

Εισαγωγή στη Γλώσσα Προγραμματισμού C



Παναγιώτης Κόκκας

Αναπληρωτής Καθηγητής , Τμήμα Φυσικής

Πανεπιστήμιο Ιωαννίνων

Ιωάννινα 2009

Εισαγωγή:

Οι σημειώσεις αυτές απευθύνονται στους πρωτοετείς φοιτητές του Τμήματος Φυσικής του Πανεπιστημίου Ιωαννίνων στα πλαίσια του μαθήματος “Γλώσσες προγραμματισμού Η/Υ”. Παρέχουν μια απλή εισαγωγή στην γλώσσα προγραμματισμού C. Πρέπει να συνδυαστούν με τις σημειώσεις που περιλαμβάνουν “*Προβλήματα στη γλώσσα προγραμματισμού C*”. Παρέχουν ένα σημαντικό βοήθημα κυρίως για το εργαστηριακό μέρος του μαθήματος. Καλύπτουν συνοπτικά την θεωρία, σε καμιά όμως περίπτωση δεν αντικαθιστούν το διδακτικό σύγγραμμα ούτε αντικαθιστούν τις παραδόσεις του μαθήματος.

Αναλυτικότερα το Κεφάλαιο I περιλαμβάνει εισαγωγή στο λειτουργικό σύστημα Linux, στο περιβάλλον επιφάνειας εργασίας KDE και στον επεξεργαστή κειμένου Emacs. Επί πλέον περιλαμβάνει οδηγίες για την σύνδεση-αποσύνδεση των χρηστών με το σύστημα Linux καθώς και εισαγωγή στην οργάνωση και στις απλές εντολές του. Τέλος περιλαμβάνει οδηγίες για τη χρήση του μεταγλωττιστή που χρησιμοποιείται για προγράμματα γραμμένα στη γλώσσα προγραμματισμού C.

Το Κεφάλαιο II περιλαμβάνει βασικές εισαγωγικές έννοιες που αφορούν τους τύπους δεδομένων, τις συναρτήσεις printf() και scanf(), διάφορους τελεστές και μαθηματικές συναρτήσεις που χρησιμοποιούνται κατά κόρον στα προγράμματά μας. Στο Κεφάλαιο III περιλαμβάνονται οι εντολές της C οι οποίες ελέγχουν την ροή εκτέλεσης σε ένα πρόγραμμα όπως οι εντολές if-else, switch, while, for και do-while.

Στο Κεφάλαιο IV αναπτύσσονται οι έννοιες των δεικτών και των πινάκων ενώ στο Κεφάλαιο V των δομών και των ενώσεων. Τέλος στο Κεφάλαιο VI περιλαμβάνονται οι συναρτήσεις οι οποίες αφορούν σε προσπέλαση αρχείων.

Η ύλη είναι οργανωμένη με τέτοιο τρόπο ώστε να καλύπτονται πλήρως οι απαιτήσεις ενός εξαμηνιαίου μαθήματος.

Κεφάλαιο Ι:

Εισαγωγή στο Linux και στο περιβάλλον εργασίας.

1.1 Τι είναι Λειτουργικό σύστημα (Operating System)

Κάθε Ηλεκτρονικός Υπολογιστής (Η/Υ) αποτελείται από δύο συνθετικά: Το **Υλικό** (Hardware) και το **Λογισμικό** (Software) του. Το Υλικό αποτελούν τα ηλεκτρικά, ηλεκτρονικά και μηχανικά μέρη του Η/Υ ενώ το Λογισμικό αποτελούν τα προγράμματα, δηλαδή οι οδηγίες για το “τι πρέπει να κάνει ο Η/Υ”.

Το βασικότερο μέρος του Λογισμικού αποτελεί το **Λειτουργικό Σύστημα** (Operating System) το οποίο αποτελείται από τα προγράμματα τα οποία είναι απαραίτητα για την αξιοποίηση του Υλικού (Hardware) και τη λειτουργία του συστήματος του Η/Υ. Αναλυτικότερα οι βασικές “αρμοδιότητες” του λειτουργικού συστήματος είναι:

- Να λειτουργεί ως ενδιάμεσος (Interface) ανάμεσα στον άνθρωπο και στη μηχανή.
- Να διαχειρίζεται τις δυνατότητες και τους πόρους (resources) του υπολογιστή έτσι ώστε να παράγεται χρήσιμο έργο (Resource Allocation).

Με αυτόν τον τρόπο το λειτουργικό σύστημα:

- Μεταφέρει εντολές ή απαιτήσεις του χρήστη στον Η/Υ.
- Δίνει χρήσιμες πληροφορίες στον χρήστη για την κατάσταση του συστήματος.
- Ενεργοποιεί και δίνει οδηγίες στην Κεντρική Μονάδα Επεξεργασίας (Central Process Unit) κατανέμοντας τον χρόνο λειτουργίας της στους διάφορους χρήστες.
- Διαχειρίζεται την Κεντρική Μνήμη (RAM) του συστήματος καθώς και τις συσκευές εξόδου και εισόδου, ελέγχοντας την ροή των δεδομένων (είσοδος) και την έξοδο των πληροφοριών (έξοδος).
- Ελέγχει την εκτέλεση των προγραμμάτων των διαφόρων χρηστών.
- Οργανώνει και διαχειρίζεται τα αρχεία του συστήματος.
- Εφαρμόζει μηχανισμούς οι οποίοι βελτιώνουν την Ασφάλεια του υπολογιστή από διάφορους κινδύνους.

1.2 Ένα σύντομο ιστορικό του Linux

Το **UNIX** είναι ένα από τα πλέον δημοφιλή λειτουργικά συστήματα παγκοσμίως, λόγω της μεγάλης βάσης υποστήριξης και διανομής του. Αρχικά στις αρχές της δεκαετίας του 1970 αναπτύχθηκε ως ένα σύστημα πολυδιεργασίας για μίνι υπολογιστές και μεγάλα συστήματα. Από τότε εξελίχθηκε και έγινε ένα από τα πιο ευρέως χρησιμοποιούμενα λειτουργικά συστήματα. Σήμερα απαντώνται πολλές

εκδόσεις του UNIX ανάλογα με την κατασκευάστρια εταιρεία Η/Υ. Για παράδειγμα σε μηχανές IBM ονομάζεται AIX, σε μηχανές SUN ονομάζεται Solaris, σε μηχανές DEC ονομάζεται Ultrix κτλ.

Το **Linux** αποτελεί ένα **ελεύθερο (Open Source)** λειτουργικό σύστημα-κλώνο του UNIX για προσωπικούς υπολογιστές. Αρχικά αναπτύχθηκε από τον Linus Torvalds ο οποίος ξεκίνησε να εργάζεται στο Linux το 1991, όταν ήταν φοιτητής στο Πανεπιστήμιο του Ελσίνκι στη Φινλανδία. Ο Linus κυκλοφόρησε την αρχική έκδοση του Linux ως ελεύθερο λογισμικό στο Internet, δημιουργώντας άθελα του ένα από τα μεγαλύτερα φαινόμενα όλων των εποχών στην ανάπτυξη λογισμικού. Σήμερα το Linux δημιουργείται και υποστηρίζεται από μια ομάδα πολλών χιλιάδων προγραμματιστών οι οποίοι συνεργάζονται μέσω του Internet. Επίσης υπάρχουν διάφορες εταιρείες για την παροχή υπηρεσιών υποστήριξης του Linux, και για τη συσκευασία τους σε διανομές εύκολες στην εγκατάστασή τους. Για τις ανάγκες του μαθήματος χρησιμοποιούμε την διανομή **Scientific Linux**.

1.3 Τα βασικά χαρακτηριστικά και δυνατότητες του Linux

Το Linux παρουσιάζει πολλά χαρακτηριστικά, τα σημαντικότερα από τα οποία είναι τα παρακάτω:

- Ταυτόχρονη εκτέλεση πολλών διεργασιών (πολυδιεργασία, multitasking). Το Linux επιτρέπει την ταυτόχρονη εκτέλεση πολλών διεργασιών – προγραμμάτων.
- Σύστημα πολλών χρηστών (multiuser). Το Linux επιτρέπει την χρήση του ίδιου υπολογιστή από πολλούς χρήστες ταυτόχρονα.
- Υποστήριξη συστημάτων πολλών επεξεργαστών. Το Linux υποστηρίζει συστήματα πολλών επεξεργαστών (όπως οι μητρικές κάρτες διπλού Pentium), με υποστήριξη μέχρι 16 επεξεργαστών σε ένα σύστημα, κάτι το οποίο είναι σημαντικό για διακομιστές υψηλών επιδόσεων και επιστημονικές εφαρμογές.
- Φορητότητα (Portability). Ο όρος φορητότητα χρησιμοποιείται στα λειτουργικά συστήματα για την δυνατότητα που έχουν αυτά να εγκαθίστανται σε υπολογιστές διαφόρων τύπων. Το Linux είναι κατ' εξοχήν φορητό λειτουργικό σύστημα, πράγμα το οποίο σημαίνει ότι μπορεί να εγκαθίσταται και να λειτουργεί αρμονικά σε υπολογιστές διαφόρων εταιρειών. Το Linux είναι δυνατό να λειτουργήσει σε μια μεγάλη ποικιλία αρχιτεκτονικών κεντρικής μονάδας επεξεργασίας (CPU), συμπεριλαμβανομένων των Intel x86, SPARC, Alpha, PowerPC, MIPS και m68k.
- Βοηθητικά προγράμματα ή προγράμματα κοινής χρήσης (utilities). Το Linux συνοδεύεται από μεγάλο αριθμό προγραμμάτων. Τα προγράμματα αυτά χωρίζονται σε δύο κατηγορίες:
 - Τα Γενικά προγράμματα τα οποία είναι μέρη του λειτουργικού συστήματος και είναι απολύτως απαραίτητα για τη λειτουργία του υπολογιστή.
 - Τα Επιμέρους προγράμματα τα οποία συνήθως εξυπηρετούν ειδικές απαιτήσεις του χρήστη. Για παράδειγμα ένας επεξεργαστής κειμένου (word processor), ένα λογιστικό πακέτο (spreadsheet) κτλ.
- Επικοινωνίες (communications). Το Linux είναι εξοπλισμένο με ειδικό λογισμικό επικοινωνίας το οποίο επιτρέπει την επικοινωνία μεταξύ διαφορετικών χρηστών ή ακόμη μεταξύ υπολογιστών διαφορετικών μεγεθών και τύπων οι οποίοι βρίσκονται σε διαφορετικές τοποθεσίες ακόμη και χώρες.

- Συνύπαρξη με άλλα λειτουργικά συστήματα. Το Linux είναι δυνατό να συνυπάρχει με επιτυχία σε ένα σύστημα στο οποίο υπάρχουν εγκατεστημένα άλλα λειτουργικά συστήματα, όπως για παράδειγμα Windows, OS/2, ή άλλες εκδόσεις του UNIX.

1.3 Η λογική οργάνωση του Linux

Το Linux αποτελείται από εκατοντάδες προγράμματα. Το σημαντικότερο αυτών αποτελεί ο **Πυρήνας (kernel)**. Ο πυρήνας είναι η ουσία του ίδιου του λειτουργικού συστήματος. Είναι ο κώδικας ο οποίος ελέγχει τη διασύνδεση μεταξύ των προγραμμάτων του χρήστη και των συσκευών υλικού, το χρονοπρογραμματισμό διεργασιών για την επίτευξη της πολυδιεργασίας, και πολλές άλλες πλευρές του συστήματος. Ο πυρήνας δεν είναι μια ξεχωριστή διεργασία η οποία εκτελείται στο σύστημα. Αντίθετα μπορείτε να θεωρήσετε τον πυρήνα ως ένα σύνολο ρουτινών, που βρίσκονται σταθερά στη μνήμη, στις οποίες κάθε διεργασία έχει πρόσβαση. Ο πυρήνας είναι όλος γραμμένος σε **γλώσσα προγραμματισμού C**, εκτός από 1000 περίπου γραμμές οι οποίες είναι γραμμένες σε Συμβολική γλώσσα χαμηλού επιπέδου (Assembly) και οι οποίες αλλάζουν από πλατφόρμα σε πλατφόρμα.

Θα ήταν πολύ δύσκολο για κάποιον χρήστη να επικοινωνεί απευθείας με τον πυρήνα. Με τον πυρήνα μπορούμε να επικοινωνήσουμε απευθείας μόνο με προγράμματα γραμμένα σε γλώσσα C. Για τον λόγο αυτόν δημιουργήθηκε το **Κέλυφος (Shell)**. Το κέλυφος είναι το πρόγραμμα το οποίο ασχολείται με την επικοινωνία με τον χρήστη, ερμηνεύοντας και εκτελώντας κάθε διαταγή του. Έτσι ο πυρήνας του Linux είναι το κεντρικό μέρος του λειτουργικού το οποίο εκτελεί τις εντολές, ενώ το κέλυφος είναι αυτό το οποίο το περιβάλλει και το κρύβει από τον χρήστη.

Στο Linux υπάρχουν αρκετά διαθέσιμα κελύφη τα οποία προέρχονται ως ένα αποτέλεσμα της εξέλιξης του λειτουργικού. Αυτά είναι:

- Το κέλυφος Bourne Again ή **bash** . Είναι το πλέον ευρέως χρησιμοποιούμενο και το πιο δυναμικό κέλυφος του Linux.
- Το κέλυφος C ή **csh** το οποίο αναπτύχθηκε στο Μπέρκλεϊ και είναι στο μεγαλύτερο μέρος του συμβατό με το κέλυφος Bourne.
- Το κέλυφος Korn ή **ksh** το οποίο είναι ίσως το πιο γνωστό στα συστήματα UNIX. Είναι συμβατό με το κέλυφος Bourne.
- Το κέλυφος Bourne ή **sh** είναι το πρωτότυπο κέλυφος το οποίο αναπτύχθηκε στα μέσα της δεκαετίας του '70.
- Το εμπλουτισμένο κέλυφος C ή **tcsh** .
- Το κέλυφος Z ή **zsh** το οποίο είναι το νεότερο κέλυφος. Είναι συμβατό με το κέλυφος Bourne.

Ο κάθε χρήστης είναι ελεύθερος να χρησιμοποιήσει το κέλυφος της αρεσκείας του. Εμείς στα πλαίσια του μαθήματός μας θα χρησιμοποιήσουμε το κέλυφος Bourne Again ή **bash**. Μία χρήσιμη εντολή η οποία μας επιτρέπει να ανακαλύψουμε ποιο είναι το κέλυφός μας είναι η ακόλουθη:

echo \$SHELL

Εκτός από το κέλυφος ο χρήστης “βλέπει” και το **Σύστημα Αρχείων (File System)**. Η δημιουργία ενός συστήματος αρχείων είναι ανάλογη με το φορμάρισμα ενός διαμερίσματος (partition) στο MS-

DOS ή σε άλλα λειτουργικά συστήματα. Το Linux υποστηρίζει διάφορους τύπους συστημάτων αρχείων για την αποθήκευση δεδομένων. Μερικά συστήματα αρχείων, όπως το σύστημα Third Extended (ext3fs), έχουν αναπτυχθεί ειδικά για το Linux. Με αυτόν τον τρόπο ο χρήστης μπορεί να οργανώσει τα δεδομένα και τα αρχεία του όπως θέλει.

Έτσι λογικά το Linux στον χρήστη ή στον προγραμματιστή, φαίνεται να αποτελείται από τα ακόλουθα τέσσερα επίπεδα, ξεκινώντας από το χαμηλότερο προς το υψηλότερο:

- Πυρήνας (Kernel) .
- Σύστημα αρχείων (File System).
- Κέλυφος (Shell).
- Εργαλεία – Εφαρμογές λογισμικού.

1.4 Τα X Windows και το περιβάλλον KDE.

Τα **X Windows** είναι η τυπική διασύνδεση γραφικών για μηχανήματα UNIX. Αποτελούν ένα δυναμικό περιβάλλον το οποίο υποστηρίζει πολλές εφαρμογές. Χρησιμοποιώντας το σύστημα X, ο χρήστης μπορεί να έχει ταυτόχρονα πολλά παράθυρα τερματικού στην οθόνη και κάθε παράθυρο να εμφανίζει μια διαφορετική περίοδο εργασίας σύνδεσης. Με την διασύνδεση X χρησιμοποιείται γενικά μια συσκευή κατάδειξης, όπως το ποντίκι. Με το Linux και τα X Windows, το σύστημα είναι ένας πραγματικός σταθμός εργασίας. Σε συνδυασμό με την δικτύωση TCP/IP, μπορείτε ακόμη και να εμφανίσετε στην οθόνη του Linux εφαρμογές των X Windows που εκτελούνται σε άλλα μηχανήματα.

Τα X Windows, που αρχικά αναπτύχθηκαν στο MIT, είναι ελεύθερα στη διανομή. Η έκδοση των X Windows που είναι διαθέσιμη για Linux είναι γνωστή ως **XFree86**. Το XFree86 είναι μια πλήρης διανομή του λογισμικού X, το οποίο περιέχει τον ίδιο τον διακομιστή X, πολλές εφαρμογές και βοηθητικά προγράμματα, βιβλιοθήκες προγραμματισμού και τεκμηρίωση.

Οι εφαρμογές X που είναι διαθέσιμες για Linux είναι πολυάριθμες. Την πλέον τυπική εφαρμογή X αποτελεί το **xterm**, το οποίο είναι ένας εξομοιωτής τερματικού που χρησιμοποιείται για τις περισσότερες εφαρμογές που βασίζονται σε κείμενο μέσα σε ένα παράθυρο X.

Το **K Desktop Environment** (Περιβάλλον επιφάνειας εργασίας K ή **KDE**) είναι ένα έργο λογισμικού Open Source το οποίο στοχεύει στην παροχή ενός σταθερού, φιλικού, και σύγχρονου περιβάλλοντος γραφικών εργασίας για συστήματα UNIX και κατά συνέπεια και Linux. Το KDE λειτουργεί επάνω από τα X Windows. Από την γέννησή του, τον Οκτώβριο του 1996, παρουσίασε θεαματική εξέλιξη. Αυτό οφείλεται εν μέρη στην επιλογή μιας εξαιρετικά καλής εργαλειοθήκης διασύνδεσης γραφικών με τον χρήστη, της Qt, καθώς και στην επακόλουθη επιλογή χρήσης της γλώσσας C++ και των αντικειμενοστραφών δυνατοτήτων της για την υλοποίηση του περιβάλλοντος. Το KDE εκτός της πλούσιας διασύνδεσης γραφικών με τον χρήστη και την άψογη διαχείριση των παραθύρων περιλαμβάνει μια πληθώρα βοηθητικών προγραμμάτων και εφαρμογών τα οποία συναγωνίζονται ευθέως πλέον τις δυνατότητες συστημάτων όπως (για παράδειγμα) η επιφάνεια εργασίας των Windows XP ή 2000.

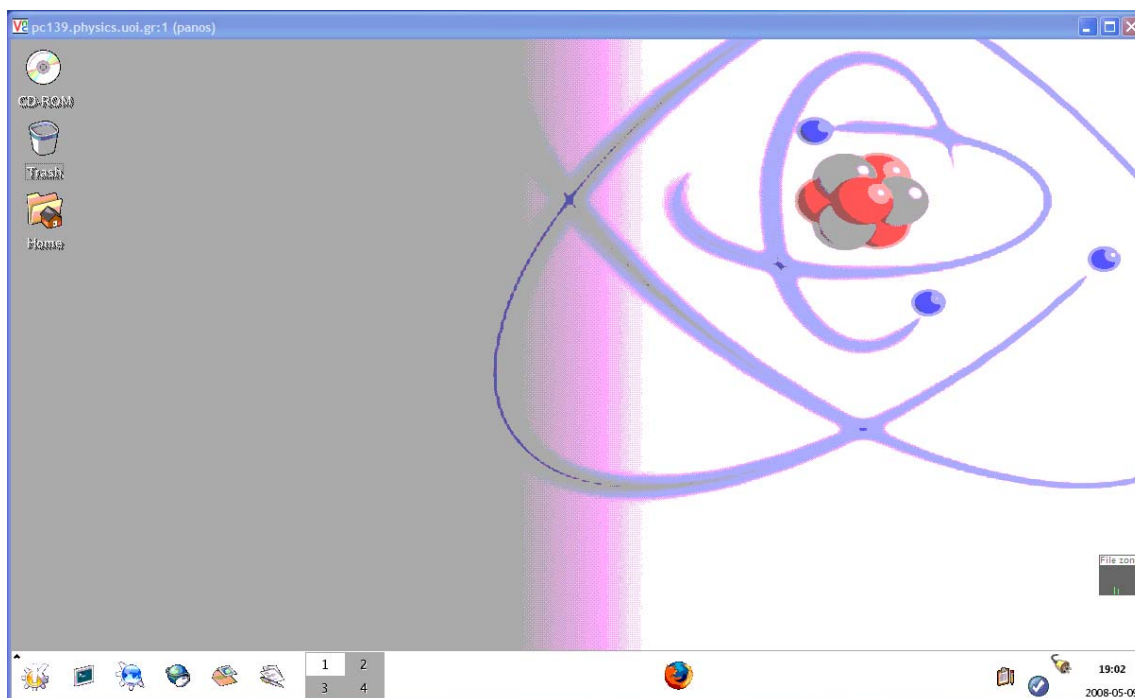
Εκτός του KDE και ένα δεύτερο περιβάλλον επιφάνειας εργασίας με το όνομα **GNOME** έχει αναπτυχθεί εξ ίσου πολύ καλά. Το GNOME παρουσιάζει τις ίδιες δυνατότητες με το KDE και αποτελεί πλέον θέμα προσωπικής επιλογής του χρήστη με ποιο από τα δύο περιβάλλοντα θα δουλέψει. Για τις ανάγκες του μαθήματος θα επιλέξουμε το περιβάλλον KDE.

1.4 Σύνδεση – Αποσύνδεση με το Linux και το περιβάλλον KDE.

Το εργαστήριο υπολογιστών του Φυσικού Τμήματος του Πανεπιστημίου Ιωαννίνων είναι εξοπλισμένο με πολλούς Υ/Η τύπου PC. Σε κάθε υπολογιστή έχουν εγκατασταθεί και συνυπάρχουν δύο λειτουργικά συστήματα τα WINDOWS και το Linux. Όταν ο χρήστης ανοίγει (power on) έναν υπολογιστή έχει την δυνατότητα να επιλέξει ανάμεσα στα δύο συστήματα. Για τις ανάγκες του μαθήματος επιλέξετε την δυνατότητα **Scientific Linux**.

Στη συνέχεια το λειτουργικό σύστημα Linux “φορτώνεται” στον υπολογιστή. Η όλη διαδικασία διαρκεί περίπου ένα λεπτό και με την ολοκλήρωσή της εμφανίζεται το ακόλουθο προτρεπτικό σήμα στην οθόνη: **Login** και στη συνέχεια: **Password**.

Κάθε χρήστης για να δουλέψει σε έναν συγκεκριμένο υπολογιστή πρέπει να είναι εφοδιασμένος με ένα **κωδικό όνομα** και ένα **κλειδί**. Στον χώρο δεξιά του **Login** και του **Password** γράφουμε το κωδικό όνομα (**user id**) και το κλειδί (**password**) αντίστοιχα. Στη συνέχεια αφού κάνουμε κλικ με το ποντίκι στο **Go!** Ξεκινάει και εμφανίζεται το **περιβάλλον επιφάνειας εργασίας KDE** (Σχήμα 1.1) και μπορούμε να αρχίσουμε την εργασία μας.



Σχήμα 1.1 : Το περιβάλλον επιφάνειας εργασίας KDE κατά την εκκίνηση.

Η χρήση του KDE είναι αρκετά εύκολη. Η λειτουργία των περισσότερων πραγμάτων είναι πολύ διαισθητική, το περιβάλλον φιλικό και έτσι ακόμη και ο αρχάριος χρήστης εύκολα μπορεί απλά να μαντέψει τι θέλει να κάνει. Κατά μήκος του κάτω μέρους της οθόνης εμφανίζεται η **παλέτα κουμπιών (panel)**. Η παλέτα κουμπιών εξυπηρετεί αρκετούς σκοπούς, μεταξύ των οποίων και την γρήγορη πρόσβαση σε εγκατεστημένες εφαρμογές. Ξεκινώντας από το αριστερό μέρος της παλέτας και κινούμενοι προς τα δεξιά (Σχήμα 1.2) διακρίνουμε τα ακόλουθα στην παλέτα κουμπιών:



Σχήμα 1.2 : Το αριστερό μέρος της παλέτας των κουμπιών του KDE.

- **Εκκίνηση των Εφαρμογών (Start Application)** (1^ο εικονίδιο). Μας δίνει το μενού με όλες τις εφαρμογές του KDE. Μέσω του μενού μπορούμε να διαλέξουμε όποια εφαρμογή θέλουμε να εκτελέσουμε.
- **Κονσόλα - Κέλυφος - Παράθυρο προσομοίωσης τερματικού (Shell – Terminal emulation window)** (2^ο εικονίδιο). Δημιουργεί ένα παράθυρο τερματικού από το οποίο μπορούμε να δώσουμε και να εκτελέσουμε τις εντολές μας στο Linux.
- **Konqueror : εφαρμογή περιήγησης στο Web (Konqueror Web Browser)** (3^ο εικονίδιο). Ξεκινάμε την περιήγησή μας στο διαδίκτυο.
- **Firefox : εφαρμογή περιήγησης στο Web (Firefox Web Browser)** (4^ο εικονίδιο). Ξεκινάμε την περιήγησή μας στο διαδίκτυο.
- **Email** (5^ο εικονίδιο). Εφαρμογή που μας επιτρέπει να διαχειριζόμαστε το ηλεκτρονικό μας ταχυδρομείο.
- Η εφαρμογή **OpenOffice** (6^ο εικονίδιο). Εφαρμογή αντίστοιχη του Office στα Windows.
- **Χώροι εργασίας (workspaces)** (Εικονίδια με χαρακτηριστικούς αριθμούς 1 έως 4). Μας παρέχει την δυνατότητα να χρησιμοποιούμε τέσσερις (έως και 16) διαφορετικούς χώρους εργασίας και με αυτόν τον τρόπο να αυξάνουμε κατά πολύ την χωρητικότητα του χώρου εργασίας μας. Ο κάθε χώρος χαρακτηρίζεται από έναν αριθμό (στην περίπτωση μας τους αριθμούς 1 έως 4). Η εναλλαγή μεταξύ των διαφορετικών χώρων εργασίας γίνεται με ένα απλό κλικ του ποντικιού στον αντίστοιχο κουμπί το οποίο χαρακτηρίζεται από έναν αριθμό. Ο χώρος εργασίας ο οποίος επιλέγεται κάθε φορά γίνεται λευκός (στην περίπτωση μας έχει επιλεγεί ο χώρος εργασίας με αριθμό 1).

