρόγραμμα, γεμάτο ενδεχόμενα λάθη.

```
#include <stdio.h>
int add(int x, int y);
int main(void)
{
    int sum;
    float a,b;
    printf("Δωσε δύο αριθμούς:");
    scanf("%f %f,&a,&b);
    sum=add(a,b);
    printf("ss=%d\n",ss);
    printf("Αθροισμα=%d\n",sum);
    return 0;
}
```

Τα ορίσματα με τα οποία πρέπει να κληθεί η **add()** πρέπει να είναι τύπου **int**, και όχι **float**. Αυτό μπορεί να έχει συνέπεια την απώλεια πληροφορίας.

Η συνάρτηση **main()** αγνοεί την ύπαρξη της **ss**, η οποία είναι τοπική μεταβλητή της **add()**.

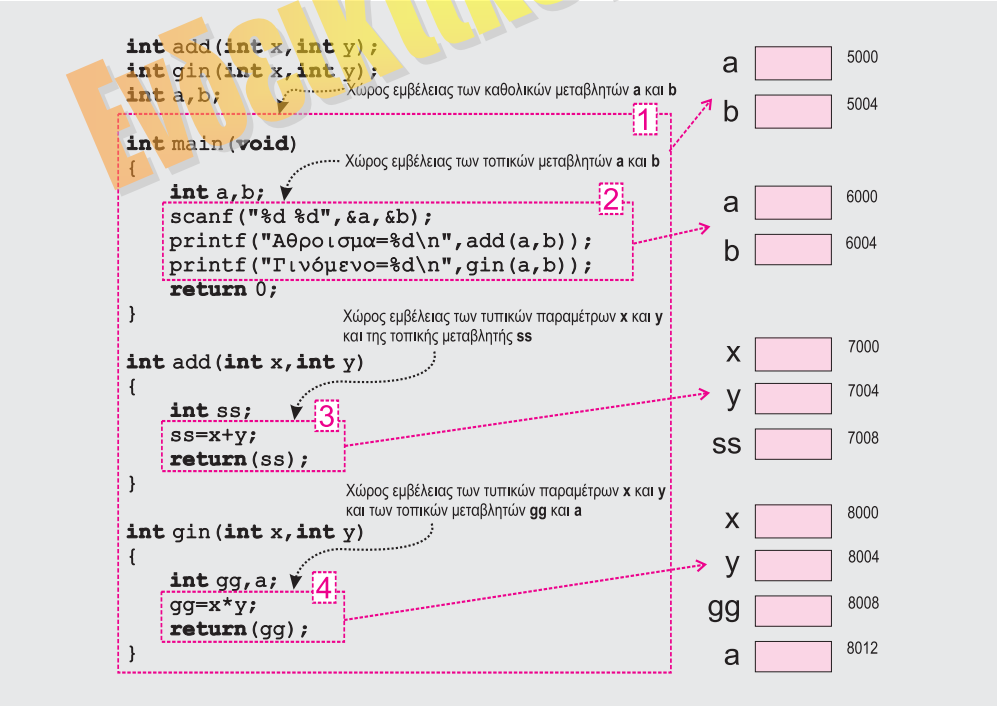
Ενδεικτικές σελίδες

Η μέθοδος μεταβίβασης τιμών σε μια συνάρτηση μέσω καθολικών μεταβλητών πρέπει να αποφεύγεται, και να χρησιμοποιείται μόνο όταν πραγματικά υπάρχει απόλυτη ανάγκη.

☞ Σε μια καθολική μεταβλητή έχουν πρόσβαση όλες οι συναρτήσεις που βρίσκονται μετά από το σημείο του προγράμματος όπου δηλώθηκε. Όταν λοιπόν θέλουμε όλες ανεξαιρέτως οι συναρτήσεις να έχουν πρόσβαση σε μια καθολική μεταβλητή, πρέπει να δηλώσουμε τη μεταβλητή αυτή στην αρχή του προγράμματος, πριν από τη main().

☞ Στην περίπτωση που έχει δηλωθεί μια τοπική μεταβλητή με όνομα ίδιο με μια καθολική μεταβλητή, τότε στο χώρο εμβέλειας της τοπικής μεταβλητής αναφερόμαστε στην τοπική μεταβλητή, ενώ οπουδήποτε αλλού αναφερόμαστε στην καθολική μεταβλητή.

Το επόμενο παράδειγμα δείχνει την εμβέλεια τοπικών και καθολικών μεταβλητών:



Σχήμα 10.1 *Εμβέλεια τοπικών και καθολικών μεταβλητών*

Ενδεικτικές σελίδες

Συχνά λάθη

- Το συχνότερο λάθος είναι ότι μπερδεύουμε την εμβέλεια των μεταβλητών, ιδιαίτερα όταν έχουμε μεταβλητές με το ίδιο όνομα.
- Ξεχνάμε να αναθέσουμε αρχική τιμή σε τοπικές στατικές μεταβλητές. Η δήλωση μιας τέτοιας μεταβλητής χωρίς αρχική τιμή δεν έχει ουσιαστικό νόημα.
- Ξεχνάμε ότι οι παράμετροι μιας συνάρτησης αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης.

Ανασκόπηση Κεφαλαίου 10

- Οι μεταβλητές που δηλώνονται μέσα σε μια συνάρτηση καλούνται τοπικές μεταβλητές και έχουν εμβέλεια μόνο μέσα στη συνάρτηση στην οποία ορίζονται.
- Τοπικές μεταβλητές μπορούν να δηλωθούν και αμέσως μετά από την αριστερή αγκύλη οποιασδήποτε σύνθετης πρότασης. Μια τέτοια τοπική μεταβλητή έχει εμβέλεια μόνο μέσα στη σύνθετη πρόταση όπου δηλώθηκε.
- Οι τυπικές παράμετροι μιας συνάρτησης είναι και αυτές τοπικές μεταβλητές, με εμβέλεια μόνο μέσα στην ίδια τη συνάρτηση.
- Οι καθολικές μεταβλητές δηλώνονται έξω από κάθε συνάρτηση (συνήθως στην αρχή του κώδικα) και έχουν εμβέλεια σε ολόκληρο το πρόγραμμα.
- Μια στατική μεταβλητή έχει περιορισμένη εμβέλεια, όπως και μια τοπική μεταβλητή, αλλά διατηρεί την τιμή της και όταν επιστρέφει η συνάρτηση στην οποία δηλώθηκε.
- Όταν κατά τη δήλωση μιας στατικής μεταβλητής δοθεί ταυτόχρονα αρχική τιμή, π.χ. `static int a=5`, η αρχική τιμή θα ανατεθεί στη στατική μεταβλητή μόνο την πρώτη φορά που θα εκτελεστεί η πρόταση δήλωσης.
- Μια τοπική μεταβλητή δεσμεύει μια θέση μνήμης η οποία απελευθερώνεται όταν λήξει η εμβέλεια της μεταβλητής, και τα περιεχόμενά της χάνονται.
- Τόσο οι καθολικές, όσο και οι στατικές μεταβλητές δεσμεύουν μια θέση μνήμης στη φάση της μεταγλώττισης, και τη διατηρούν σε όλη τη διάρκεια του προγράμματος.

Ενδεικτικές σελίδες

Ασκήσεις Κεφαλαίου 10

10.1 Ποιο θα είναι το αποτέλεσμα του επόμενου προγράμματος; Σχεδιάστε το χώρο εμβέλειας κάθε μεταβλητής. ★

```
#include <stdio.h>
void set();
void pp();
float mo(int k, int l);
float a;
int x,y;

int main(void)
{
    int x,y;
    x=y=4;
    set();
    a=mo(x,y);
    pp();
    return 0;
}

void set()
{
    int aa;
    x=10;
    y=20;
}

void pp()
{
    int x,y;
    printf("a=%f\n",a);
}

float mo(int k, int l)
{
    float mesos;
    mesos=(k+l)/2.0;
    return mesos;
}
```

Ενδεικτικές σελίδες

10.2 Τι κάνει το επόμενο πρόγραμμα; Να σχεδιαστούν (με ορθογώνια) οι εμβέλειες των μεταβλητών. ★★

```
#include <stdio.h>
void disp();
int step;
int get();

int main(void)
{
   step=2;
   disp();
   return 0;
}

void disp()
{
   int a,b;
   printf("Μια συνάρτηση\n");
   a=get();
   b=get();
   if (a==0 && b==0)
      exit();
   else
   {
      int i;
      for (i=a;i<=b;i=i+step)
         printf("%d\n",i);
   }
}

int get()
{
   int x;
   scanf("%d",&x);
   return x;
}
```

10.3 Βρείτε τα λάθη του επόμενου προγράμματος: ★

```
#include <stdio.h>
int step;

int main(void)
{
   step=2;
```

Ενδεικτικές σελίδες

10.15 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★★

```
#include <stdio.h>
void out();

int main()
{
    int i;
    for (i=1;i<=15;i++)
       out();
    return 0;
}

void out()
{
    static char ch='A';
    printf("%c\n",ch++);
    if (ch>'E') ch='A';
}
```

10.16 Τι διαφορά θα είχε στα αποτελέσματα το πρόγραμμα της προηγούμενης άσκησης (10.15), αν η συνάρτηση out() ήταν η εξής: ★★

```
void out()
{
    static char ch;
    ch='A';
    printf("%c\n",ch++);
    if (ch>'E') ch='A';
}
```

10.17 Να γραφεί συνάρτηση η οποία, κάθε φορά που καλείται, να επιστρέφει ως τιμή το επόμενο πολλαπλάσιο του 5. Δηλαδή την πρώτη φορά το 5, τη δεύτερη το 10, την τρίτη το 15 κ.ο.κ. ★★

10.18 Να γραφεί πρόγραμμα το οποίο να ζητάει από το χρήστη έναν ακέραιο αριθμό. Στη συνέχεια, το πρόγραμμα να εμφανίζει, για κάθε ψηφίο του αριθμού, τόσους αστερίσκους όση η τιμή του ψηφίου (μία γραμμή με αστερίσκους για κάθε ψηφίο), με την αντίστροφη σειρά. Για παράδειγμα αν δοθεί ο αριθμός 2461, να εμφανίσει τα τους αστερίσκους που φαίνονται στο διπλανό πλαίσιο. ★★★

```
*
******
****
**
```

Ενδεικτικές σελίδες

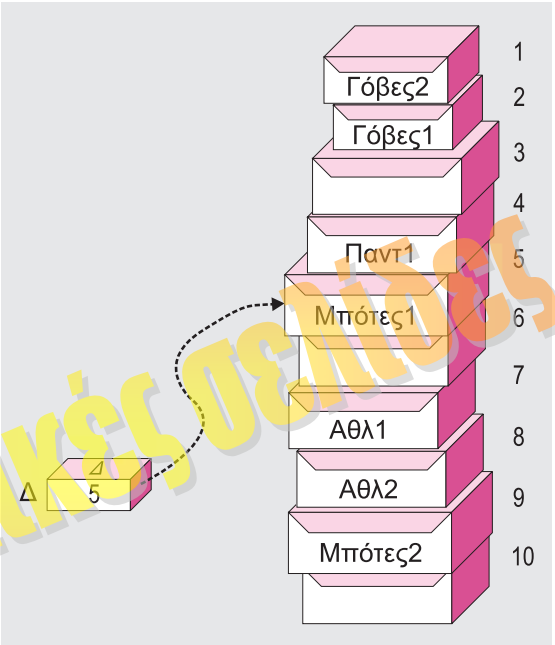
Κεφάλαιο 11

Δείκτες

Ενδεικτικές σελίδες

Δείκτες

Οι δείκτες (pointers) είναι ένα από τα πιο "παρεξηγημένα" κεφάλαια των γλωσσών προγραμματισμού. Οι αρχάριοι στον προγραμματισμό συναντούν αρκετή δυσκολία στην κατανόηση και στη χρήση τους, ενώ οι έμπειροι σπαζοκεφαλιάζουν με τις ιδιοτροπίες τους. Η βασική φιλοσοφία των δεικτών, όμως, είναι πολύ απλή.



Ας φανταστούμε κουτιά διαφόρων μεγεθών, το ένα πάνω από το άλλο, τα οποία περιέχουν παπούτσια. Κάποια κουτιά είναι άδεια και κάποια γεμάτα. Κάθε γεμάτο κουτί γράφει στο εξωτερικό του ένα ενδεικτικό όνομα ώστε να καταλαβαίνουμε τι περιέχει. Κάθε κουτί έχει έναν αύξοντα αριθμό (1, 2, 3, ...) ο οποίος προσδιορίζει τη σειρά του στη στοίβα.

Ενδεικτικές σελίδες

Σε ένα ξεχωριστό κουτί (Δ) βάζουμε τον αύξοντα αριθμό του κουτιού με τα παπούτσια που θα φορέσουμε το βράδυ, π.χ. το 5 αν θέλουμε να βάλουμε το ζευγάρι που βρίσκεται στο κουτί Μπότες1 (με α/α 5).

> Το κουτί **Δ** δείχνει το κουτί που θα χρησιμοποιήσουμε (το κουτί με α/α 5). Μπορούμε να πούμε ότι το κουτί **Δ** είναι ένας **δείκτης** ο οποίος δείχνει στο κουτί **Μπότες1**.

Θα μπορούσαμε φυσικά να θυμόμαστε το κουτί που θα χρησιμοποιούσαμε από το όνομά του (Μπότες1). Η χρήση του δείκτη **Δ**, όμως, είναι ένας έμμεσος τρόπος πρόσβασης στο κουτί Μπότες1. Έτσι, λοιπόν, για να φορέσουμε τα παπούτσια μας το βράδυ έχουμε δύο επιλογές:

- Να ανοίξουμε το κουτί Μπότες1 (αν το θυμόμαστε) — ο άμεσος τρόπος.

- Να ανοίξουμε το κουτί **Δ**, να βρούμε ότι το κουτί που μας ενδιαφέρει είναι το κουτί με α/α 5, και να ανοίξουμε το συγκεκριμένο κουτί. Με αυτόν τον τρόπο δεν χρειάζεται πια να γνωρίζουμε το όνομα του κουτιού.

Στο συγκεκριμένο παράδειγμα το κουτί-δείκτης είναι ένα κουτί το οποίο περιέχει τον α/α ενός άλλου κουτιού.

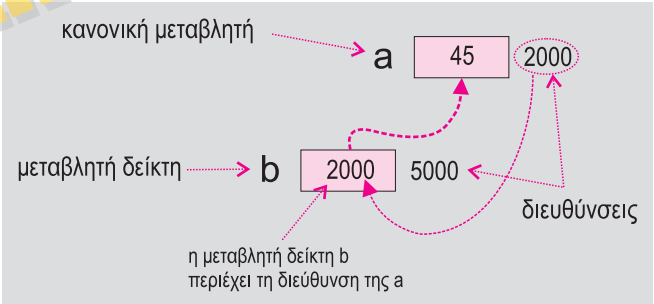
Το μέγεθος του κουτιού-δείκτη **είναι ανεξάρτητο** από το μέγεθος του κουτιού στο οποίο δείχνει, αφού σε κάθε περίπτωση θα περιέχει μόνο τον α/α του.

Το προηγούμενο παράδειγμα δείχνει με απλό και κατανοητό τρόπο τη φιλοσοφία του δείκτη. Στη συνέχεια του κεφαλαίου θα εξετάσουμε σε βάθος τη χρήση των δεικτών στη C. Η παρουσίαση των δεικτών θα ολοκληρωθεί στα επόμενα κεφάλαια, αφού γίνει πρώτα κατανοητή η χρήση των πινάκων, αλλά και των άλλων τύπων δεδομένων που υποστηρίζει η C.

Ηλεκτρονικές σελίδες

Μεταβλητές δείκτη (Pointer variables)

Οι μεταβλητές δείκτη έχουν ένα ιδιαίτερο χαρακτηριστικό. Μέσα σε μια μεταβλητή δείκτη καταχωρίζεται η διεύθυνση μιας άλλης μεταβλητής. Η φιλοσοφία μιας μεταβλητής δείκτη φαίνεται στο διπλανό σχήμα. Στη μεταβλητή δείκτη b έχει καταχωριστεί η διεύθυνση της μεταβλητής a (b=&a).



Δήλωση μιας μεταβλητής δείκτη

Μια μεταβλητή δείκτη δηλώνεται όπως οποιαδήποτε άλλη μεταβλητή, με τη διαφορά ότι πριν από το όνομά της πρέπει να υπάρχει ο τελεστής *. Για παράδειγμα, η πρόταση:

```
int a,*b;
```

δηλώνει μια μεταβλητή a τύπου **int** και μια **μεταβλητή δείκτη** b.

Το γεγονός ότι η συγκεκριμένη μεταβλητή δείκτη δηλώθηκε ως τύπου int προϊδεάζει ότι η μεταβλητή b θα περιέχει τη διεύθυνση μιας μεταβλητής τύπου int.

Όπως θα δούμε αργότερα, ο τύπος της θέσης μνήμης στην οποία "δείχνει" μια μεταβλητή δείκτη παίζει καθοριστικό ρόλο στην αριθμητική των δεικτών.

Προς το παρόν, αρκεί να έχουμε υπόψη ότι μια **μεταβλητή δείκτη** περιέχει τη **διεύθυνση** μιας άλλης μεταβλητής.

☞ Το μέγεθος μιας μεταβλητής δείκτη **δεν** εξαρτάται από τον τύπο της θέσης μνήμης στην οποία "δείχνει" η μεταβλητή. Όλες οι μεταβλητές δείκτη, ανεξάρτητα από την πρόταση δήλωσης με την οποία δηλώθηκαν (int, float κ.λπ.), έχουν το ίδιο ακριβώς μέγεθος.

☞ Το μέγεθος (σε byte) μιας μεταβλητής δείκτη διαφέρει ανάλογα με την έκδοση της C, το λειτουργικό σύστημα που χρησιμοποιείται, αλλά και τον τύπο του Η/Υ. Σε κάθε περίπτωση, ο τελεστής sizeof μπορεί να μας φανερώσει το μέγεθος των μεταβλητών δείκτη στο σύστημα που εργαζόμαστε.

Ελληνικές Εκδόσεις

Τα επόμενα παραδείγματα δίνουν μια πρώτη γεύση από τη χρήση των δεικτών.

```
int main(void)
{
    int a, *b;
    b=&a;
    scanf("%d",b);
    return 0;
}
```

Στη συνάρτηση **scanf()** μεταβιβάζεται η τιμής της μεταβλητής **b**, δηλαδή η διεύθυνση της **a**.

Στη θέση μνήμης b καταχωρίζεται η διεύθυνση της μεταβλητής a (ο τελεστής & αποδίδει τη διεύθυνση μιας μεταβλητής).

Η scanf() περιμένει ως παράμετρο τη διεύθυνση μιας θέσης μνήμης, που είναι στη συγκεκριμένη περίπτωση το περιεχόμενο της μεταβλητής δείκτη b (δηλαδή η διεύθυνση της μεταβλητής a). Επομένως η scanf() θα περιμένει να πατηθεί ένας αριθμός, και θα τον βάλει στη θέση μνήμης με διεύθυνση την τιμή της παραμέτρου της. Δηλαδή στη μεταβλητή a!

Παρατηρούμε ότι είναι δυνατό να αλλάξουμε την τιμή της μεταβλητής στην οποία δείχνει ένας δείκτης. Έτσι, στο προηγούμενο πρόγραμμα, με την πρόταση:

```
*c=44;
```

Η παράσταση ***c** αποτελεί αναφορά στη μεταβλητή **a**.

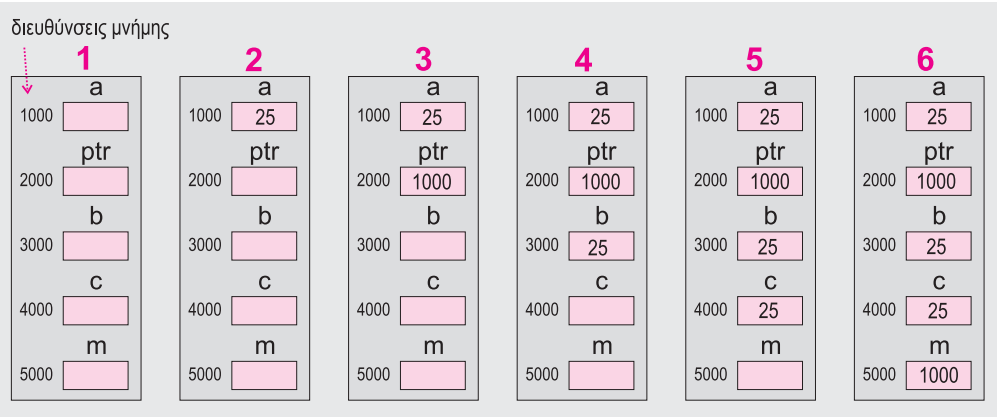
το 44 δεν καταχωρίζεται στη μεταβλητή c, αλλά στη μεταβλητή που δείχνει η c, δηλαδή στην a. Οι επόμενες προτάσεις:

```
a=44;
*c=44;
```

είναι ισοδύναμες.

Το επόμενο σχηματικό παράδειγμα συμβάλλει στην κατανόηση της χρήσης αυτών των δύο τελεστών. Υποθέτουμε ότι οι διευθύνσεις των μεταβλητών του παραδείγματος είναι οι αριθμοί που εμφανίζονται αριστερά τους.

1. int a, *ptr, b, c, *m; → δηλώνει 5 μεταβλητές (οι δύο είναι δείκτες)
2. a=25; → καταχωρίζει το 25 στη μεταβλητή a
3. ptr = &a; → καταχωρίζει στο δείκτη ptr τη διεύθυνση της a
4. b = a; → καταχωρίζει στη μεταβλητή b την τιμή της a
5. c = *ptr; → καταχωρίζει στη c το περιεχόμενο της θέσης μνήμης στην οποία δείχνει ο δείκτης ptr
6. m = ptr; → καταχωρίζει στο δείκτη m το περιεχόμενο του ptr



Ενδεικτικές σελίδες

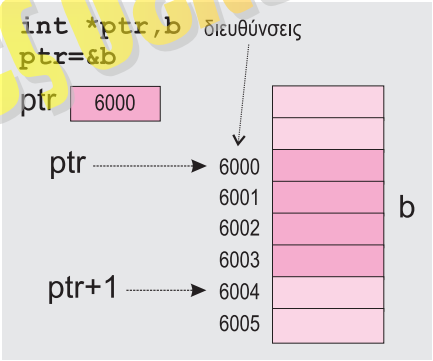
Μέχρι τώρα μάθαμε ότι ...

- Οι δείκτες είναι μεταβλητές που περιέχουν διευθύνσεις μνήμης, συνήθως διευθύνσεις άλλων μεταβλητών.
- Το μέγεθος μιας μεταβλητής δείκτη είναι συγκεκριμένο και ανεξάρτητο από τον τύπο με τον οποίο δηλώθηκε.
- Ο τελεστής & αποδίδει τη διεύθυνση μιας θέσης μνήμης.
- Ο τελεστής αναφοράς * χρησιμοποιείται σε συνδυασμό με μια μεταβλητή δείκτη για να αναφερθούμε στη μεταβλητή **στην οποία δείχνει** ο δείκτης.

Ενδεικτικές σελίδες

Αριθμητική των δεικτών

Οι δείκτες ακολουθούν τη δική τους λογική στις αριθμητικές πράξεις, που δεν είναι όμως καθόλου παράδοξη αν αναλογιστούμε τη βαθύτερη φιλοσοφία τους.



Ας θεωρήσουμε μια μεταβλητή δείκτη με όνομα ptr, η οποία δηλώνεται στο επόμενο πρόγραμμα μαζί με μια μεταβλητή a. Υποθέτουμε ότι η μεταβλητή a έχει διεύθυνση 6000.

```
int a,*ptr;
```

`ptr=&a;` ◄ *η μεταβλητή ptr θα πάρει την τιμή 6000*

`ptr=ptr+1;` ◄ *η μεταβλητή ptr θα πάρει την τιμή 6004*

`++ptr;` ◄ *η μεταβλητή ptr θα πάρει την τιμή 6008*

`ptr=ptr+3;` ◄ *η μεταβλητή ptr θα πάρει την τιμή 6020*

Το παράδοξο είναι ότι, ενώ αυξάνουμε τη μεταβλητή ptr κατά 1, αυτή αυξάνεται κατά 4, και όταν την αυξήσουμε κατά 3 αυτή αυξάνεται κατά 12!

Πίσω από αυτό το παράδοξο κρύβονται κάποια κρυφά μυστικά των δεικτών.

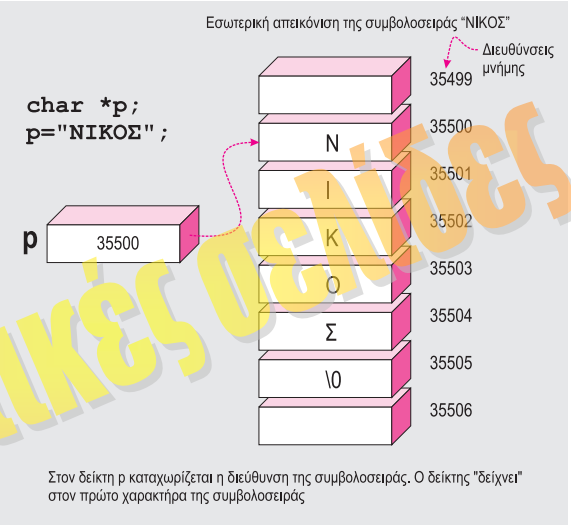
Με αυτόν τον τρόπο, όμως, σε μια πρόταση δήλωσης μπορούν να δηλωθούν μόνο μεταβλητές δείκτη.

Δείκτες και συμβολοσειρές

Όπως έχει ήδη αναφερθεί στο κεφάλαιο 5, οι συμβολοσειρές αντιμετωπίζονται από τη C ως διευθύνσεις. Επομένως, μήπως μπορούμε να καταχωρίσουμε μια συμβολοσειρά σε ένα δείκτη; Ο παρακάτω κώδικας,

```
char *p;
p="ΝΙΚΟΣ";
```

έχει αποτέλεσμα να καταχωριστεί μέσα στη μεταβλητή δείκτη p η διεύθυνση της συμβολοσειράς "ΝΙΚΟΣ", δηλαδή η διεύθυνση του πρώτου χαρακτήρα της.



Σχήμα 11.1 *Δείκτες και συμβολοσειρές*

Ενδεικτικές σελίδες

Για να δούμε όμως και τον πρακτικό χαρακτήρα αυτής της σκέψης, ας αναλύσουμε τη λειτουργία της συνάρτησης printf(), στην πιο απλή της μορφή. Γνωρίζουμε ότι η παρακάτω πρόταση,

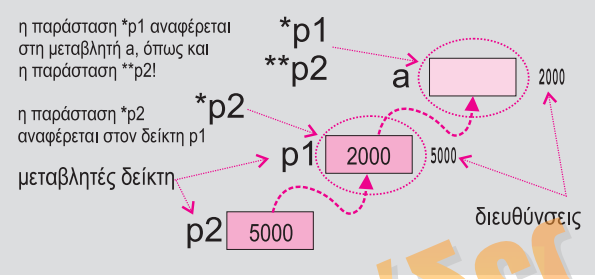
```
printf("NIKOS");
```

θα έχει αποτέλεσμα την εμφάνιση της συμβολοσειράς "ΝΙΚΟΣ". Ας διευκρινίσουμε τώρα τι ακριβώς μεταβιβάζεται στη συνάρτηση printf().