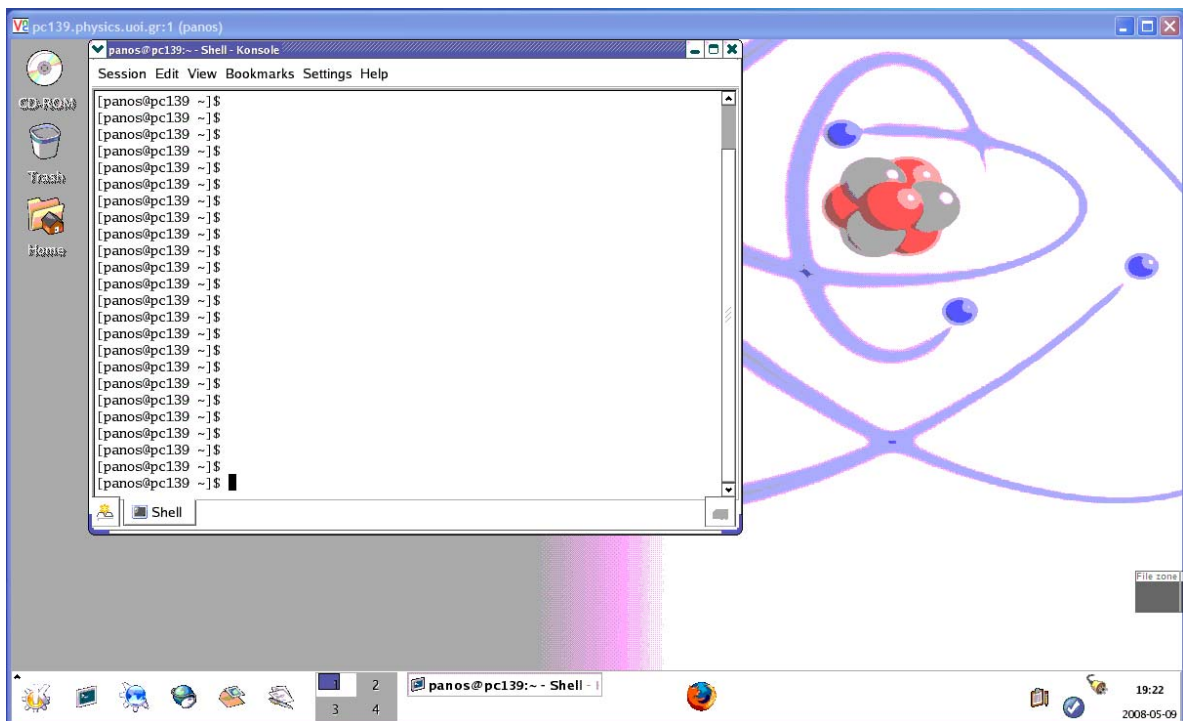
Στο δεξί μέρος της παλέτας κουμπιών (Σχήμα 1.3) διακρίνουμε τα ακόλουθα:



Σχήμα 1.3 : Το δεξί μέρος της παλέτας των κουμπιών του KDE.

- **Klipper – clipboard tool.**
- Πληροφορίες για τυχόν αναβαθμίσεις του λειτουργικού συστήματος.
- Ένδειξη ώρας.
- Ένδειξη ημερομηνίας.

Για να ξεκινήσουμε την εργασία μας στα πλαίσια του μαθήματος είναι απαραίτητη η δημιουργία μιας **κονσόλας** ή με πιο απλά λόγια το “άνοιγμα” ενός παραθύρου προσομοίωσης τερματικού. Αυτό γίνεται όπως έχουμε ήδη αναφέρει κάνοντας κλικ με το ποντίκι το 2^ο εικονίδιο στην παλέτα κουμπιών οπότε εμφανίζεται η εικόνα του Σχήματος 1.4. Στη συνέχεια μέσα στο παράθυρο το οποίο ανοίξαμε μπορούμε να εκτελέσουμε οποιαδήποτε εντολή του Linux.



Σχήμα 1.4: Δημιουργία μιας κονσόλας ή αλλιώς το “άνοιγμα” ενός παραθύρου προσομοίωσης τερματικού.

Εάν θέλουμε να τερματίσουμε την κονσόλα ή με πιο απλά λόγια να “κλείσουμε” το παράθυρο προσομοίωσης τερματικού μπορούμε είτε να εκτελέσουμε την εντολή

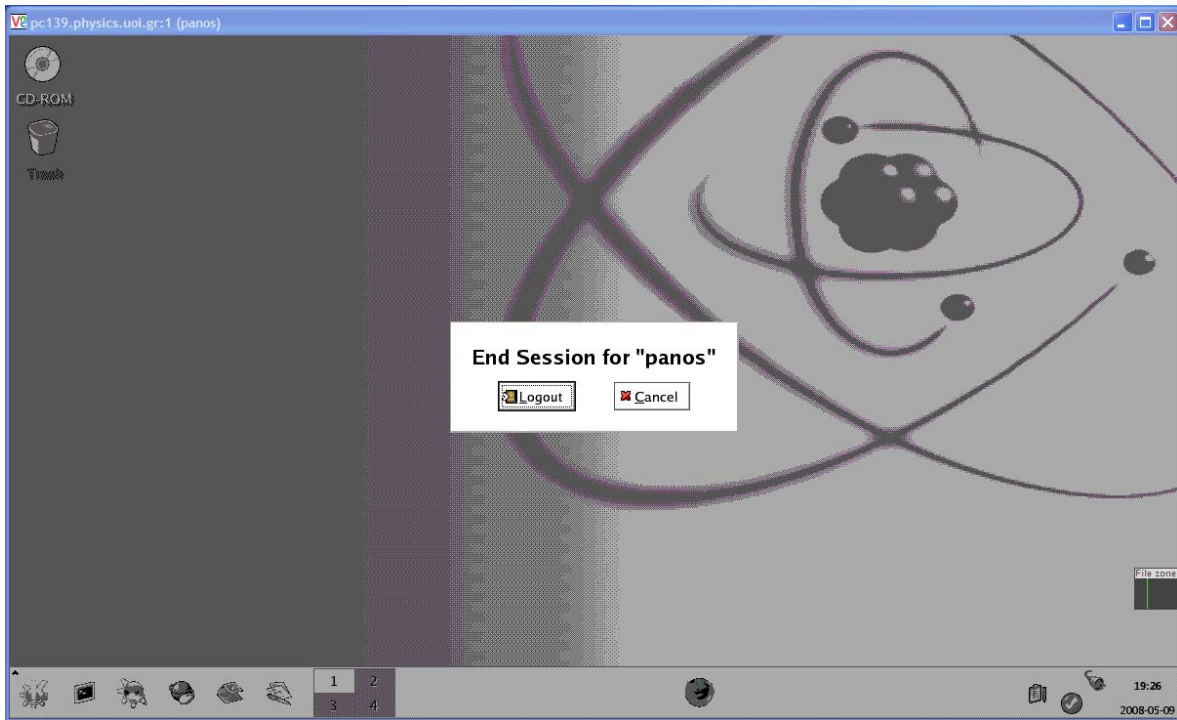
exit

είτε να κάνουμε κλικ με το ποντίκι στο επάνω δεξί μέρος και στο σύμβολο **x** του παραθύρου προσομοίωσης τερματικού.

Για να **αποσυνδεθούμε** από το περιβάλλον επιφάνειας εργασίας KDE πρέπει να ακολουθήσουμε τα εξής βήματα:

- Να τερματίσουμε ή με πιο απλά λόγια να κλείσουμε κάθε παράθυρο το οποίο έχουμε ανοίξει σε όλους τους χώρους εργασίας.
- Να κάνουμε κλικ με το ποντίκι στην **Εκκίνηση των Εφαρμογών (Start Application)** (1^ο εικονίδιο) και να επιλέξουμε τερματισμό εργασίας (**Logout**).
- Στην οθόνη του σχήματος 1.5 η οποία εμφανίζεται κάνουμε κλικ στο εικονίδιο **Logout**.

Με αυτόν τον τρόπο τερματίζεται το περιβάλλον KDE και επιστρέφουμε στην αρχική οθόνη η οποία περιέχει τα **Login** και **Password**



Σχήμα 1.5: Αποσύνδεση (Logout) από το περιβάλλον KDE.

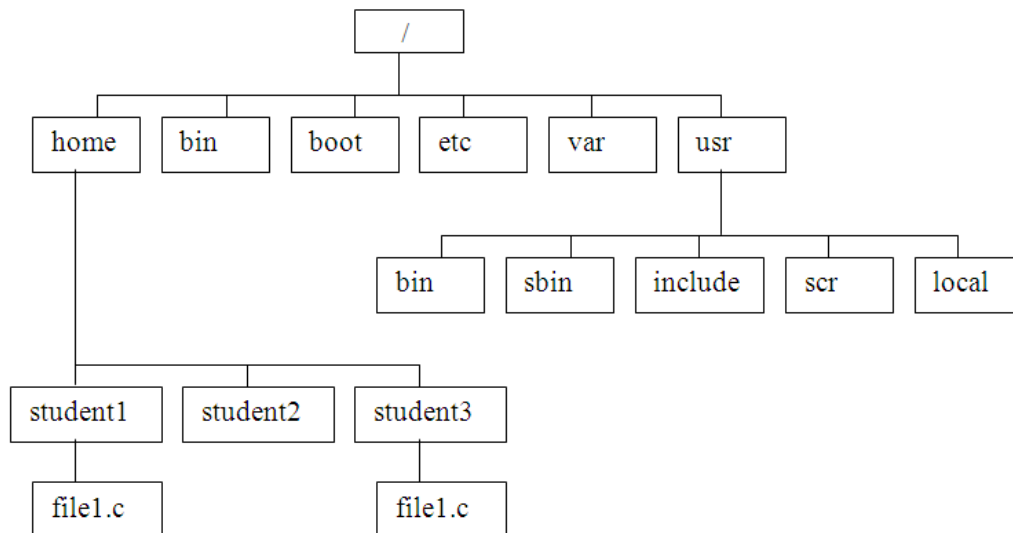
Εάν επιθυμούμε να **αποσυνδεθούμε πλήρως από το σύστημα Linux** και να κλείσουμε (power off) τον υπολογιστή μας ακολουθούμε τα εξής βήματα:

- Αφού αποσυνδεθούμε από το περιβάλλον KDE στην οθόνη η οποία προκύπτει κάνουμε κλικ στο εικονίδιο **Shutdown** και στη συνέχεια κλικ στο **OK**.
- Περιμένουμε περίπου μισό λεπτό έως ότου τερματιστεί πλήρως το Linux και να κλείσει ο υπολογιστής μας.

1.5 Το σύστημα αρχειοθέτησης του Linux.

Το σύστημα αρχειοθέτησης του Linux έχει την δομή του ανάστροφου δένδρου (tree structure) (σχήμα 1.6). Υπάρχει δηλαδή το **ριζικός κατάλογος** (root directory), ο οποίος συμβολίζεται με "/" και

που διακλαδίζεται σε περισσότερους **καταλόγους** (directories) οι οποίοι με την σειρά τους περιέχουν άλλους καταλόγους ή **αρχεία** (files).



Σχήμα 1.6: Η τυπική δομή του συστήματος αρχειοθέτησης του Linux.

Με τον όρο **αρχείο (file)** εννοούμε ένα σύνολο δεδομένων το οποίο είναι αποθηκευμένο σε μια συσκευή βοηθητικής μνήμης (σκληρός δίσκος, CD, δισκέτα κτλ.). Μερικά από τα χαρακτηριστικά ενός αρχείου είναι το όνομά του, το μέγεθός του, ο τύπος του, ο χρήστης που το δημιούργησε (ο οποίος ονομάζεται ιδιοκτήτης (owner) του), η ημερομηνία δημιουργίας του κτλ.

Ο **κατάλογος (directory)** είναι ένας φάκελος ο οποίος περιέχει ένα σύνολο από αρχεία. Ένας κατάλογος ή ένα αρχείο μπορεί να υπάρχει με το ίδιο όνομα κάτω από διαφορετικά “κλαδιά” του “δένδρου”, χωρίς να δημιουργείται σύγχυση, διότι η διαδρομή η οποία οδηγεί σε κάθε κατάλογο ή αρχείο από τον ριζικό κατάλογο είναι μοναδική και επομένως και το όνομά του.

Η διαδρομή η οποία οδηγεί σε κάθε κατάλογο ή αρχείο από τον ριζικό κατάλογο ονομάζεται “πλήρες όνομα” (**full pathname**). Για παράδειγμα στο σχήμα 1.6 υπάρχουν δύο αρχεία με το όνομα file1.c. Το πλήρες όνομα του ενός είναι “/home/student1/file1.c” ενώ του άλλου “/home/student3/file1.c”.

Ο κάθε χρήστης μπορεί να δίνει όποια ονόματα θέλει στους καταλόγους και στα αρχεία του, καλό είναι όμως τα ονόματα αυτά να περιγράφουν όσο γίνεται περισσότερο το περιεχόμενό τους.

ΠΡΟΣΟΧΗ !!! Τονίζεται πως στο Linux τα πεζά και τα κεφαλαία γράμματα αντιμετωπίζονται ως διαφορετικοί χαρακτήρες. Έτσι για παράδειγμα τα ονόματα “file1.c” και “File1.c” είναι διαφορετικά.

Στο σχήμα 1.6 παρουσιάζεται η τυπική δομή του συστήματος αρχειοθέτησης του Linux. Ο σημαντικότερος κατάλογος ο οποίος αφορά τους χρήστες είναι ο κατάλογος **/home**. Μέσα σε αυτόν τον κατάλογο υπάρχουν οι καταλόγοι των διαφόρων χρηστών οι οποίοι δουλεύουν στο συγκεκριμένο υπολογιστή. Κάθε φορά που προστίθεται ένας κωδικός για ένα νέο χρήστη στο σύστημα δημιουργείται μέσα στο /home ένας κατάλογος με το ίδιο όνομα όπως το κωδικό όνομα του χρήστη. Στο παράδειγμά μας υπάρχουν οι καταλόγοι για τρεις χρήστες με τα ονόματα “student1”, “student2”, και “student3”. Όταν για παράδειγμα ο χρήστης με το όνομα “student1” συνδέεται στο σύστημα έχει την δυνατότητα να δημιουργήσει ή να διαγράψει αρχεία και καταλόγους μόνο μέσα στον κατάλογο “/home/student1”.

Εκτός από τον κατάλογο /home, στον οποίο αποθηκεύονται τα αρχεία των διαφόρων χρηστών, υπάρχουν και αρκετοί ακόμη καταλόγοι οι οποίοι είναι σημαντικοί. Δείτε μερικούς από αυτούς, μαζί με τα περιεχόμενά τους:

- **/bin** : Περιέχει τις πιο βασικές εντολές του Linux, όπως το ls.
- **/boot** : Θέση όπου αποθηκεύεται ο πυρήνας και άλλα στοιχεία που χρησιμοποιούνται για την εκκίνηση του συστήματος.
- **/etc** : Περιέχει αρχεία τα οποία χρησιμοποιούνται από υποσυστήματα όπως η δικτύωση, το NFS και το ταχυδρομείο.
- **/var** : Περιέχει τα αρχεία διαχείρισης, όπως τα αρχεία καταγραφής (log files) που χρησιμοποιούνται από διάφορα βοηθητικά προγράμματα.
- **/usr/bin** : Περιέχει περαιτέρω εντολές του Linux.
- **/usr/sbin** : Περιέχει τις εντολές οι οποίες χρησιμοποιούνται από τον υπερχρήστη (superuser) του συστήματος. Σημειώνουμε εδώ πως ο υπερχρήστης έχει τον πλήρη έλεγχο και την διαχείριση του συστήματος.
- **/usr/include** : Είναι η τυπική θέση για τα αρχεία συμπερίληψης (include files) τα οποία χρησιμοποιούνται σε προγράμματα C, όπως το <stdio.h>.
- **/usr/src** : Αποτελεί την θέση του πηγαίου κώδικα για προγράμματα τα οποία μεταγλωττίζονται στο σύστημα.
- **/usr/local** : Περιέχει προγράμματα και αρχεία δεδομένων τα οποία έχουν προστεθεί τοπικά από τον διαχειριστή του συστήματος.

1.6 Οι απλές εντολές του Linux.

Το Linux προσφέρει έναν τεράστιο αριθμό εντολών ενώ παρέχει την δυνατότητα στον χρήστη να προσθέσει και καινούργιες. Στην παράγραφο αυτή περιγράφονται οι βασικότερες εντολές του Linux οι οποίες είναι απαραίτητες για τις ανάγκες του μαθήματος. Ξεκινάμε με τις εντολές οι οποίες αφορούν **εργασίες με καταλόγους**.

pwd : Η εντολή pwd μας δείχνει ποιος είναι ο τρέχων κατάλογος.

cd dirname : Αλλαγή του τρέχοντος καταλόγου στον νέο κατάλογο ο οποίος έχει το όνομα “dirname”.

Π.χ.:
cd c_directory
cd /usr/bin
cd ..

Η τελευταία εντολή μας μετακινεί στον γονικό κατάλογο. Στο Linux υπάρχουν δύο ειδικά ονόματα καταλόγων:

- . Συμβολίζει τον τρέχοντα κατάλογο
- .. Συμβολίζει τον γονικό κατάλογο

Τα παραπάνω ονόματα μπορούν να χρησιμοποιηθούν όπως οποιοδήποτε άλλο όνομα καταλόγου.

mkdir dirname : Δημιουργεί τον νέο κατάλογο με όνομα “dirname”.

Π.χ. mkdir new_catalog

rmdir dirname : Διαγράφει τον κατάλογο με όνομα “dirname”. Προσοχή: ο κατάλογος “dirname” πρέπει να είναι κενός διαφορετικά δεν διαγράφεται. Αυτό προστατεύει τον χρήστη από περίπτωση μεγάλου λάθους.

Π.χ. rmdir new_catalog

ls : Δείχνει τα περιεχόμενα του τρέχοντος καταλόγου. Το ls συντάσσεται και με περισσότερες παραμέτρους όπως:

ls -l : Μας δίνει περισσότερες πληροφορίες για κάθε ένα από τα περιεχόμενα του τρέχοντος καταλόγου, όπως είναι το μέγεθός του, ο τύπος του, ο χρήστης που το δημιούργησε, η ημερομηνία δημιουργίας του κτλ.

ls -lt : Εμφανίζει τα περιεχόμενα του τρέχοντος καταλόγου με ημερολογιακή σειρά. (σχήμα 1.7).

```
[student1@pc244 student1]$ ls -lt
total 1436
drwxr-xr-x  3 student1 student1  4096 Oct 21 15:24 Desktop
drwx----- 2 student1 student1  4096 Oct 17 17:29 Mail
-rw-r--r--  1 root      root      1454080 Oct 17 15:49 vnc.tar
drwxrwxr-x  3 student1 student1  4096 Aug  1 11:38 vnc
[student1@pc244 student1]$
```

Σχήμα 1.7: Η εντολή ls -lt.

Στο σχήμα 1.7 παρατηρούμε πως κάθε ένα αρχείο ή κατάλογο από τα περιεχόμενα του τρέχοντος καταλόγου χαρακτηρίζεται από 10 συγκεκριμένους χαρακτήρες π.χ.

drwxr-xr-x

Ο πρώτος χαρακτήρας δηλώνει τον τύπο και είναι συνήθως ένας από τους: **d** (κατάλογος - directory), - (αρχείο - file). Έτσι για παράδειγμα τα Desktop, Mail και vnc είναι κατάλογοι, ενώ το vnc.tar είναι αρχείο.

Οι τρεις επόμενοι χαρακτήρες **rwx** δηλώνουν τις άδειες που έχει ο χρήστης στο συγκεκριμένο αρχείο ή κατάλογο. Υπάρχουν τρεις τέτοιες άδειες στο Linux:

- Η άδεια **read** (ανάγνωση) σημαίνει ότι ο χρήστης μπορεί να δει τα περιεχόμενα του αρχείου - καταλόγου.
- Η άδεια **write** (εγγραφή) σημαίνει ότι ο χρήστης μπορεί να τροποποιήσει τα περιεχόμενα του αρχείου - καταλόγου.
- Η άδεια **execute** (εκτέλεση) σημαίνει ότι ο χρήστης μπορεί να εκτελέσει το αρχείο.

Οι επόμενοι τρεις χαρακτήρες δηλώνουν τις άδειες που έχουν τα μέλη της ομάδας που ανήκει ο κωδικός του συγκεκριμένου χρήστη, ενώ οι τελευταίοι τρεις χαρακτήρες δηλώνουν τις άδειες που έχουν οι άλλοι – υπόλοιποι χρήστες .

Οι άδειες ενός αρχείου-καταλόγου καθορίζονται από τον χρήστη με την εντολή **chmod**. Η σύνταξη της **chmod** είναι η ακόλουθη:

chmod w±άδεια filename :

Όπου w(ho) = u (user), g(group), o(other), + : παραχώρηση άδειας, - : αναίρεση άδειας. Τέλος όπου άδεια : r(read), w(write), x(execute).

Συνεχίζουμε με εντολές οι οποίες αφορούν **εργασίες με αρχεία**.

cp filename1 filename2 : Δημιουργία αντιγράφου του αρχείου με όνομα filename1 στο αρχείο με όνομα filename2.

Π.χ. `cp test1.c new_test1.c`

rm filename : Διαγραφή του αρχείου με όνομα filename.

mv filename1 filename2 : Μετονομασία του αρχείου με όνομα filename1 στο αρχείο με όνομα filename2.

more filename : Εμφάνιση των περιεχομένων του αρχείου με όνομα filename στην οθόνη.

tail -n filename : Εμφάνιση στην οθόνη των τελευταίων n γραμμών του αρχείου με όνομα filename.

diff filename1 filename2 : Εμφανίζει στην οθόνη τις διαφορές μεταξύ των αρχείων με όνομα filename1 και filename2.

Τέλος θέλουμε να τονίσουμε την χρησιμότητα της εντολής **man** η οποία μας παρέχει on-line βοήθεια για την σύνταξη οποιασδήποτε εντολής του Linux.

Π.χ. `man rm` : Μας περιγράφει την εντολή rm.

`man more` : Μας περιγράφει την εντολή more.

1.7 Ο επεξεργαστής κειμένου Emacs.

Οι επεξεργαστές κειμένου είναι οι πιο σημαντικές εφαρμογές στον κόσμο του UNIX. Χρησιμοποιούνται τόσο συχνά που πολλοί περνούν περισσότερο χρόνο σε έναν διορθωτή παρά κάπου

αλλού στο σύστημα UNIX. Το ίδιο ισχύει και στο Linux. Με την χρήση ενός επεξεργαστή κειμένου μπορούμε να αναπτύξουμε όλα τα προγράμματά μας σε οποιαδήποτε γλώσσα προγραμματισμού.

Δύο είναι οι πλέον ευρέως διαδεδομένοι επεξεργαστές κειμένου, ο **vi** και ο **Emacs**. Για τους αρχάριους χρήστες ο Emacs αποτελεί την καλύτερη λύση γιατί αποτελεί ένα ολοκληρωμένο περιβάλλον εξαιρετικά φιλικό προς τον χρήστη. Αυτός είναι και ο βασικός λόγος που τον επιλέξαμε για τις ανάγκες του μαθήματος. Το περιβάλλον του είναι γραφικό που σημαίνει ότι ο χρήστης χρησιμοποιεί το ποντίκι για να κινηθεί μέσα στο κείμενο καθώς και για να επιλέξει τις διάφορες εντολές του.

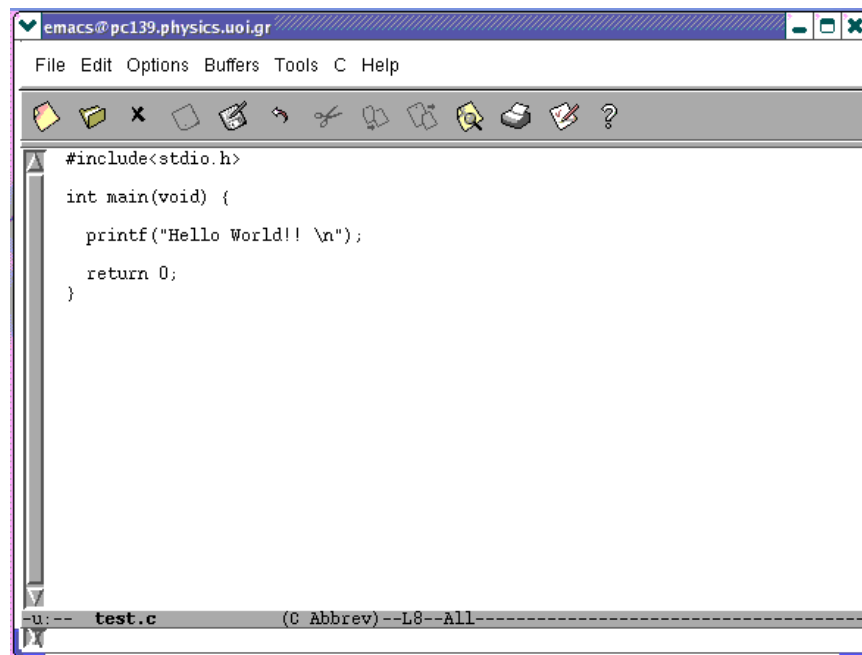
Ας υποθέσουμε ότι θέλουμε να γράψουμε το πρώτο απλό πρόγραμμα σε C, το οποίο θέλουμε να σώσουμε μέσα στο αρχείο με όνομα `test1.c`. Αφού ανοίξουμε μία κονσόλα στο περιβάλλον εργασίας KDE δίνουμε την εντολή:

emacs test1.c &

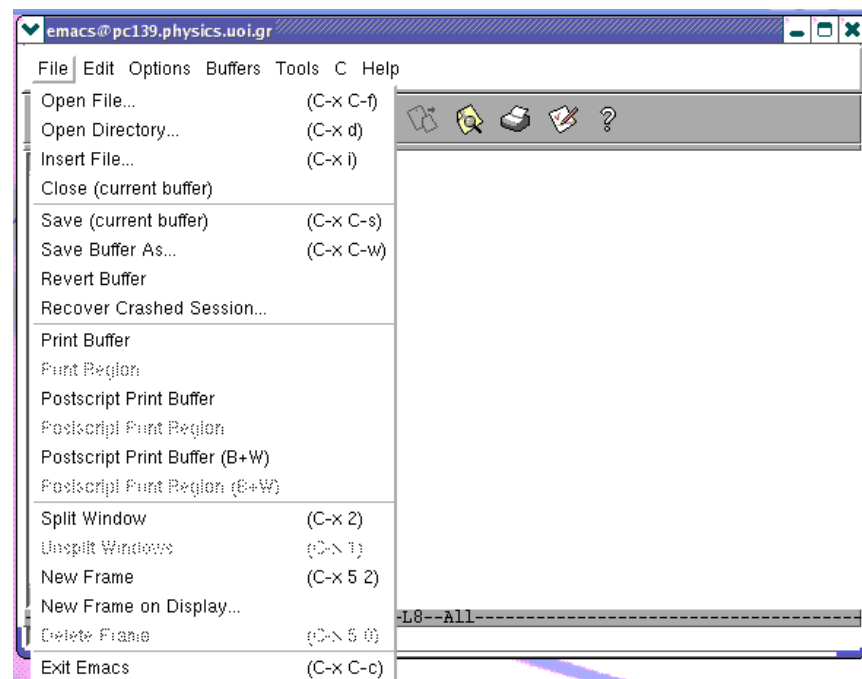
Η παραπάνω εντολή ανοίγει ένα νέο παράθυρο όπως φαίνεται στο σχήμα 1.8. Κάνοντας κλικ με το ποντίκι μέσα στο παράθυρο μπορούμε να αρχίσουμε να γράφουμε το κείμενό μας. Σημειώστε πως το σύμβολο `&` (εμπορικό και) χρησιμοποιείται ώστε να τοποθετήσει την εντολή μας στο παρασκήνιο. Με αυτόν τον τρόπο μπορούμε να χρησιμοποιήσουμε την κονσόλα που ανοίξαμε και για άλλες εντολές.

Παρατηρούμε πως στο άνω μέρος του παραθύρου του Emacs παρουσιάζεται η παλέτα κουμπιών του Emacs. Με την χρήση του ποντικιού μπορούμε απλά να δώσουμε τις διάφορες εντολές του Emacs κάνοντας κλικ στο κατάλληλο κουμπί.

Μόλις τελειώσουμε την δακτυλογράφηση του κειμένου μπορούμε να σώσουμε το κείμενο κάνοντας κλικ με το ποντίκι στο **File** και επιλέγοντας το **Save (Current Buffer)** όπως φαίνεται στο σχήμα 1.9. Εάν θέλουμε να τερματίσουμε τον Emacs στο τέλος της εργασίας μας επιλέγουμε με το ποντίκι το File και κάνουμε κλικ στο **Exit Emacs**.



Σχήμα 1.8: Το περιβάλλον του Emacs.



Σχήμα 1.9: Αποθήκευση αρχείου στο περιβάλλον του Emacs.

Κεφάλαιο II:

Εισαγωγή στη γλώσσα προγραμματισμού C.

2.1 Η γλώσσα C.

Η C είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου η οποία αναπτύχθηκε για πρώτη φορά στις αρχές της δεκαετίας του '70 από τον Dennis Ritchie στα AT&T Bell Labs. Η σημερινή μορφή της γλώσσας ακολουθεί το πρότυπο ANSI (American National Standards Institute) γιαυτό και φέρει την ονομασία "ANSI C". Ως γλώσσα υψηλού επιπέδου παρουσιάζει τα παρακάτω πλεονεκτήματα:

- **Αναγνωσιμότητα** : Τα προγράμματα διαβάζονται εύκολα.
- **Συντήρηση** : Τα προγράμματα είναι εύκολο να συντηρηθούν.
- **Μεταφερισιμότητα** : Τα προγράμματα εύκολα μεταφέρονται σε διαφορετικά λειτουργικά συστήματα και μηχανές.

Κάθε γλώσσα υψηλού επιπέδου χρειάζεται έναν **μεταγλωττιστή (compiler)** ο οποίος αναλαμβάνει να μεταφράσει τις γραμμές ενός προγράμματος σε **γλώσσα μηχανής**. Είναι η γλώσσα μηχανής αυτή η οποία γίνεται αντιληπτή από έναν Η/Υ. Με αυτόν τον τρόπο ένα πρόγραμμα αναπτύσσεται πρώτα σε γλώσσα προγραμματισμού C στη συνέχεια μεταγλωττίζεται σε γλώσσα μηχανής και εκτελείται από τον Η/Υ. Την χρήση του μεταγλωττιστή θα την δούμε σε επόμενη παράγραφο

2.2 Το πρώτο απλό πρόγραμμα C.

Στο περιβάλλον επιφάνειας εργασίας μας ανοίγουμε μία κονσόλα και δίνουμε την εντολή:

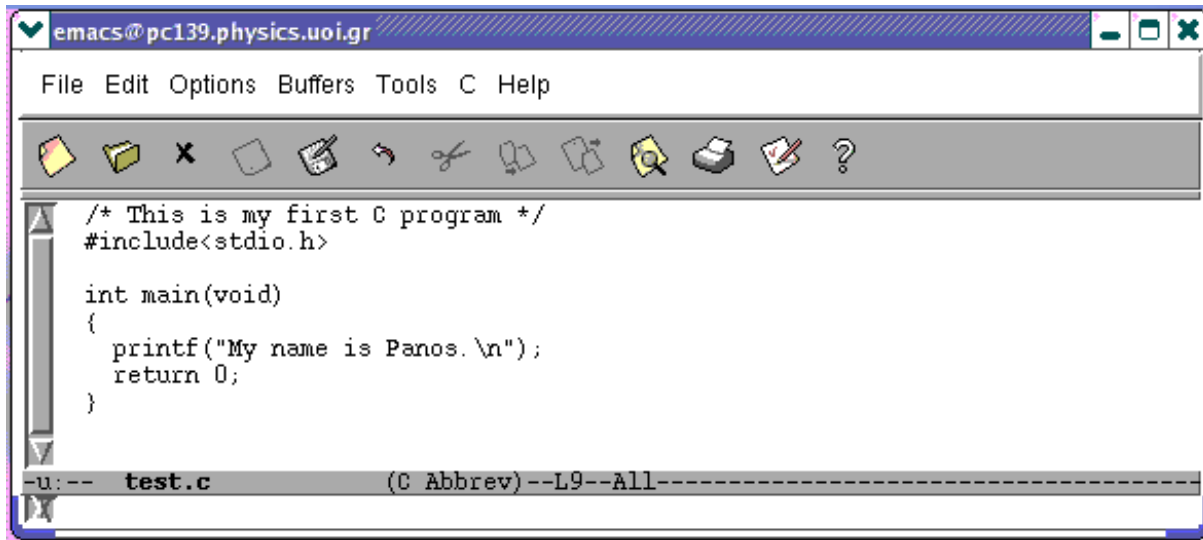
```
emacs example1.c &
```

Με αυτόν τον τρόπο ανοίγουμε ένα παράθυρο στον Emacs κάτω από το όνομα example1.c. Στη συνέχεια δακτυλογραφούμε το ακόλουθο πρόγραμμα (σχήμα 2.1):

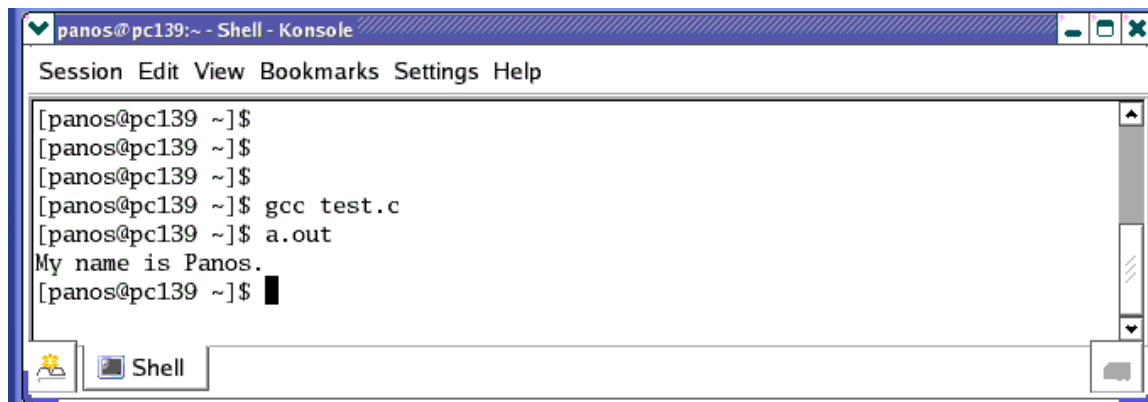
Όταν τελειώσουμε την δακτυλογράφηση αποθηκεύουμε το αρχείο μας εκτελώντας την εντολή File-Save (Current Buffer) (σχήμα 1.9). Στη συνέχεια για να εκτελέσουμε το πρόγραμμά μας πρέπει να το **μεταγλωττίσουμε (compilation)**. Αυτό γίνεται εκτελώντας την ακόλουθη εντολή:

```
gcc example1.c
```

Η παραπάνω εντολή κάνει την μεταγλώττιση και δημιουργεί το εκτελέσιμο αρχείο **a.out**. Το αρχείο αυτό περιέχει την μεταγλώττιση, του απλού προγράμματός μας, σε γλώσσα μηχανής έτοιμη να εκτελεστεί. Η εκτέλεση του αρχείου γίνεται απλά γράφοντας το όνομά του και πατώντας το Enter. Το αποτέλεσμα της εκτέλεσης εικονίζεται στο παρακάτω σχήμα 2.2:



Σχήμα 2.1: Ένα απλό πρόγραμμα σε C.



Σχήμα 2.2: Η εκτέλεση του απλού προγράμματος.

Στις επόμενες παραγράφους αναλύουμε τις απλές εντολές οι οποίες εμφανίζονται στο παραπάνω πρόγραμμα και τον μεταγλωττιστή.

2.3 Ανάλυση του απλού προγράμματος C.

Αναλύοντας μία-μία τις γραμμές του προγράμματος του σχήματος 2.1 παρατηρούμε τα ακόλουθα :

- **Σχόλια.** Η πρώτη γραμμή περιέχει ένα σχόλιο:
/* This is my first C program */

Παρατηρήστε ότι αυτή η γραμμή ξεκινά με έναν συνδυασμό καθέτου και αστερίσκου, `/*` και τελειώνει με `*/`. Στη C, το `/*` ονομάζεται **σημάδι αρχής σχολίου** και το `*/` είναι το **σημάδι τέλους σχολίου**. Ο C μεταγλωττιστής αγνοεί οτιδήποτε μεταξύ του αρχικού και του τελικού σημαδιού.

- **Η εντολή `#include` και τα αρχεία επικεφαλίδων.** Η δεύτερη γραμμή του προγράμματος είναι η ακόλουθη:

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

Η γραμμή αυτή ξεκινά με ένα `#`, το οποίο ακολουθείται από την εντολή `include`. Το `include` είναι μια εντολή του **προεπεξεργαστή** (ενός προγράμματος το οποίο κάνει κάποια προετοιμασία πριν μεταγλωττιστεί ο κώδικας) η οποία ψάχνει στην διαδρομή `/usr/include` να βρει το αρχείο `stdio.h`. Επί πλέον η εντολή `include` ζητά από τον προεπεξεργαστή να τοποθετήσει το αρχείο `stdio.h` στη θέση της εντολής μέσα στο πρόγραμμα.

Τα αρχεία τα οποία περιλαμβάνονται με την εντολή `#include`, όπως το `stdio.h`, ονομάζονται **αρχεία επικεφαλίδων** και περιέχουν μια πληθώρα συναρτήσεων έτοιμων προς χρήση για τον προγραμματιστή. Αποτελούν δηλαδή τις **βιβλιοθήκες** συναρτήσεων οι οποίες είναι διαθέσιμες στον προγραμματιστή. Εκτός από το `stdio.h` υπάρχουν και άλλα αρχεία επικεφαλίδων, όπως τα `math.h`, `stdlib.h`, `string.h` κτλ. Κάθε φορά ο προγραμματιστής πρέπει να περιλαμβάνει τα απαραίτητα αρχεία επικεφαλίδων στην αρχή του κώδικά του.

- **Η συνάρτηση `main()`** αποτελεί μια πολύ ειδική συνάρτηση της C. Κάθε πρόγραμμα πρέπει να περιέχει μία και μόνο μία συνάρτηση `main()` η οποία εκτελείται πάντα πρώτη ακόμη και εάν είναι στο κάτω μέρος του προγράμματος. Στο συγκεκριμένο παράδειγμα η συνάρτηση `main` έχει δηλωθεί ως ακέραιου τύπου (**`int`**) ενώ δεν περιέχει κάποιο όρισμα (**`void`**). Περισσότερα για τις συναρτήσεις παρουσιάζονται σε επόμενη παράγραφο.
- **Η συνάρτηση `printf()`** εμφανίζει και τυπώνει μηνύματα στην οθόνη μας. Ο χαρακτήρας `\n` ο οποίος εμφανίζεται στο τέλος του μηνύματος λέει στον υπολογιστή να μετακινήσει τον δρομέα στην αρχή της επόμενης γραμμής. Η συνάρτηση `printf()` ορίζεται στο αρχείο επικεφαλίδας `stdio.h`. Αυτός είναι και ο λόγος που περιλαμβάνουμε το συγκεκριμένο αρχείο επικεφαλίδας με την εντολή `#include` στην κορυφή του προγράμματος.
- **Η εντολή `return`.** Όλες οι συναρτήσεις της C μπορούν να επιστρέφουν τιμές. Στο συγκεκριμένο παράδειγμα η συνάρτηση `main` η οποία έχει δηλωθεί ως ακέραια συνάρτηση επιστέφει την τιμή μηδέν και με αυτόν τον τρόπο το πρόγραμμα τερματίζεται κανονικά.

2.4 Ο μεταγλωττιστής (compiler).

Στο Linux που χρησιμοποιούμε είναι διαθέσιμοι δύο μεταγλωττιστές για τη γλώσσα C, ο **cc** και ο **gcc**. Ο **cc** αποτελεί τον πιο απλό μεταγλωττιστή που συνοδεύει το σύστημα ενώ ο **gcc** αποτελεί τον GNU μεταγλωττιστή του Ιδρύματος Ελευθέρου Λογισμικού (Free Software Foundation). Ο μεταγλωττιστής gcc παρέχει σαφώς μεγαλύτερες δυνατότητες και αποτελεί τον καθιερωμένο μεταγλωττιστή για το Linux. Η εντολή

gcc example1.c

όπως εξηγήσαμε δημιουργεί το εκτελέσιμο αρχείο a.out. Εάν θέλουμε να δώσουμε στο εκτελέσιμο αρχείο ένα άλλο όνομα, πχ. το όνομα ex1 εκτελούμε την ακόλουθη εντολή:

gcc -o ex1 example1.c

Με αυτόν τον τρόπο το εκτελέσιμο αρχείο μας τώρα έχει το όνομα ex1 και όχι το a.out.

Ας εξηγήσουμε τώρα με περισσότερες λεπτομέρειες την διαδικασία μεταγλώττισης ενός προγράμματος. Ξεκινάμε με ένα πρόγραμμα το οποίο είναι γραμμένο σε γλώσσα C, και το οποίο ονομάζεται **πηγαίος κώδικας**. Το όνομα αυτού του πηγαίου κώδικα τελειώνει με την επέκταση **.c**.

Στη συνέχεια εκτελείται η εντολή gcc, όπως αναφέραμε παραπάνω, και η οποία περιλαμβάνει τις εξής τρεις διεργασίες:

- Ο **προεπεξεργαστής** είναι ένα πρόγραμμα το οποίο κάνει κάποια προετοιμασία πριν μεταγλωττιστεί ο πηγαίος κώδικας. Ποιο συγκεκριμένα ψάχνει στην διαδρομή **/usr/include** βρίσκει τα συγκεκριμένα αρχεία επικεφαλίδας και τα συμπεριλαμβάνει στον κώδικα.
- Ο **μεταγλωττιστής** είναι ένα πρόγραμμα το οποίο λαμβάνει τον κώδικα από τον προεπεξεργαστή και δημιουργεί ένα αρχείο το οποίο ονομάζεται **αντικειμενικό αρχείο (object file)**. Τα αντικειμενικά αρχεία έχουν την επέκταση **.o**. Τα αρχεία αυτά δεν εκτελούνται επειδή υπάρχει κάποιος κώδικας ο οποίος λείπει. Θα πρέπει να γίνει το επόμενο βήμα: η σύνδεση.
- Η **σύνδεση** γίνεται καλώντας ένα ειδικό πρόγραμμα το οποίο ονομάζεται **linker** (πρόγραμμα σύνδεσης) και που περιέχεται μέσα στον μεταγλωττιστή. Η σύνδεση χρησιμοποιείται για να συνδεθούν το αντικειμενικό αρχείο, η τυπική ANSI C βιβλιοθήκη και άλλες βιβλιοθήκες που έχει δημιουργήσει ο χρήστης, για να δημιουργηθεί το εκτελέσιμο αρχείο – ο δυαδικός κώδικας. Σε αυτή τη φάση, συνδυάζεται ο δυαδικός κώδικας των συναρτήσεων που καλούνται στον πηγαίο κώδικα με το αντικειμενικό αρχείο. Το αποτέλεσμα αποθηκεύεται σε ένα νέο αρχείο – το **εκτελέσιμο αρχείο (executable file)**.

Εδώ σημειώνουμε πως το αντικειμενικό αρχείο και το εκτελέσιμο αρχείο εξαρτώνται και τα δύο από την αρχιτεκτονική του υπολογιστή. Δεν συμβαίνει το ίδιο για τον πηγαίο κώδικα ο οποίος εφ' όσον έχει γραφεί σε ANSI C είναι ανεξάρτητος από τον υπολογιστή και μπορεί να μεταφέρεται όπως είναι.

Εάν ο χρήστης δεν επιθυμεί να δημιουργήσει εκτελέσιμο αρχείο αλλά να σταματήσει στην δημιουργία του αντικειμενικού αρχείου αυτό γίνεται με την εντολή:

gcc -c filename.c : Οπότε και δημιουργείται το αρχείο filename.o.

Ας υποθέσουμε πως το αρχείο filename.c περιέχει χρήσιμες συναρτήσεις για το πρόγραμμα το οποίο βρίσκεται στο αρχείο main.c. Τότε κατά την μεταγλώττιση του main.c πρέπει να συνδεθεί και το αρχείο filename.o. Αυτό γίνεται ως εξής:

gcc -o pgm main.c filename.o : οπότε το αρχείο pgm αποτελεί το εκτελέσιμο αρχείο μας.

2.5 Τα γενικά σε ένα πρόγραμμα C.

Στην παράγραφο αυτή δίνουμε τους ορισμούς σε βασικά στοιχεία της γλώσσας όπως μεταβλητές, σταθερές, αριθμητικοί τελεστές, παραστάσεις και εντολές.

- Οι **μεταβλητές** είναι οντότητες οι οποίες μπορούν να παίρνουν διαφορετικές τιμές. Επίσης δύνανται να αλλάζουν τιμές στη ροή του προγράμματος. Για παράδειγμα, σκεφτείτε τις παρακάτω τέσσερις γραμμές:

a = 1.35;

b = 32;

a = 7.5;

b = 10;

Σε αυτές ορίζονται δύο μεταβλητές οι a και b οι οποίες αρχικοποιούνται στις τιμές 1.35 και 32 αντίστοιχα και στη συνέχεια αλλάζουν τιμές. Ως ονόματα μεταβλητών μπορούμε να χρησιμοποιήσουμε μονολεκτικές λέξεις χρησιμοποιώντας πεζούς, κεφαλαίους χαρακτήρες και αριθμούς. Απαγορεύονται τα ονόματα τα οποία είναι δεσμευμένα από την γλώσσα όπως main, return, for κτλ. Επίσης απαγορεύεται η χρήση χαρακτήρων όπως (, ; ' " ? κτλ).

- Οι **σταθερές** είναι οντότητες οι οποίες αναφέρονται σε σταθερές τιμές, οι τιμές των οποίων δεν μπορούν να μεταβάλλονται από κανένα πρόγραμμα. Η χρήση των σταθερών είναι φυσιολογική και διαισθητική. Παραδείγματα σταθερών είναι οι αριθμοί 100 και 3.1415 όπως και οι χαρακτήρες 'k', 'u' κτλ.
- Στη C έχουν οριστεί οι παρακάτω **αριθμητικοί τελεστές** οι οποίοι μας επιτρέπουν να κάνουμε πράξεις:

<u>Τελεστής</u>	<u>Σημασία</u>
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο διαίρεσης

Ο τελευταίος τελεστής αφορά το υπόλοιπο της διαίρεσης δύο ακεραίων πχ. 7%5 ισούται με 2.

- Μία **παράσταση** είναι ένας συνδυασμός σταθερών, μεταβλητών, τελεστών ή και συναρτήσεων. Οι παραστάσεις χρησιμοποιούνται για να δηλώσουν **υπολογισμούς**. Για παράδειγμα, σκεφτείτε τις παρακάτω γραμμές:

```
a = 10;
b = 5;
c = (a+5)*b;
```

Ο όρος $(a+5)*b$ αποτελεί μια παράσταση. Η μεταβλητή c λαμβάνει την τιμή 75.

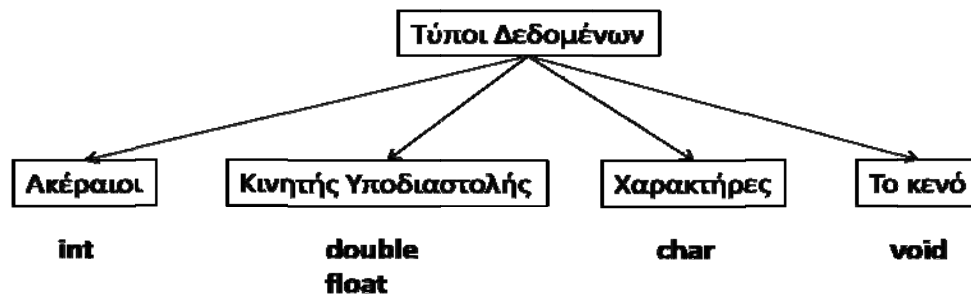
- Μία **εντολή** είναι μια οδηγία η οποία τελειώνει πάντοτε με ένα **ερωτηματικό**. Στο προηγούμενο παράδειγμα η γραμμή

```
c = (a+5)*b;
```

αποτελεί μια εντολή. Η τιμή 75 η οποία υπολογίζεται από την παράσταση $(a+5)*b$ προσδίδεται στην μεταβλητή c .

2.6 Οι τύποι δεδομένων.

Στο σχήμα 2.3 εικονίζονται οι τέσσερις βασικές κατηγορίες δεδομένων που συναντούμε στη C καθώς και οι **πέντε βασικοί τύποι δεδομένων**. Οι πέντε αυτοί τύποι χαρακτηρίζουν τις μεταβλητές τις οποίες ορίζουμε στα προγράμματά μας.



Σχήμα 2.3: Οι τέσσερις βασικές κατηγορίες δεδομένων που συναντούμε στη C καθώς και οι πέντε βασικοί τύποι δεδομένων.

Σε έναν υπολογιστή διαχωρίζουμε τους αριθμούς σε δύο κατηγορίες στους **ακεραίους** και σε αυτούς που δύνανται να έχουν δεκαδικά ψηφία. Οι τελευταίοι ονομάζονται αριθμοί **Κινητής Υποδιαστολής**. Ο λόγος αυτού του διαχωρισμού έγκειται στο ότι ο υπολογιστής κωδικοποιεί διαφορετικά του ακεραίου αριθμούς από αυτούς που διαθέτουν δεκαδικά ψηφία. Με αυτό τον τρόπο προκύπτουν δύο κατηγορίες αριθμητικών δεδομένων.

- **Ακέραια Δεδομένα:** Περιγράφονται με το πρόθεμα **int**. Ο τύπος `int` αναφέρεται σε μεταβλητές ή σταθερές οι οποίες μπορούν να διαχειριστούν ακεραίους αριθμούς. Στους περισσότερους υπολογιστές ένας ακέραιος `int` έχει μέγεθος 32 bit (4 byte). Αυτό σημαίνει πως το εύρος των ακεραίων αριθμών είναι από 2147483647 (δηλαδή $2^{31}-1$) έως -2147483648 .

Παράδειγμα δήλωσης και αρχικοποίησης δύο ακεραίων μεταβλητών:

```
int a,b;
a=5;
b=7;
```

- **Δεδομένα Κινητής Υποδιαστολής:** Περιγράφονται από τα προθέματα **double** και **float**. Οι τύποι `double` και `float` αναφέρονται σε μεταβλητές ή σταθερές οι οποίες μπορούν να διαχειριστούν αριθμούς κινητής υποδιαστολής (αριθμούς με δεκαδικά ψηφία). Ο τύπος `float` έχει μέγεθος 32 bit (4 byte) ενώ ο τύπος `double` 64 bit (8 byte). Ο τύπος `float` έχει τη δυνατότητα να συγγραφήσει αριθμούς με εύρος από $\pm 1.4 \times 10^{-45}$ έως $\pm 3.4 \times 10^{-38}$ και έως 7 σημαντικά ψηφία συνολικά. Ο τύπος `double` έχει τη δυνατότητα να συγγραφήσει εύρος αριθμών από $\pm 4.9 \times 10^{-324}$ έως $\pm 1.8 \times 10^{308}$ και έως 15 σημαντικά ψηφία συνολικά. Εδώ πρέπει να τονίσουμε πως σε αντίθεση με το ανθρώπινο μυαλό ο υπολογιστής δεν έχει άπειρη ακρίβεια στην αποθήκευση και διαχείριση των αριθμητικών δεδομένων. Είναι μία μηχανή πεπερασμένης ακρίβειας. Παρ' όλα αυτά συνήθως στους υπολογισμούς μας δεν χρειαζόμαστε ακρίβεια μεγαλύτερη από αυτή που παρέχει ο τύπος `double`, τον οποίο και θα πρέπει να χρησιμοποιούμε κατά κόρον. Ο τύπος `double` αναφέρεται συνήθως και ως διπλής ακρίβειας για ευνόητους λόγους.

Παράδειγμα δήλωσης και αρχικοποίησης δύο μεταβλητών κινητής υποδιαστολής:

```
double x;
float y;
x=3.14159;
y=7.;
```

Προσοχή! Παρ' ότι η μεταβλητή `y` αρχικοποιείται σε έναν "ακέραιο" αριθμό τον 7, καλό είναι ΠΑΝΤΟΤΕ να βάζουμε και μια τελίτσα μετά το 7 ώστε να δηλώνουμε κατευθείαν πως είναι κινητής υποδιαστολής.