> Στη συνάρτηση printf() μεταβιβάζεται η διεύθυνση της συμβολοσειράς, και ΟΧΙ οι ίδιοι οι χαρακτήρες της.

Επομένως, με βάση το παράδειγμα του σχήματος, στην printf() μεταβιβάζεται η διεύθυνση 35500. Τι κάνει τώρα η printf(); Αρχίζει και εμφανίζει στην οθόνη έναν-έναν τους χαρακτήρες, ξεκινώντας από τη διεύθυνση 3500 (την οποία της μεταβιβάσαμε), μέχρι να εντοπίσει το χαρακτήρα τερματισμού '\0'.

Δείκτες σε δείκτες



Ένας δείκτης περιέχει τη διεύθυνση μιας μεταβλητής, επομένως γιατί κάποιος δείκτης να μην μπορεί να περιέχει τη διεύθυνση ενός άλλου δείκτη; Το διπλανό σχήμα δείχνει μια μεταβλητή a, ένα δείκτη p1 ο οποίος περιέχει τη διεύθυνση της a, και ένα δείκτη p2 ο οποίος περιέχει τη διεύθυνση του δείκτη p1.

Μια μεταβλητή δείκτη-σε-δείκτη δηλώνεται όπως και μια απλή μεταβλητή δείκτη, με τη διαφορά ότι πριν από το όνομα της βάζουμε τον τελεστή * δύο φορές. Για παράδειγμα, η παρακάτω δηλωτική πρόταση:

```
int *p1,**p2;
```

δηλώνει μια μεταβλητή δείκτη p1 και μια μεταβλητή δείκτη-σε-δείκτη p2. Εναλλακτικά μια μεταβλητή δείκτη-σε-δείκτη μπορεί να δηλωθεί και ως εξής:

```
int* *p2;
```

Το παράδειγμα του παραπάνω σχήματος υλοποιείται από τις παρακάτω προτάσεις:

```
int a,*p1,**p2;
p1=&a;      // Ο δείκτης p1 τώρα δείχνει στη μεταβλητή a (διότι περιέχει τη διεύθυνση της a)
p2=&p1;     // Ο δείκτης p2 τώρα δείχνει στο δείκτη p1 (διότι περιέχει τη διεύθυνση του p1)
```

Πρόσβαση στη μεταβλητή a μπορούμε να έχουμε είτε μέσω του δείκτη p1:

```
*p1=100;    // Καταχωρίζει στη μεταβλητή που δείχνει ο p1 (στην a) το 100.
```

είτε μέσω του δείκτη p2:

```
**p2=200;   // Καταχωρίζει στη μεταβλητή που δείχνει ο δείκτης, στον οποίο δείχνει ο δείκτης p2, το 200!
```

> Η χρήση του τελεστή αναφοράς * δύο φορές (**) έχει αποτέλεσμα να αναφερόμαστε στη θέση στην οποία δείχνει ένας δείκτης, στον οποίο όμως αναφερόμαστε μέσω ενός άλλου δείκτη.

Ενδεικτικές σελίδες

Κεφάλαιο

12

Πίνακες

Ενδεικτικές σελίδες

Πίνακες (arrays)

Το κεφάλαιο αυτό είναι το πιο σημαντικό κεφάλαιο της γλώσσας. Η αλήθεια είναι ότι το έγραψα τρεις φορές, την τελευταία φορά αφού είχα ήδη στείλει το βιβλίο για τελική επιμέλεια. Ο λόγος είναι ότι συνέχεια σκεφτόμουν ποια θα ήταν η καλύτερη διδακτική σειρά που θα έπρεπε να ακολουθήσω, πράγμα καθόλου εύκολο δεδομένου ότι το κεφάλαιο αυτό εισάγει πολλές νέες έννοιες σχεδόν ταυτόχρονα. Ας μπούμε όμως στην ουσία!

Ας υποθέσουμε ότι θέλουμε να αποθηκεύσουμε 10 αριθμούς σε ισάριθμες θέσεις μνήμης και κατόπιν να υπολογίσουμε και να εμφανίσουμε το άθροισμά τους. Με όσα έχουμε μάθει μέχρι τώρα, θα έπρεπε να γράψουμε το ακόλουθο πρόγραμμα:

Ενδεικτικές σελίδες

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int a,b,c,d,e,f,k,l,m,n,sum;
    printf("Δώσε 10 αριθμούς>");
    scanf("%d %d %d %d %d %d %d %d %d %d",&a,&b,&c,&d,&e,&f,&k,&l,&m,&n);
    sum=a+b+c+d+e+f+k+l+m+n;
    printf("Αθροισμα:%d\n",sum);
    return 0;
}
```

Στις 10 θέσεις μνήμης καταχωρίζονται οι αριθμοί που πληκτρολογεί ο χρήστης.

Υπολογίζει και εμφανίζει το άθροισμα των 10 θέσεων μνήμης.

Σκεφτείτε τι θα έπρεπε να κάναμε αν θέλαμε να κάνουμε το ίδιο όχι για 10, αλλά για 100 αριθμούς! Το πρόγραμμα θα άρχιζε να γίνεται πολύ πολύπλοκο, με 100 διαφορετικές μεταβλητές, και αν σκεφτούμε για πολύ περισσότερους αριθμούς, τότε τα πράγματα θα ήταν κυριολεκτικά αδύνατα. Μη ξεχνάμε ότι μία από τις βασικές δυνατότητες ενός υπολογιστικού συστήματος είναι η επεξεργασία μεγάλου όγκου δεδομένων.

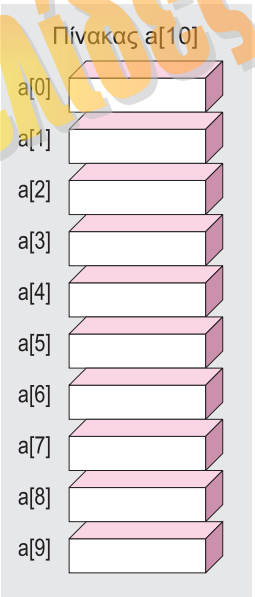
Δεν μπορεί, θα πρέπει να υπάρχει καλύτερος και πιο έξυπνος τρόπος!

Ο τρόπος για να αποθηκεύουμε μεγάλο πλήθος δεδομένων, σε οποιαδήποτε γλώσσα προγραμματισμού, είναι η χρήση **πινάκων**.

Οι **πίνακες** αποτελούν ένα διαφορετικό τρόπο διαχείρισης της μνήμης. Ο πίνακας είναι μια ομάδα θέσεων μνήμης στις οποίες γίνεται αναφορά με ένα κοινό όνομα. Μια συγκεκριμένη θέση μνήμης αυτής της ομάδας καθορίζεται από έναν ή περισσότερους αριθμούς αναφοράς. Στη C, όλοι οι πίνακες αποτελούνται από συνεχόμενες θέσεις μνήμης και μπορεί να έχουν μία ή περισσότερες διαστάσεις.

Πίνακες μίας διάστασης (1Δ)

Τον πίνακα μίας διάστασης τον φανταζόμαστε σαν μια ομάδα θέσεων μνήμης, τη μία πάνω από την άλλη. Κάθε θέση μνήμης έχει έναν αριθμό αναφοράς ο οποίος ξεκινάει από το 0 (η πρώτη 0, η δεύτερη 1, κ.ο.κ).



Στη C, η πρόταση δήλωσης ενός πίνακα είναι παρόμοια με τις προτάσεις δήλωσης οποιασδήποτε άλλης μεταβλητής:

τύπος_δεδομένων όνομα_πίνακα[μέγεθος]; π.χ. η

```
int a[10];
```

δηλώνει έναν πίνακα μίας διάστασης με όνομα a και 10 θέσεις μνήμης.

Στη C, η πρώτη θέση μνήμης ενός πίνακα έχει αριθμό αναφοράς 0. Επομένως, ένας πίνακας int a[10] θα έχει 10 θέσεις μνήμης, από την a[0] μέχρι την a[9].

Όλες οι θέσεις μνήμης ενός πίνακα είναι του τύπου που δηλώθηκε στην πρόταση δήλωσης του πίνακα.

Οι θέσεις μνήμης ενός πίνακα χρησιμοποιούνται όπως οι υπόλοιπες μεταβλητές. Για παράδειγμα, οι επόμενες προτάσεις:

```
a[5]=45;
a[8]=67;
```

θα καταχωρίσουν το 45 στην έκτη θέση μνήμης του πίνακα a και το 67 στην έατη (a[0] είναι η πρώτη θέση μνήμης, a[5] είναι η έκτη, και a[8] η έατη!).

Ενδεικτικές σελίδες

Χειρισμός πινάκων μίας διάστασης

Ο χειρισμός πινάκων μίας διάστασης συνήθως γίνεται με ένα βρόχο for που διασχίζει όλες τις θέσεις του πίνακα.

Το επόμενο πρόγραμμα δημιουργεί έναν πίνακα ακεραίων 100 θέσεων και τον γεμίζει με τυχαίους αριθμούς. Στη συνέχεια επεξεργάζεται τον πίνακα, υπολογίζει το άθροισμα των στοιχείων του, και τέλος ψάχνει αν ένας αριθμός, που έδωσε ο χρήστης, υπάρχει ή όχι μέσα στον πίνακα.

12_sum_1d.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
int main(void)
{
    int i,a[100],sum=0,ar;
    bool found=false;
    for (i=0;i<100;i++)
        a[i]=rand();
    for (i=0;i<100;i++)
        printf("%d\n",a[i]);
    for (i=0;i<100;i++)
        sum=sum+a[i];
    printf("Το άθροισμα των αριθμών είναι:%d\n",sum);
    printf("Δώσε αριθμό για εύρεση:");
    scanf("%d",&ar);
    for (i=0;i<100;i++)
        if (a[i]==ar) found=true;
    if (found)
        printf("Υπάρχει στον πίνακα\n");
    else
        printf("Δεν υπάρχει στον πίνακα\n");
    return 0;
}
```

Στις 100 θέσεις του πίνακα καταχωρίζονται τυχαίοι αριθμοί.

Εμφανίζονται στην οθόνη οι αριθμοί του πίνακα.

Υπολογίζει και εμφανίζει το άθροισμα όλων των στοιχείων του πίνακα.

Εντοπίζει αν ο αριθμός **ar** που έδωσε ο χρήστης υπάρχει μέσα στον πίνακα.

Ενδεικτικές σελίδες

Το πρόγραμμα χρησιμοποιεί τη λογική μεταβλητή found, στην οποία καταχωρίζει την τιμή true (1) στην περίπτωση που εντοπίσει τον αριθμό ar μέσα στον πίνακα. Κατόπιν ελέγχει την τιμή αυτής της μεταβλητής.

Πίνακες πολλών διαστάσεων

Η C υποστηρίζει πίνακες με περισσότερες από μία διαστάσεις. Η δήλωση ενός πίνακα με πολλές διαστάσεις γίνεται κατανοητή με το επόμενο παράδειγμα

```
int pin[4][6][10];
```

όπου δηλώνεται ένας πίνακας τριών διαστάσεων, με όρια

0~3 ⇨ για την πρώτη διάσταση

0~5 ⇨ για τη δεύτερη διάσταση

0~9 ⇨ για την τρίτη διάσταση

Ο πίνακας έχει συνολικά 4*6*10=240 θέσεις μνήμης

Όταν θέλουμε να προσπελάσουμε όλες τις θέσεις μνήμης ενός πίνακα πολλών διαστάσεων, ο ενδεδειγμένος τρόπος είναι η χρήση ένθετων βρόχων for, με τη βοήθεια των οποίων αλλάζουν οι αριθμοί αναφοράς για κάθε θέση μνήμης. Για παράδειγμα, ο κώδικας που ακολουθεί γεμίζει έναν πίνακα τριών διαστάσεων με τυχαίους αριθμούς:

```
int main(void)
{
    int pin[4][6][10],i,j,k;
    for (i=0;i<4;i++)
        for (j=0;j<6;j++)
            for (k=0;k<10;k++)
                pin[i][j][k]=rand();
    return 0;
}
```

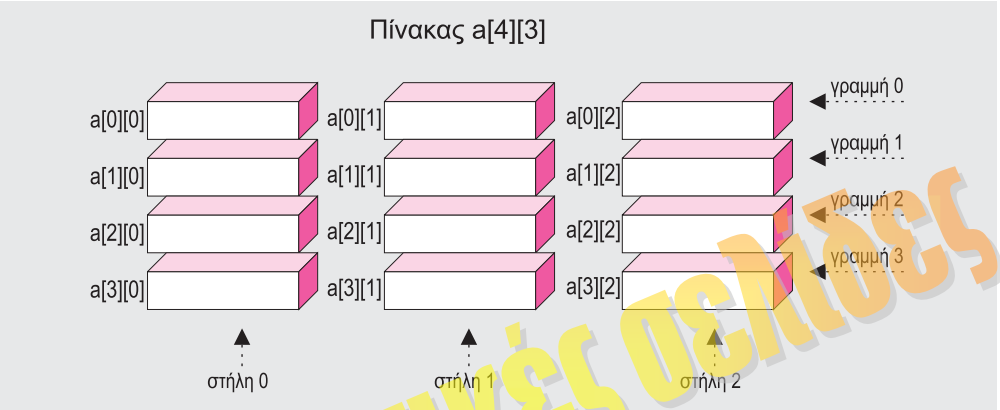
Με τα τρία ένθετα **for** γίνονται όλοι οι συνδυασμοί των **i**, **j**, και **k** (μέσα στα όρια των διαστάσεων του πίνακα), με αποτέλεσμα να προσπελάζονται όλες τις θέσεις του.

Ενδεικτικές σελίδες

Πίνακες δύο διαστάσεων (2Δ)

Ιδιαίτερη αναφορά θα κάνουμε στους πίνακες δύο διαστάσεων, οι οποίοι είναι αυτοί που χρησιμοποιούνται πιο συχνά από τους πολυδιάστατους πίνακες. Φανταζόμαστε έναν πίνακα δύο διαστάσεων σαν μια ομάδα θέσεων μνήμης που είναι διατεταγμένες σε σειρές και στήλες. Κάθε θέση μνήμης έχει δύο αριθμούς αναφοράς, από τους οποίους ο ένας προσδιορίζει τη γραμμή και ο άλλος τη

στήλη στην οποία βρίσκεται η θέση μνήμης. Ας θεωρήσουμε τον παρακάτω πίνακα a, 12 θέσεων με 4 γραμμές και 3 στήλες, ο οποίος ορίζεται από τη δηλωτική πρόταση: **int** a[4][3];



Σχήμα 12.1 *Πίνακας δύο διαστάσεων*

Ενδεικτικές σελίδες

Η προσπέλαση στις θέσεις μνήμης γίνεται με τη χρήση και των δύο αριθμών αναφοράς (έναν για κάθε διάσταση):

a[1][2] ⇨ Προσδιορίζει τη θέση που βρίσκεται στη γραμμή 1 (δεύτερη γραμμή) και στη στήλη 2 (τρίτη στήλη).

Όταν θέλουμε να προσπελάσουμε όλες τις θέσεις μνήμης ενός πίνακα δύο διαστάσεων, χρησιμοποιούμε δύο ένθετους βρόχους for, μέσω των οποίων αλλάζουν οι αριθμοί αναφοράς για κάθε διάσταση. Ο ένας βρόχος αλλάζει την τιμή του αριθμού αναφοράς για την πρώτη διάσταση και ο άλλος για τη δεύτερη. Ο επόμενος κώδικας γεμίζει έναν πίνακα τεσσάρων γραμμών και τριών στηλών με αριθμούς που υπολογίζονται από τις τιμές των μεταβλητών i και j:

```
int main(void)
{
    int a[4][3],i,j;
    for (i=0;i<4;i++)
        for (j=0;j<3;j++)
            a[i][j]=i*j;
    return 0;
}
```

Η μεταβλητή **i** χρησιμοποιείται για να προσδιορίσει τη **γραμμή** του πίνακα.

Η μεταβλητή **j** χρησιμοποιείται για να προσδιορίσει τη **στήλη** του πίνακα.

Στην κάθε θέση μνήμης του πίνακα καταχωρίζεται το γινόμενο του **i** και του **j**. Για παράδειγμα, στη θέση a[3][2] θα καταχωριστεί ο αριθμός 6!

Χειρισμός πινάκων δύο διαστάσεων

Γέμισμα πίνακα 2Δ με τυχαίους αριθμούς

Το πρόγραμμα που ακολουθεί δημιουργεί έναν πίνακα a δέκα γραμμών και πέντε στηλών. Κατόπιν, καταχωρίζει στις θέσεις μνήμης του πίνακα τυχαίους αριθμούς.

Ο εξωτερικός βρόχος for αλλάζει τον αριθμό αναφοράς για τη γραμμή (i), ενώ ο εσωτερικός βρόχος για τη στήλη (j). Επομένως η 'επίσκεψη' στον πίνακα γίνεται ανά γραμμή. Δηλαδή πρώτα γεμίζουν οι θέσεις της πρώτης γραμμής, μετά της δεύτερης, κ.ο.κ.

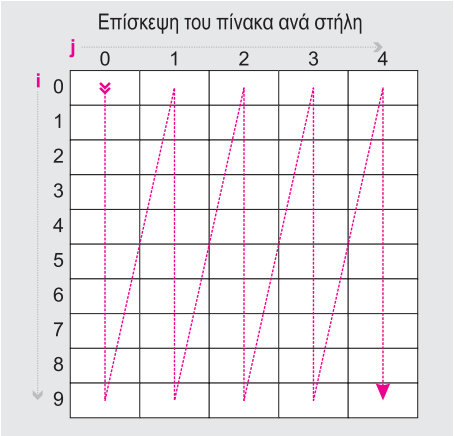
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,j,a[10][5];
    for (i=0;i<10;i++)
        for (j=0;j<5;j++)
            a[i][j]=rand();
    return 0;
}
```

Στην έκδοση που ακολουθεί η επίσκεψη του πίνακα γίνεται ανά στήλη:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,j,a[10][5];
    for (j=0;j<5;j++)
        for (i=0;i<10;i++)
            a[i][j]=rand();
    return 0;
}
```



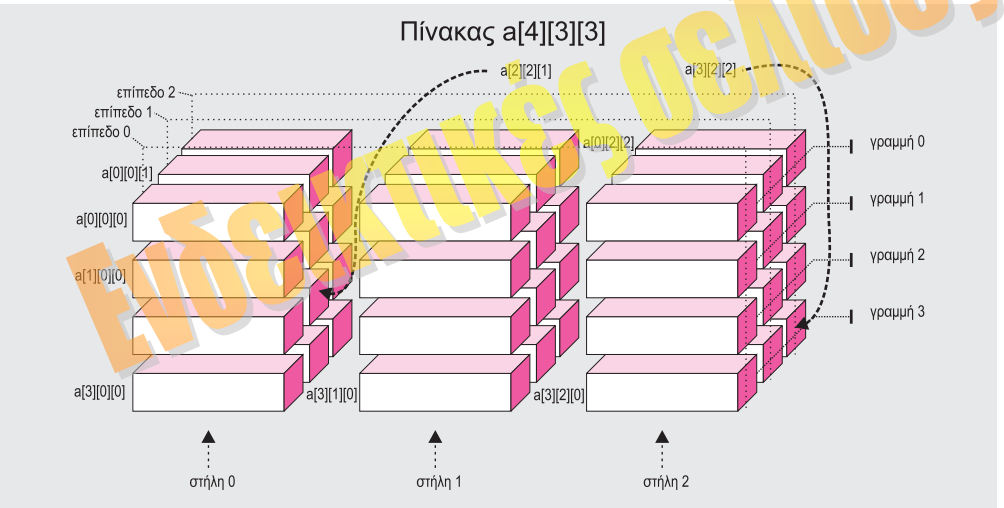
Ενδεικτικές σελίδες

Πίνακες με περισσότερες από δύο διαστάσεις

Αν και στην πραγματικότητα, οι θέσεις ενός πίνακα πολλών διαστάσεων είναι συνεχόμενες, αρκετές φορές είναι πολύ βολικό να έχουμε στο μυαλό μας μια εποπτική και σίγουρα πιο παραστατική εικόνα του πίνακα. Έναν πίνακα τριών διαστάσεων τον φανταζόμαστε σαν θέσεις μνήμης διατεταγμένες σε γραμμές, στήλες, αλλά και επίπεδα.

Στο επόμενο σχήμα, ο πίνακας a τριών διαστάσεων (a[4][3][3]) απεικονίζεται ως θέσεις μνήμης σε διάταξη τεσσάρων γραμμών, τριών στηλών, και τριών επιπέδων.

Ενδεικτικές σελίδες



Σχήμα 12.4 *Απεικόνιση πίνακα τριών διαστάσεων*

Για να αναφερθούμε σε μια θέση μνήμης πρέπει να καθορίσουμε τη γραμμή, τη στήλη, και το επίπεδο στο οποίο βρίσκεται.

Χρησιμοποιώντας τις τρεις διαστάσεις του χώρου, μπορούμε να απεικονίσουμε πίνακες μέχρι τριών διαστάσεων. Πώς μπορούμε όμως να φανταστούμε εποπτικά έναν πίνακα περισσότερων διαστάσεων; Είναι δυνατό να απεικονίσουμε στο χαρτί έναν πίνακα τεσσάρων, πέντε, ή και περισσότερων διαστάσεων;

Η απάντηση είναι ΝΑΙ, αλλά θα πρέπει να ξεχάσουμε τον τρισδιάστατο χώρο που χρησιμοποιούσαμε μέχρι στιγμής, και να σκεφτούμε τις θέσεις μνήμης σαν

Μεταβίβαση πινάκων σε συναρτήσεις

Μεταβίβαση ενός πίνακα 1Δ σε συνάρτηση

Στη C η μεταβίβαση των παραμέτρων μιας συνάρτησης γίνεται πάντα με τη μεταβίβαση κατ' αξία (pass by value). Στην περίπτωση που η παράμετρος είναι ένας πίνακας, τότε μεταβιβάζεται στη συνάρτηση η **διεύθυνση** της πρώτης θέσης μνήμης του πίνακα. Επομένως, η τυπική παράμετρος της συνάρτησης πρέπει να είναι ένας δείκτης του ίδιου τύπου όπως ο τύπος του πίνακα. Το παρακάτω πρόγραμμα καλεί τη συνάρτηση printit() για να εμφανίσει τα περιεχόμενα του πίνακα array.

ΕΝΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

12_random2.c

```
#include <stdio.h>
void printit(int *p);
int main(void)
{
    int array[100];
    ....
    printit(array);
}
void printit(int *p)
{
    int i;
    for (i=0;i<100;i++)
       printf("%d\n",p[i]);
}
```

Η διεύθυνση του πίνακα **array** μεταβιβάζεται στο δείκτη **p**.

Η συνάρτηση **printit()** έχει πρόσβαση στον πίνακα **array** μέσω του δείκτη **p**. Μέσα στη συνάρτηση ο **p** χρησιμοποιείται σαν να ήταν κανονικός πίνακας. Ας μην ξεχνάμε ότι: **p[i]⇔*(p+i)**

Ο συνήθης τρόπος με τον οποίο δηλώνεται η τυπική παράμετρος όπου θα "περάσει" η διεύθυνση ενός πίνακα, είναι ένας πίνακας του ιδίου τύπου, αλλά χωρίς να ορίζεται το μέγεθος του:

```
void printit(int p[])
{
    int i;
    for (i=0;i<100;i++)
       printf("%d\n",p[i]);
}
```

Η τυπική παράμετρος δηλώνεται ως πίνακας χωρίς μέγεθος. Το αποτέλεσμα είναι η έμμεση δημιουργία ενός δείκτη **p**. Ακόμα και αν οριστεί το μέγεθος του πίνακα **p** (π.χ. p[100]), αυτό δεν είναι λάθος, αλλά το μέγεθος δεν λαμβάνεται καθόλου υπόψη!

Εξειδικευμένοι πίνακες

Πίνακες μεταβλητού μήκους (VLAs)

Οι πίνακες μεταβλητού μήκους (Variable Length Arrays ή VLA) προστέθηκαν ως δυνατότητα στη γλώσσα C με το πρότυπο C99.

> Το μέγεθος ενός πίνακα μεταβλητού μήκους προκύπτει από τη τιμή μιας μεταβλητής και καθορίζεται κατά την ώρα της εκτέλεσης του προγράμματος.

Το ακόλουθο πρόγραμμα ζητάει από το χρήστη να πληκτρολογήσει έναν ακέραιο αριθμό, και δημιουργεί έναν πίνακα τόσων θέσεων όσος ο αριθμός που έδωσε ο χρήστης.

12_vla.c

```
#include <stdio.h>
int main(void)
{
    int i,ar;
    printf("Πόσους αριθμούς θα δώσεις ? ");
    scanf("%d",&ar);
    float pin[ar];
    for (i=0;i<ar;i++)
        scanf("%f",&pin[i]);
    return 0;
}
```

Ο αριθμός που πληκτρολογεί ο χρήστης καταχωρίζεται στη μεταβλητή **ar**.

Δημιουργείται ένας πίνακας τόσων θέσεων όση η τιμή της μεταβλητής **ar**.

Η παρακάτω συνάρτηση δημιουργεί έναν τοπικό πίνακα τόσων θέσεων όση η τιμής της παραμέτρου της:

```
void fill_random(int ar)
{
    int i, p[ar];
    for (i=0;i<ar;i++) p[i]=rand();
}
```

Δημιουργείται ο πίνακας **p** τόσων θέσεων όση η τιμή της παραμέτρου **ar**.

Γεμίζει τον πίνακα **p** με τυχαίους αριθμούς.

> Είναι προφανές ότι και στις δύο περιπτώσεις θα πρέπει να υπάρχει η μνήμη που θα δεσμευτεί για τη δημιουργία του πίνακα κατά την ώρα εκτέλεσης. Σε αντίθετη περίπτωση το πρόγραμμα μας θα κρεμάσει!

Ενδεικτικές σελίδες

Πίνακες δεικτών

Στη C μπορούμε να δηλώσουμε πίνακες των οποίων οι θέσεις μνήμης θα είναι δείκτες, δηλαδή θα περιέχουν τις διευθύνσεις άλλων θέσεων μνήμης. Ο πίνακας δεικτών δηλώνεται όπως ένας κανονικός πίνακας, με τη διαφορά ότι μπροστά από το όνομα του πρέπει να υπάρχει ο τελεστής *. Για παράδειγμα, η πρόταση:

```
char *p[5];
```

δηλώνει έναν πίνακα δεκτών σε char, πέντε θέσεων, με όνομα p. Το πρόγραμμα που ακολουθεί χρησιμοποιεί έναν τέτοιον πίνακα για να καταχωρίσει τις διευθύνεις πέντε διαφορετικών συμβολοσειρών.

Ενδεικτικές σελίδες

12_pointer_array.c

```
#include <stdio.h>
int main(void)
{
    int i;
    char *p[5];
    p[0]="ΝΙΚΟΣ";
    p[1]="ΜΑΝΩΛΗΣ";
    p[2]="ΠΑΤΡΟΚΛΟΣ";
    p[3]="ΒΕΝΕΤΙΑ;
    p[4]="ΔΕΣΠΟΙΝΑ";
    for (i=0;i<5;i++)
        printf("%s\n",p[i]);
    return 0;
}
```

Γνωρίζουμε ότι η C αντιμετωπίζει τη συμβολοσειρά ως μία διεύθυνση: τη διεύθυνση στην οποία έχει καταχωριστεί ο πρώτος της χαρακτήρας. Στο παραπάνω πρόγραμμα, σε κάθε θέση μνήμης του πίνακα p καταχωρίζεται η διεύθυνση μιας συμβολοσειράς. Μπορούμε να πούμε τώρα ότι κάθε θέση μνήμης του πίνακα p 'δείχνει' σε μια διαφορετική συμβολοσειρά. Η printf() καλείται πέντε φορές, με αποτέλεσμα να εμφανίσει τις πέντε συμβολοσειρές. Στην printf() κάθε φορά μεταβιβάζεται η διεύθυνση μίας από τις συμβολοσειρές. Οι πίνακες δεικτών έχουν εφαρμογή στη δυναμική κατανομή μνήμης. Θα τους συναντήσουμε επίσης στις παραμέτρους της γραμμής εντολών στο Κεφάλαιο 15.

Συχνά λάθη

- Νομίζουμε ότι σε έναν πίνακα 10 θέσεων, π.χ. a[10], η πρώτη θέση είναι η a[1] και η τελευταία η a[10]. Στη C η πρώτη θέση του πίνακα είναι η θέση a[0], και η τελευταία η a[9].
- Δεν έχουμε κατανοήσει πλήρως ότι το όνομα ενός πίνακα είναι ταυτόχρονα ένας δείκτης ο οποίος απλώς περιέχει μια διεύθυνση.
- Νομίζουμε ότι όταν μεταβιβάζουμε έναν πίνακα σε μια συνάρτηση, η συνάρτηση αποκτά ένα αντίγραφο του πίνακα. Αυτό δεν είναι αληθές! Η συνάρτηση έχει πρόσβαση στον **ίδιο** πίνακα ο οποίος της μεταβιβάστηκε, αλλά μέσω του ονόματος της παραμέτρου της.
- Προσπαθούμε να καταχωρίσουμε μια συμβολοσειρά σε έναν πίνακα χαρακτήρων χρησιμοποιώντας τον τελεστή ανάθεσης =. Αυτό γίνεται μόνο κατά την πρόταση δήλωσης του πίνακα, προσδίδοντας του ταυτόχρονα και αρχικές τιμές. Αν αργότερα θέλουμε να καταχωρίσουμε μια συμβολοσειρά στον πίνακα, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση βιβλιοθήκης strcpy().
- Προσπαθούμε να συγκρίνουμε συμβολοσειρές με τους γνωστούς τελεστές σύγκρισης. Αυτό δεν είναι δυνατόν! Για τη σύγκριση συμβολοσειρών πρέπει να χρησιμοποιούμε τη συνάρτηση βιβλιοθήκης strcmp().

Ενδεικτικές Σελίδες

Ανασκόπηση Κεφαλαίου 12

- Ένας πίνακας είναι μια διάταξη θέσεων μνήμης υπό ένα κοινό όνομα.
- Οι πίνακες που χρησιμοποιούνται πιο συχνά είναι οι πίνακες μίας και δύο διαστάσεων. Μπορούμε όμως να ορίσουμε πίνακες περισσότερων διαστάσεων, ανάλογα με τις ανάγκες μας.
- Ένας πίνακας ορίζει έμμεσα ένα δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- Η δήλωση ενός πίνακα γίνεται όπως και οποιασδήποτε άλλης μεταβλητής, και ακολουθείται από το πλήθος και την τιμή των διαστάσεών του. Π.χ., η πρόταση int a[4][5] δηλώνει έναν πίνακα a, δύο διαστάσεων, με είκοσι θέσεις

(4 × 5), με την πρώτη διάσταση από το 0 μέχρι το 3 και τη δεύτερη από το 0 μέχρι το 4.

- Η αναφορά σε μια θέση ενός πίνακα γίνεται με το όνομά του και τους αριθμούς αναφοράς για την κάθε διάσταση, ώστε να προσδιοριστεί η θέση μνήμης μέσα στον πίνακα. Για παράδειγμα, η a[2][3] αναφέρεται στη θέση μνήμης με αριθμούς αναφοράς 2 και 3 για την πρώτη και δεύτερη διάσταση αντίστοιχα.
- Το όνομα ενός πίνακα αποτελεί ταυτόχρονα και ένα δείκτη με τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα. Ο χειρισμός ενός πίνακα μπορεί να γίνει και με τη χρήση δεικτών.
- Για την αποθήκευση ενός συνόλου χαρακτήρων απαιτείται ένας πίνακας χαρακτήρων. Για παράδειγμα, ο πίνακας char lex[30] επιτρέπει την καταχώριση ενός συνόλου το πολύ 30 χαρακτήρων (μαζί με το χαρακτήρα τερματισμού '\0').
- Πίνακες χαρακτήρων δύο διαστάσεων χρησιμοποιούνται για την καταχώριση πολλών συμβολοσειρών.
- Ο χειρισμός των συμβολοσειρών γίνεται με τις συναρτήσεις βιβλιοθήκης strcpy(), strcmp(), strlen(), strcat(), gets(), puts(), κ.λπ. Στις συναρτήσεις αυτές μεταβιβάζονται οι διευθύνσεις των συμβολοσειρών.
- Ο χειρισμός των πινάκων γίνεται συνήθως με τη χρήση βρόχων for. Για να έχουμε πρόσβαση σε όλες τις θέσεις μνήμης ενός πίνακα, ανάλογα με το πλήθος των διαστάσεών του, χρησιμοποιούνται και αντίστοιχοι ένθετοι βρόχοι for.
- Όταν μεταβιβάζουμε έναν πίνακα ως παράμετρο σε μια συνάρτηση, στην ουσία μεταβιβάζουμε τη διεύθυνση της πρώτης θέσης μνήμης του. Η τυπική παράμετρος πρέπει να είναι ένας πίνακας του ίδιου τύπου, όπου υποχρεωτικά πρέπει να δηλώσουμε τη μέγιστη τιμή των διαστάσεών του πλην της πρώτης. Για παράδειγμα, a[] για τη μεταβίβαση ενός πίνακα μιας διάστασης και a[][4][5] για τη μεταβίβαση ενός πίνακα τριών διαστάσεων.
- Η C, με το πρότυπο C99, υποστηρίζει και πίνακες μεταβλητού μήκους. Το μέγεθος της κάθε διάστασης ενός τέτοιου πίνακα μπορεί να προσδιοριστεί κατά την ώρα εκτέλεσης του προγράμματος.

Ψηφιακές Σελίδες

Ασκήσεις Κεφαλαίου 12

12.1 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα: ★

```
#include <stdio.h>
int main(void)
{
   int a[10]={5,8,9,4,2,1,7,5,6,2},i,m=0,c=0;
   for (i=0;i<10;i++)
   {
      if (a[i]%2!=0) m++;
      if (a[i]>=5) c++;
   }
   printf("%d %d\n",m,c);
   return 0;
}
```

ΕΝΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

12.2 Να γραφεί πρόγραμμα το οποίο να γεμίζει έναν πίνακα 100 θέσεων με τυχαίους ακέραιους αριθμούς. Κατόπιν, το πρόγραμμα να βρίσκει το μικρότερο αριθμό του πίνακα, καθώς και τη θέση στην οποία βρίσκεται μέσα στον πίνακα (π.χ. ο μικρότερος είναι ο 3 και βρίσκεται στη θέση 43 του πίνακα). ★★

12.3 Να γραφεί πρόγραμμα το οποίο να ζητάει από τον χρήστη να πληκτρολογήσει 10 βαθμούς φοιτητών και να τους καταχωρίζει σε κατάλληλο πίνακα. Κατόπιν, το πρόγραμμα να εμφανίζει στην οθόνη τους βαθμούς που είναι μεγαλύτεροι από το μέσο όρο της βαθμολογίας. ★★

12.4 Να γραφεί πρόγραμμα το οποίο να ζητάει 100 αριθμούς, να τους καταχωρίζει σε έναν πίνακα μίας διάστασης, και να εμφανίζει το μέσο όρο τους. ★

12.5 Να γραφεί πρόγραμμα το οποίο να ζητάει 100 αριθμούς και να τους καταχωρίζει σε έναν πίνακα μίας διάστασης. Το πρόγραμμα θα πρέπει να κάνει τους απαραίτητους ελέγχους ώστε οι αριθμοί που δίδονται να είναι σε αύξουσα σειρά (δηλαδή κάθε αριθμός που δίδεται να είναι ίσος ή με-

12.35 Ποια από τα επόμενα αληθεύουν: ★

- ❑ Ένας πίνακας ορίζει έμμεσα και ένα δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- ❑ Σε έναν πίνακα χαρακτήρων μίας διάστασης μπορούν να καταχωριστούν πολλές συμβολοσειρές.
- ❑ Για να έχει μια συνάρτηση πρόσβαση σε έναν πίνακα μίας διάστασης, πρέπει να της διαβιβάσουμε μόνο τη διεύθυνση του πίνακα.
- ❑ Για να έχει μια συνάρτηση πρόσβαση σε έναν πίνακα με περισσότερες από μία διαστάσεις, πρέπει γνωρίζει εκτός από τη διεύθυνση του πίνακα και τις τιμές των διαστάσεών του, εκτός από την πρώτη.
- ❑ Κατά τη δήλωση ενός πίνακα, οι τιμές των διαστάσεών του μπορούν να είναι και μεταβλητές. Π.χ., η int a[b] δημιουργεί έναν πίνακα a με θέσεις μνήμης όσες και η τιμή της μεταβλητής b.

12.36 Με δεδομένο έναν πίνακα a 100 θέσεων μνήμης ο οποίος είναι γεμάτος με τυχαίους αριθμούς, να γραφεί συνάρτηση η οποία να αντιγράφει όλους τους αριθμούς του πίνακα a που είναι μεγαλύτεροι από έναν αριθμό ar σε έναν άλλον πίνακα b (πάλι 100 θέσεων) χωρίς να αφήνει κενές θέσεις. Η συνάρτηση θα επιστρέφει ως τιμή το πλήθος των αριθμών που αντέγραψε. Οι πίνακες a, b, καθώς και ο αριθμός ar θα πρέπει να μεταβιβάζονται ως παράμετροι στη συνάρτηση. ★★

12.37 Η αστυνομική διεύθυνση καταγράφει τις παραβάσεις που κάνουν τα αυτοκίνητα κάθε μήνα σε διάστημα ενός έτους. Να γραφεί πρόγραμμα το οποίο θα διαβάζει τους αριθμούς κυκλοφορίας 100 αυτοκινήτων και θα τους καταχωρίζει σε κατάλληλο πίνακα χαρακτήρων. Επίσης, για κάθε αυτοκίνητο θα διαβάζει τις παραβάσεις που έκανε κάθε μήνα για ένα ολόκληρο έτος και θα τις καταχωρίζει σε πίνακα τύπου int δύο διαστάσεων (100 × 12). Το πρόγραμμα θα πρέπει να εμφανίζει τα ακόλουθα: ★★★

- ➢ Για κάθε αυτοκίνητο, τον αριθμό κυκλοφορίας του και το συνολικό (για όλο το έτος) πλήθος παραβάσεων.
- ➢ Τα αυτοκίνητα με σύνολο παραβάσεων πάνω από 10.
- ➢ Τα αυτοκίνητα με το μικρότερο συνολικό αριθμό παραβάσεων.

Κάθε μία από τις παραπάνω απαιτήσεις να υλοποιείται από κατάλληλη συνάρτηση.

Ψηφιακές Εκδόσεις

Ενδεικτικές σελίδες

Κεφάλαιο

13

Τύποι δεδομένων οριζόμενοι από το χρήστη

Ενδεικτικές σελίδες

Τύποι δεδομένων οριζόμενοι από το χρήστη (User-Defined data types)

Μέχρι τώρα γνωρίσαμε τους τέσσερις βασικούς τύπους δεδομένων της C (int, char, float, και double), παραλλαγές τους (π.χ. long double), καθώς και το νέο τύπο δεδομένων bool που εισάγει το πρότυπο C99.

Ο προγραμματιστής είναι αναγκασμένος να προσαρμόσει τα δεδομένα του, καθώς και το πρόγραμμά του, στους τύπους δεδομένων που διαθέτει μια γλώσσα. Φανταστείτε να μετακομίζαμε το σπίτι μας και οι μόνοι αποθηκευτικοί χώροι που μας διέθετε η μεταφορική εταιρεία να ήταν κουτιά τριών-τεσσάρων συγκεκριμένων μεγεθών. Σε κάθε κουτί θα έπρεπε να αποθηκεύσουμε ένα μόνο αντικείμενο. Είναι προφανές ότι σε κάποια κουτιά θα περίσσευε χώρος, ενώ κάποια αντικείμενα θα έπρεπε να τα τεμαχίσουμε για να χωρέσουν στα κουτιά που μας διέθεσαν. Από την άλλη πλευρά, δεν θα ήταν άραγε πολύ βολικό να φτιάχναμε τα δικά μας κουτιά, στις διαστάσεις που επιθυμούσαμε, ανάλογα με τα αντικείμενα που θέλαμε να μεταφέρουμε; Οι βασικοί τύποι δεδομένων που διαθέτει μια γλώσσα παρομοιάζονται με τα κουτιά συγκεκριμένου μεγέθους που διαθέτει η μεταφορική εταιρεία, ενώ όταν φτιάχνουμε τα δικά μας κουτιά είναι σαν να δημιουργούμε δικούς μας τύπους δεδομένων.

Η C παρέχει πέντε διαφορετικούς τρόπους για να δημιουργήσουμε δικούς μας τύπους δεδομένων (πέρα από τους βασικούς τύπους που έχουμε ήδη γνωρίσει). Οι οριζόμενοι από το χρήστη τύποι δεδομένων (user defined data types) ανήκουν στις επόμενες κατηγορίες:

- **Δομές** (Structures)

  Οι δομές είναι συλλογές από μεταβλητές, οι οποίες αναφέρονται με κοινή ονομασία.

- **Πεδία εύρους ενός ή περισσότερων bit** (Bit fields)

  Είναι μια παραλλαγή δομής η οποία επιτρέπει εύκολη προσπέλαση στα bit ενός byte.

- **Ενώσεις** (Unions)

  Οι ενώσεις επιτρέπουν σε δύο ή περισσότερες μεταβλητές, ακόμη και διαφορετικού τύπου, να μοιράζονται το ίδιο τμήμα μνήμης.

Ενδεικτικός σελίδες

- **Απαριθμήσεις** (Enumerations)

  Οι απαριθμήσεις είναι λίστες συμβόλων.
- **Typedef**

  Η χρήση της **typedef** δημιουργεί ένα συνώνυμο για έναν **υπάρχοντα** τύπο δεδομένων. **Η typedef δεν δημιουργεί νέο τύπο δεδομένων**.

Δομές (structures)

Πολλές φορές στα προγράμματά μας έχουμε δεδομένα που σχετίζονται μεταξύ τους. Για παράδειγμα, το ονοματεπώνυμο, το έτος σπουδών, και οι βαθμοί ενός φοιτητή αποτελούν δεδομένα που αναφέρονται στο ίδιο άτομο. Μέχρι τώρα, αυτό που κάναμε είναι να χρησιμοποιούμε ξεχωριστές μεταβλητές ή πίνακες για την καταχώριση των δεδομένων μας. Για το συγκεκριμένο παράδειγμα θα χρειαζόμασταν έναν πίνακα χαρακτήρων (για το ονοματεπώνυμο), μία ακέραια μεταβλητή (για το έτος σπουδών), και έναν πίνακα πραγματικών αριθμών για τη βαθμολογία του φοιτητή. Η σχέση που έχουν τα δεδομένα αυτά μεταξύ τους, βρίσκεται μόνο μέσα στο μυαλό του προγραμματιστή. Στην πραγματικότητα, δεν υπάρχει καμία δέσμευση ότι αυτά τα δεδομένα αφορούν στον ίδιο φοιτητή.

Η λύση στο παραπάνω πρόβλημα είναι η δημιουργία σύνθετων τύπων δεδομένων, οι οποίοι αποτελούνται από απλούστερους βασικούς τύπους. Οι σύνθετοι αυτοί τύποι λέγονται **δομές** (structures).

Η δομή στη C είναι μια συλλογή από μεταβλητές (ή πίνακες) κάθε τύπου, οι οποίες αναφέρονται με κοινό όνομα. Η δημιουργία μιας δομής δίνει τη δυνατότητα να οριστούν μεταβλητές αυτού του τύπου.

Μια δομή αποτελείται από επιμέρους πληροφορίες, οι οποίες καλούνται πεδία.

Η δομή είναι συνήθως μια λογική ενότητα από πληροφορίες που σχετίζονται μεταξύ τους. Για παράδειγμα, το όνομα, η διεύθυνση, το τηλέφωνο, και η ηλικία ενός ατόμου είναι ένα χαρακτηριστικό παράδειγμα μιας δομής με τέσσερα πεδία:

- το πεδίο του ονόματος,
- το πεδίο της διεύθυνσης,

Ευδοκιμεσοφίες

```
for (t=0;pelatis.address[t]!='\0';t++)
    putchar(pelatis.address[t]);
```

Εμφανίζεται ο **t** χαρακτήρας του πεδίου **address** της μεταβλητής **pelatis**.

Πίνακες από δομές

Με τον ίδιο τρόπο που δηλώνεται μία μεταβλητή τύπου δομής, μπορεί να δηλωθεί και ένας πίνακας δομών:

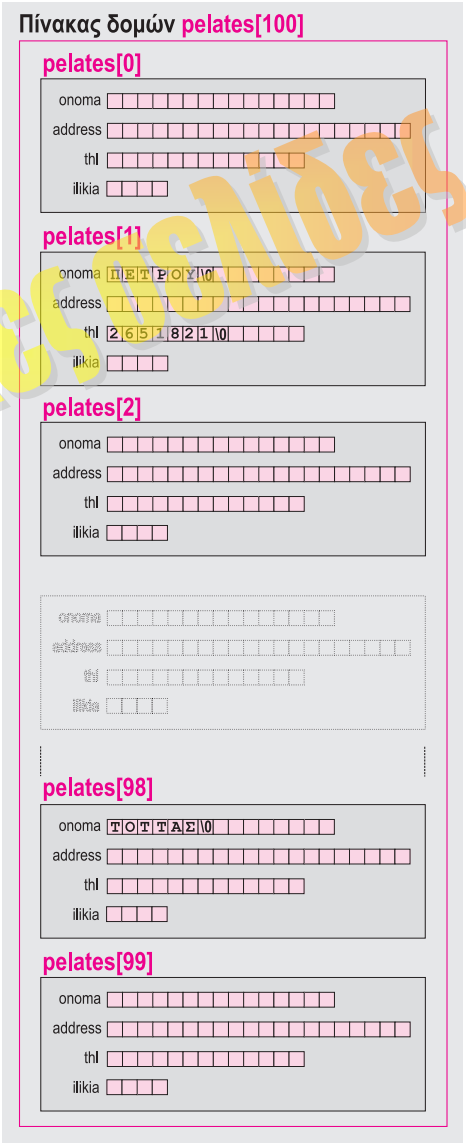
```
struct stoixeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} pelates[100];
```

Μετά από την παραπάνω δήλωση δημιουργείται ένας πίνακας 100 θέσεων με όνομα pelates, ο οποίος αποτελείται από 100 δομές τύπου stoixeia.

Η πρόσβαση στα διαφορετικά στοιχεία-δομές που αποτελούν τον πίνακα γίνεται όπως σε όλους τους πίνακες: Για παράδειγμα, το pelates[2] αναφέρεται στην τρίτη δομή του πίνακα.

Η πρόσβαση σε κάποιο πεδίο μιας δομής γίνεται με τη βοήθεια του τελεστή τελεία (.) όπως στις απλές μεταβλητές δομής: το pelates[2].ilikia αναφέρεται στο πεδίο ilikia της τρίτης δομής του πίνακα.

Οι προτάσεις που ακολουθούν έχουν αποτέλεσμα την καταχώριση των χαρακτήρων όπως εμφανίζονται στο σχήμα.

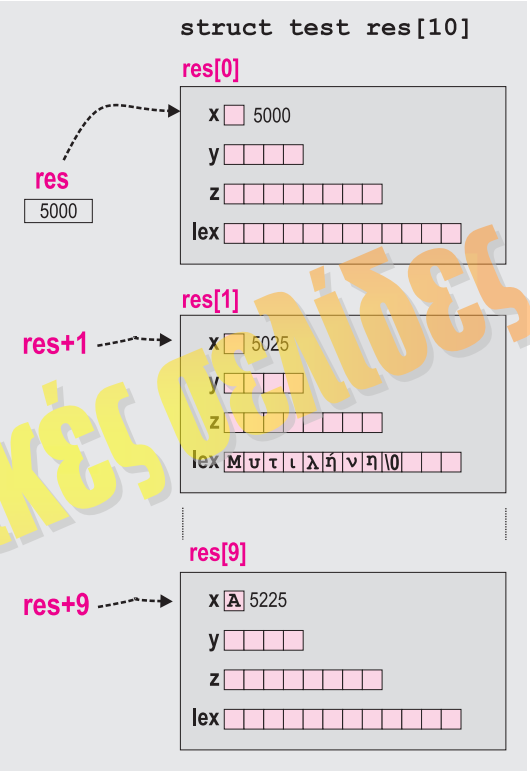


Ενδεικτικές σελίδες

Πίνακες από δομές και δείκτες

Ο ορισμός ενός πίνακα από δομές δημιουργεί ταυτόχρονα και ένα δείκτη με όνομα το όνομα του πίνακα, ο οποίος δείχνει στο πρώτο στοιχείο (δομή) του πίνακα.

Στο διπλανό σχήμα, ο δείκτης res δείχνει στην πρώτη θέση του πίνακα res[0], το res+1 στη δεύτερη θέση res[1], κ.ο.κ.



Η πρόσβαση στα στοιχεία του πίνακα μπορεί να γίνει με δύο τρόπους. Είτε με τη συνήθη χρήση των πινάκων, είτε με τη χρήση της λογικής των δεικτών.

Για παράδειγμα, οι προτάσεις

```
strcpy(res[1].lex,"Μυτιλήνη");
res[9].x='A';
```

και οι προτάσεις

```
strcpy((res+1)->lex,"Μυτιλήνη");
(res+9)->x='A';
```

έχουν ακριβώς το ίδιο αποτέλεσμα. Και στις δύο περιπτώσεις η καταχώριση των χαρακτήρων γίνεται όπως φαίνεται στο σχήμα.

Ενδεικτικές σελίδες

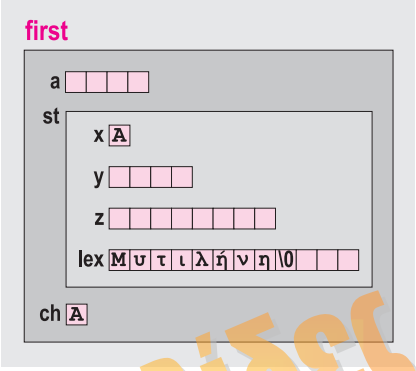
Δομές μέσα σε δομές (ένθετες δομές)

Ένα πεδίο μιας δομής μπορεί να είναι μια άλλη δομή.

```
struct test
{
    char x;
    int y;
```

```
    double z;
    char lex[12];
};

struct nea
{
    int a;
    struct test st;
    char ch;
} first;
```



Στο προηγούμενο παράδειγμα, ένα πεδίο της δομής nea, το st, είναι τύπου δομής test. Το σχήμα δείχνει την εσωτερική απεικόνιση της μεταβλητής first στη μνήμη του Η/Υ.

> Η αναφορά σε ένα πεδίο μιας ένθετης δομής γίνεται με τη χρήση του τελεστή τελεία (.) δύο φορές:

Η ακόλουθη πρόταση:

```
first.st.x='A';
```

καταχωρίζει στο πεδίο x της ένθετης δομής το χαρακτήρα 'A'

ενώ η πρόταση,

```
strcpy(first.st.lex,"Μυτιλήνη");
```

καταχωρίζει τους χαρακτήρες "Μυτιλήνη" στο πεδίο lex της ένθετης δομής st, και η πρόταση,

```
first.ch='A';
```

καταχωρίζει στο πεδίο ch της δομής first το χαρακτήρα 'A'.

Και όμως γίνεται...

Όταν έχουμε δύο μεταβλητές του ιδίου τύπου δομής, μπορούμε να καταχωρίσουμε τα περιεχόμενα των πεδίων της μιας στην άλλη με μια πρόταση της μορφής:

```
st1=st2;
```

Ενδεικτικές σελίδες

Παραδείγματα

Π13.1 Υποθέτουμε ότι έχουμε έναν πίνακα δομών με τη διπλανή γραμμογράφηση. Στον πίνακα είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Η επόμενη συνάρτηση εντοπίζει τη θέση μνήμης του πίνακα που περιέχει τα στοιχεία του μαθητή με συγκεκριμένο όνομα. Η συνάρτηση επιστρέφει ένα δείκτη στη θέση (δομή) στην οποία εντόπισε το όνομα. Αν το όνομα δεν εντοπιστεί, επιστρέφει τιμή NULL.

```
struct stoixeia
{
      char eponymo[30];
      char taxi[5];
      float mesos_oros;
      int ilikia;
} mathites[100];
```

```
struct stoixeia *find_onom(struct stoixeia math[],char onoma[])
{
   int i;

   for (i=0;i<100;i++)
   {
      if (strcmp(math[i].eponymo,onoma)==0)
         return &math[i];

   }
   return NULL;
}
```

Η **strcmp()** συγκρίνει, για κάθε μία θέση μνήμης του πίνακα, το πεδίο **eponymo** με το **onoma** που δόθηκε για αναζήτηση.

Αν εντοπιστεί το όνομα τότε επιστρέφει τη διεύθυνση της θέσης στην οποία βρέθηκε το όνομα, διαφορετικά επιστρέφει NULL.

Η κλήση της συνάρτησης για τον εντοπισμό ενός ονόματος μέσα στον πίνακα mathites γίνεται με τον παρακάτω κώδικα:

13_p13_1&2.c

```
.............
struct stoixeia *ptr;
char lex[30];
.............
printf("Δώσε όνομα για έρευνα:");
gets(lex);
ptr=find_onom(mathites,lex);
if (ptr==NULL)
{
   puts("Το όνομα δεν βρέθηκε");
}
else
```

Η συνάρτηση επιστρέφει ένα δείκτη ο οποίος καταχωρίζεται στη μεταβλητή δείκτη **ptr**.