- **Δεδομένα Χαρακτήρων:** Περιγράφονται με το πρόθεμα **char**. Ο τύπος `char` αντιπροσωπεύει ένα χαρακτήρα από τον σύνολο των χαρακτήρων οι οποίοι χρησιμοποιούνται στον υπολογιστή μας. Ένας υπολογιστής τύπου PC περιέχει ένα σύνολο από 256 χαρακτήρες. Στο σύνολο των χαρακτήρων για παράδειγμα περιέχονται όλα τα γράμματα της αλφαβήτου (Α έως Ζ και α έως z), όλα τα ψηφία (0 έως 9), όλοι οι χαρακτήρες τους οποίους βλέπουμε στο πληκτρολόγιό μας. Στον υπολογιστή μας ο κάθε χαρακτήρας καταλαμβάνει μέγεθος ίσο με **8 bit** (1 byte). Σε κάθε λοιπόν χαρακτήρα αντιστοιχείται ένας αριθμός των 8 bit, δηλαδή από το 0 (2^0-1) έως το 255 (2^8-1) ο οποίος ονομάζεται **ASCII** (American Standard Code for Information Interchange) κωδικός. Αυτό που πρέπει να γίνει κατανοητό είναι πως όταν πληκτρολογούμε για παράδειγμα τον χαρακτήρα Α, αυτός για τον υπολογιστή μας αντιστοιχεί στην αριθμητική τιμή 65 (ή σε δεκαεξαδική μορφή 0x41), ο χαρακτήρας α αντιστοιχεί στην αριθμητική τιμή 97 (ή σε δεκαεξαδική μορφή 0x61). Στον Πίνακα 2.1 εικονίζεται το σετ των κωδικών ASCII για τους βασικούς 128 χαρακτήρες του υπολογιστή.

- **Το κενό:** Η δεσμευμένη λέξη **void** είναι ένας προσδιοριστής τύπου και χρησιμοποιείται κυρίως για την δήλωση συναρτήσεων οι οποίες δεν επιστρέφουν τιμές. Χρησιμοποιείται επίσης για να δηλώσει την έλλειψη ορισμάτων σε συναρτήσεις (πχ στη συνάρτηση `main(void)`).

Εδώ πρέπει να σημειώσουμε πως εκτός των παραπάνω βασικών τύπων υπάρχουν και οι ακόλουθοι **τροποποιητές τύπων δεδομένων (qualifiers)** οι οποίοι εφαρμόζονται σε αυτούς. Έτσι έχουμε:

- **Οι τροποποιητές short και long** αναφέρονται στους τύπους δεδομένων `int`. Έτσι μπορούμε να έχουμε:

`short int`

`long int`

Ο `short int` συνήθως έχει μήκος 16 bit (2 byte) ενώ ο `long int` 32 bit (4 byte). Επίσης ο προσδιοριστής `long` εφαρμόζεται και στον τύπο `double`. Έτσι ο τύπος

`long double`

έχει συνήθως μήκος 80 bit (10 byte) και χρησιμοποιείται όταν θέλουμε την μέγιστη δυνατή ακρίβεια στους υπολογισμούς μας.

- **Οι τροποποιητές unsigned και signed** αναφέρονται στους τύπους δεδομένων `int`. Οι `unsigned` είναι πάντοτε θετικοί αριθμοί ή μηδέν ενώ οι `signed` μπορεί να είναι θετικοί μηδέν ή αρνητικοί αριθμοί. Για παράδειγμα μπορούμε να έχουμε τις ακόλουθες μεταβλητές:

`unsigned int a;`

`signed int b;`

Επίσης μπορούμε να έχουμε συνδυασμούς όπως:

`unsigned short int e;`

`signed long int f;`

Τέλος στον πίνακα 2.2 συνοψίζονται οι βασικοί τύποι δεδομένων και το εύρος τιμών τους.

Πίνακας 2.1 Οι κωδικοί ASCII για τους βασικούς 128 χαρακτήρες του υπολογιστή.

Non-Printing Characters					Printing Characters								
Name	Control	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
null	ctrl-@	0	00	NUL	32	20	Space	64	40	@	96	60	`
start header	ctrl-A	1	01	SOH	33	21	!	65	41	A	97	61	a
start of text	ctrl-B	2	02	STX	34	22	"	66	42	B	98	62	b
end of text	ctrl-C	3	03	ETX	35	23	#	67	43	C	99	63	c
end of xmit	ctrl-D	4	04	EOT	36	24	\$	68	44	D	100	64	d
enquiry	ctrl-E	5	05	ENQ	37	25	%	69	45	E	101	65	e
acknowledge	ctrl-F	6	06	ACK	38	26	&	70	46	F	102	66	f
bell	ctrl-G	7	07	BEL	39	27	'	71	47	G	103	67	g

backspace	ctrl-H	8	08	BS	40	28	(72	48	H	104	68	h
horizontal tab	ctrl-I	9	09	HT	41	29)	73	49	I	105	69	i
line feed	ctrl-J	10	0A	LF	42	2A	*	74	4A	J	106	6A	j
vertical tab	ctrl-K	11	0B	VT	43	2B	+	75	4B	K	107	6B	k
form feed	ctrl-L	12	0C	FF	44	2C	,	76	4C	L	108	6C	l
carriage feed	ctrl-M	13	0D	CR	45	2D	-	77	4D	M	109	6D	m
shift out	ctrl-N	14	0E	SO	46	2E	.	78	4E	N	110	6E	n
shift in	ctrl-O	15	0F	SI	47	2F	/	79	4F	O	111	6F	o
data line escape	ctrl-P	16	10	DLE	48	30	0	80	50	P	112	70	p
dev.cntrl.1	ctrl-Q	17	11	DC1	49	31	1	81	51	Q	113	71	q
dev.cntrl.2	ctrl-R	18	12	DC2	50	32	2	82	52	R	114	72	r
dev.cntrl.3	ctrl-S	19	13	DC3	51	33	3	83	53	S	115	73	s
dev.cntrl.4	ctrl-T	20	14	DC4	52	34	4	84	54	T	116	74	t
neg acknowledge	ctrl-U	21	15	NAK	53	35	5	85	55	U	117	75	u
sync.idel	ctrl-V	22	16	SYN	54	36	6	86	56	V	118	76	v
end of xmit block	ctrl-W	23	17	ETB	55	37	7	87	57	W	119	77	w
cancel	ctrl-X	24	18	CAN	56	38	8	88	58	X	120	78	x
end of medium	ctrl-Y	25	19	EM	57	39	9	89	59	Y	121	79	y
substitute	ctrl-Z	26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
escape	ctrl-[27	1B	ESC	59	3B	;	91	5B	[123	7B	{
file separator	ctrl-\	28	1C	FS	60	3C	<	92	5C	\	124	7C	
group separator	ctrl-]	29	1D	GS	61	3D	=	93	5D]	125	7D	}
record separator	ctrl-^	30	1E	RS	62	3E	>	94	5E	^	126	7E	~
unit separator	ctrl- _~	31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Πίνακας 2.2 . Οι βασικοί τύποι δεδομένων και το εύρος τιμών τους.

Τύπος Δεδομένων	Εύρος σε bits	Εύρος Τιμών
char	8	
int	32	-2147483648 έως 2147483647
short int	16	-32768 έως 32767
long int	32	Όπως και ο int
float	32	$\pm 1.4 \times 10^{-45}$ έως $\pm 3.4 \times 10^{38}$
double	64	$\pm 4.9 \times 10^{-324}$ έως $\pm 1.8 \times 10^{308}$

2.7 Περισσότερα για τις μεταβλητές.

Οι μεταβλητές έχουν τη δυνατότητα να αποθηκεύουν και να διαχειρίζονται δεδομένα, κατά συνέπεια είναι οι πιο βασικές οντότητες σε ένα πρόγραμμα. Για τον λόγο αυτό πρέπει να τονίσουμε εδώ τους βασικούς κανόνες που τις διέπουν.

Η **δήλωση** όλων των μεταβλητών που χρειαζόμαστε γίνεται πάντοτε πάνω-πάνω και στην αρχή κάθε συνάρτησης. Για παράδειγμα όταν αναπτύσσουμε την συνάρτηση `main` το πρώτο πράγμα που κάνουμε μετά την γραμμή

```
int main(void){
```

είναι η δήλωση όλων των μεταβλητών που χρειάζεται η συνάρτηση.

Η **εμβέλεια** κάθε μεταβλητής ξεκινά από την γραμμή της δήλωσής της μέχρι το τέλος της συνάρτησης μέσα στην οποία έχει δηλωθεί. Εάν ένα πρόγραμμα αποτελείται από δύο ή περισσότερες συναρτήσεις τότε κάθε συνάρτηση έχει τις δικές της μεταβλητές οι οποίες ισχύουν μόνο και μόνο μέσα στην συγκεκριμένη συνάρτηση.

Στην C υπάρχει η δυνατότητα να ορίζουμε μεταβλητές και έξω από τις συναρτήσεις αμέσως μετά τα αρχικά `#include`. Σε αυτή την περίπτωση οι συγκεκριμένες μεταβλητές έχουν εμβέλεια μέσα σε κάθε συνάρτηση που περιέχει το πρόγραμμα. Αυτές ειδικά οι μεταβλητές ονομάζονται *Γενικές Μεταβλητές* (global variables). Συνιστάται σε αρχάριους στον προγραμματισμό να ΜΗΝ χρησιμοποιούν τέτοιου είδους γενικές μεταβλητές.

Αμέσως μετά την δήλωση των μεταβλητών σε μία συνάρτηση πρέπει να ακολουθεί η **αρχικοποίησή** τους. Εδώ χρειάζεται προσοχή διότι όταν ορίζουμε μια μεταβλητή αυτή δεν αρχικοποιείται αυτόματα (πχ. στην τιμή 0) αλλά περιέχει μια τυχαία τιμή (η οποία βρίσκεται εκείνη την στιγμή στην μνήμη του υπολογιστή που καταλαμβάνει η συγκεκριμένη μεταβλητή που ορίσαμε). Ο προγραμματιστής οφείλει να αρχικοποιεί κάθε μεταβλητή που έχει ορίσει ΠΡΙΝ την χρησιμοποιήσει. Αφού ο προγραμματιστής δηλώσει και αρχικοποιήσει τις μεταβλητές στη συνέχεια μπορεί να αναπτύξει το πρόγραμμά του κάνοντας πράξεις, εισάγοντας λογική σε αυτό κτλ.

Σχετικά με τον τύπο των μεταβλητών που πρέπει να χρησιμοποιείτε για αριθμητικά δεδομένα στα προγράμματά σας, ως αρχάριοι, καλό θα είναι να ακολουθήσετε τους παρακάτω απλούς κανόνες.

- Μην χρησιμοποιείτε άσκοπα μεταβλητές τύπου `int`. Χρησιμοποιείτε κατά κόρον μεταβλητές τύπου `double` ή `float`. Ακόμη και εάν τα αριθμητικά δεδομένα σας σε προβλήματα είναι ακέραιοι αριθμοί χρησιμοποιείτε μεταβλητές τύπου `double`.
- Οι ακέραιοι έχουν περιορισμένη ακρίβεια και χρησιμοποιούνται μόνο σε ορισμένες περιπτώσεις όπως στην εντολή `for` και στους πίνακες όπως θα δούμε παρακάτω.
- Πράξεις με ακραίους και ειδικότερα οι διαιρέσεις είναι πολύ πιθανό να σας οδηγήσουν σε λανθασμένα αποτελέσματα. Για παράδειγμα στον παρακάτω κώδικα:

```
int a,b;  
double x;  
a=5;
```

```
b=10;  
x=a/b;
```

Το αποτέλεσμα θα είναι $x=0$ και όχι $x=0.5$. Αυτό συμβαίνει γιατί η πράξη a/b γίνεται ως ακέραια και το αποτέλεσμα χάνει τα δεκαδικά ψηφία.

- Όταν σε μια μεταβλητή `double` ή `float` αντιστοιχούμε ακεραίους αριθμούς (αριθμητικές σταθερές) καλό είναι να βάζουμε στο τέλος τη τελεία (.). Για παράδειγμα ο κώδικας B είναι προτιμότερος του A:

<u>Κώδικας A</u>	<u>Κώδικας B</u>
<code>double x, y;</code>	<code>double x, y;</code>
<code>x=4;</code>	<code>x=4.;</code>
<code>y=9;</code>	<code>y=9.;</code>
<code>z=x*y;</code>	<code>z=x*y;</code>

- Την τελεία πρέπει αν την βάζουμε σε κάθε αριθμητική σταθερά που εμπλέκεται σε παραστάσεις, δηλώνοντας με αυτόν τον τρόπο πως πρόκειται για αριθμητική σταθερά κινητής υποδιαστολής. Για παράδειγμα ο κώδικας:

```
double x, y, z;  
x=2.;  
y=10.;  
z=(1/2)*x*y;
```

Δίνει αποτέλεσμα $z=0$ διότι ο όρος $(1/2)$ αποτελεί διαίρεση ακεραίων και κατά συνέπεια το αποτέλεσμα θα είναι επίσης ακέραιος δηλαδή 0. Ο σωστός κώδικας είναι:

```
double x, y, z;  
x=2.;  
y=10.;  
z=(1./2.)*x*y;
```

Ο οποίος δίνει αποτέλεσμα $z=10$.

2.8 Μετατροπές του τύπου δεδομένων (casting).

Θα πρέπει να είμαστε πολύ προσεκτικοί όταν μετατρέπουμε έναν τύπο δεδομένων σε κάποιον άλλον. Αυτό εν γένει δεν είναι πάντοτε δυνατό. Για παράδειγμα δεν μπορούμε πάντοτε να μετατρέπουμε έναν `double` ή `float` σε `int` γιατί απλά ένας ακέραιος δεν είναι σε θέση να διαχειριστεί πολύ μεγάλους αριθμούς (άνω των 2 δις). Ομοίως δεν μπορούμε πάντοτε να μετατρέπουμε έναν `double` σε `float`. Ακολουθούν βασικά παραδείγματα μετατροπής:

- Στο παρακάτω παράδειγμα ο `a` θα πάρει την τιμή 2.

```
int a;  
double x;  
x=2.7;  
a=(int)x;
```

- Στο παρακάτω παράδειγμα ο `a` θα προσλάβει μια εντελώς λάθος τιμή.

```
int a;
double x;
x=1.234e24;
a=(int)x;
```

- Στα παρακάτω παραδείγματα ο `x` θα προσλάβει την τιμή 25.

<code>int a;</code>	<code>int a;</code>
<code>double x;</code>	<code>float x;</code>
<code>a=25;</code>	<code>a=25;</code>
<code>x=(double)a;</code>	<code>x=(float)a;</code>

Όπως παρατηρούμε στα παραπάνω παραδείγματα η μετατροπή ενός τύπου σε άλλον γίνεται με το όνομα του νέου τύπου σε παρένθεση πριν την μεταβλητή. Αυτό ονομάζεται μετατροπή τύπου (ή casting).

Γενικά η C επιτρέπει μέσα στην ίδια έκφραση την ανάμειξη διαφορετικών τύπων δεδομένων. Αυτό όμως είναι ένα πράγμα που ο αρχάριος προγραμματιστής ΠΡΕΠΕΙ να αποφεύγει. Δεν υπάρχει κάποιος σοβαρός λόγος στα προγράμματά σας να αναμειγνύετε διαφορετικούς αριθμητικούς τύπους δεδομένων. Ξεκινήστε δηλώνοντας όλες τις μεταβλητές σας να είναι του αυτού τύπου (πχ όλες double). Αν παρ' όλα αυτά χρειάζεστε μετατροπές ακολουθήστε τους κανόνες των παραπάνω παραδειγμάτων.

Πρέπει να γνωρίζουμε επίσης πως η C όταν αναμειγνύουμε διαφορετικούς τύπους δεδομένων στην ίδια έκφραση κάνει "αυτόματα" τις μετατροπές ακολουθώντας αυστηρούς κανόνες. Οι μετατροπές μπορούν να γίνουν όπως στο παρακάτω διάγραμμα από δεξιά προς τα αριστερά:

long double ← double ← float ← int ← short int

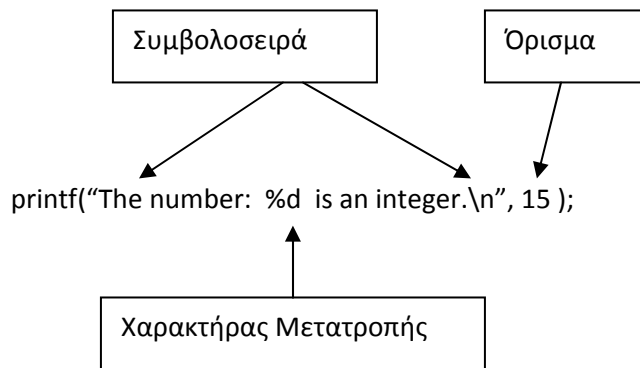
2.9 Φορμαρισμένη έξοδος - η συνάρτηση printf().

Η συνάρτηση **printf()** χρησιμοποιείται για να τυπώνει μηνύματα στην οθόνη του υπολογιστή μας, και εμπεριέχεται στο αρχείο επικεφαλίδας **stdio.h**.

Στο σχήμα 2.3 βλέπουμε ένα απλό παράδειγμα σύνταξης της συνάρτησης printf(). Το συγκεκριμένο παράδειγμα τυπώνει στην οθόνη μας την ακόλουθη συμβολοσειρά:

The number 15 is an integer.

Γενικά η συμβολοσειρά περιέχει χαρακτήρες, καθώς και χαρακτήρες μετατροπής για τα ορίσματα τα οποία αναγράφονται στο τέλος της εντολής. Παρατηρείστε πως οι χαρακτήρες μετατροπής ξεκινούν με το σύμβολο %. Οι βασικοί χαρακτήρες μετατροπής εικονίζονται στον πίνακα 2.3

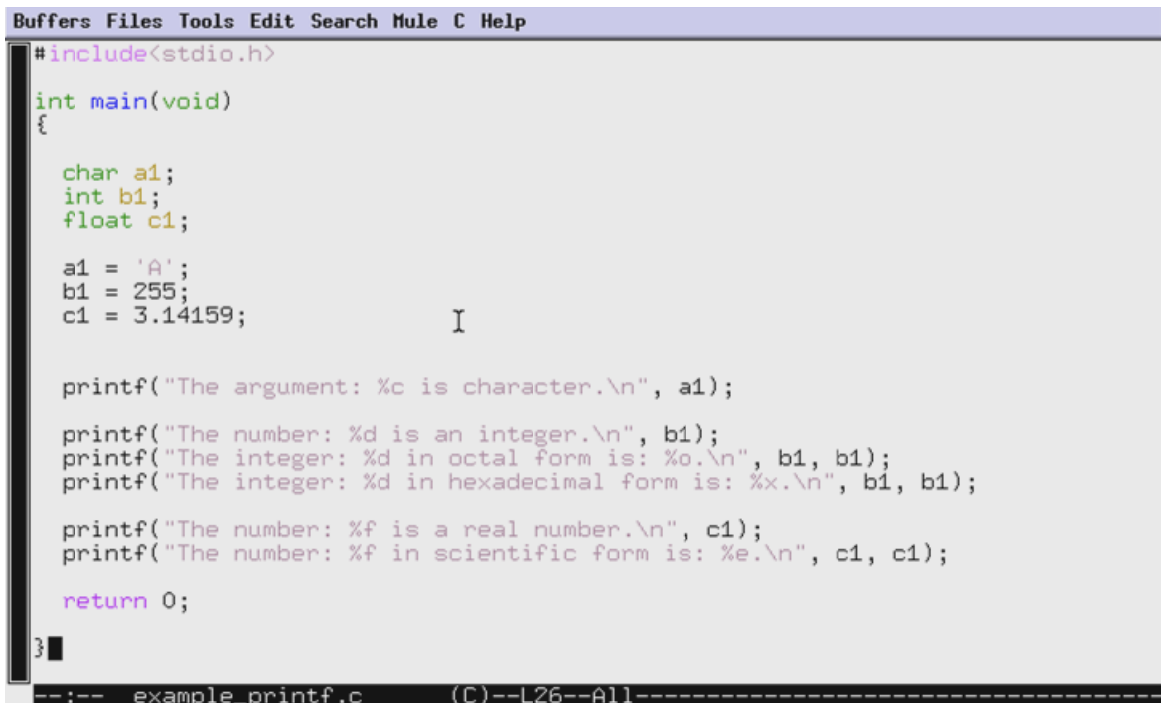


Σχήμα 2.4 Παράδειγμα σύνταξης της συνάρτησης printf().

Πίνακας 2.3 Οι βασικοί χαρακτήρες μετατροπής της συνάρτησης printf().

Χαρακτήρας μετατροπής	Τύπος ορίσματος – Μορφή εκτύπωσης
d,i	Τύπος int.
o	Τύπος int. Εκτύπωση σε Οκταδική μορφή.
x,X	Τύπος int. Εκτύπωση σε Δεκαεξαδική μορφή.
u	Τύπος unsigned int.
c	Τύπος char. Εκτύπωση ενός χαρακτήρα.
s	Τύπος char. Εκτύπωση πίνακα χαρακτήρων.
f	Τύπος float, double.
e,E	Τύπος float, double. Επιστημονική εκτύπωση.
g,G	Τύπος float, double. Επιστημονική ή κοινή εκτύπωση ανάλογα με την ακρίβεια.

Προσοχή ο χαρακτήρας '\n' ο οποίος μπαίνει στο τέλος της συμβολοσειράς είναι ο χαρακτήρας της νέας γραμμής και μετακινεί τον κέρσορα του υπολογιστή στην επόμενη γραμμή. Παράληψη του χαρακτήρα '\n' θα "κολλήσει" την επόμενη εκτύπωση στην προηγούμενη. Σημειώνουμε επίσης πως εάν θέλουμε να εκτυπώσουμε το σύμβολο % αυτό γίνεται ως %%. Από την παραπάνω λίστα παραλείψαμε τον χαρακτήρα μετατροπής r ο οποίος αναφέρεται σε δείκτες και θα τον συναντήσουμε αργότερα. Στο παρακάτω σχήμα 2.5 εικονίζεται ένα απλό πρόγραμμα στο οποίο ορίζονται, αρχικοποιούνται και εκτυπώνονται τρεις απλές μεταβλητές. Όπως φαίνεται σε αυτό χρησιμοποιούνται οι περισσότεροι από τους παραπάνω χαρακτήρες μετατροπής του πίνακα 2.3.

A screenshot of a C program in a text editor. The editor has a menu bar with 'Buffers', 'Files', 'Tools', 'Edit', 'Search', 'Mule', 'C', and 'Help'. The code is as follows:

```
#include<stdio.h>

int main(void)
{
    char a1;
    int b1;
    float c1;

    a1 = 'A';
    b1 = 255;
    c1 = 3.14159;

    printf("The argument: %c is character.\n", a1);

    printf("The number: %d is an integer.\n", b1);
    printf("The integer: %d in octal form is: %o.\n", b1, b1);
    printf("The integer: %d in hexadecimal form is: %x.\n", b1, b1);

    printf("The number: %f is a real number.\n", c1);
    printf("The number: %f in scientific form is: %e.\n", c1, c1);

    return 0;
}
```

The status bar at the bottom shows '--:-- example_printf.c (C)--L26--All-----'.

Σχήμα 2.5 Πρόγραμμα παράδειγμα για την χρήση μερικών από τους χαρακτήρες μετατροπής.

Η εκτέλεση του παραπάνω προγράμματος εμφανίζει στην οθόνη τα ακόλουθα:

The argument: A is a character.

The number: 255 is an integer.

The integer: 255 in octal form is: 377.

The integer: 255 in hexadecimal form is: ff.

The number: 3.141590 is a real number.

The number: 3.141590 in scientific form is: 3.141590e+00

Η συνάρτηση printf() επιτρέπει στον χρήστη τον καθορισμό του **ελάχιστου πλάτους πεδίου** κατά την εκτύπωση ενός ορίσματος. Αυτό γίνεται με την παρεμβολή ενός αριθμού μεταξύ του συμβόλου % και του χαρακτήρα μετατροπής. Για παράδειγμα το **%12f** καθορίζει ότι το αποτέλεσμα ενός πραγματικού αριθμού θα τυπωθεί με τουλάχιστον 12 χαρακτήρες. Ο αριθμός στοιχίζεται στα δεξιά. Εάν θέλουμε αριστερή στοίχιση του αριθμού τότε χρησιμοποιούμε το **%-12f**.

Επίσης η συνάρτηση printf() επιτρέπει τον καθορισμό του αριθμού δεκαδικών ψηφίων. Για παράδειγμα το **%.3f** καθορίζει ότι το αποτέλεσμα ενός πραγματικού αριθμού θα τυπωθεί με τρία δεκαδικά ψηφία, ενώ το **%12.3f** καθορίζει ότι το αποτέλεσμα ενός πραγματικού αριθμού θα τυπωθεί με τουλάχιστον 12 χαρακτήρες εκ των οποίων τρία ψηφία θα είναι δεκαδικά. Με αυτόν τον τρόπο καθορίζουμε την ακρίβεια κατά την εκτύπωση σε ένα πραγματικό αριθμό. Δείτε το επόμενο παράδειγμα και παρατηρήστε με προσοχή τις εκτυπώσεις:


```
double x;
int a;
x=3.14159;
a=35;
printf("x=%f\n",x);      =>   x=3.14159
printf("x=%.3f\n",x);    =>   x=3.141
printf("x=%10.3f\n",x);  =>   x=      3.141
printf("x=%d\n",a);      =>   a=35
printf("a=%10d\n",a);    =>   a=      35
```

Εκτυπώσεις

2.10 Οι βασικές μαθηματικές συναρτήσεις στη C.

Σε αυτή την παράγραφο θα αναφερθούμε στις βασικές μαθηματικές συναρτήσεις που περιλαμβάνει η C. Οι συναρτήσεις αυτές χρησιμοποιούν το αρχείο επικεφαλίδας *math.h* άρα στην κορυφή κάθε προγράμματος ο προγραμματιστής οφείλει να αναγράφει το **#include<math.h>**.

Στους ακόλουθους πίνακες αναγράφονται οι βασικότερες μαθηματικές συναρτήσεις. Σημειώνουμε πως όλες οι συναρτήσεις επιστρέφουν τύπο δεδομένων double. Στους παρακάτω πίνακες οι μεταβλητές x,y είναι τύπου double ενώ η μεταβλητή n είναι τύπου int. Στις τριγωνομετρικές συναρτήσεις οι γωνίες εκφράζονται σε ακτίνια.

Για περισσότερες μαθηματικές συναρτήσεις πρέπει να ανατρέξετε στο βιβλίο σας.

- Τετραγωνική ρίζα, εκθετικές-λογαριθμικές και άλλες βασικές συναρτήσεις.