Ενδεικτικές σελίδες

Ασκήσεις Κεφαλαίου 13

13.1 Να γραφεί πρόγραμμα το οποίο να καταχωρίζει σε μία μεταβλητή δομής τα στοιχεία ενός αυτοκινήτου (αριθμός κυκλοφορίας, χρώμα, κατασκευαστής, κυβικά, ιπποδύναμη). Να επιλέξετε εσείς το σωστό τύπο δομής που πρέπει να χρησιμοποιήσετε. ★★

13.2 Να γραφεί πρόγραμμα το οποίο να καταχωρίζει σε έναν πίνακα από δομές τα στοιχεία για μέχρι 100 αυτοκίνητα (αριθμός κυκλοφορίας, χρώμα, κατασκευαστής, κυβικά, ιπποδύναμη). Να επιλέξετε εσείς το σωστό τύπο δομής που πρέπει να χρησιμοποιήσετε. Η διαδικασία καταχώρισης να σταματάει όταν δοθεί κενό για τον αριθμό κυκλοφορίας. ★★★

13.3 Υποθέτουμε ότι έχουμε έναν πίνακα δομών με τη διπλανή γραμμογράφηση. Στον πίνακα αυτόν είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Να γραφεί συνάρτηση η οποία να εμφανίζει τη λίστα των μαθητών. Η κάθε γραμμή να περιέχει το επώνυμο, την τάξη, και το μέσο όρο του κάθε μαθητή (με ένα δεκαδικό ψηφίο). ★★

```
struct stoixeia
{
      char eponymo[30];
      char taxi[5];
      float mesos_oros;
      int ilikia;
} mathites[100];
```

13.4 Να γραφεί ακόμη μια συνάρτηση, σύμφωνα με τα δεδομένα της προηγούμενης άσκησης, η οποία να επιστρέφει ως τιμή τη μεγαλύτερη από τις ηλικίες των 100 μαθητών. ★★

13.5 Εντοπίστε τα λάθη στο επόμενο πρόγραμμα: ★

```
#include <stdio.h>
enum days {mon,tue,wed,thu,fri,sat,sun} birth_day,today;
typedef int meres;

int main(void)
{
   meres a,b;
   days my_date;
   a=1;
```

ΕΛΛΗΝΙΚΕΣ ΟΔΗΓΙΕΣ

```
    my_date=mon;
    wed=5;
    meres=23;
    today=3;
    birth_day=fri;
    return 0;
}
```

13.6 Ποια από τα επόμενα αληθεύουν: ★

- ❑ Μια δομή δεν μπορεί να περιέχει δύο πεδία με το ίδιο όνομα.
- ❑ Σε μια μεταβλητή τύπου enum δεν μπορούμε να καταχωρίσουμε άλλη τιμή, πέρα από τις τιμές που ορίσαμε στη δήλωση του τύπου enum στον οποίο ανήκει.
- ❑ Για να έχουμε πρόσβαση στο πεδίο μιας δομής μέσω ενός δείκτη, χρησιμοποιούμε τον τελεστή βέλους –>.
- ❑ Όταν μεταβιβάζουμε μια δομή ως παράμετρο σε μια συνάρτηση, η παράμετρος **πρέπει** να είναι του ίδιου τύπου δομής με το όρισμα που θα της μεταβιβάσουμε.
- ❑ Όταν έχουμε δύο μεταβλητές του ιδίου τύπου δομής, μπορούμε με τον τελεστή ανάθεσης (=) να καταχωρίσουμε όλα τα περιεχόμενα της μίας στην άλλη. Για παράδειγμα, η πρόταση filos=pelatis καταχωρίζει όλα τα πεδία της μεταβλητής δομής pelatis στα αντίστοιχα πεδία της μεταβλητής δομής filos.

13.7 Μια χρηματιστηριακή εταιρεία σημειώνει για κάθε μετοχή που αγοράζει ή πουλάει τα εξής στοιχεία: Το όνομα της μετοχής, την ποσότητα, την τιμη, και το σύμβολο 'Α' αν πρόκειται για αγορά, ή το σύμβολο 'Π' αν πρόκειται για πώληση. Να γραφεί πρόγραμμα το οποίο να διαβάζει και να καταχωρίζει σε έναν πίνακα δομών τα στοιχεία για 50 διαφορετικές κινήσεις μετοχών. Το πρόγραμμα θα εμφανίζει: το σύνολο των μετοχών που αγοράστηκαν, το σύνολο των μετοχών που πουλήθηκαν, καθώς και τη συνολική αξία (ποσότητα × τιμή) των μετοχών που διακινήθηκαν. ★★★

13.8 Μια αεροπορική εταιρεία πραγματοποιεί κάθε μέρα 10 πτήσεις με διαφορετικούς τύπους αεροσκαφών. Για κάθε πτήση της, η εταιρεία κρατάει τα εξής στοιχεία: Τον αριθμό της πτήσης, το πλήθος των θέσεων του αερο-

Ενδεικτικές σελίδες

Ενδεικτικές σελίδες

Κεφάλαιο

14

Κανάλια επικοινωνίας - Χειρισμός αρχείων

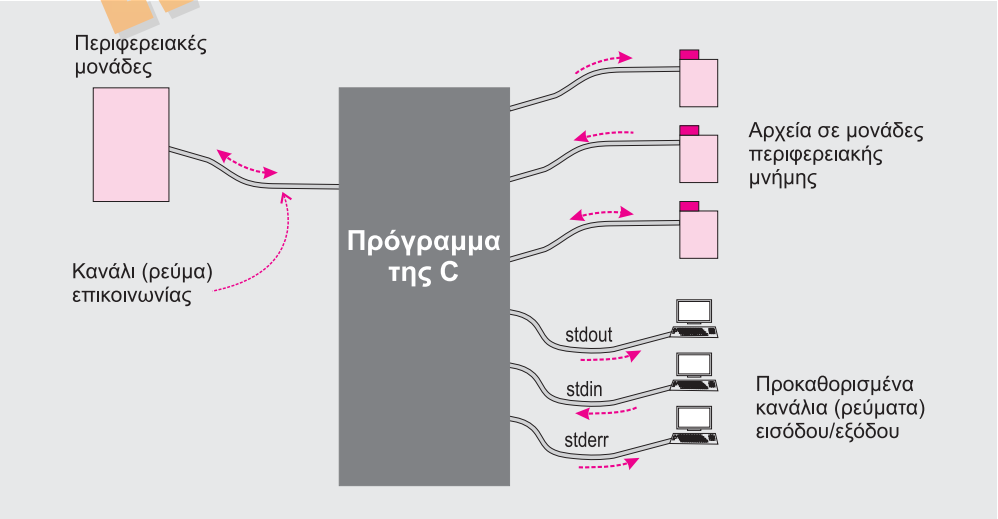
Ενδεικτικές σελίδες

Κανάλια (ρεύματα) επικοινωνίας

Στη C, η επικοινωνία του προγράμματός μας με τις περιφερειακές συσκευές του Η/Υ επιτυγχάνεται μέσω των **ρευμάτων** (streams). Μπορείτε να φανταστείτε το ρεύμα σαν μια λογική διασύνδεση του προγράμματός σας με κάποια από τις συσκευές που απαρτίζουν έναν Η/Υ. Στη C, τόσο η είσοδος όσο και η έξοδος υλοποιείται με ακολουθίες από byte. Το ρεύμα είναι στην πραγματικότητα ένα κανάλι μέσω του οποίου μπορούμε να διαβάσουμε ή να εγγράψουμε (αποστείλουμε) ακολουθίες από byte, από ή προς μια περιφερειακή συσκευή.

Ένα **ρεύμα εισόδου** δέχεται byte από μια περιφερειακή συσκευή, π.χ. από ένα αρχείο στο δίσκο, από το πληκτρολόγιο, ή από μια άλλη συσκευή εισόδου. Ένα **ρεύμα εξόδου** στέλνει μια σειρά από byte σε μια συσκευή εξόδου, όπως στην οθόνη, σε ένα αρχείο, σε έναν εκτυπωτή κ.λπ. Υπάρχουν και τα **ρεύματα εισόδου/εξόδου**, τα οποία μπορούν και να εισάγουν και να εξάγουν πληροφορίες από και προς τη συσκευή με την οποία είναι συνδεδεμένα (π.χ. αρχείο στο δίσκο, μόντεμ, κ.λπ). Μπορούμε να φανταστούμε τα ρεύματα εισόδου και εξόδου της C σαν "κανάλια" μέσω των οποίων το πρόγραμμά μας ανταλλάσει πληροφορίες με τις περιφερειακές συσκευές του συστήματός μας.

Ενδεικτικές Σελίδες



Σχήμα 14.1 *Κανάλια (ρεύματα) εισόδου και εξόδου της C*

Κάποια από αυτά τα "κανάλια" είναι μόνιμα συνδεδεμένα με συγκεκριμένες συσκευές, ενώ κάποια άλλα απαιτούν την εκτέλεση συγκεκριμένων εντολών του προγράμματός μας για να δημιουργηθούν. Τα κανάλια τα οποία είναι μόνιμα συνδεδεμένα με συγκεκριμένες περιφερειακές συσκευές λέγονται **προκαθορισμένα ρεύματα** εισόδου/εξόδου. Με την έναρξη της εκτέλεσης ενός προγράμματος της C "ανοίγουν" αυτόματα τα προκαθορισμένα ρεύματα εισόδου/εξόδου, τα οποία συνδέουν τις τυπικές περιφερειακές συσκευές.

Προκαθορισμένα ρεύματα

Στη C υπάρχουν τρία προκαθορισμένα ρεύματα (κανάλια) τα οποία συνδέουν τις τυπικές (*standard*) συσκευές ενός Η/Υ. Τα ρεύματα αυτά ορίζονται στο stdio.h και είναι:

stdout Τυπική έξοδος (οθόνη)

stdin Τυπική είσοδος (πληκτρολόγιο)

stderr Τυπική έξοδος για μηνύματα λάθους (οθόνη)

Ένα πρόγραμμα της C χρησιμοποιεί ως προεπιλεγμένη (*default*) συσκευή εξόδου την οθόνη και ως προεπιλεγμένη συσκευή εισόδου το πληκτρολόγιο[17].

Πέρα από τα προκαθορισμένα ρεύματα εισόδου/εξόδου, τα ρεύματα (κανάλια) που θα δημιουργήσουμε στα προγράμματά μας μπορούν να συνδεθούν με οποιαδήποτε περιφερειακή συσκευή, η οποία π.χ. μπορεί να είναι ένα αρχείο στο δίσκο μας, ο εκτυπωτής μας, ένα μόντεμ, κ.λπ.

Είδη ρευμάτων

Η C διαθέτει δύο τρόπους διαχείρισης των ρευμάτων: Έναν **χαμηλού επιπέδου** (low-level) και έναν **υψηλού επιπέδου** (high level). Ο πρώτος, χαμηλού επιπέδου, τρόπος αντιμετωπίζει το ρεύμα σαν μια απλή σειρά από byte, και η διαχείρισή του περιορίζεται στην εισαγωγή ή εξαγωγή συγκεκριμένου πλήθους byte από ή προς το συγκεκριμένο ρεύμα. Ο δεύτερος, υψηλού επιπέδου, τρόπος, ο

[17] Οι προκαθορισμένες συσκευές εξόδου και εισόδου του Η/Υ μπορούν να ανακατευθυνθούν από το λειτουργικό σύστημα, και να μην είναι πλέον η οθόνη και το πληκτρολόγιο.

Ενδεικτικές σελίδες

Αρχεία κειμένου (text files)

Τα αρχεία κειμένου είναι αρχεία τα οποία, με τη συνάρτηση fopen(), έχουν ανοιχτεί σε κατάσταση κειμένου. Στα αρχεία κειμένου οι πληροφορίες εγγράφονται σε μορφή χαρακτήρων, όπως δηλαδή θα εμφανίζονται στην οθόνη.

Η εγγραφή και η ανάγνωση πληροφοριών σε αρχεία κειμένου γίνεται μέσω των συναρτήσεων του παρακάτω πίνακα:

Συνάρτηση	Λειτουργία
fputc()	Γράφει ένα χαρακτήρα σε αρχείο
fgetc()	Διαβάζει ένα χαρακτήρα από κάποιο αρχείο
fprintf()	Γράφει όπως η printf, αλλά σε αρχείο
fscanf()	Διαβάζει όπως η scanf, αλλά από αρχείο
fgets()	Διαβάζει ένα σύνολο χαρακτήρων (αλφαριθμητικό) από αρχείο
fputs()	Γράφει ένα σύνολο χαρακτήρων (αλφαριθμητικό) σε αρχείο

Ενδεικτικές σελίδες

fputc()

Η συνάρτηση **fputc()** γράφει ένα χαρακτήρα μέσα σε ένα αρχείο. Το συντακτικό της είναι:

int fputc(int *ch*, FILE **fp*)

όπου ***ch*** ο χαρακτήρας που θα γραφεί στο αρχείο.

fgetc()

Η συνάρτηση **fgetc()** διαβάζει ένα χαρακτήρα από ένα αρχείο. Το συντακτικό της είναι:

int fgetc(FILE **fp*)

Η συνάρτηση επιστρέφει ένα χαρακτήρα "end-of-file" ("τέλος αρχείου") όταν φτάσει στο τέλος του αρχείου. Στο stdio.h έχει ανατεθεί η τιμή του χαρακτήρα "end-of-file" στο σύμβολο EOF.

Όταν διαβάζουμε ένα αρχείο κειμένου χαρακτήρα-χαρακτήρα, ελέγχουμε πάντα για το χαρακτήρα EOF ο οποίος σηματοδοτεί το τέλος του αρχείου. Για παράδειγμα, το επόμενο πρόγραμμα διαβάζει έναν-έναν τους χαρακτήρες του αρχείου **text.txt** και τους εμφανίζει στην οθόνη.

14_read_text_file.c

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    char ch;
    fp=fopen("text.txt","r");
    if (fp==NULL) exit(2);
    while (ch!=EOF)
    {
        ch=fgetc(fp);
        putchar(ch);
    }
    fclose(fp);
    return 0;
}
```

Ανοίγει το αρχείο **text.txt** ως αρχείο κειμένου, για ανάγνωση πληροφοριών.

Διαβάζει έναν-έναν τους χαρακτήρες από το αρχείο μέχρι να διαβάσει το χαρακτήρα EOF.

Κλείνει το αρχείο

Ενδεικτικές σελίδες

fprintf()

Η συνάρτηση **fprintf()** συμπεριφέρεται όπως η printf(), με τη διαφορά ότι, αντί να εμφανίσει τα αποτελέσματά της στην οθόνη, τα γράφει σε αρχείο. Το συντακτικό της, παρόμοιο με της printf(), είναι το ακόλουθο:

int fprintf(FILE ****fp***, char ****αλφ_μορφ*****, *παρ1*****, *παρ2*****, ...)**

Τα αποτελέσματα της fprintf() εγγράφονται στο αρχείο που καθορίζεται από το δείκτη ***fp***. Η fprintf() επιστρέφει τον αριθμό των χαρακτήρων που έγραψε στο αρχείο.

Το πρόγραμμα που ακολουθεί ανοίγει ένα αρχείο κειμένου με όνομα tyxaioi.txt για εγγραφή δεδομένων και στη συνέχεια εγγράφει μέσα στο αρχείο 100 τυχαίους αριθμούς.

14_write_randoms.c

```
#include <stdio.h>
```

```
// Διάβασε την 4η εγγραφή
fread(&eggr,37,1,fp);
// Εμφάνισε τα πεδία της eggr
printf("Ονομα:%s\n",eggr.onoma);
printf("Τηλέφωνο:%s\n",eggr.tilefono);
printf("Ηλικία:%d\n",eggr.ilikia);
```

Η **fread()** διαβάζει ένα τμήμα μήκους 37 byte και το καταχωρίζει στη μνήμη ξεκινώντας από τη διεύθυνση **&eggr**, δηλαδή το καταχωρίζει στη μεταβλητή **eggr**.

Εμφάνιση των πεδίων της εγγραφής.

Ο επόμενος κώδικας ζητάει να πληκτρολογήσουμε στοιχεία για Όνομα, Τηλέφωνο, Ηλικία, τα καταχωρίζει στη μεταβλητή δομής eggr, και τα γράφει στην τέταρτη εγγραφή του αρχείου, διαγράφοντας τα παλαιά της περιεχόμενα.

```
printf("Δώσε όνομα:");
gets(eggr.onoma);
printf("Δώσε τηλέφωνο:");
gets(eggr.tilefono);
printf("Δώσε ηλικία:");
scanf("%d",&eggr.ilikia);
fseek(fp,111,0);
fwrite(&eggr,37,1,fp);
.....
.....
```

Στα πεδία της μεταβλητής **eggr** καταχωρίζονται τα στοιχεία που πληκτρολογούνται.

Η **fseek()** τοποθετεί το δείκτη θέσης στην αρχή της τέταρτης εγγραφής

Η **fwrite()** γράφει στο αρχείο ένα τμήμα από 37 byte, το οποίο διαβάζει από τη διεύθυνση **&eggr**, δηλαδή από τη μεταβλητή **eggr**.

Ενδεικτικές σελίδες

Παραδείγματα

Π14.1 Το επόμενο πρόγραμμα ζητάει λέξεις και τις καταχωρίζει σε ένα αρχείο κειμένου με όνομα lexeis.txt. Το πρόγραμμα σταματάει όταν δοθεί ως λέξη το κενό.

14_p14_1.c

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char lex[40];
    FILE *fp;
    fp=fopen("lexeis.txt","w");
    while (1)
    {
       printf("Δώσε λέξη:");
```

```
        gets(lex);
        if (strcmp(lex,"")==0)
           break;
        fputs(lex,fp);   // Γράφει τη λέξη στο αρχείο
        fputc('\n',fp);  // Επιβάλλει αλλαγή γραμμής στο αρχείο
     }
     fclose(fp);
     return 0;
}
```

Αν αντί για λέξη δώσουμε κενό (δηλαδή πατήσουμε απλώς ένα <ENTER>), η επαναληπτική διαδικασία διακόπτεται.

Π14.2 Υποθέτουμε ότι το αρχείο data1.txt περιέχει έναν άγνωστο πλήθος ακέραιων αριθμών. Το παρακάτω πρόγραμμα βρίσκει το μικρότερο και το μεγαλύτερο αριθμό του αρχείου.

14_p14_2.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int ar,min,max,pl;
    FILE *fp;
    fp=fopen("data1.txt","r");
    pl=0;
    while (!feof(fp))
    {
       fscanf(fp,"%d",&ar);
       pl++;
       if (pl==1)
          min=max=ar;
       if (ar>max) max=ar;
       if (ar<min) min=ar;
    }
    fclose(fp);
    printf("min=%d max=%d\n",min,max);
    return 0;
}
```

Ανοίγει το αρχείο **data1.txt** για ανάγνωση. Ο χειρισμός του αρχείου γίνεται μέσω του δείκτη **fp**.

Διαβάζουμε έναν-έναν όλους τους αριθμούς

Όταν διαβάσουμε τον πρώτο αριθμό (pl==1), τον θέτουμε ως αρχική τιμή στις μεταβλητές **min** και **max**.

Αν ο αριθμός είναι μεγαλύτερος από το **max**, τότε καταχωρίζουμε στο **max** αυτόν τον αριθμό. Αν είναι μικρότερος από το **min**, τότε τον καταχωρίζουμε στο **min**.

Ενδεικτικές σελίδες

Ο χειρισμός του αρχείου γίνεται μέσω του δείκτη fp. Η μεταβλητή pl χρησιμοποιείται για να μετράει τους αριθμούς που διαβάζονται. Στις μεταβλητές min και max τίθεται ως αρχική τιμή ο πρώτος αριθμός που διαβάζεται. Η επαναληπτική διαδικασία τερματίζεται όταν φτάσουμε στο τέλος του αρχείου και η συνάρτηση feof() επιστρέψει τιμή true.

Ενδεικτικές σελίδες

Κεφάλαιο

15

Προχωρημένα θέματα

Ενδεικτικές σελίδες

Προχωρημένα θέματα

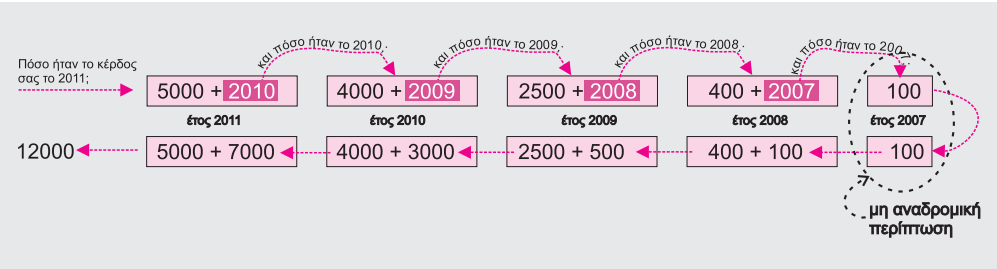
Αναδρομή

Αναδρομή είναι η διαδικασία κατά την οποία ορίζουμε μια λειτουργία σε σχέση με την ίδια τη λειτουργία. Λίγο δυσνόητο, αλλά ας δώσουμε ένα απλό παράδειγμα:

Αν ρωτήσουμε το διευθυντή μιας εταιρείας "*Πόσα ήταν τα κέρδη της εταιρείας σας το 2011*" μπορεί να μας απαντήσει ως εξής: "*Ήταν 5.000€ περισσότερα από το 2010*". Εμείς φυσικά θα ρωτήσουμε: "*Και πόσα ήταν το 2010*" και θα λάβουμε παρόμοια απάντηση, π.χ. "*4.000€ περισσότερα απ' ό,τι το 2009*" κ.ο.κ.

Αυτό θα επαναλαμβάνεται συνέχεια, όχι όμως επ' άπειρο επειδή κάποια στιγμή ο διευθυντής δεν θα μπορεί να μας απαντήσει με τον ίδιο τρόπο, αλλά θα πρέπει να μας πει ότι είχανε συγκεκριμένα κέρδη (π.χ. 100€) διότι θα έχουμε φτάσει πια στη χρονιά που ιδρύθηκε η εταιρεία.

Τότε εμείς (που σημειώναμε τις ερωτήσεις που κάναμε και τις απαντήσεις που παίρναμε) θα μπορέσουμε, ξεκινώντας από την τελευταία απάντηση που πήραμε (τα 100€) και προχωρώντας προς τα πίσω, να υπολογίσουμε τα κέρδη του 2011 που αρχικά ζητήσαμε.



Σχήμα 15.1 *Παράδειγμα αναδρομικής διαδικασίας*

Στο παραπάνω σχήμα υπάρχει μία περίπτωση στην οποία ο διευθυντής δεν μας παραπέμπει στα κέρδη του προηγούμενου έτους, αλλά μας απαντάει συγκεκριμένα. Είναι η τελευταία ερώτηση, που αφορά στο έτος 2007: Μας απαντάει συγκεκριμένα ότι το κέρδος ήταν 100€. Έτσι ξεκαθαρίζουν όλα και μπορούμε να απαντήσουμε στην αρχική ερώτηση γυρνώντας βήμα-βήμα προς τα εμπρός.

Ενδεικτικές σελίδες

Στη C, μια συνάρτηση μπορεί να καλεί τον εαυτό της. Αν στο σώμα μιας συνάρτησης μια πρόταση καλεί την ίδια τη συνάρτηση, τότε η συνάρτηση λέγεται **αναδρομική** (recursive).

☞ Κάθε αναδρομική διαδικασία πρέπει να έχει τουλάχιστον μία **μη αναδρομική περίπτωση**, διαφορετικά θα συνεχίζεται επ' άπειρο.

Ας θεωρήσουμε την επόμενη συνάρτηση, η οποία επιστρέφει ως τιμή το παραγοντικό ενός αριθμού n (1*2*3 .....* n). Για παράδειγμα, το παραγοντικό του 3 είναι 6 (1*2*3) και του 8 είναι 40320 (1*2*3*4*5*6*7*8).

```
int par(int n)
{
    int i,p;
    p=1;
    for (i=1;i<=n;i++)
       p=p*i;
    return p;
}
```

Ενδεικτικές σελίδες

Η προηγούμενη συνάρτηση υπολογίζει (με τον κλασικό τρόπο) και επιστρέφει ως τιμή το παραγοντικό ενός αριθμού n.

Το παραγοντικό (n!) ενός αριθμού n είναι το γινόμενο όλων των ακέραιων αριθμών από το 1 μέχρι το n. Επίσης, το παραγοντικό (n!) ενός αριθμού n ορίζεται ως το γινόμενο του n και του παραγοντικού του προηγούμενου αριθμού, δηλαδή n! = n * (n-1)!, άρα το παραγοντικό του 6 είναι 6 * 5!. Ο υπολογισμός λοιπόν του παραγοντικού μπορεί να γίνει με αναδρομική διαδικασία.

```
int par(int n)
{
    int p;
    if (n==1)
       return 1;
    p=n*par(n-1);
    return p;
}
```

Έλεγχος για μη αναδρομική περίπτωση. Στην περίπτωση αυτή δεν ξανακαλείται η συνάρτηση, αλλά επιστρέφει συγκεκριμένη τιμή (το 1).

Αν ο κώδικας του παραπάνω προγράμματος βρίσκεται στο αρχείο 15_space.c, ο μεταγλωττιστής της C θα δημιουργήσει ένα εκτελέσιμο αρχείο με όνομα 15_space.exe.

Στη γραμμή εντολών του λειτουργικού συστήματος, η χρήση του εκτελέσιμου πλέον αρχείου γίνεται ως εξής:

```
15_space data.txt dataout.txt
```

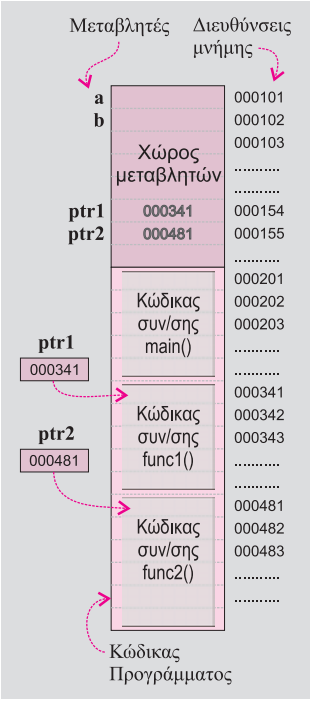
όπου οι παράμετροι data.txt και dataout.txt αποτελούν τα ονόματα των αρχείων εισόδου και εξόδου αντίστοιχα. Το πρόγραμμα θα διαβάσει τους χαρακτήρες από το αρχείο data.txt και θα δημιουργήσει ένα αρχείο dataout.txt, στο οποίο θα γράψει τους ίδιους χαρακτήρες αφού έχει αφαιρέσει όλους τους χαρακτήρες διαστήματος.

ΕΙΔΙΚΕΣ ΕΚΔΟΣΕΙΣ

Δείκτες σε συναρτήσεις !!!

Ένα πρόγραμμα καταλαμβάνει ένα συγκεκριμένο χώρο στην κεντρική μνήμη του Η/Υ. Ο χώρος αυτός χρησιμοποιείται τόσο για τις μεταβλητές, όσο και για το μεταγλωττισμένο κώδικα του προγράμματος.

Επομένως, ο κώδικας κάθε συνάρτησης που χρησιμοποιεί το πρόγραμμά μας καταλαμβάνει ένα συγκεκριμένο χώρο στη μνήμη και ξεκινάει από μια συγκεκριμένη διεύθυνση μνήμης. Η διεύθυνση αυτή αποτελεί τη **διεύθυνση της συνάρτησης**.



Στη C ένας δείκτης μπορεί να δείχνει σε μια συνάρτηση! Ένας δείκτης σε συνάρτηση δεν είναι τίποτα άλλο παρά μια μεταβλητή δείκτη η οποία περιέχει τη διεύθυνση μιας συνάρτησης.

Στο διπλανό σχήμα, οι συναρτήσεις func1() και func2() έχουν διευθύνσεις 000341 και 000481 αντίστοιχα. Αν καταχωριστούν οι διευθύνσεις των συναρτήσεων στις μεταβλητές δείκτη ptr1 και ptr2, θα

μπορούσε να γίνεται κλήση των συναρτήσεων μέσω των δεικτών αυτών, χωρίς τη χρήση του ονόματός τους.

Αν αποθηκεύσουμε τη διεύθυνση μιας συνάρτησης σε μια μεταβλητή δείκτη, θα μπορούμε να καλέσουμε τη συνάρτηση όχι πια με το όνομά της, αλλά χρησιμοποιώντας τον συγκεκριμένο δείκτη (ο οποίος δείχνει στον κώδικα της συνάρτησης στη μνήμη του Η/Υ).

Δήλωση δείκτη σε συνάρτηση

Το επόμενο παράδειγμα δείχνει τον τρόπο με τον οποίο δηλώνεται μια μεταβλητή δείκτη σε συνάρτηση:

```
int (*ptr) (int x, int y);
```

Η πρόταση αυτή δηλώνει μια μεταβλητή δείκτη με όνομα ptr, ο οποίος θα δείχνει σε συναρτήσεις τύπου int με δύο παραμέτρους επίσης τύπου int.

Το γενικό συντακτικό της δήλωσης ενός δείκτη σε συνάρτηση είναι το ακόλουθο:

int (*_όνομα_δείκτη_) (τύπος _παράμετρος1_, τύπος _παράμετρος2_, ...);

Ο τύπος και οι παράμετροι της συνάρτησης που θα δείχνει ο δείκτης.

Το όνομα της μεταβλητής δείκτη.

Ο τύπος των συναρτήσεων που θα δείχνει ο δείκτης.

Ένας δείκτης σε συνάρτηση προορίζεται να δείχνει σε συναρτήσεις με συγκεκριμένο "αποτύπωμα". Το **αποτύπωμα** μιας συνάρτησης αποτελείται από τον τύπο της, καθώς και από το πλήθος και τον τύπο των παραμέτρων της.

Ανάθεση τιμής σε μεταβλητή δείκτη σε συνάρτηση

Η ανάθεση τιμής σε δείκτη σε συνάρτηση γίνεται απλώς με την ανάθεση της διεύθυνσης της συνάρτησης στη μεταβλητή δείκτη, ή με την ανάθεση του ονόματος της συνάρτησης. Η διεύθυνση μιας συνάρτησης αποδίδεται με τη χρήση του τελεστή & πριν από το όνομά της.

Ενδεικτικές σελίδες

```
{
    return x+y;
}
```

Στον παραπάνω κώδικα, για να αντιμετωπιζόταν η συνάρτηση func() ως εμβόλιμη, θα έπρεπε να έχει οριστεί πριν από τη main(). Σε αυτή την περίπτωση θα ήταν περιττή η πρόσθια δήλωσή της.

Συνοψίζοντας, ισχύουν τα παρακάτω βασικά θέματα που αφορούν τις εμβόλιμες συναρτήσεις:

- Το πρόθεμα inline είναι απλώς μια προτροπή στο μεταγλωττιστή να προσπαθήσει να αντιμετωπίσει τη συνάρτηση ως εμβόλιμη.
- Οι εμβόλιμες συναρτήσεις πρέπει να είναι σχετικά μικρές.
- Για να αντιμετωπιστεί μια inline συνάρτηση ως εμβόλιμη, θα πρέπει ο ορισμός της να έχει γίνει πριν από την πρώτη κλήση της.
- Τελικά είναι απόφαση του μεταγλωττιστή αν μια inline συνάρτηση θα την αντιμετωπίσει ως εμβόλιμη ή όχι.

Ενδεικτικές σελίδες

Προγράμματα με πολλά πηγαία αρχεία

Μέχρι τώρα, όλα τα προγράμματα με τα οποία ασχοληθήκαμε αποτελούνταν από ένα μόνο πηγαίο αρχείο. Όμως, ένα πρόγραμμα της C μπορεί να "σπάσει" σε περισσότερα πηγαία αρχεία, κάθε ένα από τα οποία μεταγλωττίζεται ξεχωριστά. Η τεχνική αυτή βοηθάει στη καλύτερη δομή του προγράμματος, και επίσης κάνει πιο εύκολη την αποσφαλμάτωση καθώς και τη συντήρησή του. Για παράδειγμα, η αλλαγή στον κώδικα μιας συνάρτησης θα οδηγήσει στη μεταγλώττιση μόνο του αρχείου που περιέχει τη συγκεκριμένη συνάρτηση, και όχι ολόκληρου του προγράμματος.

Η κατάστρωση ενός προγράμματος στη C περνάει από δύο στάδια: Κατά το στάδιο της **μεταγλώττισης**, ο μεταγλωττιστής μεταφράζει τον πηγαίο κώδικα σε αντικειμενικό κώδικα (object code), δηλαδή σε κώδικα γλώσσας μηχανής ο οποίος μπορεί να εκτελεστεί άμεσα από τον επεξεργαστή.

Κατά το δεύτερο στάδιο, το στάδιο της **σύνδεσης**, το πρόγραμμα σύνδεσης (linker ή συνδετής) συνενώνει όλα τα αρχεία αντικειμενικού κώδικα, μαζί με

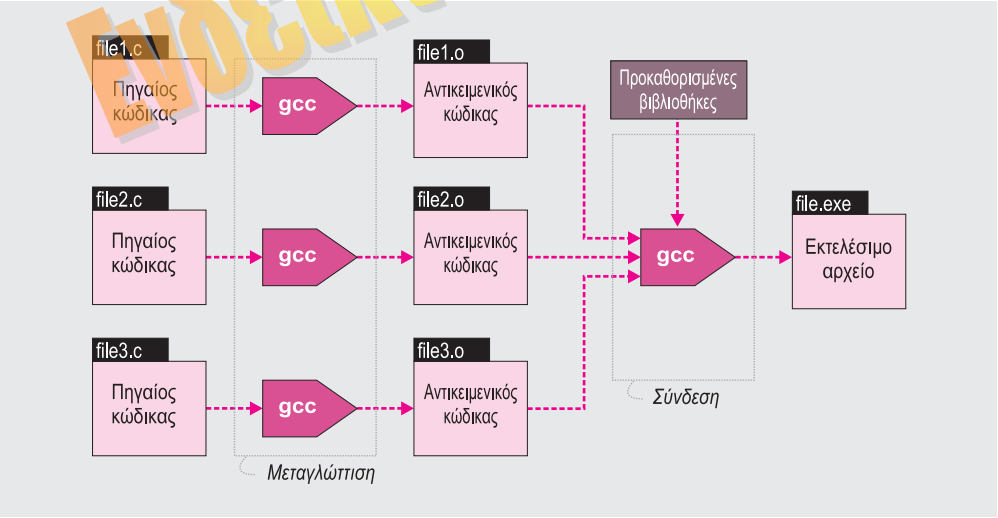
κώδικα που προέρχεται από τις προκαθορισμένες βιβλιοθήκες της γλώσσας, σε ένα ενιαίο εκτελέσιμο αρχείο. Το σχήμα που ακολουθεί δείχνει τα διαφορετικά αυτά στάδια:

Ο μεταγλωττιστής GCC

Το ολοκληρωμένο περιβάλλον DEV C++ , το οποίο πιθανώς να χρησιμοποιήσατε για τη συγγραφή των προγραμμάτων σας, χρησιμοποιεί το μεταγλωττιστή **GCC**. Ο μεταγλωττιστής GCC είναι ένα πρόγραμμα 'δύο σε ένα': Μπορεί να χρησιμοποιηθεί ως μεταγλωττιστής αλλά και ως πρόγραμμα σύνδεσης!

Το σχήμα που ακολουθεί δείχνει εποπτικά τη διαδικασία μεταγλώττισης ενός προγράμματος το οποίο έχει γραφεί σε τρία ξεχωριστά αρχεία πηγαίου κώδικα:

Ενδεικτικές σελίδες



Σχήμα 15.2 *Μεταγλώττιση και σύνδεση ξεχωριστών αρχείων με χρήση του GCC*

Δυστυχώς, για να χρησιμοποιήσετε το μεταγλωττιστή GCC, θα πρέπει να εγκαταλείψετε το εύχρηστο γραφικό περιβάλλον του DEV C++ και να ανοίξετε ένα παράθυρο γραμμής εντολών (command line). Η χρήση του μεταγλωττιστή από

Παραδείγματα

Π15.1 Η επόμενη αναδρομική συνάρτηση δέχεται ως παράμετρο έναν αριθμό και τον εμφανίζει αντίστροφα. Για παράδειγμα, αν δοθεί ο 4672 εμφανίζει τον 2764.

```
int reverse(int n)
{
   if (n==0)
      return;
   else
   {
      printf("%d",n%10);
      return reverse(n/10);
   }
}
```

Μη αναδρομική περίπτωση

Εμφανίζει το υπόλοιπο της διαίρεσης με το 10, δηλαδή το λιγότερο σημαντικό ψηφίο του αριθμού.

Καλεί αναδρομικά τη συνάρτηση με παράμετρο το πηλίκο της ακέραιας διαίρεσης του αριθμού διά του 10.

☞ Η συνάρτηση εμφανίζει το υπόλοιπο της διαίρεσης της τιμής της παραμέτρου της με το 10 (n%10). Αυτό πρακτικά είναι το τελευταίο ψηφίο του αριθμού της παραμέτρου.

☞ Η συνάρτηση καλείται αναδρομικά με το πηλίκο της ακέραιας διαίρεσης της τιμής της παραμέτρου της με το 10 (n/10).

☞ Κάθε αναδρομική συνάρτηση πρέπει να έχει μία τουλάχιστον μη αναδρομική περίπτωση. Στη συγκεκριμένη περίπτωση αυτό συμβαίνει όταν το n==0.

Το πρόγραμμα που ακολουθεί ζητάει έναν αριθμό από το πληκτρολόγιο και χρησιμοποιώντας τη συνάρτηση reverse() τον εμφανίζει αντίστροφα.

15_p15_1.c

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int ar;
   printf("Δώσε αριθμό:");
   // Διαβάζει έναν αριθμό από το πληκτρολόγιο
   scanf("%d",&ar);
   // Εμφάνιση του αριθμού με αντίστροφη σειρά των ψηφίων του
   reverse(ar);
   printf("\n----------\n");
   return 0;
}
```

Εκπαιδευτικές σελίδες

Π15.2 Ο επόμενος τύπος, που προέρχεται από τον Leonard Euler (1707-1783), υπολογίζει προσεγγιστικά το π. Το πρόγραμμα που ακολουθεί χρησιμοποιεί αυτή τη σχέση για να υπολογίσει την τιμή του π. Το πρόγραμμα χρησιμοποιεί αναδρομική συνάρτηση για τον υπολογισμό του δεξιού μέλους της σχέσης.

$$\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}$$

Η συνάρτηση δέχεται ως παράμετρο το **n**. Για τον υπολογισμό χρησιμοποιούνται οι 1000 πρώτοι όροι.

15_p15_2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double melos();

int main(void)
{
   double p;
   p=sqrt(melos(1000)*6);
   printf("p=%f\n",p);
   return 0;
}

double melos(int n)
{
   double res;
   if (n==1)
      return 1.0;
   else
   {
      res=1.0/(n*n)+melos(n-1);
      return res;
   }
}
```

Επιστρέφει την τιμή του δεξιού μέλους της σχέσης για τους 1000 πρώτους όρους.

Υπολογίζει την τιμή του **p (π)** επιλύοντας την παραπάνω σχέση ως προς **p**.

Μη αναδρομική περίπτωση.

Αναδρομική κλήση της συνάρτησης.

Η συνάρτηση θα κληθεί αναδρομικά 999 φορές! Αυτό σημαίνει ότι θα εισαγάγει τις τιμές των τοπικών της μεταβλητών στη στοίβα 999 φορές. Αν η συνάρτηση κληθεί με όρισμα κάποια μεγάλη τιμή (μεγαλύτερη από το 1000 του παραδείγματος), είναι δυνατόν να προξενήσει λάθος υπερχείλισης στοίβας (stack overflow). Δοκιμάστε τη με τιμή 100000.

ΕΛΕΣΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

Ενδεικτικές σελίδες

Κεφάλαιο

16

Αναζήτηση και ταξινόμηση

Ενδεικτικές σελίδες

Αναζήτηση και ταξινόμηση

Η αναζήτηση και η ταξινόμηση αποτελούν τα πλέον "προκλητικά" θέματα στη διαχείριση δεδομένων.

Η αναζήτηση αφορά τον εντοπισμό μιας συγκεκριμένης τιμής σε ένα σύνολο δεδομένων, ενώ η ταξινόμηση αφορά τη διάταξη ενός συνόλου δεδομένων σε αύξουσα ή φθίνουσα σειρά.

Τα σύνολα δεδομένων σε έναν Η/Υ είναι αποθηκευμένα με ένα συγκεκριμένο τρόπο και καλούνται δομές δεδομένων[19]. Η πιο απλή δομή δεδομένων είναι ένας πίνακας.

Η δομή δεδομένων μπορεί να είναι εσωτερική (αποθηκευμένη στην κεντρική μνήμη του Η/Υ) —όπως είναι οι πίνακες, οι συνδεδεμένες λίστες, τα δυαδικά δένδρα κ.λπ.— ή εξωτερική, οπότε αναφερόμαστε σε δεδομένα αποθηκευμένα σε αρχεία.

Είτε πρόκειται για εσωτερικές, είτε για εξωτερικές δομές, οι αλγόριθμοι που χρησιμοποιούνται για την αναζήτηση και την ταξινόμηση είναι ίδιοι. Αυτό που διαφέρει είναι ο τρόπος υλοποίησής τους για κάθε περίπτωση.

Βασικός στόχος κάθε αλγόριθμου αναζήτησης και ταξινόμησης είναι η ταχύτητα. Οι διάφοροι αλγόριθμοι που έχουν σχεδιαστεί διαφοροποιούνται μόνο ως προς την απόδοσή τους. Συνήθως οι πιο απλοί στη σύλληψη αλλά και στην υλοποίηση είναι οι λιγότερο αποδοτικοί. Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά οι πιο "κλασικές" μέθοδοι αναζήτησης και ταξινόμησης.

Σε όλες τις περιπτώσεις τα δεδομένα είναι αποθηκευμένα σε πίνακες. Οι ίδιες όμως μέθοδοι, με μικρές παραλλαγές, μπορούν να χρησιμοποιηθούν και σε δεδομένα αποθηκευμένα σε αρχεία περιφερειακών μονάδων αποθήκευσης.

Στην περίπτωση που έχουμε έναν πίνακα από δομές, η αναζήτηση έγκειται στον εντοπισμό ενός στοιχείου με συγκεκριμένη τιμή ενός πεδίου, ενώ η ταξινόμηση αφορά στη διάταξη των στοιχείων του πίνακα με βάση κάποιο από τα επιμέρους πεδία του.

Ενδεικτικές σελίδες

[19] Στις δομές δεδομένων θα αναφερθούμε διεξοδικά στο Κεφάλαιο 18. Η πιο απλή δομή δεδομένων που υποστηρίζει η C είναι ο πίνακας.

Δυαδική αναζήτηση (binary search)

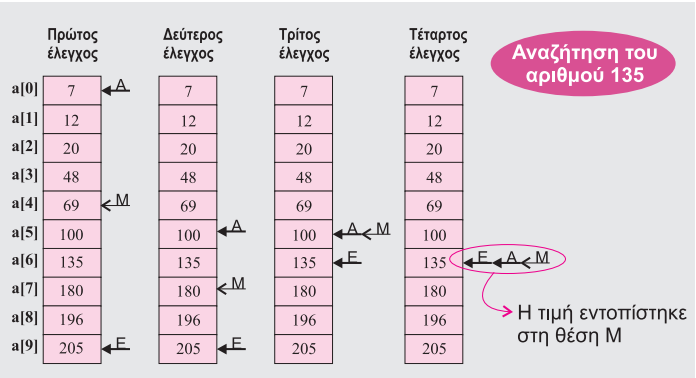
Η **δυαδική αναζήτηση** αποτελεί τον πιο αποδοτικό τρόπο αναζήτησης, ο οποίος όμως προϋποθέτει ότι τα δεδομένα είναι ταξινομημένα είτε κατά αύξουσα, είτε κατά φθίνουσα σειρά.

Φανταστείτε να ψάχνουμε στον τηλεφωνικό κατάλογο για κάποιο όνομα. Δεν θα ήταν καθόλου έξυπνο να αρχίσουμε από την αρχή και να ψάχνουμε ένα-ένα τα ονόματα μέχρι να εντοπίσουμε αυτό που μας ενδιαφέρει. Αντιθέτως, αν ακολουθήσουμε την επόμενη τεχνική, ο εντοπισμός θα είναι πολύ πιο γρήγορος:

Ανοίγουμε τον κατάλογο περίπου στα μισά, βλέπουμε ένα όνομα, και αποφασίζουμε αν αυτό που ψάχνουμε βρίσκεται στο πρώτο μισό τμήμα του καταλόγου ή στο δεύτερο. Αν π.χ. στη σελίδα που ανοίξαμε υπάρχει το όνομα *Παπαδόπουλος* και εμείς ψάχνουμε το *Ταμβακέλλης*, προφανώς το όνομα που ψάχνουμε θα βρίσκεται στο δεύτερο κομμάτι του καταλόγου, μετά από το *Παπαδόπουλος*. Παίρνουμε τώρα το δεύτερο κομμάτι του καταλόγου και κάνουμε ακριβώς τα ίδια. Εντοπίζουμε δηλαδή μια σελίδα στα μισά του και βλέπουμε σε ποιο τμήμα βρίσκεται το όνομα που ψάχνουμε κ.ο.κ. Αυτό γίνεται συνέχεια, κόβοντας στα δύο τα κομμάτια που ψάχνουμε, μέχρι να καταλήξουμε σε μία μόνο σελίδα. Αυτή η σελίδα ή θα περιέχει το όνομα που ψάχνουμε, ή θα συμπεράνουμε ότι το όνομα δεν υπάρχει στον κατάλογο.

Ηλεκτρονικές σελίδες

Θεωρούμε το διπλανό πίνακα a, 10 θέσεων, ο οποίος περιέχει ακέραιους αριθμούς με αύξουσα ταξινόμηση, και αναζητούμε μέσα στον πίνακα τον αριθμό 135.



Η αναζήτηση γίνεται πάντα σε ένα τμήμα του πίνακα που ξεκινάει από τη θέση που δείχνει το A (Από) μέχρι τη θέση που δείχνει το E (Έως). Αρχικά τα A και E δείχνουν στην πρώτη (0) και στην τελευταία θέση (9) του πίνακα αντίστοιχα.

```
        else if (ar>a[meson])
            return rbinary(a,meson+1,eos,ar);
        else
            return meson;
    }
    else
        return -(apo + 1);
}
```

Μη αναδρομική περίπτωση.

Μη αναδρομική περίπτωση.

☞ Οι παραπάνω συναρτήσεις προϋποθέτουν ότι ο πίνακας a είναι ταξινομημένος με αύξουσα σειρά. Με ελάχιστες τροποποιήσεις, μπορούν να εφαρμοστούν και σε πίνακα με φθίνουσα ταξινόμηση. **Σε κάθε περίπτωση, όμως, για να εφαρμοστεί η δυαδική αναζήτηση, τα δεδομένα πρέπει να είναι ταξινομημένα.**

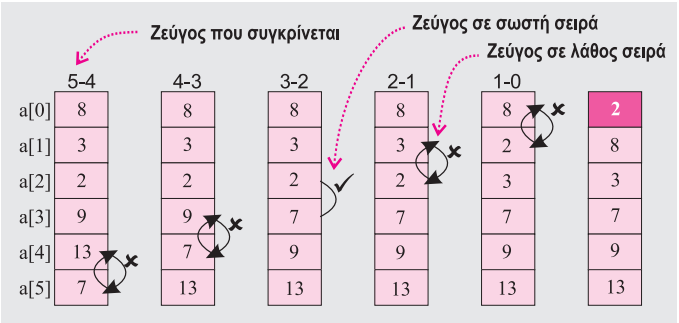
☞ Στη δυαδική αναζήτηση, το μέγιστο πλήθος αναζητήσεων σε n στοιχεία είναι $\log_2 n$. Για παράδειγμα, σε 8 στοιχεία είναι 3 ($\log_2 8=3$), ενώ σε 1024 στοιχεία είναι 10 ($\log_2 1024=10$).

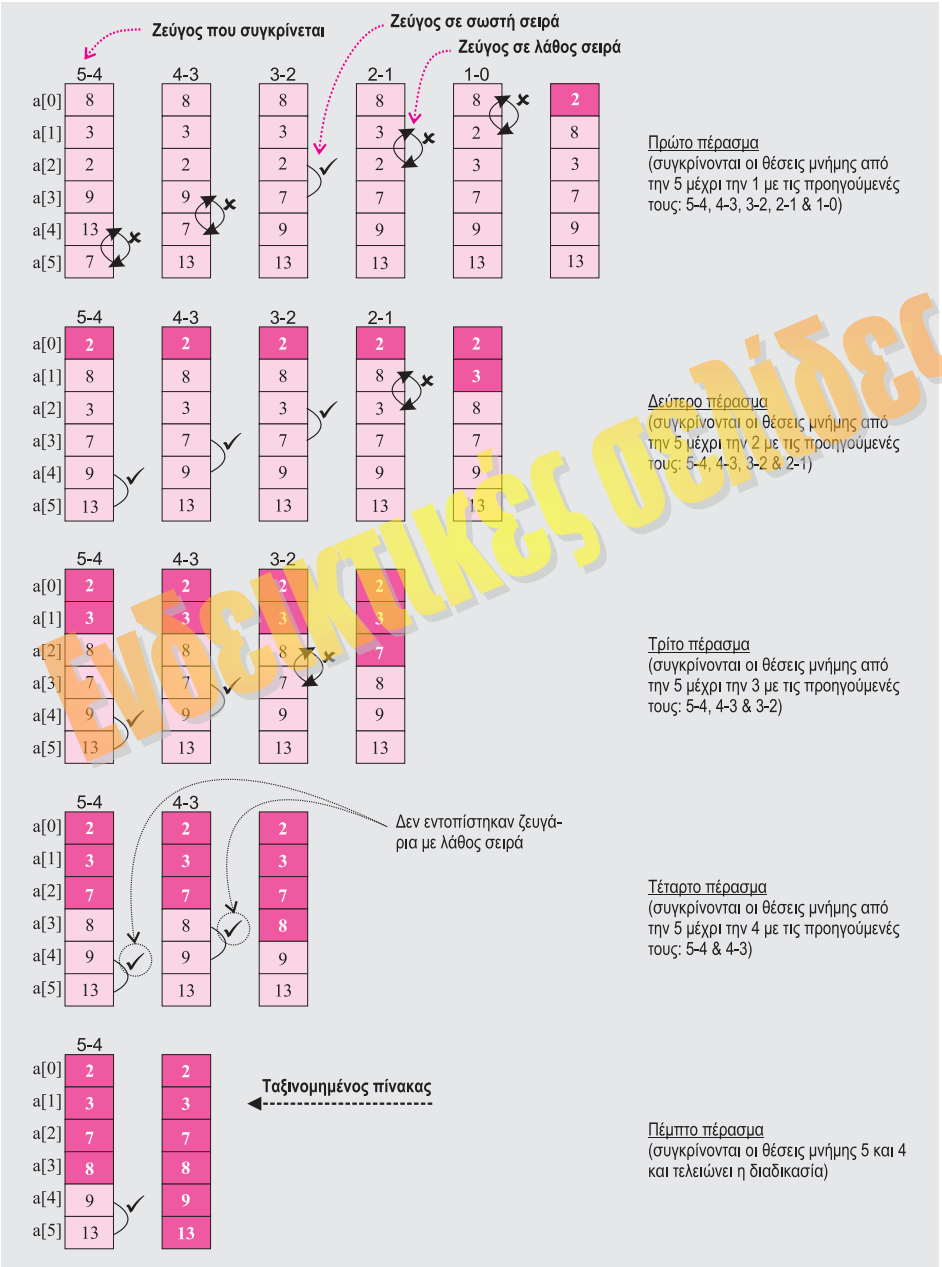
Ενδεικτικές σελίδες

Ταξινόμηση φυσαλίδας (bubble sort)

Η ταξινόμηση φυσαλίδας είναι ίσως ο πιο απλός αλγόριθμος ταξινόμησης. Η λογική στην οποία στηρίζεται είναι η εξής:

Ξεκινάμε από το τέλος του πίνακα και ελέγχουμε ανά ζεύγη τα γειτονικά κελιά του (το κάθε κελί με το προηγούμενο του). Αν είναι σε διαφορετική σειρά (από αυτή που θέλουμε) αντιμεταθέτουμε τα περιεχόμενα τους, διαφορετικά τα αφήνουμε όπως είναι. Μετά το πρώτο πέρασμα, στη κορυφή του πίνακα θα βρεθεί η μικρότερη τιμή. Το επόμενο "πέρασμα" γίνεται για τις υπόλοιπες θέσεις του πίνακα (πλην της πρώτης,) και στο τέλος του θα βρεθεί στη δεύτερη θέση του πίνακα η επόμενη μικρότερη τιμή. Κάθε φορά ο έλεγχος εφαρμόζεται σε ολοένα μικρότερο τμήμα του πίνακα, με αποτέλεσμα κάθε φορά η μικρότερη τιμή να μετακινείται στη κορυφή





Ενδεικτικές σελίδες

Σχήμα 16.1 *Ταξινόμηση πίνακα με τη μέθοδο της "φυσαλίδας" (bubble sort)*

Συχνά λάθη

- Προσπαθούμε να κάνουμε δυαδική αναζήτηση σε πίνακα που δεν είναι ταξινομημένος.
- Η ταξινόμηση συμβολοσειρών προϋποθέτει τη σύγκριση τους. Κατά λάθος χρησιμοποιούμε τους συγκριτικούς τελεστές > και <, αντί της συνάρτησης strcmp() που αποτελεί το μόνο τρόπο σύγκρισης δύο αλφαριθμητικών.

Ανασκόπηση Κεφαλαίου 16

- Αναζήτηση είναι η διαδικασία εντοπισμού ενός στοιχείου σε μια δομή δεδομένων.
- Στη σειριακή αναζήτηση, ο έλεγχος των δεδομένων γίνεται με τη σειρά από το πρώτο μέχρι το τελευταίο, έως ότου να γίνει ο εντοπισμός του προς αναζήτηση στοιχείου.
- Η δυαδική αναζήτηση μπορεί να εφαρμοστεί μόνο σε ταξινομημένα δεδομένα.
- Ταξινόμηση είναι η διαδικασία κατά την οποία γίνεται η διάταξη κάποιων δεδομένων σε αύξουσα ή φθίνουσα σειρά.
- Η ταξινόμηση αριθμητικών δεδομένων με αύξουσα σειρά διατάσσει τα δεδομένα από το μικρότερο προς το μεγαλύτερο, ενώ με φθίνουσα σειρά τα διατάσσει αντίστροφα.
- Η ταξινόμηση αλφαριθμητικών δεδομένων διατάσσει τα δεδομένα με απόλυτη αλφαβητική σειρά.
- Υπάρχουν διάφοροι αλγόριθμοι ταξινόμησης, με διαφορετική πολυπλοκότητα και απόδοση.
- Ο αλγόριθμος φυσαλίδας (bubble sort) είναι ο πιο δημοφιλής και απλός αλγόριθμος ταξινόμησης, ιδανικός για μικρό πλήθος δεδομένων.
- Ο αλγόριθμος quick sort είναι ο πιο πολύπλοκος, αλλά και ο πιο αποδοτικός αλγόριθμος ταξινόμησης, ιδιαίτερα όταν έχουμε μεγάλο πλήθος δεδομένων με τυχαία διασπορά τιμών.

Ενδεικτικές σελίδες

Ασκήσεις Κεφαλαίου 16

16.1 Να γραφεί πρόγραμμα το οποίο να ζητάει μια λέξη, να ταξινομεί τους χαρακτήρες της λέξης κατά αλφαβητική σειρά, και να την εμφανίζει στην οθόνη. Για παράδειγμα, αν πληκτρολογηθεί η λέξη ΑΝΑΝΑΣ θα εμφανίσει τη λέξη ΑΑΑΝΝΣ. ★★

16.2 Να γραφεί πρόγραμμα (ή συνάρτηση) που να ταξινομεί κατά αύξουσα αλφαβητική διάταξη, **ως προς το δεύτερο χαρακτήρα** τους, τις συμβολοσειρές ενός πίνακα χαρακτήρων δύο διαστάσεων με 100 γραμμές και 40 στήλες. ★★★

16.3 Αν υποθέσουμε ότι στον πίνακα χαρακτήρων lex υπάρχει καταχωρισμένη η λέξη "ΧΑΡΟΥΜΕΝΟΣ", ποιο θα είναι το περιεχόμενο του πίνακα lex μετά από την εκτέλεση του επόμενου τμήματος κώδικα; ★★