Συνάρτηση	Περιγραφή
sqrt(x)	Τετραγωνική ρίζα του x, με $x \geq 0$.
exp(x)	Εκθετική συνάρτηση e^x
log(x)	Φυσικός λογάριθμος $\ln(x)$, με $x > 0$
log10(x)	δεκαδικός λογάριθμος $\log_{10}(x)$, με $x > 0$
pow(x,y)	ύψωση σε δύναμη x^y
fabs(x)	Απόλυτη τιμή $ x $
ldexp(x,n)	$x \cdot 2^n$

- Τριγωνομετρικές συναρτήσεις. Προσοχή! τα ορίσματα είναι όλα σε rad.

Συνάρτηση	Περιγραφή
$\cos(x)$	συνημίτονο του x
$\sin(x)$	ημίτονο του x
$\tan(x)$	εφαπτομένη του x
$\arccos(x)$	τόξο συνημιτόνου του x στο διάστημα $[0, \pi]$, $x \in [-1,1]$
$\arcsin(x)$	τόξο ημιτόνου του x στο διάστημα $[-\pi/2, \pi/2]$, $x \in [-1,1]$
$\arctan(x)$	τόξο εφαπτομένης του x στο διάστημα $[-\pi/2, \pi/2]$
$\arctan2(y,x)$	τόξο εφαπτομένης του y/x στο διάστημα $[-\pi, \pi]$
$\cosh(x)$	υπερβολικό συνημίτονο του x
$\sinh(x)$	υπερβολικό ημίτονο του x
$\tanh(x)$	υπερβολική εφαπτομένη του x

Παράδειγμα: Να αναπτύξετε ένα πρόγραμμα το οποίο να υπολογίζει και να εκτυπώνει το συνημίτονο, το ημίτονο και την εφαπτομένη της γωνίας των 60° .

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

int main(void)
{
    double x;                /*Ορισμός μεταβλητής x */
    x=60.;                   /*Αρχικοποίηση μεταβλητής x */
    x=x*3.14159/180.;        /* Μετατροπή από μοίρες σε rad */

    printf("cos(60)=%f\n",cos(x));    /*Εκτύπωση συνημιτόνου */
    printf("sin(60)=%f\n",sin(x));    /*Εκτύπωση ημιτόνου */
    printf("tan(60)=%f\n",tan(x));    /*Εκτύπωση εφαπτομένης */

    return 0;                /* Τερματισμός συνάρτησης main() και προγράμματος */
}
```

Ας υποθέσουμε ότι το παραπάνω πρόγραμμα αποθηκεύεται στο αρχείο `example.c`. Η μεταγλώττιση του αρχείου αυτού γίνεται ως εξής:

```
gcc example.c -lm
```

Το εκτελέσιμο αρχείο θα ονομαστεί `a.out` όπως έχουμε ήδη αναφέρει. **Προσοχή!!!** Κατά την μεταγλώττιση πρέπει να χρησιμοποιήσουμε τον όρο `-lm` (link mathematics) για να υποδείξουμε στον μεταγλωττιστή ότι χρησιμοποιούμε μαθηματικές συναρτήσεις. Εάν εκτελέσουμε το παραπάνω πρόγραμμα θα εμφανιστούν στη οθόνη μας τα εξής:

```
cos(60)=0.500001
```

```
sin(60)=0.866025
```

```
tan(60)=1.732047
```

2.11 Οι βασικοί τελεστές στη C και η ντιρεκτίβα `#define`.

Εκτός από τους αριθμητικούς τελεστές, τους οποίους έχουμε ήδη αναφέρει, στη C συναντάμε και τους ακόλουθους βασικούς τελεστές:

- Οι **συσχετιστικοί τελεστές** είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
>	Μεγαλύτερο
>=	Μεγαλύτερο ή ίσο
<	Μικρότερο
<=	Μικρότερο ή ίσο

- Οι **τελεστές ισότητας** είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
==	Ίσο με
!=	Άνισο με

- Οι **λογικοί τελεστές** είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
&&	Λογικός τελεστής AND (και)
	Λογικός τελεστής OR (ή)
!	Λογικός τελεστής NEGATION (όχι)

Τους συσχετιστικούς τελεστές, τους τελεστές ισότητας και τους λογικούς τελεστές τους συναντάμε κυρίως στις εντολές **if**, **for**, **while**, **do-while**. Οι παραπάνω τελεστές χρησιμοποιούνται για συγκρίσεις μεταξύ αριθμών, μεταβλητών και παραστάσεων. Εάν η σύγκριση είναι **αληθής** τότε το αποτέλεσμα είναι 1 διαφορετικά εάν είναι **ψευδής** τότε το αποτέλεσμα είναι μηδέν. Ας δώσουμε ένα παράδειγμα. Έστω:

```
int a, b, c, d;
```

```
a = 15;
```

```
b = 7;
```

```
c = 25;
```

```
d = 15;
```

τότε η παράσταση `a>10` επιστρέφει τον αριθμό 1

η παράσταση `a>b` επιστρέφει τον αριθμό 1

η παράσταση `a>c` επιστρέφει τον αριθμό 0

η παράσταση `a>=b` επιστρέφει τον αριθμό 1

η παράσταση `a<b` επιστρέφει τον αριθμό 0

η παράσταση `a<c` επιστρέφει τον αριθμό 1

η παράσταση `a<=b` επιστρέφει τον αριθμό 0

η παράσταση `a==b` επιστρέφει τον αριθμό 0

η παράσταση `a==d` επιστρέφει τον αριθμό 1

η παράσταση `a!=b` επιστρέφει τον αριθμό 1

η παράσταση `a!=d` επιστρέφει τον αριθμό 0

επίσης

η παράσταση `(a>b)&&(a<c)` επιστρέφει τον αριθμό 1

η παράσταση `(a>b)&&(a>c)` επιστρέφει τον αριθμό 0

η παράσταση `(a>b)|| (a<c)` επιστρέφει τον αριθμό 1

η παράσταση `(a>b)|| (a>c)` επιστρέφει τον αριθμό 1

η παράσταση `(a>c)|| (d>c)` επιστρέφει τον αριθμό 0

η παράσταση `!(a>b)` επιστρέφει τον αριθμό 0

η παράσταση `!(a==b)` επιστρέφει τον αριθμό 1

- Ο τελεστής αύξησης και ο τελεστής μείωσης εικονίζεται παρακάτω:

Τελεστής	Περιγραφή
++	Τελεστής αύξησης κατά 1
--	Τελεστής μείωσης κατά 1

Οι τελεστές ++ και -- χρησιμοποιούνται όταν θέλουμε να προσθέσουμε ή να αφαιρέσουμε το 1 από μία μεταβλητή. Έτσι

το ++a; ισοδυναμεί με το a=a+1;
 ενώ το --a; ισοδυναμεί στο a=a-1;

Εδώ πρέπει να σημειώσουμε πως οι τελεστές ++ και -- μπορούν να χρησιμοποιηθούν είτε ως **προθεματικοί** τελεστές (δηλ. πριν την μεταβλητή, όπως ++a ή --a) είτε ως **μεταθεματικοί** (δηλ. μετά την μεταβλητή, όπως a++ ή a--). Και στις δύο περιπτώσεις η τιμή του a αυξάνει ή ελαττώνεται κατά 1. Για παράδειγμα στην παράσταση ++a η τιμή του a αυξάνει πριν χρησιμοποιηθεί η τιμή της, ενώ στην παράσταση a++ η τιμή του a αυξάνει αφού χρησιμοποιηθεί η τιμή της. Έτσι έστω ότι το a ισούται με 5 τότε η παράσταση

b = a++;

δίνει στο b την τιμή 5 ενώ η παράσταση

b=++a;

την τιμή 6. Το a και στις δύο περιπτώσεις γίνεται 6.

- Ο τελεστής αντιστοίχισης είναι ο:

Τελεστής	Περιγραφή
=	Τελεστής αντιστοίχισης

Ο τελεστής = αντιστοιχεί την δεξιά τιμή μιας παράστασης στην αριστερή. Έτσι η παράσταση a=b; αντιστοιχεί την τιμή του b στο a.

- οι τελεστές αντικατάστασης είναι οι ακόλουθοι

Τελεστής	Περιγραφή
+=	Τελεστής πρόσθεσης και αντιστοίχισης
-=	Τελεστής αφαίρεσης και αντιστοίχισης

<code>*=</code>	Τελεστής πολ/μου και αντιστοίχησης
<code>/=</code>	Τελεστής διαίρεσης και αντιστοίχησης
<code>%=</code>	Τελεστής υπολοίπου και αντιστοίχησης

Τα παρακάτω παραδείγματα δίνουν την έννοια των παραπάνω τελεστών:

To <code>a += b;</code>	ισοδυναμεί με το	<code>a = a+b;</code>
To <code>a -= b;</code>	ισοδυναμεί με το	<code>a = a-b;</code>
To <code>a *= b;</code>	ισοδυναμεί με το	<code>a = a*b;</code>
To <code>a /= b;</code>	ισοδυναμεί με το	<code>a = a/b;</code>
To <code>a %= b;</code>	ισοδυναμεί με το	<code>a = a %b;</code>

- Οι **τελεστές πράξεων με bits** είναι οι ακόλουθοι :

Τελεστής	Περιγραφή
<code>&</code>	AND για bit
<code> </code>	OR για bit
<code>^</code>	XOR για bit
<code>~</code>	NOT για bit
<code>>></code>	Ολίσθηση αριστερά
<code><<</code>	Ολίσθηση δεξιά

Οι παραπάνω τελεστές αφορούν πράξεις σε επίπεδο bits και για να τους καταλάβουμε πρέπει να διαχειριζόμαστε τους διάφορους αριθμούς σε δυαδική μορφή. Οι τελεστές `&`, `|`, `^` και `~` αντιστοιχούν στις απλές πράξεις της **άλγεβρας Boole**. Οι τελεστές `>>` και `<<` προκαλούν ολίσθηση στα δεξιά και στα αριστερά αντίστοιχα. Έτσι για παράδειγμα εάν η μεταβλητή `a` είναι ο δυαδικός αριθμός 01101000 τότε η παράσταση

`b = a >> 2;`

δίνει στη μεταβλητή `b` την τιμή 00011010.

Εδώ θα κάνουμε μια παρένθεση και θα αναφερθούμε σε σταθερές οι οποίες ονομάζονται **Συμβολικές Σταθερές**. Σε πολλά προγράμματα επιθυμούμε να χρησιμοποιούμε πολλές φορές κάποιες τιμές οι οποίες συνήθως είναι φυσικές σταθερές όπως ο αριθμός $\pi=3.14159$, η επιτάχυνση της

βαρύτητας $g=9.81\text{m/sec}^2$ κτλ. Οι τιμές αυτές μπορούν να εισαχθούν σε κάποιες σταθερές που ονομάζονται συμβολικές σταθερές. Οι συμβολικές σταθερές μπορούν να οριστούν μία φορά και όποτε χρειάζεται μπορούν να χρησιμοποιηθούν σε οποιοδήποτε σημείο του προγράμματος. Οι συμβολικές σταθερές είναι οντότητες που μπορούν να δημιουργηθούν με την χρήση της ντιρεκτίβας **#define**. Η γενική μορφή της ντιρεκτίβας **#define** είναι:

#define όνομα_μακροεντολής ακολουθία_χαρακτήρων

και προκαλεί την αντικατάσταση μιας ακολουθίας χαρακτήρων με μία άλλη. Ακολουθεί ένα απλό πρόγραμμα το οποίο υπολογίζει την περίμετρο και το εμβαδό ενός κύκλου ακτίνας 10cm.

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

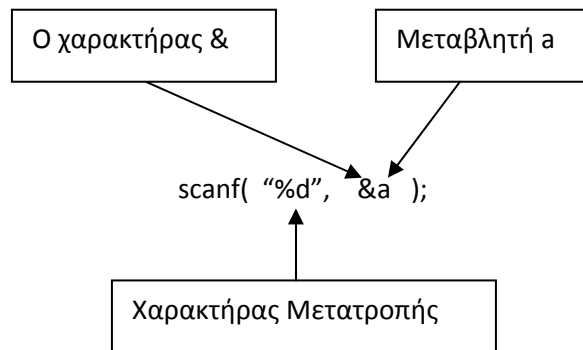
#define pi 3.14159          /* Δημιουργία συμβολικής σταθεράς pi=3.14159 */

int main(void)
{
    double R, S, E;          /* Δήλωση μεταβλητών */
    R=10.;                   /* Αρχικοποίηση ακτίνας */
    S=2.*pi*R;               /* Υπολογισμός περιμέτρου - χρήση του pi */
    E=pi*pow(R,2.);          /* Υπολογισμός εμβαδού - χρήση του pi */
    printf("S=%f cm E=%f cm^2 \n",S,E);    /*Εκτύπωση αποτελεσμάτων */
    return 0;
}
```

2.12 Φορμαρισμένη είσοδος - η συνάρτηση scanf().

Η συνάρτηση **scanf()** χρησιμοποιείται για να διαβάζουμε διάφορους τύπους δεδομένων ή συμβολοσειρές από την τυπική είσοδο (πληκτρολόγιο). Η εκτέλεση της συνάρτησης **scanf()** σταματά την ροή του προγράμματος έως ότου ο χρήστης πληκτρολογήσει τα απαραίτητα δεδομένα. Στη συνέχεια το πρόγραμμα συνεχίζει λαμβάνοντας υπ' όψη και τα δεδομένα που έχουν πληκτρολογηθεί. Η συνάρτηση **scanf()** όπως και η **printf()** εμπεριέχεται στο αρχείο επικεφαλίδας **stdio.h**. Στο σχήμα 2.6 βλέπουμε ένα απλό παράδειγμα σύνταξης της συνάρτησης **scanf()**.

Στο παράδειγμα του σχήματος 2.6 εισάγουμε έναν ακέραιο αριθμό στην μεταβλητή **a**. Οι χαρακτήρες μετατροπής που χρησιμοποιούμε στην συνάρτηση **scanf()** εικονίζονται στον πίνακα 2.4. Εδώ πρέπει να τονίσουμε την χρήση του **Τελεστή Διεύθυνσης &**, ο οποίος είναι απαραίτητος και αναγράφεται πριν το όνομα της μεταβλητής. Ο τελεστής **&** υπολογίζει την διεύθυνση στην μνήμη του υπολογιστή που καταλαμβάνει μια μεταβλητή. Περισσότερα για τον τελεστή **&** θα μάθουμε όταν αναφερθούμε στους δείκτες σε επόμενο κεφάλαιο.



Σχήμα 2.6 Παράδειγμα σύνταξης της συνάρτησης scanf().

Πίνακας 2.4 Οι βασικοί χαρακτήρες μετατροπής της συνάρτησης scanf().

Χαρακτήρας μετατροπής	Τύπος μεταβλητής
d	int.
c	char.
s	Πίνακας χαρακτήρων.
lf	double.
f	float.
c	char.

Στο παρακάτω σχήμα 2.7 εικονίζεται ένα απλό παράδειγμα για την χρήση της συνάρτησης scanf(). Στο παράδειγμα ζητούμε να αναπτύξουμε ένα πρόγραμμα στο οποίο να εισάγουμε από το πληκτρολόγιο δύο αριθμούς int και έναν float, να τους αποθηκεύσουμε σε κατάλληλες μεταβλητές και στη συνέχεια να τους εκτυπώσουμε.

```

Buffers Files Tools Edit Search Mule C Help
#include<stdio.h>
int main(void)
{
    int a,b;
    float c;

    printf("Please insert two integers:\n");
    scanf("%d %d",&a, &b);

    printf("Please insert one real number:\n");
    scanf("%f",&c);

    printf("The integers you have typed are: %d %d\n",a,b);
    printf("The real number you have typed is: %f\n",c);

    return 0;
}
-1:-- example_scanf.c (C)--L19--A11-----
Wrote /home/student1/kef2/example_scanf.c
  
```

Σχήμα 2.7 Απλό παράδειγμα για την χρήση της συνάρτησης scanf().

Η εκτέλεση του παραπάνω προγράμματος είναι η ακόλουθη (με έντονα γράμματα αναγράφονται οι αριθμοί τους οποίους πληκτρολογούμε):

Please insert two integers:

12345 98765

Please insert one real number:

987.654321

The integers you have typed are: 12345 98765

The real number you have typed is: 987.654321

2.13 Οι συναρτήσεις εισόδου-εξόδου χαρακτήρων `getchar()`, `putchar()`.

Ειδικά για τους χαρακτήρες, η C είναι εφοδιασμένη με κατάλληλες συναρτήσεις εισόδου-εξόδου. Η συνάρτηση **`getchar()`** μπορεί να χρησιμοποιηθεί για εισαγωγή χαρακτήρων από το πληκτρολόγιο, ενώ η συνάρτηση **`putchar()`** μπορεί να χρησιμοποιηθεί για εκτύπωση χαρακτήρων στην οθόνη του υπολογιστή. Ακολουθούν παραδείγματα σύνταξης των δύο νέων συναρτήσεων και ο αντίστοιχος ισοδύναμος κώδικας με τις συναρτήσεις `scanf()` και `printf()`.

Κώδικας με την `getchar()`

```
char c;  
c=getchar();
```

Ισοδύναμος κώδικας με την `scanf()`

```
char c;  
scanf("%c",&c);
```

Κώδικας με την `putchar()`

```
char c;  
c='k';  
putchar(c);
```

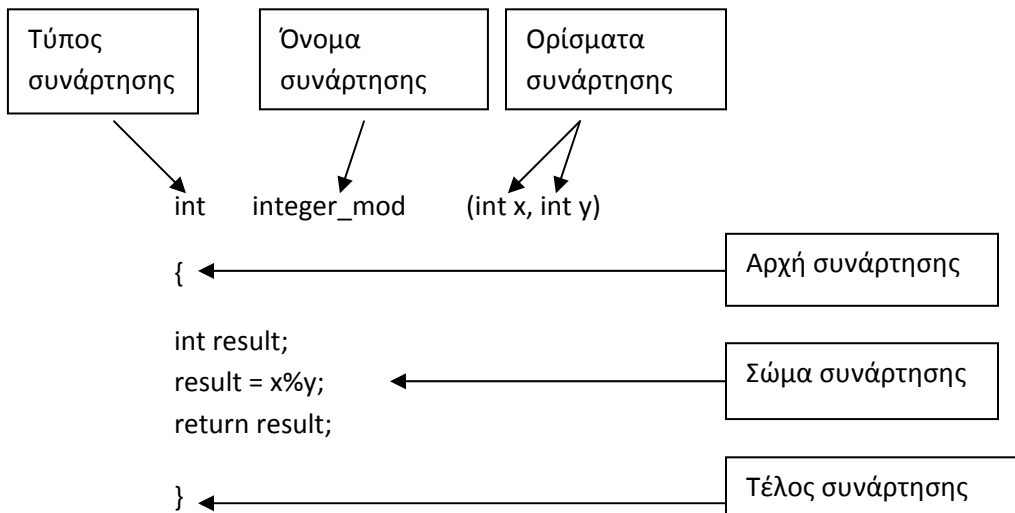
Ισοδύναμος κώδικας με την `printf()`

```
char c;  
c='k';  
printf("%c",&c);
```

Σημειώνουμε εδώ πως στην `putchar()` μπορεί να χρησιμοποιηθεί ως όρισμα ο κωδικός ASCII (πίνακας 2.1) για κάθε χαρακτήρα πχ. η κλήση `putchar(65)` θα τυπώσει τον χαρακτήρα 'Α'.

2.14 Οι συναρτήσεις στη C.

Οι συναρτήσεις στη γλώσσα προγραμματισμού C αποτελούν τα βασικά δομικά τμήματα από τα οποία αποτελούνται τα διάφορα προγράμματα. Ένα πρόγραμμα μπορεί να περιέχει πολλές συναρτήσεις. Μία από αυτές είναι η συνάρτηση `main()`. Η συνάρτηση `main()` είναι μία ειδική συνάρτηση η οποία δεν μπορεί να απουσιάζει από κανένα πρόγραμμα. Όταν ο υπολογιστής εκτελεί ένα πρόγραμμα ψάχνει βρίσκει την συνάρτηση `main()` και αρχίζει να εκτελεί σειριακά μία-μία τις γραμμές της. Το πρόγραμμα σταματά αυτόματα όταν εκτελεστεί και η τελευταία γραμμή της `main()`.

[illegible]

- Ο **τύπος της συνάρτησης** δείχνει τι είδους τιμή επιστρέφει η συνάρτηση μετά την εκτέλεσή της. Εάν η τιμή που επιστρέφει είναι `int` τότε και η συνάρτηση οφείλει να είναι τύπου `int`. Εάν η τιμή που επιστρέφει είναι `double` τότε και η συνάρτηση οφείλει να είναι τύπου `double` κτλ. Εάν μια συνάρτηση δεν επιστρέφει κάποιο αποτέλεσμα τότε οφείλει να είναι τύπου `void`.
- Καλό είναι το **όνομα της συνάρτησης** να δίνεται με τέτοιο τρόπο ώστε να δείχνει τι μπορεί να κάνει η συνάρτηση. Στο όνομα της συνάρτησης μπορούν να χρησιμοποιηθούν όλοι οι χαρακτήρες (Α έως Ζ και α έως z), όλα τα ψηφία (0 έως 9) και ο χαρακτήρας υπογράμμισης (`_`). Σημειώνουμε πως το όνομα της συνάρτησης δεν μπορεί να ξεκινά με ψηφίο ενώ απαγορεύεται η χρήση των αριθμητικών συμβόλων (+, -, *, /), τα εισαγωγικά (""), η απλή απόστροφος ('), η τελεία (.) και μερικοί άλλοι χαρακτήρες όπως τα #, ?, &, @ κτλ.
- Τα **ορίσματα της συνάρτησης** αποτελούν τις μεταβλητές οι οποίες εισάγονται και χρησιμοποιούνται από αυτή. Με αυτόν τον τρόπο η συνάρτηση μπορεί να χρησιμοποιηθεί όσες φορές θέλουμε κάθε φορά με διαφορετικές τιμές στα ορίσματα. Οι μεταβλητές που μπαίνουν ως ορίσματα έχουν εμβέλεια μόνο μέσα στην συγκεκριμένη συνάρτηση και δεν χρειάζεται να ξαναορισθούν μέσα σε αυτή. Εάν μια συνάρτηση δεν έχει ορίσματα τότε χρησιμοποιείται το `void` στη θέση των ορισμάτων. Τέτοιο παράδειγμα αποτελεί το `int main(void)`.
- Μία συνάρτηση **αρχίζει** πάντοτε με αριστερή αγκύλη ({) και τελειώνει με **δεξιά** αγκύλη (}).

- Το **σώμα μιας συνάρτησης** περιέχει πάνω-πάνω όλες τις δηλώσεις των μεταβλητών οι οποίες έχουν εμβέλεια μόνο μέσα στην συγκεκριμένη συνάρτηση. Στη συνέχεια μπορεί να περιέχει τις παραστάσεις, τους υπολογισμούς και τις εντολές οι οποίες απαιτούνται. Τέλος εφ' όσον επιστρέφει κάποιο αποτέλεσμα αυτό γίνεται με την εντολή return.

Στις παρακάτω γραμμές εικονίζεται η ανάλυση της συνάρτησης του σχήματος 2.7.

```
int integer_mod (int x, int y)    /* Ορισμός συνάρτησης τύπου int, με όνομα: integer_mod
                                   και με ορίσματα x και y */
{
    int result;                  /* Δήλωση μεταβλητής με το όνομα result */
    result = x%y;                /* Η μεταβλητή result λαμβάνει την τιμή της πράξης x%y */
    return result;               /* Επιστροφή του αποτελέσματος μέσω της μεταβλητής result */
}
```

Σημειώνουμε εδώ πως μία συνάρτηση καλείται από κάποια άλλη απλά με το όνομά της. Ας δούμε τώρα ένα πλήρες παράδειγμα το οποίο χρησιμοποιεί την παραπάνω συνάρτηση. Ζητούμε να αναπτύξουμε ένα πρόγραμμα στο οποίο να εισάγονται από το πληκτρολόγιο δύο ακέραιοι αριθμοί, να αποθηκεύονται σε δύο κατάλληλες μεταβλητές (τις a και b) στη συνέχεια να καλείται η συνάρτηση και το αποτέλεσμα να επιστρέφεται σε μία άλλη μεταβλητή (την c), η οποία τελικά να εκτυπώνεται. Το πρόγραμμα έχει ως εξής:

```
#include<stdio.h>

int integer_mod (int x, int y);    /* Δήλωση πρωτότυπο της συνάρτησης */

int main(void)
{
    int a,b,c;                    /* Δήλωση μεταβλητών a,b,c */
    printf("Eisagete ta a kai b:");
    scanf("%d %d",&a,&b);        /* Εισαγωγή δεδομένων */
    c=integer_mod(a,b);           /* Κλήση συνάρτησης */
    printf("Υπολοιπο diairesis %d\n",c);
    return 0;
}

int integer_mod (int x, int y)
{
    int result;
    result = x%y;
    return result;
}
```

Προσοχή! εάν θέλουμε να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την δηλώσουμε πάνω-πάνω στο πρόγραμμά μας αμέσως μετά τα #include, όπως φαίνεται στο παραπάνω παράδειγμα. Αυτό ονομάζεται δήλωση πρωτότυπου της συνάρτησης.

Παράδειγμα: Να γραφεί μια συνάρτηση η οποία να υπολογίζει την τιμή της μαθηματικής παράστασης

$$f(x, y, z) = x^3 + y^2 + z$$

Στη συνέχεια να αναπτύξετε ένα πρόγραμμα στο οποίο να εισάγονται από το πληκτρολόγιο δεδομένα, τα οποία να αποθηκεύονται σε τρεις κατάλληλες μεταβλητές (τις x,y και z). Στη συνέχεια να καλείται η συνάρτηση και το αποτέλεσμα να επιστρέφεται σε μία άλλη μεταβλητή (πχ την a), η οποία τελικά να εκτυπώνεται. Το πλήρες πρόγραμμα με τις κατάλληλες επεξηγήσεις έχει ως εξής:

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

double  fx(double x, double y, double z);      /* Δήλωση πρωτότυπο της συνάρτησης */

int main(void)
{
    double x,y,z,a;                            /* Δήλωση μεταβλητών a,y,z,a */
    printf("Dose to x: ");
    scanf("%lf",&x);                            /* Εισαγωγή του x */
    printf("Dose to y: ");
    scanf("%lf",&y);                            /* Εισαγωγή του y */
    printf("Dose to z: ");
    scanf("%lf",&z);                            /* Εισαγωγή του z */
    a=fx(x,y,z);                               /* Κλίση συνάρτησης */
    printf("a=%f\n",a);
    return 0;
}

double  fx(double x, double y, double z);
{
    double e;                                  /* Δήλωση μεταβλητής e */
    e=pow(x,3.)+pow(y,2.)+z ;                 /* Υπολογισμός παράστασης */
    return e;                                /* Επιστροφή του αποτελέσματος */
}
```

Κεφάλαιο III:

Οι εντολές ελέγχου της ροής ενός προγράμματος.

3.1 Εντολές ελέγχου της ροής.

Στο παρόν κεφάλαιο ασχολούμαστε με την σύνταξη των εντολών της C οι οποίες εισάγουν λογική και ελέγχουν την ροή εκτέλεσης σε ένα πρόγραμμα. Οι εντολές με τις οποίες θα ασχοληθούμε είναι οι ακόλουθες:

- **if-else και else-if**
- **switch**
- **for**
- **while**
- **do-while**
- **break και continue**
- **goto και ετικέτες**

3.2 Οι εντολές if-else και else-if.

Η βασική εντολή στη C με την οποία μπορούμε να αναπτύξουμε λογική στα προγράμματά μας είναι η εντολή **if**. Η πιο απλή σύνταξη της εντολής **if** είναι η ακόλουθη:

```
if (παράσταση) {  
    γραμμή_1;  
    γραμμή_2;  
    .  
    .  
}
```

Εάν η **παράσταση** είναι αληθής τότε εκτελούνται όλες οι γραμμές οι οποίες περικλείονται μέσα στις αγκύλες. Εάν η **παράσταση** δεν είναι αληθής τότε όλες οι γραμμές, οι οποίες περικλείονται μέσα στις αγκύλες, αγνοούνται και το πρόγραμμα συνεχίζει παρακάτω.

Εάν ο αριθμός των γραμμών προς εκτέλεση είναι 1 τότε μπορούν να παραληφθούν οι αγκύλες. Η χρησιμοποίηση των αγκυλών πάντως δεν απαγορεύεται ακόμη και εάν έχουμε μία μόνο γραμμή. Η μορφή της εντολής **if** σε αυτή την περίπτωση γίνεται απλά:

```
if (παράσταση)  
    γραμμή_1;
```

Προσοχή! Καλό είναι για ένα αρχάριο προγραμματιστή ακόμη και εάν υπάρχει μία μόνο γραμμή μέσα στην if να χρησιμοποιεί τις αγκύλες.

Η γενική μορφή της εντολής **if-else** είναι η ακόλουθη:

```
if (παράσταση) {  
    γραμμή_1;  
    γραμμή_2;  
    .  
}  
else {  
    γραμμή_A;  
    γραμμή_B;  
    .  
}
```

Εάν η **παράσταση** είναι αληθής τότε εκτελούνται όλες οι γραμμές οι οποίες περικλείονται από τις πρώτες αγκύλες, δηλαδή η γραμμή_1, η γραμμή_2 κτλ. Εάν η **παράσταση** δεν είναι αληθής τότε εκτελούνται όλες οι γραμμές οι οποίες περικλείονται από τις δεύτερες αγκύλες, δηλαδή η γραμμή_A, η γραμμή_B κτλ.

Στην **παράσταση** στην οποία αναφερόμαστε παραπάνω χρησιμοποιούμε τους συσχετικούς τελεστές, τους τελεστές ισότητας και τους λογικούς τελεστές τους οποίους συναντήσαμε στο κεφάλαιο II. Η παράσταση μπορεί να είναι είτε ψευδής (να παίρνει δηλαδή την τιμή μηδέν), είτε αληθής (να λαμβάνει δηλαδή μη μηδενική τιμή). Ακολουθούν μερικά απλά παραδείγματα της εντολής if:

```
double a, b, c;  
a=3.4;  
b=7.8;  
c=9.2;  
  
if(a<b){                /* Έλεγχος εάν a<b. Επειδή είναι αληθές θα εκτελεστεί η γραμμή */  
    printf("a<b\n");    /* Η γραμμή εκτελείται και εκτυπώνεται το a<b */  
}  
  
if(a==c){                /* Έλεγχος εάν a=b. Επειδή είναι ψευδές δεν θα εκτελεστεί η γραμμή */  
    printf("a=b \n");  /* Η γραμμή δεν εκτελείται, δεν εκτυπώνεται κάτι */  
}  
  
If((a<b)&&(a<c)){        /* Έλεγχος εάν a<b και ταυτόχρονα εάν a<c. Είναι αληθές */  
    printf("a<b and a<c \n"); /* Η γραμμή εκτελείται και εκτυπώνεται το a<b and a<c */  
}  
  
If((a<b) || (a>c)){      /* Έλεγχος εάν a<b ή a>c. Είναι αληθές */  
    printf("a<b or a>c \n"); /* Η γραμμή εκτελείται και εκτυπώνεται το a<b or a>c */  
}
```

Η γενική μορφή της εντολής **if-else** συμπεριλαμβανομένης και της **else-if** είναι η ακόλουθη:

```
if (παράσταση_1) {  
    γραμμή_1_1;  
    γραμμή_1_2;  
    .  
}  
else if(παράσταση_2) {  
    γραμμή_2_1;  
    γραμμή_2_2;  
    .  
}  
else if(παράσταση_3) {  
    γραμμή_3_1;  
    γραμμή_3_2;  
    .  
}  
.  
.  
.  
.  
else {  
    γραμμή_A;  
    γραμμή_B;  
    .  
}
```

Εάν η **παράσταση_1** είναι αληθής τότε εκτελούνται οι γραμμές: γραμμή_1_1, γραμμή_1_2 κτλ. Εάν η **παράσταση_2** είναι αληθής τότε εκτελούνται οι γραμμές: γραμμή_2_1, γραμμή_2_2 κτλ. Εάν η **παράσταση_3** είναι αληθής τότε εκτελούνται οι γραμμές: γραμμή_3_1, γραμμή_3_2 κτλ. Εάν καμία από τις: **παράσταση_1, παράσταση_2, παράσταση_3** κτλ. δεν αληθεύουν, τότε εκτελούνται οι γραμμές μετά το **else**, δηλαδή η γραμμή_A, η γραμμή_B κτλ.

Καταλαβαίνουμε πως η τελευταία σύνταξη είναι η γενικότερη και πως οι προηγούμενες αποτελούν υποπεριπτώσεις αυτής. Σημειώνουμε πως μπορούμε να έχουμε μεγάλο αριθμό από εντολές **else if** τη μία μετά από την άλλη.

Ας δούμε ένα απλό παράδειγμα. Να γραφεί ένα πρόγραμμα το οποίο να συγκρίνει μεταξύ τους δύο ακεραίους αριθμούς. Οι δύο ακέραιοι να εισαχθούν στο πρόγραμμα με την χρήση της συνάρτησης **scanf()**. Μια πιθανή λύση με την χρήση της εντολής **if** είναι το πρόγραμμα το οποίο εικονίζεται στο επόμενο σχήμα 3.1

```

#include<stdio.h>

int main(void)
{
    int a,b;

    printf("Dose toys akeraious a kai b:\n");
    scanf("%d %d",&a,&b);

    if(a==b){
        printf("O akeraios a=%4d einai isos me ton akeraio b=%4d\n",a,b);
    }

    if(a>b){
        printf("O akeraios a=%4d einai megalyteros apo ton akeraio b=%4d\n",a,b);
    }

    if(a<b){
        printf("O akeraios a=%4d einai mikroteros apo ton akeraio b=%4d\n",a,b);
    }

    return 0;
}

```

--:-- example_if_1.c (C Abbrev)--L23--All-----

Σχήμα 3.1 Παράδειγμα με την εντολή if.

Μια δεύτερη πιθανή λύση με την χρήση της εντολής if-else και else-if είναι το πρόγραμμα το οποίο εικονίζεται στο επόμενο σχήμα 3.2

```

#include<stdio.h>

int main(void)
{
    int a,b;

    printf("Dose toys akeraious a kai b:\n");
    scanf("%d %d",&a,&b);

    if(a==b)
        printf("O akeraios a=%4d einai isos me ton akeraio b=%4d\n",a,b);
    else if(a>b)
        printf("O akeraios a=%4d einai megalyteros apo ton akeraio b=%4d\n",a,b);
    else
        printf("O akeraios a=%4d einai mikroteros apo ton akeraio b=%4d\n",a,b);

    return 0;
}

```

--:-- example_if_2.c (C Abbrev)--L18--All-----

Σχήμα 3.2 Παράδειγμα με την εντολή if-else και else-if.

Όποια από τις παραπάνω δύο λύσεις του προβλήματος ακολουθήσουμε παίρνουμε το αποτέλεσμα το οποίο εικονίζεται στο ακόλουθο σχήμα 3.3


```

[student1@pc244 kef3]$ gcc example_if_1.c
[student1@pc244 kef3]$ a.out
Dose toys akeraious a kai b:
123 234
0 akeraios a= 123 einai mikroteros apo ton akeraio b= 234
[student1@pc244 kef3]$ a.out
Dose toys akeraious a kai b:
9876 8765
0 akeraios a=9876 einai megalyteros apo ton akeraio b=8765
[student1@pc244 kef3]$ a.out
Dose toys akeraious a kai b:
34 34
0 akeraios a= 34 einai isos me ton akeraio b= 34
[student1@pc244 kef3]$ █

```

Σχήμα 3.3 Εκτέλεση των προγραμμάτων τα οποία εικονίζονται στα σχήματα 3.1 και 3.2.

Τελειώνοντας με την εντολή if πρέπει να αναφέρουμε την ύπαρξη του **Τριαδικού Τελεστή (? :)**. Η γενική του σύνταξη είναι η ακόλουθη:

παράσταση_1 ? παράσταση_2 : παράσταση_3

η οποία αντιστοιχεί σε μια απλή εντολή if-else, όπως ακριβώς στην παρακάτω έκφραση:

```

if (παράσταση_1) {
    παράσταση_2;
}
else {
    παράσταση_3;
}

```

3.3 Η εντολή switch.

Στην προηγούμενη παράγραφο είδαμε πως η εντολή if χρησιμοποιείται όταν πρόκειται να παρθούν συνεχόμενες σχετικές αποφάσεις. Εάν υπάρχουν πολλές αποφάσεις που πρέπει να ληφθούν οι ένθετες εντολές if μπορεί να γίνουν περίπλοκες ο κώδικας μεγάλος και δυσανάγνωστος, πράγμα το οποίο μπορεί να οδηγήσει σε λάθη λογικής. Η C μας παρέχει την εντολή **switch** την οποία συνήθως χρησιμοποιούμε όταν ο αριθμός των αποφάσεων είναι μεγάλος. Επίσης η εντολή switch (η οποία δρα ως διακόπτης επιλογής) ενδείκνυται να χρησιμοποιείται όταν αναπτύσσουμε διάφορα μενού στα προγράμματά μας. Η γενική μορφή της switch είναι η ακόλουθη:

```

switch (παράσταση) {
    case σταθερή-παράσταση_1 :
        γραμμή_1_1;
        γραμμή_1_2;
        .

```

```

        .
        break;
    case σταθερή-παράσταση_2 :
        γραμμή_2_1;
        γραμμή_2_2;
        .
        .
        break;
    case σταθερή-παράσταση_2 :
        γραμμή_2_1;
        γραμμή_2_2;
        .
        .
        break;

    .
    .
    .
    default :
        γραμμή_A;
        γραμμή_B;
        .
        .
        break;
}

```

Πρώτα υπολογίζεται η **παράσταση** και στη συνέχεια εκτελείται η περίπτωση (**case**) της οποίας η **σταθερή-παράσταση** ταυτίζεται με αυτή. Όλες οι παραστάσεις case πρέπει να είναι διαφορετικές. Εάν καμία περίπτωση δεν ικανοποιείται τότε εκτελείται η **default**. Στο τέλος κάθε εντολής case και της default αναγράφεται η εντολή **break** η οποία τερματίζει την εντολή switch και το πρόγραμμα συνεχίζει μετά από τέλος αυτής. Η χρήση της εντολής break είναι **απαραίτητη** γιατί οι case εξυπηρετούν απλώς ως επιγραφές.

Ας δούμε ένα παράδειγμα. Να γραφεί ένα πρόγραμμα το οποίο ως είσοδο να έχει έναν ακέραιο από το 1 έως το 7 και ως έξοδο να τυπώνει την αντίστοιχη ημέρα της εβδομάδας. Μια πιθανή λύση του προβλήματος εικονίζεται στο σχήμα 3.4. Η εκτέλεση και τα αποτελέσματα του προγράμματος του σχήματος 3.4 εικονίζονται στο σχήμα 3.5.

```

#include<stdio.h>

int main(void)
{
    int day;
    printf("Doste thn hmera tis ebdomadas apo to 1 eos to 7\n");
    scanf("%d",&day);

    switch(day){
    case 1:
        printf("H ptoti mera tis ebdomadas einai h Kiriaki.\n");
        break;
    case 2:
        printf("H deyteri mera tis ebdomadas einai h Deytera.\n");
        break;
    case 3:
        printf("H triti mera tis ebdomadas einai h Triti.\n");
        break;
    case 4:
        printf("H tetarti mera tis ebdomadas einai h Tetarti.\n");
        break;
    case 5:
        printf("H pempti mera tis ebdomadas einai h Pempti.\n");
        break;
    case 6:
        printf("H ekti mera tis ebdomadas einai h Paraskeyi.\n");
        break;
    case 7:
        printf("H ebdomi mera tis ebdomadas einai to Sabato.\n");
        break;
    }
    return 0;
}
--:-- example_case.c (C Abbrev)--L33--A11-----

```

Σχήμα 3.4 Παράδειγμα με την εντολή case.

```

[student1@pc244 kef3]$ gcc example_case.c
[student1@pc244 kef3]$ a.out
Doste thn hmera tis ebdomadas apo to 1 eos to 7
2
H deyteri mera tis ebdomadas einai h Deytera.
[student1@pc244 kef3]$ a.out
Doste thn hmera tis ebdomadas apo to 1 eos to 7
5
H pempti mera tis ebdomadas einai h Pempti.
[student1@pc244 kef3]$ a.out
Doste thn hmera tis ebdomadas apo to 1 eos to 7
7
H ebdomi mera tis ebdomadas einai to Sabato.
[student1@pc244 kef3]$ █

```

Σχήμα 3.5 Η εκτέλεση του προγράμματος το οποίο εικονίζεται στο σχήμα 3.4.

3.4 Η εντολή for.

Συνεχίζουμε με τις εντολές που αφορούν την δημιουργία βρόγχων μέσα σε ένα πρόγραμμα. Με την λέξη βρόγχος εννοούμε πως ένα σύνολο γραμμών (που ανήκει στον βρόγχο) μπορεί να εκτελείται επανειλημμένα όσες φορές το επιθυμεί ο προγραμματιστής.

Οι εντολές που σχετίζονται με τους βρόγχους είναι η **for**, η **while** και η **do-while**. Ξεκινούμε με την πλέον δημοφιλέστερη εντολή αυτού του είδους, την εντολή **for**, της οποίας η γενική σύνταξη είναι η ακόλουθη:

```
for (παράσταση_1; παράσταση_2; παράσταση_3){  
    γραμμή_1;  
    γραμμή_2;  
    .  
    .  
}
```

Το κυρίως σώμα της for αποτελείται από ένα σύνολο γραμμών (γραμμή_1, γραμμή_2 κτλ.) οι οποίες εκτελούνται ανάλογα με την λογική που προσδιορίζουν οι παραστάσεις: **παράσταση_1**, **παράσταση_2** και **παράσταση_3**. Ποιο συγκεκριμένα η ακριβής εκτέλεση της παραπάνω εντολής έχει ως ακολούθως:

- Πρώτα εκτελείται η **παράσταση_1**.
- Στη συνέχεια ελέγχεται εάν η **παράσταση_2** είναι αληθής ή ψευδής. Εάν είναι αληθής εκτελούνται οι γραμμές που αποτελούν το κυρίως σώμα της for. Εάν είναι ψευδής τερματίζεται εδώ η εντολή for.
- Στη συνέχεια εκτελείται η **παράσταση_3**.
- Ελέγχεται πάλι εάν η **παράσταση_2** είναι αληθής ή ψευδής. Εάν είναι αληθής εκτελούνται οι γραμμές που αποτελούν το κυρίως σώμα της for. Εάν είναι ψευδής τερματίζεται εδώ η εντολή for.
- Εκτελείται πάλι η **παράσταση_3**.
- Ελέγχεται πάλι εάν η **παράσταση_2** είναι αληθής ή ψευδής. Εάν είναι αληθής εκτελούνται οι γραμμές που αποτελούν το κυρίως σώμα της for. Εάν είναι ψευδής τερματίζεται εδώ η εντολή for.
- Εκτελείται πάλι η **παράσταση_3** και ελέγχεται εάν η **παράσταση_2** είναι αληθής ή ψευδής. Αυτό συνεχίζεται έως ότου κάποια στιγμή η **παράσταση_2** γίνει ψευδής οπότε και τερματίζεται η for.

Με απλά λόγια η εντολή for αποτελεί ένα βρόγχο ό οποίος εκτελείται **n** φορές (όπου **n** ακέραιος αριθμός). Ο αριθμός **n** εξαρτάται από την λογική που προσδιορίζουν οι παραστάσεις: **παράσταση_1**, **παράσταση_2** και **παράσταση_3**.

Παράδειγμα: Ας δούμε ένα απλό παράδειγμα με την εντολή for. Να αναπτύξετε ένα πρόγραμμα το οποίο να παράγει και να τυπώνει τους ακεραίους αριθμούς από το 1 έως το 10.

```
#include<stdio.h>

int main(void){
    int i;                                /* Δήλωση ακεραίας μεταβλητής i */

    for(i=1; i<=10; i++){                /* Εντολή for. Το i λαμβάνει τιμές από 1 έως 10 με βήμα 1 */
        printf("%d \n",i);              /* Εκτύπωση της ακεραίας μεταβλητής i */
    }

    return 0;                             /* Τερματισμός του προγράμματος */
}
```

Στο παραπάνω απλό πρόγραμμα το σώμα της εντολής for (δηλαδή η γραμμή `printf("%d \n",i);`) εκτελείται συνολικά δέκα φορές. Η μεταβλητή `i` αρχικοποιείται στην τιμή 1 και ελέγχει την εκτέλεση της εντολής for. Κάθε φορά αυξάνει κατά ένα. Το for τερματίζεται όταν πλέον δεν ισχύει η παράσταση `i<=10`. Ποιο αναλυτικά η λογική που προσδιορίζει την εντολή for καθορίζεται από τις ακόλουθες τρεις παραστάσεις:

- **παράσταση_1: `i=1`** (Αρχικοποίηση της μεταβλητής `i` στην τιμή 1).
- **παράσταση_2: `i<=10`** (Έλεγχος εάν η τιμή της μεταβλητής `i` είναι μικρότερη ή ίση του 10).
- **παράσταση_3: `i++`** (Η μεταβλητή `i` αυξάνει την τιμή της κατά ένα).

Προσοχή! Η **παράσταση_2** καθορίζει τον τερματισμό των επαναλήψεων. Ο προγραμματιστής θα πρέπει να είναι προσεκτικός σε αυτό το σημείο. Πιθανό λάθος στην **παράσταση_2** μπορεί να οδηγήσει σε ατέρμονα βρόγχο, δηλαδή σε άπειρο αριθμό επαναλήψεων. Αυτό συμβαίνει όταν η **παράσταση_2** είναι πάντοτε αληθής. Σε αυτή την περίπτωση το πρόγραμμα θα εγκλωβιστεί μέσα στον βρόγχο και θα τρέχει συνέχεια. Ο χρήστης σε αυτή την περίπτωση πρέπει να τερματίσει το πρόγραμμα πατώντας στο πληκτρολόγιο **ctrl-c**. Για παράδειγμα το ακόλουθο πρόγραμμα θα τρέχει επ' άπειρον τυπώνοντας ακεραίους αριθμούς ξεκινώντας από το 1:

```
#include<stdio.h>

int main(void){
    int i;                                /* Δήλωση ακεραίας μεταβλητής i */

    for(i=1; i>0; i++){                  /* Ατέρμονας βρόγχος! Η παράσταση i>0 είναι πάντοτε αληθής */
        printf("%d \n",i);              /* Εκτύπωση της ακεραίας μεταβλητής i */
    }

    return 0;
}
```

Σε περίπτωση που κάποιος προγραμματιστής επιθυμεί την δημιουργία ατέρμονα βρόγχου μπορεί να χρησιμοποιήσει την ακόλουθη σύνταξη:

```
for ( ; ; ){
    γραμμή_1;
```

```

        γραμμή_2;
        .
        .
    }

```

Σε αυτή την περίπτωση όμως πρέπει στο σώμα της εντολής for να συμπεριλάβει τις κατάλληλες γραμμές κώδικα οι οποίες κάτω από τις κατάλληλες συνθήκες θα τερματίζουν τον βρόγχο (χρειάζεται για παράδειγμα κάποια εντολή if() συνοδευόμενη από την εντολή break).

3.5 Η εντολή while.

Η εντολή **while** όπως και η for χρησιμοποιείται για την κατασκευή βρόγχων μέσα σε ένα πρόγραμμα. Η σύνταξη της εντολής while είναι η ακόλουθη:

```

while (παράσταση){
    γραμμή_1;
    γραμμή_2;
    .
    .
}

```

Η εκτέλεσή της είναι πολύ απλή. Όσο η **παράσταση** είναι αληθής εκτελείται το σώμα της εντολής (δηλαδή η γραμμή_1, γραμμή_2 κτλ.). Αυτό συνεπάγεται πως ο προγραμματιστής πρέπει με κατάλληλη λογική να ελέγχει τη έξοδο από τον βρόγχο.

Παράδειγμα: Ας δούμε το απλό παράδειγμα που αναπτύξαμε με την εντολή for, πως μπορεί να γραφεί κάνοντας χρήση της while. Να αναπτύξετε ένα πρόγραμμα το οποίο να παράγει και να τυπώνει τους ακεραίους αριθμούς από το 1 έως το 10.

```

#include<stdio.h>

int main(void){
    int i;                                /* Δήλωση ακεραίας μεταβλητής i */
    i=1;                                  /* Αρχικοποίηση της μεταβλητής i στο 1 */
    while(i<=10){                          /* Εντολή while. Έλεγχος εάν i<=10 */
        printf("%d \n",i);                 /* Εκτύπωση της ακεραίας μεταβλητής i */
        i++;                               /* Αύξηση της μεταβλητής i κατά 1 */
    }

    return 0;                              /* Τερματισμός του προγράμματος */
}

```

Όπως παρατηρούμε πάλι χρησιμοποιούμε μια ακέραια μεταβλητή *i* με την τιμή της οποίας ελέγχεται πλήρως ο βρόγχος. Η μεταβλητή *i* αρχικοποιείται στο 1. Ο βρόγχος του `while` εκτελείται εάν $i \leq 10$, ενώ πριν την επόμενη επανάληψη η τιμή της μεταβλητής *i* αυξάνεται κατά 1.

Σημειώνουμε εδώ πως οι εντολές `for` και `while` είναι ισοδύναμες. Αυτό φαίνεται παρακάτω όπου γίνεται σύγκριση των δύο εντολών. Είναι θέμα γούστου ποια από τις δύο εντολές μπορεί να χρησιμοποιήσει κάποιος.

Η εντολή `for`

```
for (παράσταση_1; παράσταση_2; παράσταση_3){  
    γραμμή_1;  
    γραμμή_2;  
    .  
    .  
}
```

Η εντολή `while`

```
παράσταση_1;  
while(παράσταση_2){  
    γραμμή_1;  
    γραμμή_2;  
    .  
    παράσταση_3;  
}
```

3.6 Η εντολή `do-while`.

Η εντολή **`do-while`** είναι η τρίτη εντολή της C με την οποία μπορούμε να δημιουργήσουμε βρόγχους σε ένα πρόγραμμα. Η σύνταξη της εντολής `do-while` είναι η ακόλουθη:

```
do{  
    γραμμή_1;  
    γραμμή_2;  
    .  
    .  
}while(παράσταση);
```

Η εκτέλεσή της είναι πολύ απλή. Το σώμα της εντολής (δηλαδή η γραμμή_1, γραμμή_2 κτλ.), εκτελείται μία φορά και στη συνέχεια ελέγχεται εάν η ***παράσταση*** είναι αληθής ή ψευδής. Εάν η παράσταση είναι αληθής το σώμα της εντολής ξαναεκτελείται έως ότου η παράσταση γίνει ψευδής. Αυτό συνεπάγεται πως ο προγραμματιστής πρέπει με κατάλληλη λογική να ελέγχει τη έξοδο από τον βρόγχο.

Παράδειγμα: Ας δούμε το απλό παράδειγμα που αναπτύξαμε με την εντολή `for` και την εντολή `while` πως μπορεί να γραφεί κάνοντας χρήση της `do-while`. Να αναπτύξετε ένα πρόγραμμα το οποίο να παράγει και να τυπώνει τους ακραίους αριθμούς από το 1 έως το 10.