```
int i=0,p1,p2;
char ch1,ch2,temp;
ch1=ch2=lex[0];
p1=p2=0;
while (lex[i]!='\0')
{
   if (ch1>lex[i])
   {
      ch1=lex[i];
      p1=i;
   }
   if (ch2<lex[i])
   {
      ch2=lex[i];
      p2=i;
   }
   i++;
}
temp=lex[0];
lex[0]=lex[p2];
lex[p2]=temp;
temp=lex[i-1];
lex[i-1]=lex[p1];
lex[p1]=temp;
```

Ενδεικτικές σελίδες

16.4 Να γραφεί συνάρτηση η οποία θα προσθέτει ένα χαρακτήρα, παρεμβάλλοντάς τον στη σωστή του θέση, μέσα σε έναν ταξινομημένο πίνακα χαρακτήρων. Για παράδειγμα, αν ο πίνακας περιέχει τους χαρακτήρες "ΑΓΓΔΚΜΧ" και ο χαρακτήρας που πρόκειται να προστεθεί είναι ο 'Ε', η συνάρτηση θα έχει αποτέλεσμα να περιέχει ο πίνακας τους χαρακτήρες "ΑΓΓΔΕΚΜΧ". Η συνάρτηση να δέχεται δύο παραμέτρους: τον πίνακα χαρακτήρων και το χαρακτήρα που πρόκειται να προστεθεί. ★★★

16.5 Ποια από τα επόμενα αληθεύουν: ★

- ❑ Η δυαδική αναζήτηση προϋποθέτει ταξινομημένα δεδομένα.
- ❑ Οι διαδικασίες ταξινόμησης είναι γενικά χρονοβόρες σε μεγάλους πίνακες.
- ❑ Σε ένα μεγάλο πίνακα με τυχαίους αριθμούς, η καλύτερη μέθοδος ταξινόμησης είναι η μέθοδος της φυσαλίδας.
- ❑ Σε ένα μισοταξινομημένο πίνακα η καλύτερη μέθοδος ταξινόμησης είναι η *quick sort*.
- ❑ Σε έναν πίνακα με τυχαίους αριθμούς, η μόνη μέθοδος αναζήτησης είναι η σειριακή.
- ❑ Η μέθοδος ταξινόμησης φυσαλίδας βασίζεται στη σύγκριση και αντιμετάθεση γειτονικών θέσεων μνήμης.
- ❑ Η μέθοδος ταξινόμησης *quick sort* βασίζεται στο διαχωρισμό των θέσεων μνήμης του πίνακα με βάση μια τιμή διαχωρισμού.

16.6 Να γραφεί πρόγραμμα το οποίο να ζητάει ονόματα και να τα καταχωρίζει σε έναν πίνακα συμβολοσειρών. Το πρόγραμμα να έχει τη δυνατότητα καταχώρισης το πολύ 50 ονομάτων, αλλά να σταματά όταν αντί για όνομα πατηθεί απλώς <enter>. Τέλος, το πρόγραμμα να εμφανίζει τα ονόματα που δόθηκαν με απόλυτη αλφαβητική σειρά. ★★★

16.7 Δέκα αθλητές προκρίθηκαν για τον τελικό του αγωνίσματος ρίψης ακοντίου. Κάθε αθλητής ρίχνει έξι βολές. Να γραφεί πρόγραμμα το οποίο να ζητάει τα ονόματα και τις ρίψεις των δέκα αθλητών και να τα καταχωρίζει σε κατάλληλους πίνακες. Το πρόγραμμα να εμφανίζει μια λίστα με τα ονόματα των αθλητών καθώς και την καλύτερη βολή τους με φθίνουσα

Ενδεικτικές σελίδες

Κεφάλαιο

17

Δυναμική διαχείριση μνήμης

Ενδεικτικές σελίδες

Δυναμική διαχείριση μνήμης

Η μέχρι στιγμής εκμετάλλευση της μνήμης από τη C γίνεται είτε με τη χρήση μεταβλητών, είτε με τη χρήση πινάκων. Οι μεταβλητές και οι πίνακες μπορεί να είναι τριών κατηγοριών, ανάλογα με τον τρόπο και τη θέση που δηλώνονται:

- Καθολικές
- Στατικές
- Τοπικές

Οι καθολικές και οι στατικές μεταβλητές καταλαμβάνουν συγκεκριμένο χώρο στη μνήμη και διατηρούν τα περιεχόμενά τους σε όλη τη διάρκεια του προγράμματος.

Η κατανομή της μνήμης των καθολικών και των στατικών μεταβλητών λέγεται **στατική κατανομή**, και οι θέσεις μνήμης δεσμεύονται κατά τη στιγμή της μεταγλώττισης.

Οι τοπικές μεταβλητές (που λέγονται επίσης *αυτόματες* μεταβλητές) δεσμεύουν χώρο στη μνήμη και διατηρούν το περιεχόμενο τους όσο διαρκεί η εμβέλειά τους. Για παράδειγμα, όταν μια τοπική μεταβλητή δηλωθεί μέσα σε μια συνάρτηση, δεσμεύει χώρο μνήμης ανάλογα με τον τύπο της. Όταν η συνάρτηση επιστρέφει, ο χώρος που δέσμευε η μεταβλητή απελευθερώνεται και τα περιεχόμενά της χάνονται.

Η κατανομή της μνήμης των τοπικών μεταβλητών λέγεται **αυτόματη κατανομή**, και οι θέσεις μνήμης δεσμεύονται προσωρινά (κατά τη δήλωσή τους) και αποδεσμεύονται μόλις λήξει η εμβέλεια τους.

Έτσι, οι προτάσεις

```
char a,b;
int num[100];
```

δεσμεύουν δύο byte, για τις μεταβλητές a και b, και 400 byte (4 × 100) για τον πίνακα num. Οι θέσεις αυτές δεσμεύονται τη στιγμή της δήλωσης και αποδεσμεύονται μόλις λήξει η εμβέλεια των μεταβλητών.

Ενδεικτικές σελίδες

Αν όμως, για παράδειγμα, θέλουμε έναν πίνακα στον οποίο να καταχωρίζουμε τους αριθμούς των αυτοκινήτων που περνάνε μπροστά από ένα σημείο παρατήρησης, το ερώτημα είναι πόσο μεγάλος πρέπει να είναι αυτός ο πίνακας.

Αν τον ορίσουμε μικρό (π.χ. με 100 θέσεις), είναι πιθανό να μην επαρκεί επειδή μπορεί να περάσουν περισσότερα αυτοκίνητα. Αν τον ορίσουμε αρκετά μεγάλο, προβλέποντας (αν αυτό είναι δυνατόν) τη μέγιστη κίνηση που θα μπορούσαμε να έχουμε (π.χ. με 1000 θέσεις), το πιθανότερο είναι να σπαταλήσουμε άσκοπα μνήμη την οποία δεν θα χρησιμοποιήσουμε ποτέ.

Πρέπει να έχουμε υπόψη ότι δεν είναι δυνατό να μεταβάλουμε το μέγεθος ενός πίνακα. Ο πίνακας έχει πάντα το μέγεθος που ορίζουμε με την πρόταση δήλωσής του.

Δυναμικές σελίδες

Ένα άλλο πρόβλημα προκύπτει όταν δεν γνωρίζουμε το μέγεθος του πίνακα όταν γράφουμε το πρόγραμμα, αλλά το γνωρίζουμε κατά την ώρα εκτέλεσης του προγράμματος. Για παράδειγμα, μπορεί το πρόγραμμα να ζητάει τον αριθμό των μαθητών μιας τάξης και μετά να θέλουμε να φτιάξουμε έναν πίνακα με τόσες θέσεις όσες ο αριθμός των μαθητών. Στις αρχικές εκδόσεις της C, δεν ήταν δυνατόν να δηλώσουμε έναν πίνακα που οι διαστάσεις του να καθορίζονται από μεταβλητές. Όμως, με το πρότυπο C99 και τους πίνακες μεταβλητού μήκους που ήδη έχουμε συναντήσει, προτάσεις όπως η επόμενη:

int num[a];

είναι αποδεκτές, και υπάρχει δυνατότητα οι διατάσεις, και επομένως το μέγεθος ενός πίνακα, να καθορίζονται από τιμές μεταβλητών κατά τη στιγμή της δήλωσης του πίνακα (δείτε *Πίνακες μεταβλητού μήκους* σελ. 505). Και πάλι όμως, από τη στιγμή που θα δηλωθεί ο πίνακας, με όποιες τιμές και να έχουν οι διαστάσεις του, δεν μπορούμε αργότερα να μεταβάλουμε το μέγεθος του.

Η λύση βρίσκεται στο μηχανισμό δυναμικού καταμερισμού μνήμης που διαθέτει η C.

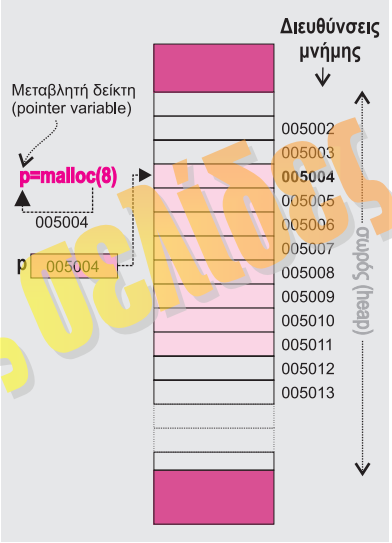
Η C μας εφοδιάζει με ένα μηχανισμό **δυναμικής κατανομής** μνήμης, καθώς και με συναρτήσεις μέσω των οποίων αυτός υλοποιείται. Μπορούμε να δεσμεύουμε περισσότερη μνήμη όταν την έχουμε ανάγκη, ή να απελευθερώνουμε μνήμη την οποία πλέον δεν χρειαζόμαστε.

Η συνάρτηση malloc()

#include <stdlib.h>

void *malloc(unsigned int *size*)

Η συνάρτηση malloc() δεσμεύει από το σωρό ένα τμήμα (μπλοκ) μνήμης, μεγέθους (σε byte) όσο η τιμή της παραμέτρου ***size***. Η malloc() επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του μπλοκ που δέσμευσε.



> Η συνάρτηση malloc() διαθέτει μία παράμετρο τύπου unsigned int και η τιμή που επιστρέφει είναι ένας δείκτης (pointer) τύπου void στο τμήμα το οποίο δέσμευσε.

Στην περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για το μέγεθος του τμήματος, η malloc() επιστρέφει δείκτη NULL. **Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση.**

Ενδεικτικές σελίδες

Ο συνήθης τρόπος χρήσης της malloc() φαίνεται στο επόμενο τμήμα κώδικα:

```
char *p;
p=(char *)malloc(8);
if (p==NULL)
{
    puts("Δεν υπάρχει αρκετή μνήμη");
    exit(2);
}
gets(p);
putchar(p[3]);
```

- Μετατροπή του δείκτη **void** που επιστρέφει η **malloc()** σε δείκτη προς δεδομένα τύπου **char**. Όπως θα δούμε αργότερα , η μετατροπή τύπου μπορεί να παραλειφθεί.
- Έλεγχος του δείκτη που επέστρεψε η **malloc()**.
- Καταχώριση των χαρακτήρων που πληκτρολογούνται, στο χώρο μνήμης που δείχνει ο δείκτης **p**.
- Εμφάνιση του 4ου χαρακτήρα από αυτούς που πληκτρολογήθηκαν. Ο χώρος που δεσμεύτηκε χρησιμοποιείται ως πίνακας, με όνομα το όνομα του δείκτη **p**!

Αν θυμηθούμε λίγο τη φιλοσοφία των δεικτών και τη σχέση τους με τους πίνακες, οι παραστάσεις p[3] και *(p+3) είναι ισοδύναμες και αναφέρονται στο τέταρτο από τα δεδομένα που ξεκινάνε από τη θέση όπου δείχνει ο δείκτης p.

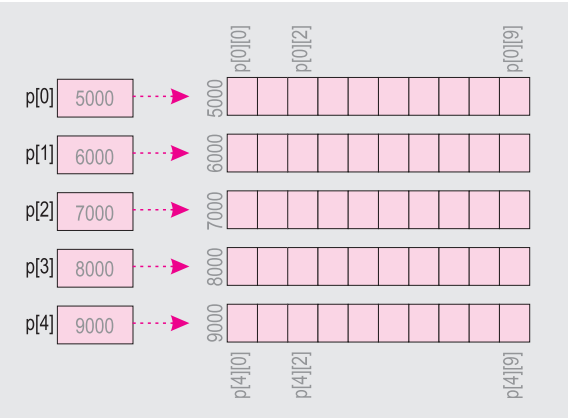
Το πρόγραμμα αρχικά ζητάει να πληκτρολογήσει ο χρήστης το μέγεθος του πίνακα. Κατόπιν, με χρήση της calloc(), δεσμεύει το χώρο μνήμης που απαιτεί ένας πίνακας ακεραίων του μεγέθους που έδωσε ο χρήστης. Η διεύθυνση που επιστρέφει η calloc() καταχωρίζεται στο δείκτη p. Ο δείκτης p αντιμετωπίζεται πλέον σαν πίνακας με τόσες θέσεις, όσο το μέγεθος που έδωσε ο χρήστης. Ο πίνακας p γεμίζει με τυχαίους αριθμούς, οι οποίοι κατόπιν εμφανίζονται στην οθόνη. Τέλος, με χρήση της free() ο δεσμευμένος χώρος απελευθερώνεται, και φυσικά χάνονται όλα του τα περιεχόμενα.

Δυναμικοί πίνακες δύο διαστάσεων

Η δημιουργία ενός δυναμικού πίνακα δύο διαστάσεων δεν είναι τόσο απλή υπόθεση. Ο λόγος είναι ότι δεν αρκεί πλέον η δέσμευση του απαιτούμενου χώρου μνήμης, αλλά χρειαζόμαστε και ένα μηχανισμό υπολογισμού μιας θέσης μέσω δύο αριθμοδεικτών.

Ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε δυναμικά έναν πίνακα ακεραίων, διαστάσεων 5 × 10, με όνομα p. Η λογική που θα ακολουθήσουμε είναι η ακόλουθη:

Αρχικά, θα δημιουργήσουμε δυναμικά έναν πίνακα **δεικτών σε πίνακες** τύπου int, πέντε θέσεων, με όνομα p. Σε κάθε θέση του πίνακα δεικτών θα καταχωρίσουμε τη διεύθυνση ενός πίνακα 10 θέσεων, ο οποίος θα δεσμεύεται δυναμικά.



Το διπλανό σχήμα δείχνει παραστατικά αυτή τη λογική. Η κάθε θέση του πίνακα δεικτών p θα δείχνει σε ένα τμήμα μνήμης 10 θέσεων int, το οποίο έχει δεσμευτεί δυναμικά. Στις θέσεις των τμημάτων αυτών έχουμε πρόσβαση χρησιμοποιώντας το δείκτη p, σαν πίνακα δύο διαστάσεων 5 × 10! Παρατηρούμε ότι οι θέσεις μνήμης ενός δυναμικά δεσμευμένου πίνακα δύο διαστάσεων δεν είναι κατανάγκη συνεχόμενες.

Ενδεικτικές σελίδες

Αν εκτελέσουμε το παραπάνω πρόγραμμα, θα παρατηρήσουμε ότι δουλεύει μια χαρά και χωρίς τη μετατροπή τύπου. Η μετατροπή τύπου χρησιμοποιείται ώστε να είναι το πρόγραμμά μας συμβατό με τους περισσότερους μεταγλωττιστές, και ιδιαίτερα με τους μεταγλωττιστές της C++.

Δημιουργία δυναμικών πινάκων

Δυναμικοί πίνακες μίας διάστασης

Η δημιουργία ενός δυναμικού πίνακα μίας διάστασης είναι πολύ απλή διαδικασία. Οι συναρτήσεις δυναμικής κατανομής μνήμης επιστρέφουν τη διεύθυνση του τμήματος μνήμης που δέσμευσαν. Η εκχώρηση αυτής της διεύθυνσης σε ένα δείκτη μάς δίνει τη δυνατότητα να χρησιμοποιούμε το δείκτη σαν πίνακα. Το πρόγραμμα που ακολουθεί επιδεικνύει τη δημιουργία και τη χρήση ενός δυναμικού πίνακα στον οποίο καταχωρίζονται τυχαίοι αριθμοί.

17_dynamic_1d.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int *p,i,n;
    printf("Δώσε διάσταση πίνακα :");
    scanf("%d",&n);
    p=calloc(n,sizeof(int));
    if (p==NULL)
    {
        puts("Δεν υπάρχει αρκετή μνήμη");
        exit(1);
    }
    for (i=0;i<n;i++)
        p[i]=rand();
    for (i=0;i<n;i++)
        printf("%d\n",p[i]);
    free(p);
    return 0;
}
```

Ο χρήστης πληκτρολογεί το μέγεθος της διάστασης του πίνακα, το οποίο καταχωρίζεται στη μεταβλητή **n**.

Η **calloc()** δεσμεύει χώρο για την καταχώριση **n** ακεραίων. Η διεύθυνση που επιστρέφει καταχωρίζεται στο δείκτη **p**.

Έλεγχος του δείκτη που επέστρεψε η **calloc()**.

Καταχώριση **n** τυχαίων αριθμών, στο χώρο μνήμης που δείχνει ο δείκτης **p**. Αντιμετωπίζουμε το χώρο σαν έναν πίνακα με όνομα **p**!

Εμφάνιση των αριθμών που καταχωρίστηκαν.

Απελευθερώνει το δεσμευμένο χώρο.

Ενδεικτικές σελίδες

Κεφάλαιο

18

Δυναμικές δομές δεδομένων

Ενδεικτικές σελίδες

Δυναμικές δομές δεδομένων

Μέχρι στιγμής, η μοναδική δομή δεδομένων που έχουμε γνωρίσει είναι ο πίνακας. Οι πίνακες αποτελούν **στατικές** δομές καταχώρισης δεδομένων και έχουν ένα μεγάλο μειονέκτημα: πρέπει να καθορίσουμε το μέγεθος τους στην πρόταση δήλωσής τους, χωρίς να έχουμε τη δυνατότητα να το αλλάξουμε (αυξήσουμε ή μειώσουμε) κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Το αποτέλεσμα είναι η χρήση πινάκων μεγάλου μεγέθους, ώστε να είμαστε σίγουροι ότι θα μπορούμε να αποθηκεύσουμε το μεγαλύτερο πιθανό πλήθος δεδομένων που θα έχουμε. Όμως η λύση αυτή οδηγεί σε υπερκατανάλωση μνήμης, η οποία τις περισσότερες φορές δεν είναι απαραίτητη. Επίσης, αν θέλουμε να διατηρούμε τα δεδομένα του πίνακα συνεχώς ταξινομημένα, η διαδικασία επαναταξινόμησης (ή παρεμβολής) μετά από την προσθήκη κάθε νέου στοιχείου είναι αρκετά χρονοβόρα, ιδιαίτερα σε πίνακες μεγάλου μεγέθους. Η λύση στα προηγούμενα προβλήματα είναι η χρήση δυναμικών δομών, οι οποίες δεσμεύουν μνήμη όταν τη χρειάζονται και την απελευθερώνουν όταν δεν τους είναι πια απαραίτητη. Η χρήση δυναμικών δομών, εκτός από το ότι λύνει το πρόβλημα της κατανομής μνήμης, δίνει τη δυνατότητα να διατηρούμε τα δεδομένα συνεχώς ταξινομημένα, ώστε να μην είναι πια απαραίτητες οι χρονοβόρες διαδικασίες αναζήτησης και ταξινόμησης.

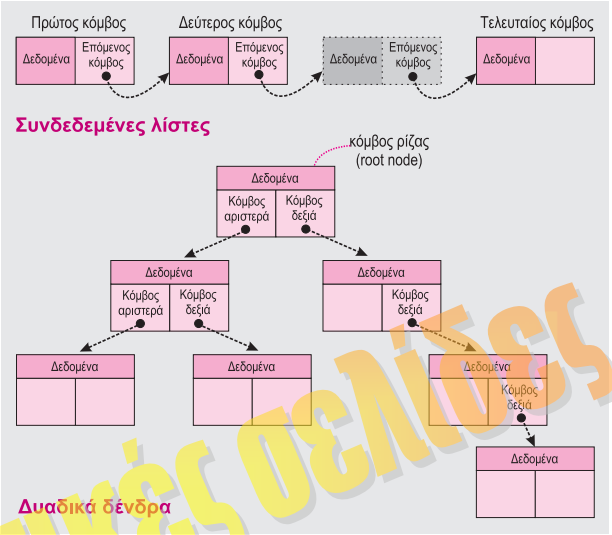
Δείγμα σελίδας

Το κεφάλαιο αυτό δεν φιλοδοξεί να καλύψει όλες τις δυναμικές δομές δεδομένων, οι οποίες από μόνες τους θα απαιτούσαν ένα ολόκληρο βιβλίο για να αναπτυχθούν. Θα εξετάσουμε όμως τις πιο συχνά χρησιμοποιούμενες δυναμικές δομές δεδομένων, οι οποίες είναι:

- Οι συνδεδεμένες λίστες (linked lists)
- Τα δυαδικά δένδρα αναζήτησης (binary search trees)

Το δομικό στοιχείο των δυναμικών δομών δεδομένων είναι ο **κόμβος**. Μια δομή δεδομένων αποτελείται από πολλούς συνδεδεμένους κόμβους. Κάθε κόμβος είναι ένα σύνολο πληροφοριών που περιέχει τόσο τα δεδομένα, όσο και πληροφορίες για τον επόμενο ή τους επόμενους κόμβους. Ο μηχανισμός δυναμικής διαχείρισης μνήμης χρησιμοποιείται για τη δέσμευση της μνήμης που απαιτεί ο κάθε κόμβος. Η διάταξη των κόμβων διαφέρει ανάλογα με το είδος της δομής δεδομένων (συνδεδεμένη λίστα ή δυαδικό δένδρο).

Στις συνδεδεμένες λίστες, κάθε κόμβος έχει έναν προηγούμενο και έναν επόμενο κόμβο (εκτός φυσικά από τον πρώτο και τον τελευταίο). Η δομή των δυαδικών δένδρων διαφέρει. Κάθε κόμβος του δένδρου μπορεί να συνδεθεί με έναν κόμβο προς τα αριστερά του και έναν κόμβο προς τα δεξιά του. Ο πρώτος κόμβος του δένδρου λέγεται κόμβος "ρίζας".



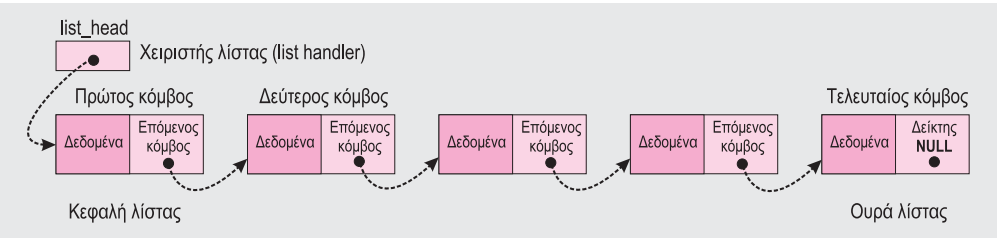
Ενδεικτικές σελίδες

Συνδεδεμένες λίστες

Απλά συνδεδεμένη λίστα (simple linked list))

Η απλά συνδεδεμένη λίστα αποτελείται από στοιχεία (κόμβους, nodes) τα οποία περιέχουν δεδομένα. Κάθε στοιχείο της λίστας περιέχει και ένα δείκτη ο οποίος δείχνει στο επόμενο στοιχείο της λίστας. Επομένως, κάθε στοιχείο της λίστας αποτελείται από:

- Το τμήμα δεδομένων, όπου καταχωρίζονται τα δεδομένα του στοιχείου, και
- Το δείκτη **Επόμενος κόμβος**, που δείχνει στο επόμενο στοιχείο (κόμβο) της λίστας.



Σχήμα 18.1 *Απλά συνδεδεμένη λίστα*

```
    p=list_head;
    while (p!=NULL)
    {
        printf("%d\n",p->data);
        p=p->next;
    }
}
```

Στο δείκτη **p** καταχωρίζεται η διεύθυνση του πρώτου κόμβου της λίστας.

Εμφανίζονται τα δεδομένα του κόμβου και στο δείκτη **p** καταχωρίζεται η διεύθυνση του επόμενου κόμβου. Αυτό επαναλαμβάνεται μέχρι να φτάσουμε στο τέλος της λίστας (p==NULL).

Η συνάρτηση διασχίζει όλη τη λίστα μέχρι να εντοπίσει τον τελευταίο της κόμβο.

Το επόμενο πρόγραμμα δημιουργεί μια απλά συνδεδεμένη λίστα με δεδομένα θετικούς ακέραιους αριθμούς που δίνονται από το χρήστη. Η διαδικασία σταματάει μόλις ο χρήστης δώσει αρνητικό αριθμό, και τότε εμφανίζει με τη σειρά όλα τα δεδομένα των κόμβων της λίστας. Το πρόγραμμα χρησιμοποιεί τις δύο συναρτήσεις που αναφέραμε προηγουμένως.

Ενδεικτικές σελίδες

18_list.c

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *next;
}*list_head;

void add_node_to_list(int d);
void display_all_nodes();

int main(void)
{
    int timi;
    list_head=NULL;
    while (1)
    {
        printf("Δώσε δεδομένα νέου κόμβου:");
        scanf("%d",&timi);
        if (timi<0) break;
```

Η **δομή** του **κόμβου** της λίστας. Δηλώνεται επίσης ο δείκτης **list_head** ο οποίος θα δείχνει στην αρχή (στον πρώτο κόμβο) της λίστας.

Η **δομή** του **κόμβου** της λίστας. Δηλώνεται επίσης ο δείκτης **list_head** ο οποίος θα δείχνει στην αρχή (στον πρώτο κόμβο) της λίστας.

Όταν ο χρήστης πληκτρολογήσει αρνητική τιμή, η επαναληπτική διαδικασία σταματάει.

```
        add_node_to_list(timi); // Προσθέτει ένα νέο κόμβο
    }
    display_all_nodes();        // Εμφανίζει τις τιμές όλων των κόμβων
    return 0;
}
```

Η επόμενη συνάρτηση δέχεται ως παράμετρο έναν ακέραιο αριθμό και προσπαθεί να τον εντοπίσει στα δεδομένα της συνδεδεμένης λίστας. Η συνάρτηση επιστρέφει ως τιμή ένα δείκτη στον κόμβο στον οποίο εντόπισε τα δεδομένα, διαφορετικά (αν δεν το εντοπίσει) επιστρέφει τιμή NULL.