```
#include<stdio.h>  
  
int main(void){  
    int i;                               /* Δήλωση ακεραίας μεταβλητής i */  
    i=1;                                 /* Αρχικοποίηση της μεταβλητής i στο 1 */
```

```

do{                                     /* Αρχή της εντολής do-while */
    printf("%d \n",i);                 /* Εκτύπωση της ακεραίας μεταβλητής i */
    i++;                              /* Αύξηση της μεταβλητής i κατά 1 */
}while(i<=10);                         /* Τέλος της εντολής do-while. Έλεγχος εάν i<=10 */

return 0;                             /* Τερματισμός του προγράμματος */
}

```

Όπως παρατηρούμε πάλι χρησιμοποιούμε μια ακέραια μεταβλητή *i* με την τιμή της οποίας ελέγχεται πλήρως ο βρόγχος. Η μεταβλητή *i* αρχικοποιείται στο 1. Ο βρόγχος του `while` εκτελείται πάντοτε μία φορά και στο τέλος ελέγχεται εάν $i \leq 10$. Η τιμή της μεταβλητής *i* αυξάνεται κατά 1 πριν ακριβώς από τον έλεγχο.

Σημειώνουμε εδώ πως η εντολή `do-while` δεν είναι ακριβώς ισοδύναμη με τις εντολές `for` και `while` διότι το σώμα της εντολής εκτελείται πάντοτε μία τουλάχιστον φορά. Αυτό συμβαίνει γιατί ο έλεγχος αληθείας της *παράστασης* γίνεται στο τέλος της εντολής και όχι στην αρχή όπως συμβαίνει στις εντολές `for` και `while`. Η εντολή `do-while` γενικά χρησιμοποιείται πολύ λιγότερο από τις εντολές `for` και `while`.

3.7 Οι εντολές `break` και `continue`.

Η εντολή **`break`** χρησιμοποιείται όταν θέλουμε να σταματήσουμε την εκτέλεση ενός βρόγχου και να βγούμε από αυτόν. Δείτε το ακόλουθο παράδειγμα:

```

int i;                                /* Δήλωση ακεραίας μεταβλητής i */
for(i=1; i<=10; ++i){                /* Εντολή for. Το i ξεκινά από 1 έως 10 με βήματα του 1 */
    if(i==6){                          /* Μόλις το i πάρει την τιμή εκτελείται το break */
        break;
    }
    printf("%d \n",i);                /* Εκτύπωση του i */
}

```

Στο παραπάνω παράδειγμα ο βρόγχος έχει προγραμματιστεί να τρέξει δέκα φορές (το *i* ξεκινά από 1 έως 10 με βήματα του 1). Κάθε φορά εκτυπώνεται η τιμή της μεταβλητής *i*. Μόλις όμως το *i* γίνει ίσο με το 6 εκτελείται το `break` το οποίο βρίσκεται μέσα στην `if()`. Αυτό έχει ως αποτέλεσμα τον τερματισμό της εκτέλεσης του βρόγχου και την έξοδο από αυτόν. Ο παραπάνω κώδικας λοιπόν θα εκτυπώσει τις τιμές από το 1 έως και το 5.

Η εντολή **`continue`** χρησιμοποιείται όταν θέλουμε να επιστρέψουμε στην κορυφή του βρόγχου αγνοώντας τις υπόλοιπες γραμμές του σώματός του. Το παρακάτω παράδειγμα κάνει κατανοητή την χρήση του `continue`:


```

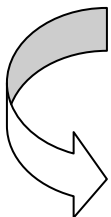
int i;                /* Δήλωση ακέραιας μεταβλητής i */
for(i=1; i<=10; ++i){ /* Εντολή for. Το i ξεκινά από 1 έως 10 με βήματα του 1 */
    if((i==4) || (i==7)){ /* Μόλις το i πάρει τις τιμές 4 ή 7 εκτελείται το continue */
        continue;
    }
    printf("%d \n",i); /*Εκτύπωση του i */
}

```

Στο παραπάνω παράδειγμα ο βρόγχος έχει προγραμματιστεί να τρέξει δέκα φορές (το i ξεκινά από 1 έως 10 με βήματα του 1). Κάθε φορά εκτυπώνεται η τιμή της μεταβλητής i. Μόλις όμως το i γίνει ίσο με το 4 ή το 7 εκτελείται το continue το οποίο βρίσκεται μέσα στην if(). Αυτό έχει ως αποτέλεσμα την επιστροφή κάθε φορά στην κορυφή του βρόγχου. Ο παραπάνω κώδικας λοιπόν θα εκτυπώσει στη σειρά τις ακόλουθες τιμές: 1 2 3 5 6 8 9 και 10. Δεν θα εκτυπωθούν δηλαδή οι τιμές 4 και 7.

3.8 Η εντολή goto και οι ετικέτες.

Η εντολή **goto** αποτελεί κατάλοιπο από άλλες γλώσσες προγραμματισμού (όπως η Fortran) και σήμερα αποφεύγουμε να την χρησιμοποιούμε. Ο λόγος είναι απλός. Η goto επιτρέπει την μετάβαση σε όποιο σημείο του κώδικα επιθυμούμε. Αυτός ο τρόπος προγραμματισμού σήμερα δεν είναι αποδεκτός μιας και μπορεί πολύ εύκολα να χαθεί η λογική μέσα στο πρόγραμμά μας. Παραθέτουμε εδώ την εντολή για λόγους πληρότητας. Η σύνταξή της είναι η ακόλουθη:



```

γραμμή_προγράμματος;
γραμμή_προγράμματος;
γραμμή_προγράμματος;
goto sinexeia_edw;    /* Μετάβαση στην ετικέτα sinexeia_edw */
γραμμή_προγράμματος;
γραμμή_προγράμματος;
γραμμή_προγράμματος;
sinexeia_edw:        /* Ετικέτα με όνομα: sinexeia_edw */
γραμμή_προγράμματος;
γραμμή_προγράμματος;

```

Κεφάλαιο IV:

Δείκτες και πίνακες.

4.1 Δείκτες.

Η C, όπως έχουμε αναφέρει, είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου η οποία αναπτύχθηκε για πρώτη φορά το 1972 από τον Dennis Ritchie στα AT&T Bell Labs. Στόχος ήταν η δημιουργία μιας γλώσσας μέσω της οποίας θα υπήρχε η δυνατότητα ανάπτυξης λειτουργικών συστημάτων (όπως το λειτουργικό σύστημα Unix). Μία τέτοια γλώσσα θα έπρεπε να δίνει την δυνατότητα στον προγραμματιστή με εύκολο τρόπο να επικοινωνεί κατ'ευθείαν με την μνήμη του υπολογιστή και έμμεσα με το υλισμικό (hardware) του υπολογιστή. Για τον λόγο αυτό δημιουργήθηκαν οι **δείκτες**.

Οι δείκτες θα μπορούσαμε να πούμε πως αποτελούν την μεγάλη “δύναμη” της γλώσσας στον προγραμματισμό. Ως έννοιες για κάποιον αρχάριο είναι μάλλον δυσνόητες. Ένας καλός όμως προγραμματιστής στη C σήμερα οφείλει να ξέρει να χρησιμοποιεί τους δείκτες με ευχέρεια. Στα πλαίσια του εισαγωγικού μαθήματος μας, σε αυτή την παράγραφο θα κάνουμε μια απλή εισαγωγή στις έννοιες των δεικτών. Ο κάθε φοιτητής για περισσότερες λεπτομέρειες πρέπει να ανατρέξει στο διδακτικό σύγγραμμα.

Ας ξεκινήσουμε με τις ακόλουθες δύο γραμμές προγράμματος όπου ορίζουμε την μεταβλητή x που είναι τύπου float, την οποία και αρχικοποιούμε στην τιμή 3.1415.

```
float x;          /* Δήλωση μεταβλητής x τύπου float */  
x=3.1415;        /* Αρχικοποίηση της μεταβλητής x */
```

Μέχρι τώρα την παραπάνω διαδικασία την έχουμε κάνει αυτόματα πολλές φορές. Τι όμως ακριβώς συμβαίνει στον υπολογιστή μας κατά την εκτέλεση των δύο αυτών γραμμών;

Τα πάντα εξελίσσονται στην μνήμη (RAM) του υπολογιστή μας. Μπορούμε να φανταστούμε την μνήμη σαν μια τεράστια ταινία γεμάτη κυψελίδες όπως φαίνεται στην παρακάτω εικόνα. Σε κάθε κυψελίδα (μεγέθους στο συγκεκριμένο παράδειγμα τεσσάρων bytes) μπορούν να αποθηκευτούν τα δεδομένα των μεταβλητών που χρησιμοποιούμε στα προγράμματά μας. Κάθε κυψελίδα όμως έχει και μία συγκεκριμένη διεύθυνση, έτσι ώστε ο υπολογιστής να γνωρίζει που να ανατρέξει κάθε φορά που το πρόγραμμα ζητά τα συγκεκριμένα δεδομένα. Η διεύθυνση κάθε κυψελίδας είναι συνήθως ένας μεγάλος αριθμός τον οποίο, στην γλώσσα των υπολογιστών, τον αναγράφουμε σε δεκαεξαδικό σύστημα. Στην παρακάτω εικόνα εικονίζονται μια σειρά κυψελίδων με τις αντίστοιχες διευθύνσεις τους.

Διευθύνσεις Μνήμης	Περιεχόμενο Μνήμης RAM
bfffdfdc	
bfffdfe0	3.1415
bfffdfe4	
bfffdfe8	
.....	

x →

Όταν λοιπόν εκτελείται η γραμμή

```
float x;          /* Δήλωση μεταβλητής x τύπου float */
```

ο υπολογιστής ανατρέχει στην πρώτη κυψελίδα της μνήμης του η οποία είναι διαθέσιμη και δεσμεύει τον κατάλληλο χώρο για την μεταβλητή x. Στο συγκεκριμένο παράδειγμα δεσμεύει την κυψελίδα με διεύθυνση bfffdfe0. Όταν εκτελείται η δεύτερη γραμμή

```
x=3.1415;        /* Αρχικοποίηση της μεταβλητής x */
```

ο υπολογιστής πηγαίνει στην κυψελίδα με την διεύθυνση bfffdfe0 και αποθηκεύει τον αριθμό 3.1415. Με αυτόν τον τρόπο ο υπολογιστής σε κάθε μελλοντική αναφορά στην μεταβλητή x γνωρίζει που να ανατρέξει για να βρει τα δεδομένα.

Την διεύθυνση στη μνήμη που καταλαμβάνει κάθε μεταβλητή στο πρόγραμμά μας μπορούμε πολύ εύκολα να βρούμε κάνοντας χρήση του **Τελεστή Διεύθυνσης &**. Για την διεύθυνση παραδείγματος χάριν της μεταβλητής x αρκεί να γράψουμε &x. Στις επόμενες δύο γραμμές χρησιμοποιώντας την συνάρτηση printf() εκτυπώνουμε τόσο το περιεχόμενο της μεταβλητής x όσο και την διεύθυνσή της στην μνήμη του υπολογιστή μας, για το παραπάνω παράδειγμα.

Αποτέλεσμα εκτύπωσης

```
printf("%f \n",x);    =>    3.1415
printf("%p \n",&x);   =>    bfffdfe0
```

Παρατηρείστε πως για την εκτύπωση διευθύνσεων στην printf() χρησιμοποιούμε τον χαρακτήρα μετατροπής %p.

Ας δούμε τώρα την έννοια του **δείκτη**. Θεωρείστε τις ακόλουθες γραμμές προγράμματος :

```
float x;          /* Δήλωση μεταβλητής x τύπου float */
float *point_x;   /* Δήλωση δείκτη *point_x τύπου float */
x=3.1415;         /* Αρχικοποίηση της μεταβλητής x */
point_x=&x;       /* Η μεταβλητή point_x λαμβάνει τιμή αυτήν της διεύθυνσης του x */
```

Στην πρώτη γραμμή δηλώνουμε μια μεταβλητή `x` τύπου `float`. Στη δεύτερη γραμμή ορίζουμε έναν δείκτη με όνομα `point_x` χρησιμοποιώντας τον **Τελεστή Έμμεσης Αναφοράς** `*`. Μην συγχέετε το σύμβολο του πολλαπλασιασμού, το οποίο μπαίνει ανάμεσα από δύο μεταβλητές (πχ. `x*y`), με τον τελεστή έμμεσης αναφοράς, ο οποίος μπαίνει μπροστά από μία μεταβλητή όταν αυτή ορίζεται για να δηλώσει πως είναι δείκτης. Ο δείκτης `*point_x` οφείλει να είναι του ίδιου τύπου με την μεταβλητή `x`. Στην τρίτη γραμμή αρχικοποιούμε την μεταβλητή `x` στην τιμή 3.1415. Τέλος στην τέταρτη γραμμή αντιστοιχούμε στην μεταβλητή `point_x` την τιμή της διεύθυνσης του `x`.

Στο επόμενο σχήμα εικονίζεται το διάγραμμα στην μνήμη του υπολογιστή μας μετά την εκτέλεση των παραπάνω τεσσάρων γραμμών.



Στις επόμενες γραμμές χρησιμοποιώντας την συνάρτηση `printf()` εκτυπώνουμε το περιεχόμενο της μεταβλητής `x`, την διεύθυνσή του `x` στην μνήμη του υπολογιστή μας, την τιμή της μεταβλητής `point_x`, την διεύθυνσή της `point_x` στην μνήμη του υπολογιστή μας και τέλος την τιμή του δείκτη `*point_x`.

Αποτέλεσμα εκτύπωσης

```
printf("%f \n",x);           =>      3.1415
printf("%p \n",&x);          =>      bffffdfe0
printf("%p \n",point_x);     =>      bffffdfe0
printf("%p \n",&point_x);    =>      bffffdfe4
printf("%f \n",*point_x);    =>      3.1415
```

Ενδιαφέρον μεγάλο παρουσιάζει η τελευταία γραμμή από την οποία συμπεραίνουμε πως ο δείκτης `*point_x` δείχνει το περιεχόμενο της κυψελίδας της μεταβλητής `x`. Με αυτόν τον τρόπο είτε αναφερόμαστε στην μεταβλητή `x` είτε στον δείκτη `*point_x` είναι το ίδιο πράγμα. Για παράδειγμα γράφοντας:

```
*point_x=6.2;
```

αλλάζουμε στην ουσία την τιμή της μεταβλητής x και την κάνουμε 6.2.

Τους δείκτες μπορούμε να τους χρησιμοποιούμε σε πολλές περιπτώσεις όπως να περνάμε με αναφορά (by reference) μεταβλητές σε συναρτήσεις, να επικοινωνούμε έμμεσα με το υλισμικό (hardware) του υπολογιστή κτλ. Αυτά τα πράγματα όμως ξεφεύγουν από ένα εισαγωγικό μάθημα προγραμματισμού και οι φοιτητές που ενδιαφέρονται πρέπει να ανατρέξουν στις αντίστοιχες παραγράφους του διδακτικού βιβλίου και στο internet.

4.2 Μονοδιάστατοι πίνακες.

Η ομαδοποίηση της πληροφορίας στον προγραμματισμό είναι πολύ σημαντική και επιβάλλεται έτσι ώστε να είναι εύκολη η επεξεργασία της. Για παράδειγμα εάν έχουμε δέκα διαφορετικά αριθμητικά δεδομένα του ίδιου τύπου, μπορούμε να ορίσουμε δέκα διαφορετικές μεταβλητές και να τα επεξεργαστούμε. Ο τρόπος αυτός όμως δεν είναι ο ενδεδειγμένος. Σε τέτοιες περιπτώσεις χρησιμοποιούμε πίνακες δεδομένων. Στην παράγραφο αυτή θα ασχοληθούμε με μονοδιάστατους πίνακες.

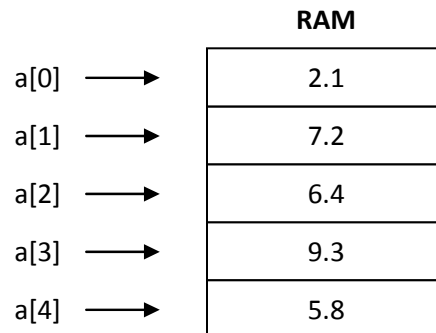
Για παράδειγμα ας υποθέσουμε ότι έχουμε πέντε αριθμούς τύπου float τους 2.1, 7.2, 6.4, 9.3, και 5.8. Ας ορίσουμε τώρα έναν πίνακα ο οποίος να αποτελείται από πέντε στοιχεία και ας τον αρχικοποιήσουμε στις παραπάνω τιμές. Αυτό γίνεται ως ακολούθως:

```
float a[5];      /* Δήλωση του πίνακα a με πέντε στοιχεία */
a[0]=2.1;        /* Αρχικοποίηση των πέντε στοιχείων */
a[1]=7.2;
a[2]=6.4;
a[3]=9.3;
a[4]=5.8;
```

Εδώ πρέπει να σημειώσουμε και να προσέξουμε τα ακόλουθα:

- Κάθε πίνακας έχει ένα όνομα όπως ακριβώς και μία απλή μεταβλητή. Ισχύουν οι ίδιοι κανόνες ονοματολογίας με τις απλές μεταβλητές.
- Όλα τα στοιχεία του πίνακα είναι του ίδιου τύπου.
- Το όνομα του πίνακα ακολουθεί ο αριθμός των στοιχείων του μέσα σε [].
- Η αρίθμηση των στοιχείων του πίνακα ξεκινά από το μηδέν και σταματά στο $n-1$, όπου n η διάσταση του πίνακα. Στο παραπάνω παράδειγμα το πρώτο στοιχείο του είναι το $a[0]$ και το τελευταίο το $a[4]$.
- Προσοχή! Δεν υπάρχει το στοιχείο $a[5]$ και κάθε αναφορά σε αυτό προκαλεί σφάλμα στην εκτέλεση του προγράμματος με μήνυμα *segmentation fault*. Το ίδιο σφάλμα παίρνουμε εάν ξεφύγουμε από τα όρια του πίνακα, πχ να καλέσουμε το στοιχείο $a[7]$ το οποίο δεν υπάρχει για τον παραπάνω πίνακα.

- Τα στοιχεία κάθε πίνακα κατατάσσονται στην μνήμη του υπολογιστή στην σειρά. Για παράδειγμα για τον παραπάνω πίνακα η κατάταξη των στοιχείων του στην μνήμη του υπολογιστή φαίνεται στο παρακάτω σχήμα:



Σημειώνουμε πως στο παραπάνω σχήμα κάθε κυψελίδα μνήμης αποτελείται από τέσσερα bytes.

Η αρχικοποίηση ενός πίνακα μπορεί να γίνει και με τους ακόλουθους δύο τρόπους:

```
float a[5]={ 2.1, 7.2, 6.4, 9.3, 5.8.};
float a[]={ 2.1, 7.2, 6.4, 9.3, 5.8.};
```

Προσέξτε πως στον τελευταίο τρόπο παραλείψαμε την διάσταση του πίνακα. Όταν αρχικοποιούμε έναν μονοδιάστατο πίνακα με τον παραπάνω τρόπο έχουμε το δικαίωμα να μην δηλώσουμε την διάσταση του. Ο υπολογιστής προσμετρά τα δεδομένα μέσα στις αγκύλες και την καθορίζει αυτόματα.

Στις παρακάτω γραμμές κώδικα παραθέτουμε ένα απλό παράδειγμα δημιουργίας ενός πίνακα τύπου double με δέκα στοιχεία. Ο πίνακας γεμίζει με τους αριθμούς 0., 1., 2.,9. Με αυτό το παράδειγμα δείχνουμε πως μπορούμε με την εντολή for να διαχειριστούμε εύκολα έναν πίνακα.

```
int i;                /* Δήλωση μεταβλητής i */
double x[10];         /* Δήλωση πίνακα x δέκα στοιχείων */

for(i=0; i<10; i++){
    x[i]=(double)i;    /* Αρχικοποίηση των στοιχείων με τους αριθμούς 0., 1., ....9. */
}
```

Η πλήρης εκτύπωση ενός πίνακα μπορεί να γίνει με την χρήση μιας εντολής for και της συνάρτησης printf(). Ακολουθεί ένα παράδειγμα εκτύπωσης ενός πίνακα:

```
int i;                /* Δήλωση μεταβλητής i */
double x[7]={1.3, 5.6, 8.1, 9.5, 6.3, 6.5, 5.2 }; /* Δήλωση-αρχικοποίηση πίνακα x */

for(i=0; i<7; i++){
    printf("%f \n",x[i]); /* Εκτύπωση των 7 στοιχείων του πίνακα */
}
```

Ας δούμε τώρα, με ένα παράδειγμα, τον τρόπο με τον οποίο μπορούμε να εισάγουμε από το πληκτρολόγιο ένα-ένα τα στοιχεία ενός πίνακα. Για παράδειγμα θα θεωρήσουμε ένα πίνακα `double a[10]`. Ακολουθούν οι γραμμές προγράμματος για την εισαγωγή των δεδομένων από το πληκτρολόγιο

```
int i;           /* Δήλωση μεταβλητής i */
double a[10];    /* Δήλωση πίνακα a δέκα στοιχείων */

for(i=0; i<10; i++){
    printf("Eisagete to stoixeiο a[%d]: ", i);    /* Μήνυμα για τον χρήστη */
    scanf("%lf", &a[i]);                        /* Εισαγωγή των δεδομένων */
}
```

Όπως παρατηρούμε για να διαχειριστούμε έναν μονοδιάστατο πίνακα χρειαζόμαστε μία εντολή `for`. Κάθε φορά εισάγουμε τα δεδομένα σε ένα στοιχείο. Πριν την εισαγωγή των δεδομένων, με την συνάρτηση `scanf()`, στο κατάλληλο στοιχείο του πίνακα καλό είναι να τυπώνουμε ένα μήνυμα στον χρήστη του προγράμματος για το σε ποιο ακριβώς στοιχείο βρίσκεται κάθε φορά.

4.3 Πίνακες χαρακτήρων και συμβολοσειρές.

Ένας πίνακας των οποίων τα στοιχεία είναι χαρακτήρες ονομάζεται **πίνακας χαρακτήρων**. Για παράδειγμα ο πίνακας

```
char a[5]={'H', 'e', 'l', 'l', 'o'};
```

είναι ένας πίνακας με πέντε στοιχεία τα οποία σχηματίζουν την αγγλική λέξη Hello.

Γενικά η εισαγωγή κειμένου και η επεξεργασία του στα προγράμματα είναι πολύ σημαντική γιατί τον λόγο στη C υπάρχουν οι **συμβολοσειρές** (strings). Ένα παράδειγμα συμβολοσειράς είναι το ακόλουθο:

```
char b[6]={'H', 'e', 'l', 'l', 'o', '\0'};
```

Η παραπάνω συμβολοσειρά `b` είναι ένας πίνακας χαρακτήρων με 6 στοιχεία και περιέχει την λέξη Hello. Παρατηρήστε πως έχει ένα στοιχείο περισσότερο από όσα χρειάζονται, το τελευταίο, το οποίο περιέχει τον κενό χαρακτήρα `\0`. Ο κενός χαρακτήρας είναι απαραίτητος και δηλώνει το τέλος της συμβολοσειράς. Η παραπάνω συμβολοσειρά μπορεί να αρχικοποιηθεί και με τους ακόλουθους δύο τρόπους:

```
char b[6]="Hello";
char b[]="Hello";
```

Στην τελευταία περίπτωση η συμβολοσειρά αποκτά αυτόματα την διάστασή της. Άλλα παραδείγματα συμβολοσειρών αποτελούν τα ακόλουθα:

```
char s[]="University of Ioannina";
char name[]="George Adams";
```

Για να εκτυπώσουμε συμβολοσειρές κάνουμε χρήση της συνάρτησης `printf()` και του χαρακτήρα μετατροπής `s`. Για παράδειγμα:

```
char a[]="University of Ioannina";
printf("%s\n", a);           /* Εκτυπώνεται η φράση University of Ioannina */
```

Για να εισάγουμε δεδομένα από το πληκτρολόγιο σε μία συμβολοσειρά χρησιμοποιούμε τη συνάρτηση scanf(), όπως στο επόμενο παράδειγμα:

```
char b[20];                 /* Δήλωση συμβολοσειράς */
scanf("%s",&b);             /* Εισαγωγή δεδομένων στην συμβολοσειρά b */
```

Η C είναι εφοδιασμένη με έτοιμες συναρτήσεις οι οποίες σχετίζονται με τις συμβολοσειρές. Για να έχει την δυνατότητα ο προγραμματιστής να τις χρησιμοποιήσει πρέπει να συμπεριλάβει στο πρόγραμμά του το αρχείο επικεφαλίδας string.h (δηλαδή #include<string.h>). Έστω οι a και b κατάλληλες συμβολοσειρές. Μερικές βασικές συναρτήσεις που σχετίζονται με συμβολοσειρές είναι οι ακόλουθες:

- strlen(a) : Επιστρέφει το μήκος της συμβολοσειράς a.
- strcmp(a,b) : Συγκρίνει τα περιεχόμενα της συμβολοσειράς a με αυτά της b. Εάν αυτά είναι ίδια επιστρέφει μηδέν. Επιστρέφει τιμή >0 ή <0 ανάλογα με την αλφαβητική τους κατάταξη. Η συνάρτηση αυτή είναι πολύ χρήσιμη όταν ψάχνουμε μια λέξη μέσα σε ένα κείμενο ή μέσα σε μια βάση δεδομένων.
- strcpy(a,"hello") : Κοπiάρει την λέξη πχ hello μέσα στην συμβολοσειρά a.
- strcat(a," there") : Προσθέτει στο τέλος της συμβολοσειράς την λέξη πχ there. Οι δύο τελευταίες συναρτήσεις έχουν ως αποτέλεσμα η συμβολοσειρά a να γίνει η φράση hello there.

Για περισσότερες συναρτήσεις σχετικές με συμβολοσειρές ο αναγνώστης πρέπει να ανατρέξει στο διδακτικό βιβλίο ή το internet.

4.4 Χρήση πίνακα ως όρισμα σε συνάρτηση.

Ας δούμε με ένα απλό παράδειγμα πως μπορούμε να περάσουμε έναν ολόκληρο πίνακα ως όρισμα σε μία συνάρτηση. Υποθέτουμε πως έχουμε ένα πίνακα double x[5] και θέλουμε να γράψουμε μια συνάρτηση η οποία να υπολογίζει το άθροισμα των στοιχείων της. Το όλο πρόγραμμα έχει ως ακολούθως:

```
#include<stdio.h>

double sum(double a[]);      /* Δήλωση πρωτότυπο της συνάρτησης */

int main(void){
    double x[5]={5.2, 7.1, 6.3, 9.2, 4.7}; /* Δήλωση-αρχικοποίηση του πίνακα x[5] */
    double y;                 /* Δήλωση μιας μεταβλητής y */
    y=sum(x);                 /* Κλήση της συνάρτησης */
    printf("Athroisma stoixeion = %f\n",y); /* Εκτύπωση Αποτελέσματος */
    return 0;                 /* Τερματισμός του προγράμματος */
}
```



```

double sum(double a[]){                               /* Αρχή της συνάρτησης sum() */
    double s;                                         /* Δήλωση μιας μεταβλητής s για τον υπολογισμό του αθροίσματος */
    s=0.;                                             /* Αρχικοποίηση της s στο μηδέν */
    for(i=0; i<5; i++){
        s=s+a[i];                                    /* Υπολογισμός αθροίσματος των στοιχείων του πίνακα */
    }
    return s;                                         /* Επιστροφή του αποτελέσματος */
}

```

Τα βασικά σημεία του παραπάνω προγράμματος είναι τα ακόλουθα:

- Κατά την κλήση της συνάρτησης ($y=\text{sum}(x)$) ο πίνακας x περνά ως μια απλή μεταβλητή χωρίς την διάστασή του (δηλαδή χωρίς τα σύμβολα []).
- Στην συνάρτηση ως όρισμα βάζουμε ένα πίνακα χωρίς διαστάσεις δηλαδή
`double sum(double a[]).`
 ο πίνακας αποκτά αυτόματα διάσταση ίση με αυτή του πίνακα της συνάρτησης `main()`.

4.5 Πίνακες πολλών διαστάσεων.

Η έννοια του πίνακα μπορεί να επεκταθεί και σε περισσότερες από μία διαστάσεις. Με αυτόν τον τρόπο μπορούμε να ορίσουμε πίνακες δύο, τριών, τεσσάρων κτλ. διαστάσεων. Ας πάρουμε ένα παράδειγμα ενός πίνακα δύο διαστάσεων. Ο παρακάτω πίνακας A είναι ένας πίνακας δύο διαστάσεων με τρεις γραμμές, τέσσερις στήλες και συνολικά δώδεκα στοιχεία

$$A(3 \times 4) = \begin{bmatrix} 1.2 & 4.3 & 7.8 & 9.3 \\ 6.3 & 8.9 & 6.7 & 3.4 \\ 1.9 & 9.8 & 4.1 & 6.6 \end{bmatrix}$$

Ακολουθεί ο ορισμός του πίνακα στη C και η αρχικοποίηση των στοιχείων του:

```

double a[3][4];

a[0][0]=1.2;
a[0][1]=4.3;
a[0][2]=7.8;
a[0][3]=9.3;

```

Πρώτη γραμμή

```

a[1][0]=6.3;
a[1][1]=8.9;
a[1][2]=6.7;
a[1][3]=3.4;

```

Δεύτερη γραμμή

```

a[2][0]=1.9;
a[2][1]=9.8;
a[2][2]=4.1;
a[2][3]=6.6;

```

Τρίτη γραμμή

Εκτός από την παραπάνω δήλωση και αναλυτική αρχικοποίηση του πίνακα μπορούμε να χρησιμοποιήσουμε και τους παρακάτω δύο τρόπους:

```
double a[3][4]={1.2, 4.3, 7.8, 9.3, 6.3, 8.9, 6.7, 3.4, 1.9, 9.8, 4.1, 6.6};
```

```
double a[][4]={1.2, 4.3, 7.8, 9.3, 6.3, 8.9, 6.7, 3.4, 1.9, 9.8, 4.1, 6.6};
```

Προσέξτε πως στον τελευταίο τρόπο παραλείψαμε την πρώτη διάσταση. Όπως και στους μονοδιάστατους πίνακες έχουμε το δικαίωμα να μην δηλώσουμε την πρώτη μόνο διάσταση ενός πολυδιάστατου πίνακα. Ο υπολογιστής προσμετρά τα δεδομένα μέσα στις αγκύλες και την καθορίζει αυτόματα.

Με την ίδια λογική όπως παραπάνω μπορούμε να ορίσουμε πίνακες τριών ή περισσότερων διαστάσεων. Για παράδειγμα ο πίνακας b[3][4][5] έχει συνολικά 60 στοιχεία τα οποία και πρέπει προσεκτικά να τα αρχικοποιήσουμε ακολουθώντας τους παραπάνω κανόνες.

Ας δούμε τώρα, με ένα παράδειγμα, τον τρόπο με τον οποίο μπορούμε να εισάγουμε ένα-ένα τα στοιχεία ενός πίνακα. Για παράδειγμα θα θεωρήσουμε ένα πίνακα double a[3][4]. Ακολουθούν οι γραμμές προγράμματος για την εισαγωγή των δεδομένων από το πληκτρολόγιο.