```
struct node *find_node(int d)
{
    struct node *p;
    p=list_head;
    while (p!=NULL)
    {
        if (p->data)==d) return p;
        p=p->next;
    }
    return NULL;
}
```

Στο δείκτη **p** καταχωρίζεται η διεύθυνση του πρώτου κόμβου της λίστας.

Ελέγχονται τα δεδομένα κάθε κόμβου με τη σειρά. Μόλις εντοπιστεί το στοιχείο επιστρέφεται ως τιμή η διεύθυνση του κόμβου στον οποίο εντοπίστηκε.

Αν δεν εντοπιστεί το στοιχείο, επιστρέφεται τιμή NULL.

☞ Στα παραδείγματα στο τέλος του κεφαλαίου θα βρείτε επιπλέον κώδικα διαχείρισης συνδεδεμένων λιστών.

Η συνάρτηση που ακολουθεί διαγράφει έναν κόμβο από την κορυφή της λίστας:

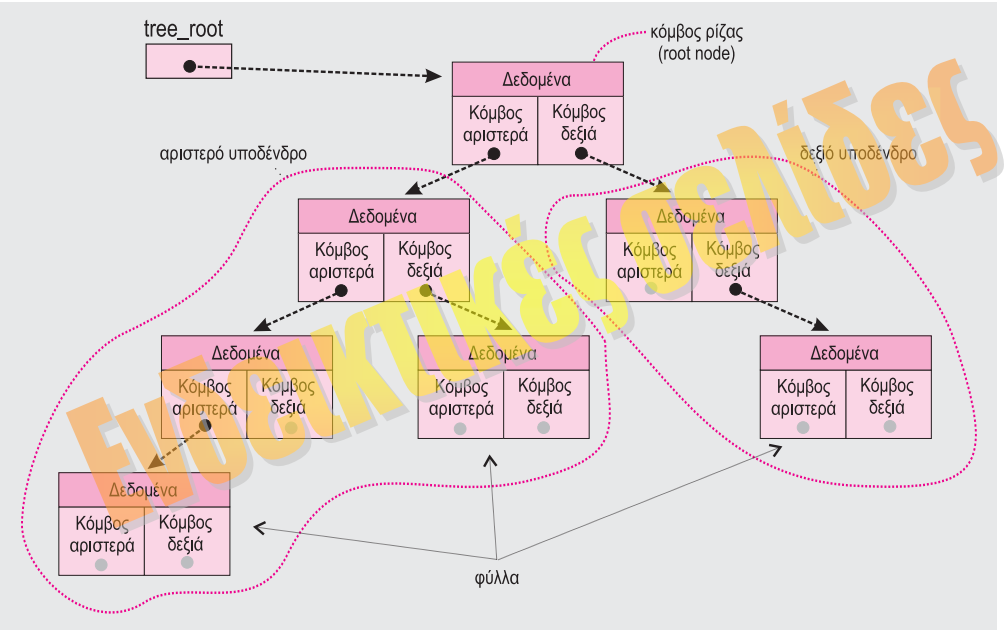
```
void delete_node()
{
    struct node *temp;
    // αν η λίστα είναι κενή επιστρέφει αμέσως
    if (list_head==NULL) return;
    // αποθηκεύει προσωρινά στη μεταβλητή temp τη διεύθυνση του δεύτερου κόμβου
    temp = list_head->next;
    free(list_head);
    /* Ο χειριστής της λίστας list_head παίρνει ως τιμή
    τη διεύθυνση του μέχρι στιγμής δεύτερου κόμβου, ο οποίος
```

Απελευθερώνει το χώρο της μνήμης που καταλαμβάνει ο πρώτος κόμβος.

Ενδεικτικές σελίδες

Δυαδικά δένδρα

Το δυαδικό δένδρο αποτελείται από έναν κόμβο που αποτελεί τη **ρίζα** του δένδρου (root node), ένα **υποδένδρο αριστερά** και ένα **υποδένδρο δεξιά**. Κάθε υποδένδρο είναι και αυτό ένα δυαδικό δένδρο, με τον κόμβο ρίζας του και το αριστερό και το δεξιό υποδένδρο του.



Σχήμα 18.9 *Η δομή ενός δυαδικού δένδρου*

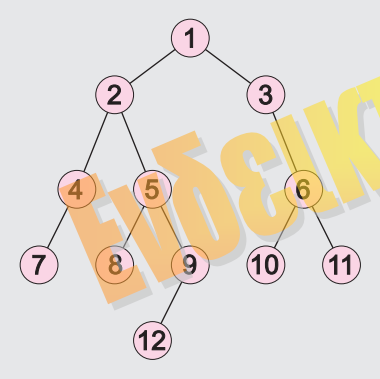
Κάθε κόμβος του δυαδικού δένδρου περιέχει, εκτός από το τμήμα δεδομένων, ένα δείκτη προς το αριστερό υποδένδρο του και ένα δείκτη προς το δεξιό. Οι κόμβοι οι οποίοι δεν έχουν υποδένδρο ονομάζονται **φύλλα** του δυαδικού δένδρου.

Η πρόσβαση σε ένα δυαδικό δένδρο γίνεται μέσω ενός δείκτη ο οποίος περιέχει τη διεύθυνση του κόμβου ρίζας του δυαδικού δένδρου (root node).

- Ένα δυαδικό δένδρο μπορεί να είναι άδειο, δηλαδή να μην περιέχει κανέναν κόμβο. Τότε ο δείκτης ο οποίος δείχνει στον κόμβο ρίζας θα πρέπει να έχει τιμή NULL.

Ενδεικτικές σελίδες

- Όταν ένα δυαδικό δένδρο αποτελείται από έναν μόνο κόμβο, τότε αυτός είναι ο κόμβος ρίζας.
- Η πρόσβαση σε κάθε κόμβο του δένδρου γίνεται μέσω μιας **μοναδικής** διαδρομής, η οποία ξεκινάει από τον κόμβο ρίζας του.
- Ένα δυαδικό δένδρο έχει **μέγεθος**. Το μέγεθος ενός δυαδικού δένδρου είναι το πλήθος των κόμβων που περιέχει.
- Κάθε κόμβος του δένδρου έχει **βάθος**. Το βάθος του κόμβου είναι η απόστασή του (σε κόμβους) από τον κόμβο ρίζας του δένδρου.
- Ένα δυαδικό δένδρο έχει **ύψος**. Το ύψος ενός δυαδικού δένδρου είναι το βάθος του βαθύτερου κόμβου του, δηλαδή του κόμβου που απέχει περισσότερο από τη ρίζα του δένδρου.



Ας υποθέσουμε το δυαδικό δένδρο του σχήματος, με κόμβο ρίζας τον κόμβο 1.

Το **μέγεθός** του είναι 12, διότι αποτελείται από 12 κόμβους.

Το **ύψος** του είναι 4, διότι ο πλέον απομακρυσμένος κόμβος του (ο 12) απέχει τέσσερις κόμβους από τη ρίζα του (1). Το βάθος του κόμβου 1 είναι 0, ενώ του κόμβου 6 είναι 2 (απέχει δύο κόμβους από τη ρίζα).

Ενδεικτικές Σελίδες

Δυαδικά δένδρα αναζήτησης

Τα δυαδικά δένδρα αναζήτησης (binary search tree - BST) είναι μια συγκεκριμένη κατηγορία δυαδικών δένδρων, και είναι πολύ χρήσιμα στον προγραμματισμό επειδή είναι εξαιρετικά αποδοτικά στη διαδικασία αναζήτησης και ταξινόμησης των κόμβων που περιέχουν. Οι τεχνικές και οι αλγόριθμοι που θα αναπτυχθούν στη συνέχεια του κεφαλαίου αφορούν στα δυαδικά δένδρα αναζήτησης.

Η λογική με την οποία είναι διατεταγμένοι οι κόμβοι ενός δυαδικού δένδρου αναζήτησης είναι η εξής:

- Ένα από τα δεδομένα του κόμβου αποτελεί το **κλειδί** με βάση το οποίο θα είναι διατεταγμένο το δυαδικό δένδρο.

Όλες οι συναρτήσεις έχουν υλοποιηθεί σύμφωνα με όσα αναφέραμε στις προηγούμενες ενότητες. Το πρόγραμμα εμφανίζει αρχικά ένα μενού επιλογών μέσω του οποίου μπορεί ο χρήστης να επιλέξει την επιθυμητή λειτουργία.

18_binary_tree.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct node
{
    int data;
    struct node *left;
    struct node *right;
};

struct node *root;
struct node *newnode(int num);
struct node *find(int key);
struct node *insert(int num);
void display(struct node *ptr);
bool rm(int key);
struct node *find_left_most(struct node *rt);
struct node *find_right_most(struct node *rt);

int main(void)
{
    char ch;
    int a;
    struct node *new;
    root=NULL;
    while (1)
    {
        printf("0->Εξοδος 1->Προσθήκη 2->Εύρεση ");
        printf("3->Διαγραφή 4->Εμφάνιση:\n");
        ch=getch();
        switch(ch)
        {
```

Η δομή **node** προσδιορίζει τη μορφή του κάθε κόμβου του δένδρου.

Πρόσθιες δηλώσεις συναρτήσεων

Εμφανίζει το μενού επιλογών

Περιμένει να πατηθεί η επιλογή του μενού

Ενδεικτικές σελίδες

Κεφάλαιο

19

Ο προμεταγλωττιστής της C

Ενδεικτικές σελίδες

Ο προμεταγλωττιστής της C

Όπως έχει ήδη αναφερθεί στα προηγούμενα κεφάλαια, μέσα στον πηγαίο κώδικα του προγράμματός μας μπορούμε να συμπεριλάβουμε οδηγίες (compiler directives) προς τον προμεταγλωττιστή της C. Οι οδηγίες αυτές δεν είναι στην πραγματικότητα κομμάτι της γλώσσας, αλλά αποτελούν τμήμα του περιβάλλοντος μεταγλώττισης. Έχουμε ήδη συναντήσει και χρησιμοποιήσει τις οδηγίες **#include** και **#define**.

Οι οδηγίες εκτελούνται από τον προμεταγλωττιστή της C (C precompiler ή C preprocessor), ένα στάδιο πριν ξεκινήσει η διαδικασία της μεταγλώττισης.

Οι πιο συχνά χρησιμοποιούμενες οδηγίες του προμεταγλωττιστή της C είναι οι εξής:

```
#include
#define
#undef
#if
#else
#elif
#endif
#ifdef
#ifndef
#error
```

Οι παραπάνω οδηγίες ορίζονται στο αρχικό πρότυπο (ANSI) της γλώσσας και υποστηρίζονται από όλους τους μεταγλωττιστές. Νεώτερα πρότυπα, αλλά και μεμονωμένοι μεταγλωττιστές, υποστηρίζουν ένα εκτεταμένο σετ οδηγιών, στις οποίες όμως δεν θα αναφερθούμε.

#include

Έχουμε ήδη χρησιμοποιήσει την οδηγία #include σε όλα σχεδόν τα προγράμματα της C που έχουμε καταστρώσει μέχρι τώρα. Γνωρίζουμε ότι η οδηγία #include συντάσσεται με τον επόμενο τρόπο:

#include <*αρχείο*> ή **#include "*αρχείο*"**

Ενδεικτικές σελίδες

Ο πρώτος τρόπος αφορά σε αρχεία κεφαλίδας της προκαθορισμένης βιβλιοθήκης της C, ενώ ο δεύτερος τρόπος αφορά σε δικά μας αρχεία κεφαλίδας. Και ο ένας και ο άλλος τρόπος αναγκάζουν το μεταγλωττιστή να συμπεριλάβει κατά τη φάση της μεταγλώττισης το αρχείο το οποίο ορίζεται.

Για παράδειγμα, το αποτέλεσμα της πρότασης:

```
#include <stdio.h>
```

αναγκάζει το μεταγλωττιστή να συμπεριλάβει το αρχείο κεφαλίδας stdio.h της προκαθορισμένης βιβλιοθήκης, ενώ η πρόταση:

```
#include "mylib.h"
```

αναγκάζει το μεταγλωττιστή να συμπεριλάβει το αρχείο mylib.h, από τον ίδιο φάκελο στον οποίο βρίσκεται και το πηγαίο αρχείο (.c) του προγράμματός μας. Αυτός ο δεύτερος τρόπος μάς δίνει τη δυνατότητα να καθορίσουμε και τη διαδρομή, απόλυτη ή σχετική, στην οποία βρίσκεται το αρχείο. Για παράδειγμα:

```
#include "c:/test/mylib.h"      // όταν το αρχείο βρίσκεται στον φάκελο c:/test
#include "../mylib.h"           // όταν το αρχείο βρίσκεται στον προηγούμενο φάκελο
```

Ενδεικτικές σελίδες

#define

Την οδηγία #define την έχουμε συναντήσει επίσης σε προγράμματα που έχουμε χρησιμοποιήσει μέχρι τώρα. Η οδηγία ορίζει ένα αναγνωριστικό και ένα σύνολο χαρακτήρων που θα αντικαταστήσει αυτό το αναγνωριστικό κατά τη διαδικασία της μεταγλώττισης του πηγαίου κώδικα. Η σύνταξή της είναι:

#define *αναγνωριστικό χαρακτήρες*

Ο προμεταγλωττιστής της C αντικαθιστά το ***αναγνωριστικό*** με τους ***χαρακτήρες***, σε όποιο σημείο του πηγαίου κώδικα και αν το βρει. Συνήθως χρησιμοποιούμε την οδηγία #define για να ορίσουμε κάποιες σταθερές του προγράμματός μας. Για παράδειγμα, στο πρόγραμμα που ακολουθεί ανατίθεται στο αναγνωριστικό SIZE η τιμή 100. Ο προμεταγλωττιστής αντικαθιστά το αναγνωριστικό SIZE με το 100, σε όποιο σημείο του προγράμματος και αν το εντοπίσει. Αυτό γίνεται πριν αρχίσει η κανονική διαδικασία της μεταγλώττισης.

```
#include <stdio.h>
#define SIZE 100
```

```
#include <stdio.h>
#define SIZE 1000
int main(void)
{
    #if SIZE>500000
        #error Πολύ μεγάλο μέγεθος πίνακα
    #else
        int a[SIZE];
    #endif
    .....
    return 0;
}
```

Στην περίπτωση που στο αναγνωριστικό **SIZE** έχει ανατεθεί τιμή μεγαλύτερη από 500000, η διαδικασία μεταγλώττισης σταματάει και εμφανίζεται το μήνυμα που ακολουθεί την οδηγία **#error**.

Σε αντίθετη περίπτωση, η μεταγλώττιση συνεχίζεται και δημιουργείται ένας πίνακας ακεραίων με μέγεθος SIZE.

Ενδεικτικές σελίδες

Αποσφαλμάτωση (debugging)

Πρόσφατα διάβασα σε μία δημοσίευση κάτι που μου άρεσε πάρα πολύ:

*Ο Η/Υ έχει ένα **πλεονέκτημα** και ένα **μειονέκτημα**:*
*Το πλεονέκτημα είναι ότι **κάνει ακριβώς ΟΤΙ ΤΟΥ ΛΕΜΕ!***
*και το μειονέκτημα είναι ότι **κάνει ΑΚΡΙΒΩΣ ότι του λέμε!***

Η αλήθεια είναι ότι κανένας προγραμματιστής, όσο καλός και να είναι, δεν μπορεί να φτιάχνει προγράμματα που όλα να δουλεύουν απροβλημάτιστα από την αρχή. Και δεν αναφέρομαι σε συντακτικά λάθη, τα οποία ούτως ή άλλως τα εντοπίζει ο μεταγλωττιστής, αλλά σε λογικά σφάλματα τα οποία μπορεί να ταλαιπωρούν τον προγραμματιστή (αλλά και το χρήστη) για μεγάλο χρονικό διάστημα μέχρι να διορθωθούν.

Τα λογικά σφάλματα εντοπίζονται κατά την εκτέλεση του προγράμματος (run time), όπου απλώς διαπιστώνουμε ότι το πρόγραμμά μας δεν κάνει αυτά που θα έπρεπε να κάνει!

Η διαδικασία εντοπισμού και διόρθωσης των λογικών λαθών λέγεται αποσφαλμάτωση (debugging). Οι προγραμματιστές χρησιμοποιούν διάφορες τεχνικές για να μπορέσουν να εντοπίσουν τα λογικά σφάλματα. Συνήθως οι τεχνικές αυτές περιλαμβάνουν προτάσεις εξόδου, οι οποίες προστίθενται στον

Παραδείγματα

Π19.1 Στο πρόγραμμα που ακολουθεί χρησιμοποιείται η οδηγία #define για να ορίσει τόσο αναγνωριστικά-σταθερές, όσο και μακροεντολές. Μελετήστε το προσεκτικά:

19_p19_1.c

```
#include <stdio.h>
#include <stdlib.h>

#define PROGRAMMA int main(void)
#define PERIMENE system("pause")
#define OLA_KALA return 0;
#define AKERAIOI int
#define ARXH {;
#define TELOS };
#define ALLAXE_GRAMMH printf("\n");

#define PRINT(X) printf("%d",X);
#define GET(X) scanf("%d",&X);
#define PROSTHESE(X,Y) (X)+(Y)

PROGRAMMA
ARXH
    AKERAIOI a,b,c;
    GET(a);
    GET(b);
    c=PROSTHESE(a,b);
    PRINT(c);
    ALLAXE_GRAMMH;
    PERIMENE;
    OLA_KALA;
TELOS
```

Ορισμοί αναγνωριστικών-σταθερών

Ορισμοί αναγνωριστικών-μακροεντολών

Προμεταγλωττιστής

```
int main(void)
{
  int a,b,c;
  scanf("%d",&a);
  scanf("%d",&b);
  c=(a)+(b);
  printf("%d",c);
  printf("\n");
  system("pause");
  return 0;
}
```

Υποδειγματικές σελίδες

Ο προμεταγλωττιστής της C, πριν από την έναρξη της μεταγλώττισης, θα αντικαταστήσει όλα τα αναγνωριστικά που έχουν οριστεί με την οδηγία #define, με τις συμβολοσειρές οι οποίες τους έχουν ανατεθεί. Στις μακροεντολές γίνεται επίσης αντικατάσταση των παραμέτρων. Ο κώδικας που θα προκύψει φαίνεται στο διπλανό πλαίσιο και είναι αυτός που τελικά θα μεταγλωττιστεί.

Παρατηρούμε ότι με χρήση της #define μπορούμε να αλλάξουμε ριζικά την εικόνα ενός προγράμματος χωρίς να επηρεάσουμε την ουσία του! Αυτό μερικές φορές (όχι όλες!) κάνει τον κώδικα πιο ευανάγνωστο.

Ενδεικτικές σελίδες

Κεφάλαιο

20

Από τη C, στη C++

Ενδεικτικές σελίδες

Από τη C, στη C++

Μετά από τη σπουδή σας στη C, το επόμενο φυσικό σας βήμα θα είναι σίγουρα μια γλώσσα αντικειμενοστρεφούς προγραμματισμού. Με δεδομένη τη γνώση που αποκτήσατε μαθαίνοντας τη γλώσσα C, η C++ είναι το πιο εύκολο βήμα στο δύσκολο δρόμο του αντικειμενοστρεφούς προγραμματισμού[21].

Βασισμένη στη γλώσσα C, η C++ αναπτύχθηκε από τον Bjarne Stroustrup στα εργαστήρια Bell, την περίοδο 1983-1985. Πριν από το 1983, ο Bjarne Stroustrup δανείστηκε τα αντικειμενοστρεφή χαρακτηριστικά της Simula (η οποία θεωρείται η πρώτη αντικειμενοστρεφής γλώσσα προγραμματισμού) και, συνδυάζοντάς τα με την αποδοτικότητα και την ευελιξία της C, δημιούργησε μια έκδοση της C που ονομάστηκε "C with Classes". Αυτή αποτέλεσε την πρώτη "πειραματική" έκδοση της C++, η οποία πήρε την τελική της ονομασία το 1983. Οι συνεχείς βελτιώσεις και προσθήκες στη γλώσσα συνεχίστηκαν μέχρι το 1997, οπότε οριστικοποιήθηκε το πρότυπο της γλώσσας και πιστοποιήθηκε από διεθνείς οργανισμούς (ANSI, ISO κ.λπ.).

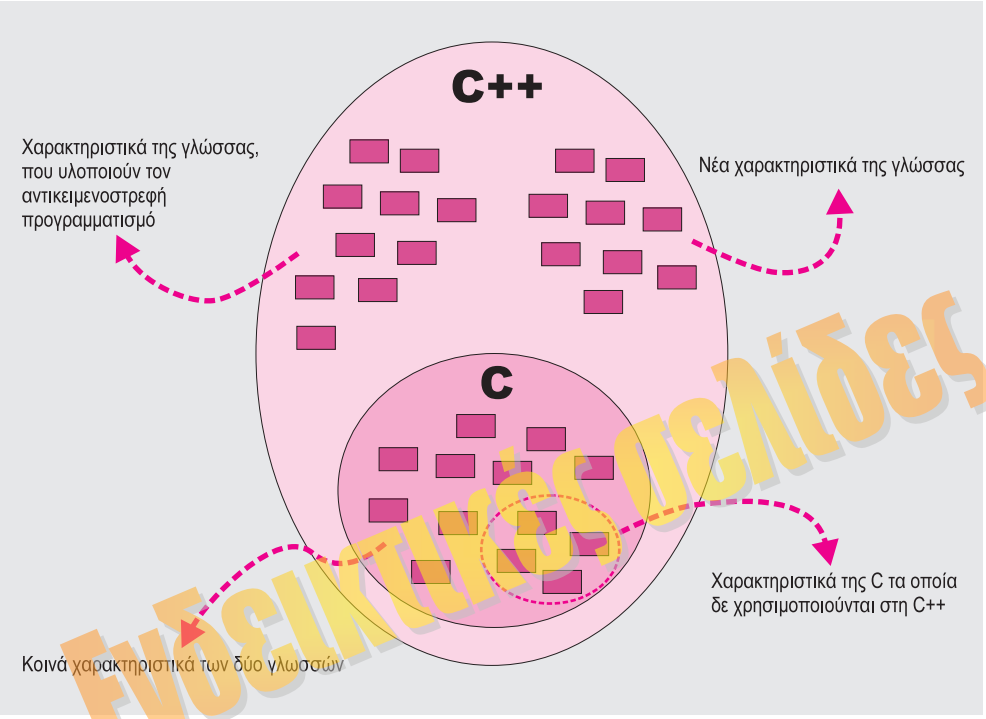
Αρχικά η C++ σχεδιάστηκε για το λειτουργικό σύστημα UNIX. Σήμερα οι διαφορετικές εκδόσεις της γλώσσας καλύπτουν όλα τα λειτουργικά συστήματα και αποτελούν αναμφίβολα την επιλογή των περισσότερων επαγγελματιών προγραμματιστών για την ανάπτυξη πολύπλοκων εφαρμογών. Με τη C++ οι προγραμματιστές μπορούν να βελτιώσουν την ποιότητα των προγραμμάτων τους, και ο κώδικας που παράγουν μπορεί να επαναχρησιμοποιηθεί πιο εύκολα.

C++, η κληρονομιά από τη C

Η C++ ξεκίνησε ως μια επέκταση της C, προσθέτοντας στην πλέον δημοφιλή αυτή γλώσσα αντικειμενοστρεφείς δυνατότητες. Όμως η βάση της γλώσσας και οι "παραδοσιακές" δομές της παραμένουν ίδιες όπως της C.

Μπορούμε λοιπόν να ισχυριστούμε ότι η C αποτελεί ένα υποσύνολο της C++. Ένα πρόγραμμα γραμμένο σε C, με ελάχιστες ή καθόλου αλλαγές, θα μεταγλωττιστεί χωρίς πρόβλημα από οποιονδήποτε μεταγλωττιστή της C++.

[21] Τα περισσότερα κείμενα του παρόντος κεφαλαίου προέρχονται από το βιβλίο μου "Η γλώσσα C++ σε βάθος", εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ, 2008.



ΕΝΔΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

Σχήμα 20.1 *Η σχέση των γλωσσών C και C++*

C++, μια αντικειμενοστρεφής γλώσσα

Η C++, μαζί με την Java και τη C#, αποτελούν τις πλέον δημοφιλείς και χαρακτηριστικές γλώσσες αντικειμενοστρεφούς προγραμματισμού (Object Oriented Programming, OOP).

Οι αντικειμενοστρεφείς γλώσσες προγραμματισμού δίνουν περισσότερη έμφαση στα δεδομένα, παρά στον κώδικα. Το πρόγραμμα αναπτύσσεται γύρω από τα αντικείμενα που απαρτίζουν το πρόβλημα, και τις διαδικασίες (τον κώδικα) που επενεργούν επάνω σε αυτά τα αντικείμενα. Για παράδειγμα, αν θέλαμε να αναπτύξουμε ένα πρόγραμμα διαχείρισης ενοικιαζόμενων αυτοκινήτων, κάθε αυτοκίνητο που είχαμε προς ενοικίαση θα αποτελούσε ένα αντικείμενο. Η ενοικίαση ενός αυτοκινήτου θα γινόταν με την εφαρμογή μιας διαδικασίας (με-

θόδου) επάνω σε αυτό το αντικείμενο. Η επιστροφή του αυτοκινήτου από τον πελάτη στην εταιρεία με την εφαρμογή μιας άλλης μεθόδου, κ.ο.κ.

Η C++ υποστηρίζει πλήρως τα βασικά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού: Την **Ενθυλάκωση** (Encapsulation), τον **Πολυμορφισμό** (Polymorphism), και την **Κληρονομικότητα** (Inheritance). Έννοιες που θα γίνουν κατανοητές αργότερα σε αυτό το κεφάλαιο, και θα αναλυθούν διεξοδικά σε επόμενα κεφάλαια του βιβλίου.

Το χαρακτηριστικό αυτό της γλώσσας, σε συνδυασμό με τις τεράστιες δυνατότητες που παρέχει, την ευελιξία της, και την απόδοσή της, την καθιστούν σήμερα την πρώτη επιλογή των προγραμματιστών για την κατασκευή μεγάλων και απαιτητικών εφαρμογών.

Ο αντικειμενοστρεφής προγραμματισμός συγκεντρώνει όλα τα στοιχεία του δομημένου και του τμηματικού προγραμματισμού, καταργώντας το χάσμα που υπάρχει μεταξύ δεδομένων και διαδικασιών, και εισάγοντας παράλληλα νέες έννοιες και μια διαφορετική φιλοσοφία όσον αφορά στο σχεδιασμό και στην ανάπτυξη των προγραμμάτων.

Ενδεικτικές σελίδες

Αντικειμενοστρεφής προγραμματισμός

Μια από τις καθημερινές μου συνήθειες είναι να φτιάχνω τον πρωινό μου καφέ. Ας προσπαθήσουμε να αναλύσουμε λίγο αυτή τη διαδικασία.

Η πρώτη ύλη που διαθέτουμε (τα δεδομένα μας) είναι ο καφές, η ζάχαρη, το νερό, και το γάλα, αποθηκευμένα στα ντουλάπια και στο ψυγείο της κουζίνας μας. Εμείς παίρνουμε τα υλικά που χρειαζόμαστε με απευθείας πρόσβαση στους αποθηκευτικούς μας χώρους και κάνουμε την απαραίτητη επεξεργασία για να φτιάξουμε τον καφέ μας. Ο συγκάτοικός μας έχει και αυτός πρόσβαση στα ίδια υλικά και φτιάχνει και αυτός τον καφέ του ακολουθώντας την ίδια ή διαφορετική διαδικασία.

Παρατηρούμε ότι με την παραπάνω διαδικασία ...

- Τα υλικά μας (δεδομένα) είναι εκτεθειμένα στον οποιονδήποτε έχει πρόσβαση στους αποθηκευτικούς μας χώρους (ντουλάπια, ψυγείο). Με αυτόν τον τρόπο μπορεί να γίνει αλόγιστη χρήση των υλικών ακόμη και για δια-

φορετικό σκοπό (π.χ. να ξοδέψουμε ζάχαρη και γάλα για την κατασκευή ενός κέϊκ).

- Η επεξεργασία των υλικών για την κατασκευή του καφέ δεν είναι αυτοματοποιημένη, και ο καθένας μπορεί να ακολουθεί μια εντελώς διαφορετική διαδικασία.
- Αν αλλάξει ο χώρος όπου αποθηκεύουμε τα υλικά (π.χ. τα βάζουμε σε άλλο ντουλάπι), πρέπει να αλλάξουμε και τη διαδικασία επεξεργασίας ώστε να τα παίρνουμε από το νέο αποθηκευτικό χώρο.
- Στην περίπτωση που θέλουμε να φτιάχνουμε διαφορετικά είδη καφέ (π.χ. καπουτσίνο, φραπέ, εσπρέσο), θα πρέπει κάθε φορά να χρησιμοποιούμε διαφορετική διαδικασία επεξεργασίας.
- Αν θελήσουμε να φτιάξουμε καφέ όχι στο σπίτι μας, αλλά σε έναν άλλο χώρο, θα πρέπει καταρχήν να μάθουμε πού είναι τα υλικά μας (τα δεδομένα μας), ενώ ίσως χρειαστεί να τροποποιήσουμε τη διαδικασία κατασκευής του καφέ ανάλογα με τις δυνατότητες του χώρου στον οποίο βρισκόμαστε. Για παράδειγμα, θα ακολουθήσουμε διαφορετική διαδικασία για να ζεστάνουμε το νερό ανάλογα με το αν ο χώρος διαθέτει υγραέριο, ηλεκτρική κουζίνα, ή φούρνο μικροκυμάτων.

Ενδεικτικές σελίδες

Μετά από αρκετό καιρό υπομονής, η κατάσταση δεν πήγαινε άλλο. Πότε ανακάλυπτα την τελευταία στιγμή ότι τελείωσε ο καφές γιατί τον είχε χρησιμοποιήσει ο συγκάτοικος μου, πότε η ζάχαρη επειδή έφτιαχνε γλυκό η φίλη μου, και πότε δεν έβρισκα καθόλου τα υλικά επειδή αποφάσισε η καθαρίστρια να τα βάλει σε διαφορετικό ντουλάπι. Και δεν φτάνουν όλα αυτά, κάθε φορά ο καφές ήταν διαφορετικός, είτε γιατί έπεφτε λίγο περισσότερη ζάχαρη, είτε γιατί δεν υπολόγιζα σωστά το γάλα!

Σκεφτόμουν, λοιπόν, μήπως υπάρχει κάποιο αντικείμενο που να φτιάχνει καφέ και να με βγάλει από όλη αυτή τη διαδικασία. Έτσι αποφάσισα να αγοράσω μια μηχανή για καφέ και αμέσως όλα μου τα προβλήματα λύθηκαν.

- Τα υλικά πλέον ήταν μέσα στη μηχανή. Κανείς δεν είχε πρόσβαση σε αυτά και κανείς δεν διανοείτο να διαρρήξει τη μηχανή για να βγάλει λίγο καφέ ή ζάχαρη.
- Ποτέ πια δεν χρειάστηκε να ψάχνω πού είναι ο καφές, η ζάχαρη, ή το γάλα· η δε μηχανή από μόνη της με ειδοποιούσε όταν κόντευαν να τελειώσουν τα υλικά που περιέχει.

- Κάθε φορά πατούσα απλώς το κουμπί και ο καφές ετοιμαζόταν χωρίς ιδιαίτερο κόπο και, το κυριότερο, πάντα με τις ίδιες αναλογίες.
- Η μηχανή είχε τη δυνατότητα να φτιάχνει καφέ διαφόρων ειδών. Εγώ απλώς πατούσα το σωστό κουμπί, χωρίς να με απασχολεί η διαφορετική διαδικασία που ακολουθούσε η μηχανή για να φτιάξει τον καφέ που ζήτησα.
- Και φυσικά, όταν πήγαινα σε κάποιον άλλο χώρο, απλώς έπαιρνα τη μηχανή μαζί μου ή φρόντιζα να υπάρχει μια ακριβώς ίδια.

Συγχαρητήρια, μόλις αντιληφθήκατε τις βασικές αρχές και τα πλεονεκτήματα του αντικειμενοστρεφούς προγραμματισμού.

Ενδεικτικές σελίδες

Κλάσεις και αντικείμενα (classes and objects)

Παρατηρούμε ότι από τη στιγμή που αποφασίσαμε να κατασκευάσουμε (ή να αγοράσουμε) ένα αντικείμενο (καφετιέρα) το οποίο εκτελεί την εργασία που μέχρι τώρα υλοποιούσαμε με αποσπασματικές διαδικασίες, λύθηκαν τα περισσότερα προβλήματα που είχαμε.

Μήπως λοιπόν η καλύτερη προσέγγιση και στην κατασκευή των προγραμμάτων βρίσκεται στη χρήση **αντικειμένων**; Τα προγράμματα με τις παραδοσιακές, διαδικασιακές, τεχνικές προγραμματισμού ρίχνουν το βάρος περισσότερο στις διαδικασίες, παρά στα δεδομένα. Ας επισημάνουμε επίσης ότι ο φυσικός και ο τεχνητός κόσμος στον οποίο ζούμε είναι πιο κοντά στη φιλοσοφία του **αντικειμενοστρεφούς**, παρά του διαδικασιακού προγραμματισμού.

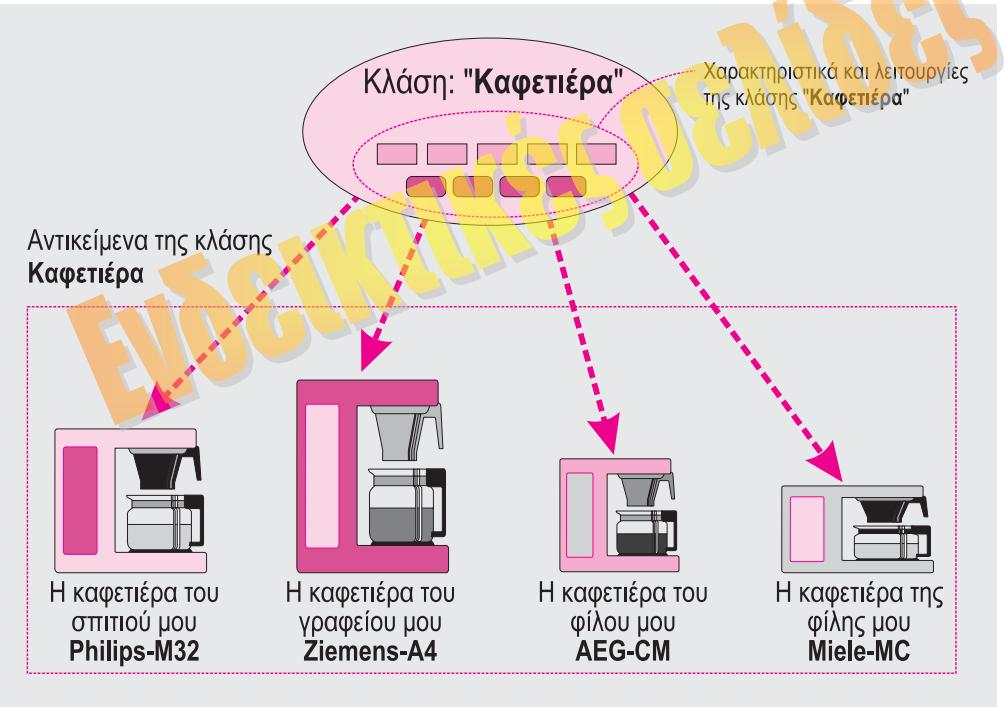
Με μια κλασική διαδικασιακή γλώσσα προγραμματισμού, όπως η C, αναγκαζόμαστε να προσαρμόσουμε το πρόβλημα που καλούμαστε να λύσουμε στις δομές και στις δυνατότητες που παρέχει η γλώσσα. Στις γλώσσες αντικειμενοστρεφούς προγραμματισμού κάνουμε ακριβώς το αντίθετο: Προσαρμόζουμε τη γλώσσα και τις δομές της ώστε να εξυπηρετούν στη διαχείριση του συγκεκριμένου προβλήματος.

Στις αντικειμενοστρεφείς γλώσσες προγραμματισμού αντιμετωπίζουμε τις διάφορες οντότητες του προγράμματος ως αντικείμενα. Ο κώδικας του προγράμματος καθορίζει τις διαδικασίες που επενεργούν σε αυτά τα αντικείμενα.

Οι **κλάσεις** (classes) και τα **αντικείμενα** (objects) είναι έννοιες πάνω στις οποίες βασίζεται όλη η φιλοσοφία και τα χαρακτηριστικά των γλωσσών αντικειμενοστρεφούς προγραμματισμού.

Η **κλάση** (class) είναι μια έννοια η οποία προσδιορίζει μια κατηγορία αντικειμένων, και ταυτόχρονα περιγράφει τα κοινά χαρακτηριστικά τους.

Για παράδειγμα, η κλάση "Καφετιέρα" περιγράφει μια κατηγορία συσκευών για την κατασκευή καφέ.



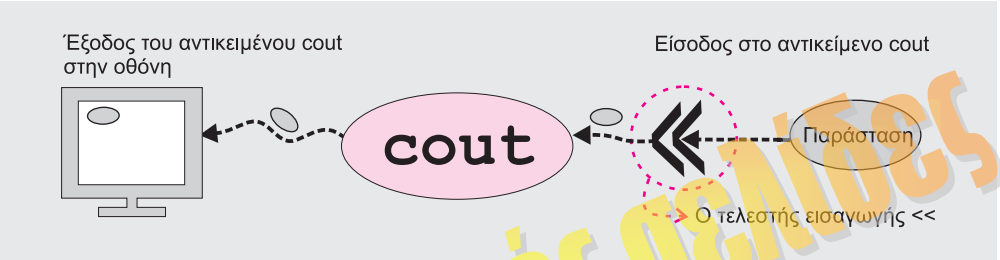
Ενδεικτικές σελίδες

Σχήμα 20.2 *Παράδειγμα κλάσης - η κλάση "Καφετιέρα"*

Όλα τα αντικείμενα αυτής της κλάσης έχουν κάποια κοινά χαρακτηριστικά και λειτουργίες. Π.χ. όλες οι καφετιέρες έχουν χώρο για νερό, καφέ, γάλα, και ζάχαρη. Επίσης όλες διαθέτουν λειτουργία για να φτιάξουν καφέ σκέτο, γλυκό, ή μέτριο. Ένα αντικείμενο-καφετιέρα έχει επομένως κάποια **χαρακτηριστικά** (χρώμα, υλικά, κ.λ.π.) και κάποιες **λειτουργίες**, οι οποίες διαχειρίζονται τα υλικά για να φτιάξουν καφέ!

Το αντικείμενο cout

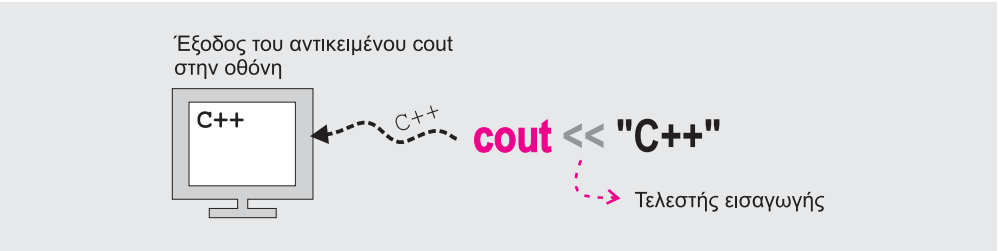
Το αντικείμενο **cout** είναι συνδεδεμένο με την προκαθορισμένη μονάδα εξόδου, που είναι συνήθως η οθόνη. Το αντικείμενο **cout** έχει μία είσοδο. Οτιδήποτε στείλουμε στην είσοδο του **cout** προωθείται στην έξοδό του, δηλαδή στην οθόνη.



Το αντικείμενο **cout** ακολουθείται από τον **τελεστή εισαγωγής** << και την παράσταση που θέλουμε να εμφανιστεί στην οθόνη.

Ο τελεστής εισαγωγής << αποστέλλει το αποτέλεσμα της παράστασης που βρίσκεται δεξιά του, στην είσοδο του αντικειμένου **cout** που βρίσκεται στα αριστερά του.

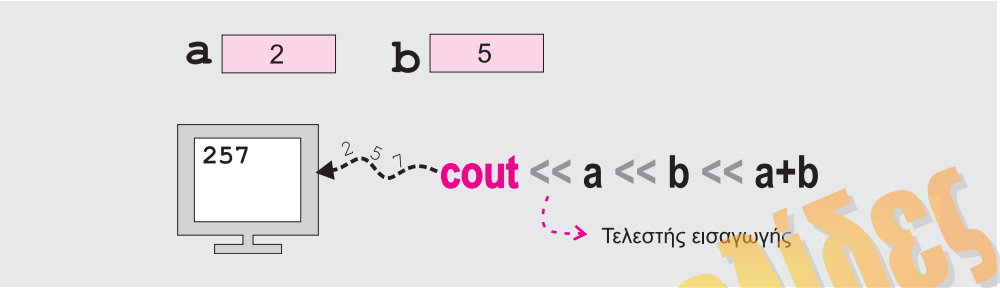
Στο παρακάτω παράδειγμα, το αλφαριθμητικό "C++" αποστέλλεται στην είσοδο του αντικειμένου **cout**, οπότε εμφανίζεται στην έξοδό του, δηλαδή στην οθόνη.



Μπορούμε να αποστείλουμε περισσότερες παραστάσεις στο αντικείμενο **cout**, χρησιμοποιώντας διαδοχικούς τελεστές εισαγωγής. Στην οθόνη εμφανίζεται το αποτέλεσμα της παράστασης. Για παράδειγμα, αν οι παραστάσεις είναι μεταβλητές, τότε εμφανίζεται στην οθόνη το περιεχόμενό τους, ενώ αν είναι αριθμητικές πράξεις εμφανίζεται το αποτέλεσμά τους.

Ενδεικτικές σελίδες

Στο παρακάτω παράδειγμα, αποστέλλονται στην είσοδο του αντικειμένου **cout** τρεις παραστάσεις (δύο μεταβλητές και μία πράξη). Στην οθόνη θα εμφανιστεί το περιεχόμενο των μεταβλητών, καθώς και το αποτέλεσμα της πράξης.



Παρατηρούμε ότι τα αποτελέσματα των παραστάσεων εμφανίζονται το ένα μετά το άλλο, χωρίς να εμφανίζονται κενά διαστήματα μεταξύ τους. Το σχήμα που ακολουθεί δείχνει το γενικό τρόπο χρήσης του αντικειμένου **cout**.

Το παρακάτω τμήμα κώδικα δείχνει τη χρήση του αντικειμένου cout:

```
int a;
char c;
float b;
a=15;
b=a/2.0;
c='A';
cout <<"a+2="<<a+2<<" b="<<b<<" c="<<c<<endl;
cout <<"Τέλος\n";
cout <<"Προγράμματος"<<endl;
```

```
a+2=17 b=7.5 c=A
Τέλος
Προγράμματος
```

Ας αναλύσουμε την παρακάτω πρόταση:

```
cout <<"a+2="<<a+2<<" b="<<b<<" c="<<c<<endl;
```

Ελληνικές Σελίδες

Ενδεικτικές σελίδες

Παράρτημα

Α

Συναρτήσεις βιβλιοθήκης της C

Ενδεικτικές σελίδες

Η τυπική βιβλιοθήκη της C

Η τυπική βιβλιοθήκη της C περιλαμβάνει κάθε είδους συναρτήσεις, οι οποίες είναι ομαδοποιημένες κατά κατηγορίες και δηλώνονται σε διαφορετικά αρχεία κεφαλίδας (header files). Θεωρήσαμε πιο χρήσιμο να αναφέρουμε τις συναρτήσεις ανάλογα με το αρχείο κεφαλίδας στο οποίο δηλώνονται. Για να χρησιμοποιήσουμε οποιαδήποτε από αυτές τις συναρτήσεις, θα πρέπει με την οδηγία **#include** να συμπεριλάβουμε στον κώδικα του προγράμματος το αρχείο κεφαλίδας όπου δηλώνεται.

Σε ορισμένα προγραμματιστικά περιβάλλοντα (όπως στο DEV C++), αρκετές συναρτήσεις βιβλιοθήκης δηλώνονται, πέρα από το αρχείο κεφαλίδας το οποίο καθορίζει το πρότυπο ANSI, και στα αρχεία κεφαλίδας **stdlib.h** ή **stdio.h**. Σε αυτή την περίπτωση, αν συμπεριλάβουμε τα αρχεία αυτά στον κώδικά μας, επαρκούν για τη χρήση των περισσότερων συναρτήσεων βιβλιοθήκης.

Ενδεικτικές σελίδες

<ctype.h>

int islower(int c);

Η συνάρτηση **islower**() επιστρέφει μη μηδενική τιμή αν η παράμετρος c είναι πεζός χαρακτήρας, ενώ διαφορετικά επιστρέφει τιμή 0.

Παράδειγμα

Το επόμενο πρόγραμμα ζητάει συνέχεια χαρακτήρες μέχρι να πληκτρολογηθεί ένας κεφαλαίος χαρακτήρας.

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    char ch;
    do
    {
       ch=getch();
       putchar(ch);
    } while (islower(ch));
    return 0;
}
```

int isupper(int c);

Η συνάρτηση **isupper()** επιστρέφει μη μηδενική τιμή αν η παράμετρος c είναι κεφαλαίος χαρακτήρας, διαφορετικά επιστρέφει τιμή 0.

int isprint(int c);

Η συνάρτηση **isprint()** επιστρέφει μη μηδενική τιμή αν η παράμετρος c είναι εκτυπώσιμος χαρακτήρας (συμπεριλαμβανομένου και του διαστήματος), διαφορετικά επιστρέφει τιμή 0.

int tolower(int c);

Η συνάρτηση **tolower()** επιστρέφει τον αντίστοιχο πεζό χαρακτήρα της παραμέτρου c. Δεν υποστηρίζει ελληνικούς χαρακτήρες.

Παράδειγμα

Η πρόταση,

```
tolower('A')
```

επιστρέφει το χαρακτήρα 'a'.

int toupper(int c);

Η συνάρτηση **toupper()** επιστρέφει τον αντίστοιχο κεφαλαίο χαρακτήρα της παραμέτρου c. Δεν υποστηρίζει ελληνικούς χαρακτήρες.

Παράδειγμα

Η πρόταση,

```
toupper('a')
```

επιστρέφει το χαρακτήρα 'A'.

<math.h>

double exp(double x);

Η συνάρτηση **exp()** επιστρέφει τη βάση των φυσικών λογαρίθμων e υψωμένη στη δύναμη της παραμέτρου x (e^x).

Παράδειγμα

Η επόμενη πρόταση καταχωρίζει στη μεταβλητή f την τιμή e^4.

```
f=exp(4)
```

Ενδεικτικές σελίδες

double log(double x);

Η συνάρτηση **log()** επιστρέφει το φυσικό λογάριθμο της παραμέτρου x. Η συνάρτηση επιστρέφει μήνυμα λάθους αν η παράμετρος x είναι μικρότερη ή ίση από το 0.

Παράδειγμα

Το επόμενο πρόγραμμα εμφανίζει τους φυσικούς λογάριθμους των αριθμών από το 1 μέχρι το 10.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   int i;
   for (i=1;i<=10;i++)
      printf("%d %f\n",i,log((double)i));
   return 0;
}
```

Παρατηρούμε τη μετατροπή[22] του i σε τύπο double ώστε να συμφωνεί με τον τύπο της παραμέτρου που απαιτεί η συνάρτηση log().

Ενδεικτικές σελίδες

double log10(double x);

Η συνάρτηση **log10()** επιστρέφει το δεκαδικό λογάριθμο της παραμέτρου x. Η συνάρτηση επιστρέφει μήνυμα λάθους αν η παράμετρος x είναι μικρότερη ή ίση του μηδενός.

double pow(double x, double y);

Η συνάρτηση **pow()** επιστρέφει την τιμή της πρώτης παραμέτρου x υψωμένη στη δύναμη της δεύτερης παραμέτρου y (x^y).

Παράδειγμα

Το επόμενο πρόγραμμα εμφανίζει τις δυνάμεις του 2 από το 0 έως το 7 (2^0, 2^1,.., 2^7).

[22] Σε πολλούς μεταγλωττιστές η μετατροπή γίνεται αυτόματα, και δεν είναι απαραίτητη η χρήση της μετατροπής τύπου (type casting).

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   int i;
   for (i=0;i<=7;i++)
      printf("%d %f\n",i,pow(2,i));
   return 0;
}
```

double sqrt(double x);

Η συνάρτηση **sqrt()** επιστρέφει ως τιμή την τετραγωνική ρίζα της παραμέτρου x. Η συνάρτηση επιστρέφει μήνυμα λάθους αν η παράμετρος x είναι μικρότερη από το 0.

Παράδειγμα

Ο επόμενος κώδικας υπολογίζει τις ρίζες της εξίσωσης β' βαθμού $ax^2+bx+c=0$.

```
d=b*b-4*a*c;
r1=(-b+sqrt(d))/(2*a);
r1=(-b-sqrt(d))/(2*a);
printf("r1=%f r2=%f\n",r1,r2);
```

double fabs(double x);

Η συνάρτηση **fabs()** επιστρέφει την απόλυτη τιμή του δεκαδικού αριθμού x.

int abs(int x);

Η συνάρτηση **abs()** επιστρέφει την απόλυτη τιμή του ακέραιου αριθμού x.

double sin(double x);

Η συνάρτηση **sin()** επιστρέφει το ημίτονο της γωνίας x. Η τιμή της x πρέπει να δοθεί σε ακτίνια.

```
#include <stdio.h>
#include <math.h>
#define PI 3.141593
int main(void)
{
   int i;
   double rad;
   for (i=0;i<=360;i=i+10)
```

Ενδεικτικές σελίδες

Ενδεικτικές σελίδες

Παράρτημα

Β

Το ολοκληρωμένο περιβάλλον του DEV C++

Ενδεικτικές σελίδες

Το ολοκληρωμένο περιβάλλον του DEV C++

Το DEV C++ είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης για τις γλώσσες C και C++. Χρησιμοποιεί το μεταγλωττιστή GCC και παράγει 32μπιτες εφαρμογές (Win32) τόσο για τη γραμμή εντολών (console applications), όσο και για το περιβάλλον γραφικών των Windows (GUI applications). Το περιβάλλον DEV C++ διατίθεται δωρεάν στα πλαίσια της γενικής άδειας χρήσης της GNU (GNU GENERAL PUBLIC LICENSE).

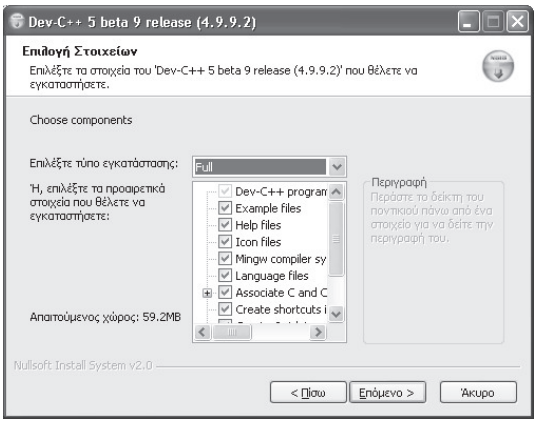
Στο συνοδευτικό CD, αλλά και στο δικτυακό τόπο του παρόντος βιβλίου **http://c.bytes.gr**, θα βρείτε το πρόγραμμα εγκατάστασης του ολοκληρωμένου περιβάλλοντος DEV C++.

Όλα τα προγράμματα που υπάρχουν σε αυτό το βιβλίο έχουν δοκιμαστεί στο περιβάλλον του DEV C++.

ΕΛΛΕΙΚΤΙΚΕΣ ΣΕΛΙΔΕΣ

Εγκατάσταση του DEV C++

Για να εγκαταστήσετε το ολοκληρωμένο περιβάλλον του DEV C++, εκτελέστε το αρχείο devcpp_setup.exe από το συνοδευτικό CD.



- Επιλέξτε ως γλώσσα εγκατάστασης την Ελληνική.
- Επιλέξτε ως τύπο εγκατάστασης Full.
- Αποδεχθείτε τον προτεινόμενο φάκελο C:\Dev-Cpp για την εγκατάσταση του προγράμματος.

Την πρώτη φορά που θα εκτελέσετε το πρόγραμμα θα σας ρωτήσει τη γλώσσα του περιβάλλοντος. Αν επιθυμείτε, επιλέξτε Ελληνική.

Το περιβάλλον του DEV C++ διαθέτει τη δυνατότητα αυτόματης συμπλήρωσης των εντολών. Αν επιθυμείτε την ενεργοποίησή της, ενεργοποιήστε την επιλογή *code completion feature* στο πλαίσιο διαλόγου που θα εμφανιστεί.

Οι βασικές λειτουργίες του DEV C++

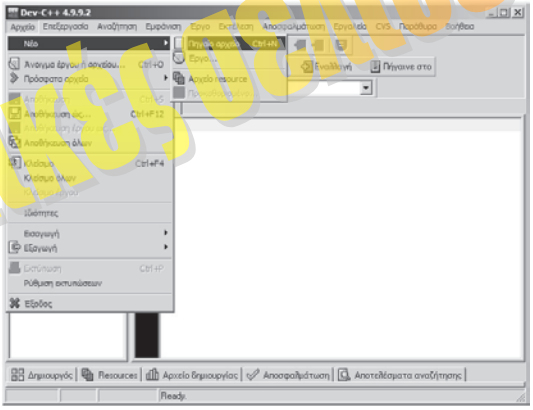
Στη συνέχεια αναφέρουμε τις βασικές λειτουργίες του περιβάλλοντος ώστε να μπορείτε μόνοι σας να δοκιμάζετε τα παραδείγματα του βιβλίου ή τα προγράμματα που καταστρώνετε μόνοι σας.

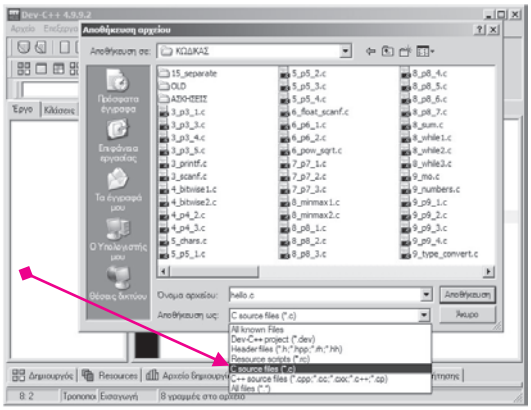
Το περιβάλλον του DEV C++ μας δίνει τη δυνατότητα να φτιάχνουμε τόσο απλά προγράμματα με ένα πηγαίο μόνο αρχείο, όσο και πολύπλοκες εφαρμογές (έργα) με πολλά πηγαία αρχεία.

Δημιουργία απλού προγράμματος με ένα πηγαίο αρχείο

Ενδεικτικές σελίδες

Από την επιλογή του μενού: **Αρχείο-Νέο-Πηγαίο αρχείο**



Αποθήκευση του αρχείου

Προσοχή πρέπει να δοθεί ώστε να αποθηκευτεί το αρχείο ως πηγαίο αρχείο της C.