```
double a[3][4];          /* Δήλωση του πίνακα */
int i,j;                 /* Δήλωση των ακεραίων i και j για τη διαχείριση του πίνακα */

for(i=0; i<3; i++){
    for(j=0; j<4; j++){
        printf("Dose to stoixeio a[%d][%d]:",i,j);      /* Μήνυμα για τον χρήστη */
        scanf("%lf",&a[i][j]);                          /* Εισαγωγή των δεδομένων */
    }
}
```

Όπως παρατηρούμε για να διαχειριστούμε έναν πίνακα δύο διαστάσεων χρειαζόμαστε δύο εντολές for την μία μέσα στην άλλη. Η πρώτη "τρέχει" πάνω στις γραμμές και η δεύτερη στις στήλες του πίνακα. Κάθε φορά εισάγουμε τα δεδομένα σε ένα στοιχείο. Πριν την εισαγωγή των δεδομένων, με την συνάρτηση scanf(), στο κατάλληλο στοιχείο του πίνακα καλό είναι να τυπώνουμε ένα μήνυμα στον χρήστη του προγράμματος για το σε ποιο ακριβώς στοιχείο βρίσκεται κάθε φορά.

Ας δούμε τώρα πως μπορούμε να εκτυπώσουμε έναν πίνακα δύο διαστάσεων σε μορφή γραμμών-στηλών πλήρως στοιχισμένων. Παίρνουμε για παράδειγμα τον παραπάνω πίνακα double a[3][4]. Ο κώδικας για την εκτύπωσή του θα έχει ως ακολούθως:

```
int i,j;                 /* Δήλωση των ακεραίων i και j για τη διαχείριση του πίνακα */

for(i=0; i<3; i++){
    for(j=0; j<4; j++){
        printf("%10.2f",a[i][j]);                      /*Εκτύπωση του στοιχείου a[i][j] */
    }
}
```

```
        printf("\n");        /* Αλλαγή γραμμής */  
    }
```

Όπως παρατηρούμε για να εκτυπώσουμε έναν πίνακα δύο διαστάσεων χρειαζόμαστε δύο εντολές for την μία μέσα στην άλλη. Η πρώτη “τρέχει” πάνω στις γραμμές και η δεύτερη στις στήλες του πίνακα. Κάθε φορά εκτυπώνουμε ένα στοιχείο. Για την εκτύπωση χρησιμοποιούμε το κατάλληλο εύρος πεδίου και αριθμό δεκαδικών ψηφίων. Με αυτόν τον τρόπο επιτυγχάνουμε την πλήρη στοίχιση των στοιχείων κατά την εκτύπωση του. Τέλος χρησιμοποιούμε μία εκτύπωση με τον χαρακτήρα της νέας γραμμής “\n” (μεταξύ των δύο for) έτσι ώστε να αλλάζουμε γραμμή.

Κεφάλαιο V:

Δομές και ενώσεις.

5.1 Δομές.

Όπως αναφέραμε στο προηγούμενο κεφάλαιο η ομαδοποίηση της πληροφορίας στον προγραμματισμό είναι ιδιαίτερα σημαντική. Ένα παράδειγμα ομαδοποίησης της πληροφορίας αποτελούν οι πίνακες οι οποίοι αποτελούνται από στοιχεία του ιδίου τύπου. Είδαμε επίσης πόσο απλά μπορούμε να διαχειριστούμε τα δεδομένα ενός πίνακα.

Το ερώτημα το οποίο τίθεται είναι εάν υπάρχει η δυνατότητα στη C να ορίσουμε οντότητες οι οποίες όμως να αποτελούνται από στοιχεία διαφορετικών τύπων δεδομένων. Για παράδειγμα έστω πως έχουμε το ακόλουθο σύνολο μεταβλητών:

```
float a;      /* Μεταβλητή τύπου float */
int b;        /* Μεταβλητή τύπου int */
float c[3];   /* Πίνακας τύπου float με τρία στοιχεία */
char d[3];    /* Συμβολοσειρά */
a=7.1;
b=10;
c[0]=2.1;
c[1]=4.5;
c[2]=6.8;
d="Hi";
```

Μπορούμε να ορίσουμε μια οντότητα που να έχει μέλη τις παραπάνω μεταβλητές, παρ' ότι αυτές είναι διαφορετικού τύπου μεταξύ τους; Η απάντηση είναι ναι. Τέτοιες οντότητες στην C ονομάζονται **δομές**. Ας δούμε πως μπορούμε να κατασκευάσουμε μια δομή:

- Πρώτα οφείλουμε να ορίσουμε την *περιγραφή της δομής* που περιλαμβάνει τον σχεδιασμό της. Για το παραπάνω παράδειγμα έχουμε:

```
struct my_example{
    float a;
    int b;
    float c[3];
    char d[3];
};
```

Στον ορισμό της δομής οφείλουμε να χρησιμοποιήσουμε ένα όνομα της αρεσκείας μας (ακολουθώντας πάντα τους κανόνες ονοματολογίας που έχουμε αναφέρει). Στον παραπάνω ορισμό χρησιμοποιήσαμε το όνομα *my_example*. Στη συνέχεια μέσα σε αγκύλες αναγράφουμε τα στοιχεία της δομής τα οποία ονομάζονται *μέλη*.

- Το δεύτερο βήμα είναι η δήλωση (κατασκευή μια δομής) με βάση τον παραπάνω ορισμό. Αυτό γίνεται ως εξής:

```
struct my_example s1;
```

Στην παραπάνω γραμμή δηλώσαμε (κατασκευάσαμε) την δομή s1 με βάση τον ορισμό μας στο πρώτο βήμα. Οι δομές που δηλώνουμε μπορούν να έχουν ονόματα της αρεσκείας μας ακολουθώντας πάντα τους κανόνες ονοματολογίας που έχουμε αναφέρει.

- Το τρίτο βήμα είναι η αρχικοποίηση των μελών της δομής. Για το παραπάνω παράδειγμα έχουμε:

```
s1.a=7.1;
s1.b=10;
s1.c[0]=2.1;
s1.c[1]=4.5;
s1.c[2]=6.8;
s1.d="Hi";
```

Όπως παρατηρούμε για να αναφερθούμε στα μέλη της δομής, αυτό γίνεται με το όνομα της δομής και το όνομα του μέλους διαχωριζόμενα από μια τελεία (δηλαδή *όνομα_δομής.όνομα_μέλους*). Για τα μέλη κάθε δομής ισχύουν ότι και για τις απλές μεταβλητές. Μπορούν να εμπλακούν σε υπολογισμούς, να εκτυπωθούν μέσω της printf(), να εισαχθούν σε αυτά δεδομένα μέσω της scanf(), να περάσουν ως ορίσματα σε συναρτήσεις κτλ.

Ένας άλλος τρόπος αρχικοποίησης των μελών της δομής μπορεί να γίνει κατά την δήλωσή της και εικονίζεται στην παρακάτω γραμμή:

```
struct my_example s1={7.1, 10, 2.1, 4.5, 6.8, "Hi" };
```

Όπως παρατηρούμε τα δεδομένα μπαίνουν μέσα σε αγκύλες και αρχικοποιούν τα μέλη της δομής με την σειρά ακριβώς που αυτά εμφανίζονται στην περιγραφή της. Σημειώνουμε επίσης πως ο ορισμός και η δήλωση μιας δομής μπορεί να γίνει ταυτόχρονα με τον ακόλουθο τρόπο:

```
struct my_example{
    float a;
    int b;
    float c[3];
    char d[3];
}s1;
```

Οι δομές αποτελούν σημαντικές οντότητες στον προγραμματισμό σε C. Με τις δομές ο προγραμματιστής έχει την δυνατότητα να ομαδοποιεί την πληροφορία και ουσιαστικά να ορίζει και να δημιουργεί τους δικούς του τύπους δεδομένων που χρειάζεται στο πρόγραμμά του.

Στο παρακάτω σχήμα εικονίζεται η κατάταξη της πληροφορίας στην μνήμη του υπολογιστή για την συγκεκριμένη δομή s1 την οποία δημιουργήσαμε και αρχικοποιήσαμε παραπάνω. Η πληροφορία κατατάσσεται ακριβώς σειριακά.

RAM	
s1.a →	7.1
s1.b →	10
s1.c[0] →	2.1
s1.c[1] →	4.5
s1.c[2] →	6.8
s1.d[0] →	H
s1.d[1] →	i
s1.d[2] →	\0

Παράδειγμα: Να αναπτύξετε ένα πρόγραμμα στο οποίο να ορίσετε μια δομή που να περιγράφει σημεία στο επίπεδο. Με βάση τον ορισμό να δηλώσετε δύο δομές και να τις αρχικοποιήσετε για τα σημεία A(2,3) και B(4,5). Στη συνέχεια να υπολογίσετε και να εκτυπώσετε την απόσταση κάθε σημείου από την αρχή των αξόνων και την απόσταση μεταξύ τους.

Ένα σημείο στο επίπεδο ορίζεται με δύο πραγματικούς αριθμούς που αποτελούν τις συντεταγμένες του. Άρα ο ορισμός της δομής που περιγράφει σημεία στο επίπεδο μπορεί να είναι ο ακόλουθος:

```
struct simeio{
    double x;
    double y;
};
```

Μία λύση του παραπάνω προβλήματος είναι η ακόλουθη:

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

int main(void){

    struct simeio{          /* Ορισμός δομής σημείου στο επίπεδο */
        double x;
        double y;
    };

    struct simeio A={2., 3.}; /* Δήλωση-αρχικοποίηση σημείου A */
    struct simeio B={4.,5.};  /* Δήλωση-αρχικοποίηση σημείου B */
    double ra, rb, rab;       /* Δήλωση μεταβλητών ra, rb, rab */
```

```

ra=sqrt(pow(A.x,2.)+pow(B.y,2.));          /* Υπολογισμός απόστασης σημείου A */
rb=sqrt(pow(B.x,2.)+pow(B.y,2.));          /* Υπολογισμός απόστασης σημείου B */
rab=sqrt(pow(B.x-A.x,2.)+pow(B.y-A.y,2.)); /* Υπολογισμός απόστασης σημείων A-B */

printf("Apostasi simeiou A apo arxi=%f \n",ra); /* Εκτύπωση αποτελεσμάτων */
printf("Apostasi simeiou B apo arxi=%f \n",rb);
printf("Apostasi simeion A-B =%f \n",rab);

return 0;
}

```

Παράδειγμα: Να ορίσετε μια δομή που να περιγράφει μιγαδικούς αριθμούς. Με βάση τον ορισμό να δηλώσετε δύο μιγαδικούς αριθμούς τους $a=3+5i$ και $b=5-8i$.

```

struct migadikos{          /* Ορισμός δομής μιγαδικών */
    double re;
    double im;
};
struct migadikos a={3., 5.}; /* Δήλωση-αρχικοποίηση μιγαδικού a */
struct migadikos b={5., -8}; /* Δήλωση-αρχικοποίηση μιγαδικού b */

```

Παράδειγμα: Να ορίσετε μια δομή που να περιγράφει τους φοιτητές του Τμήματος Φυσικής. Δημιουργήστε έναν φοιτητή με στοιχεία "Anagnostou", "Dimitrios" με αριθμό μητρώου 3241.

Κάθε φοιτητής έχει επώνυμο, όνομα και αριθμό μητρώου. Άρα ο ορισμός της δομής που περιγράφει φοιτητές μπορεί να είναι ο ακόλουθος:

```

struct foititis{          /* Ορισμός δομής φοιτητών */
    char eponimo[20];
    char onoma[20];
    int AM;
};
struct foititis a={ "Anagnostou", "Dimitrios", 3241};

```

Στον παραπάνω κώδικα χρησιμοποιήσαμε για το επώνυμο και για το όνομα συμβολοσειρές με μέγεθος 20, ενώ για τον αριθμό μητρώου μία μεταβλητή τύπου int.

5.4 Πίνακες Δομών.

Στη C έχουμε την δυνατότητα να ορίσουμε πίνακες των οποίων τα στοιχεία είναι δομές. Ως παράδειγμα ας θεωρήσουμε την παραπάνω δομή η οποία περιγράφει φοιτητές. Με βάση αυτόν τον ορισμό μπορούμε να δηλώσουμε έναν πίνακα δομών ως εξής:

```

struct foititis{                /* Ορισμός δομής φοιτητών */
    char epônimo[20];
    char onoma[20];
    int AM;
};
struct foititis A[200];

```

Ο παραπάνω πίνακας A[200] είναι ένας πίνακας δομών με διακόσια στοιχεία. Κάθε στοιχείο είναι δομή και αντιπροσωπεύει έναν φοιτητή. Για παράδειγμα ο πρώτος φοιτητής αντιστοιχεί στα μέλη:

```

A[0].epônimo
A[0].onoma
A[0].AM

```

Ο δεύτερος φοιτητής στα μέλη:

```

A[1].epônimo
A[2].onoma
A[3].AM

```

και ου το καθεξής. Όπως καταλαβαίνουμε ένας πίνακας δομών σαν τον παραπάνω μπορεί να θεωρηθεί ως μια πολύ απλή βάση δεδομένων.

5.3 Δομές και συναρτήσεις.

Ας δούμε με ένα απλό παράδειγμα πως μπορούμε να περνάμε δομές ως ορίσματα σε μία συνάρτηση. Στο παρακάτω πρόγραμμα χρησιμοποιούμε την δομή migadikos που αναπτύξαμε σε προηγούμενη παράγραφο, με βάση την οποία μπορούμε να ορίσουμε μιγαδικούς αριθμούς στα προγράμματά μας. Στη συνέχεια αναπτύσσουμε μία συνάρτηση η οποία απλά τυπώνει έναν μιγαδικό αριθμό που παίρνει ως όρισμα. Την συνάρτηση αυτή την καλούμε μέσα από την main().

```

#include<stdio.h>
struct migadikos{              /* Ορισμός δομής μιγαδικών */
    double re;
    double im;
};
void fx(struct migadikos b);    /* Δήλωση πρωτότυπο της συνάρτησης */

int main(void){
    struct migadikos a={2., 3.};    /* Δήλωση-αρχικοποίηση δομής a */
    fx(a);                          /* Κλήση της συνάρτησης fx με όρισμα τον a */
    return 0;
}

void fx(struct migadikos b){      /* Γενική συνάρτηση εκτύπωσης μιγαδικού */
    printf("(%)i(%)",b.re, b.im);
}

```


Όπως παρατηρούμε στο παραπάνω πρόγραμμα ορίζουμε την δομή που περιγράφει μιγαδικούς αριθμούς ψηλά-ψηλά στο πρόγραμμά μας και πριν από κάθε συνάρτηση. Με αυτό τον τρόπο κοινοποιούμε τον ορισμό της σε όλες τις συναρτήσεις που ακολουθούν. Στην συνέχεια αναφερόμαστε στη δήλωση πρωτότυπο της συνάρτησης μας. Στην συνάρτηση `main()` δηλώνουμε και αρχικοποιούμε έναν μιγαδικό τον `a`. Στο επόμενο βήμα καλούμε την συνάρτηση με όρισμα τον συγκεκριμένο μιγαδικό `a`. Η εκτέλεση του προγράμματος μεταφέρεται τώρα στην συνάρτηση:

```
void fx(struct migadikos b)
```

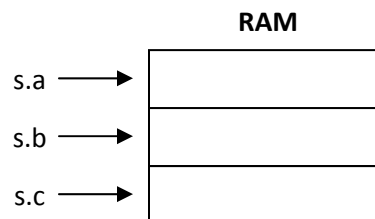
Η συνάρτηση αυτή παίρνει ένα όρισμα το οποίο είναι μιγαδικός. Είναι τύπου `void` γιατί δεν επιστρέφει κάποια τιμή αλλά απλά τυπώνει τον μιγαδικό με την συνάρτηση `printf()` που περιέχει.

5.4 Ενώσεις.

Οι ενώσεις είναι οντότητες οι οποίες μοιάζουν σε όλα με τις δομές εκτός από την διαχείριση μνήμης που κάνουν για την αποθήκευση των μελών τους. Ας δώσουμε ένα από παράδειγμα δομής και ένωσης και ας δούμε την διαφορά τους:

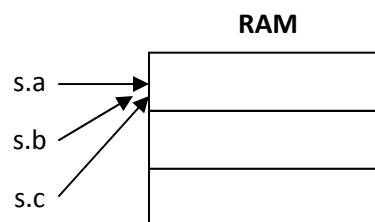
Ορισμός Δομής

```
struct various{  
    float a;  
    int b;  
    char c;  
};  
struct various s;
```



Ορισμός Ένωσης

```
union various{  
    float a;  
    int b;  
    char c;  
};  
union various s;
```



Όπως παρατηρούμε μία ένωση ορίζεται με το όνομα **union** αντί του `struct`. Το κάθε μέλος της δομής καταλαμβάνει τον ξεχωριστό χώρο στη μνήμη του υπολογιστή που του αναλογεί. Σε αντίθεση τα μέλη μιας ένωσης καταλαμβάνουν κοινό χώρο (δεσμεύεται χώρος κατάλληλος ώστε να είναι δυνατόν να αποθηκευθεί το μεγαλύτερο μέλος). Αυτό έχει ως αποτέλεσμα αφ ενός οικονομία στην χρήση της μνήμης του υπολογιστή αφετέρου όμως κάθε φορά μόνο ένα μέλος της ένωσης κάθε φορά περιέχει

δεδομένα. Οι ενώσεις γενικά είναι πολύ πιο δύσχρηστες από τις δομές και χρησιμοποιούνται σπάνια μόνο σε περιπτώσεις όπου ο προγραμματιστής θέλει να κάνει οικονομία στην μνήμη του υπολογιστή. Ειδικότερα σήμερα δεν χρησιμοποιούνται πολύ μιας και οι σύγχρονοι υπολογιστές διαθέτουν πολύ μεγάλη μνήμη.

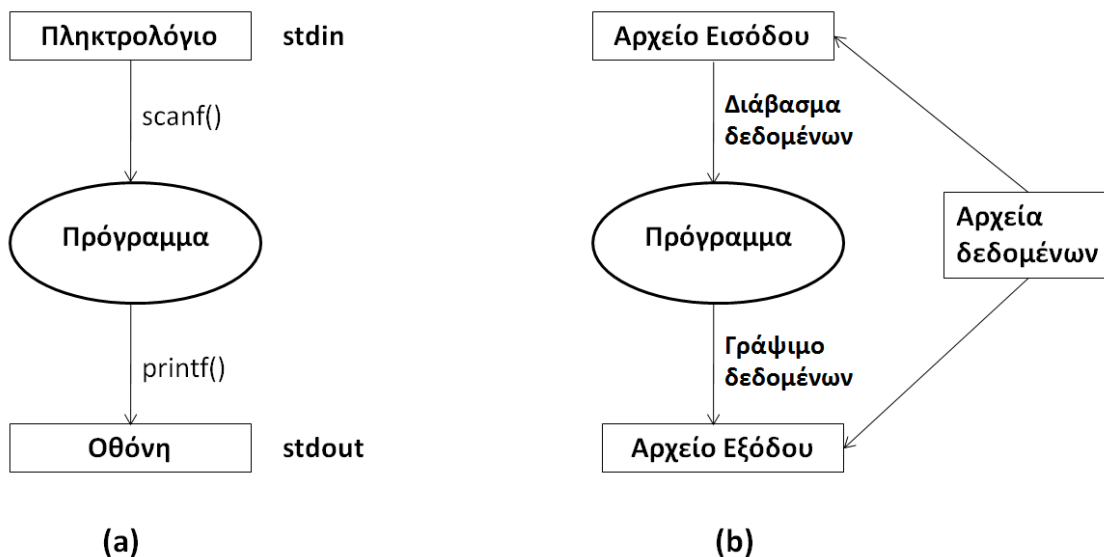
Κεφάλαιο VI:

Προσπέλαση Αρχείων.

5.1 Αρχεία δεδομένων.

Έως τώρα σε ένα πρόγραμμα έχουμε μάθει να εισάγουμε δεδομένα από το πληκτρολόγιο χρησιμοποιώντας την συνάρτηση `scanf()` και να εκτυπώνουμε δεδομένα στην οθόνη του υπολογιστή μας με την συνάρτηση `printf()`. Το πληκτρολόγιο για το πρόγραμμά μας αποτελεί την καθιερωμένη είσοδο γιαυτό και ονομάζεται **stdin** (standard input), ενώ η οθόνη την καθιερωμένη έξοδο και ονομάζεται **stdout** (standard output), όπως εικονίζεται στο σχήμα 5.1 (a).

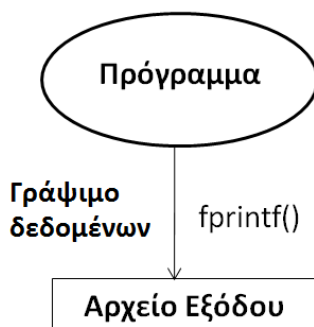
Η χρήση του πληκτρολογίου και της οθόνης ως είσοδος-έξοδος στα προγράμματά μας δεν είναι πάντοτε η ενδεδειγμένη. Σκεφθείτε εάν έχετε να εισάγετε μερικές εκατοντάδες αριθμούς στο πρόγραμμά σας, αυτό είναι πρακτικά αδύνατο να γίνει από το πληκτρολόγιο. Το ίδιο ισχύει εάν ένα πρόγραμμα παράγει πολλές γραμμές στην έξοδό του. Σε αυτή την περίπτωση η λύση είναι να χρησιμοποιήσουμε κατάλληλα **Αρχεία Δεδομένων**, τα οποία το πρόγραμμα χρησιμοποιεί ως είσοδο-έξοδο (σχήμα 5.2 (b)).



Σχήμα 5.1: α) Η καθιερωμένη είσοδος-έξοδος ενός προγράμματος. β) Τα αρχεία δεδομένων ως είσοδος-έξοδος ενός προγράμματος.

5.2 Δημιουργία αρχείου και εγγραφή δεδομένων σε αυτό.

Σε αυτή την παράγραφο θα ασχοληθούμε με το πώς μέσα από ένα πρόγραμμα μπορούμε να δημιουργήσουμε ένα νέο αρχείο και να γράψουμε μέσα σε αυτό δεδομένα. Θα ασχοληθούμε δηλαδή με το πρόβλημα που εικονίζεται στο σχήμα 5.2.



Σχήμα 5.2: Δημιουργία του αρχείου εξόδου και εγγραφή δεδομένων σε αυτό.

Ας υποθέσουμε πως θέλουμε να αναπτύξουμε ένα πρόγραμμα το οποίο να δημιουργήσει ένα αρχείο δεδομένων και να γράψει σε αυτό τα δεδομένα που περιέχει μια μεταβλητή $x=3.14159$. Η λύση φαίνεται παρακάτω:

```
#include<stdio.h>

int main(void){
    double x;           /* Δήλωση μεταβλητής x */
    FILE *fp;           /* Δήλωση δείκτη αρχείου με όνομα fp */
    x=3.14159;          /* Αρχικοποίηση μεταβλητής x */
    fp=fopen("arxeio.data","w"); /* Δημιουργία αρχείου arxeio.data για εγγραφή */
    fprintf(fp,"%f",x);  /* Εγγραφή των δεδομένων στο αρχείο */
    fclose(fp);         /* Κλείσιμο του αρχείου */
    return 0;
}
```

Για την δημιουργία και την εγγραφή δεδομένων σε αρχείο απαιτούνται τα εξής βήματα:

- Το πρώτο βήμα είναι η δήλωση ενός δείκτη αρχείου. Αυτό επιτυγχάνεται με την γραμμή:
FILE *fp;
Ο δείκτης fp (μπορούμε να χρησιμοποιήσουμε εδώ όποιο όνομα θέλουμε σύμφωνα με τους κανόνες ονοματολογίας των μεταβλητών) είναι τύπου FILE, και μπορεί να διαχειρίζεται ένα αρχείο.
- Η δημιουργία του αρχείου γίνεται στην γραμμή:
fp=fopen("arxeio.data","w");

Εδώ εμπλέκεται η συνάρτηση `fopen()` η οποία παίρνει δύο ορίσματα: Το πρώτο όρισμα είναι το όνομα του αρχείου που θέλουμε να δημιουργήσουμε, στο συγκεκριμένο παράδειγμα το όνομα που επιλέξαμε είναι το *arχειο.data*. Το δεύτερο όρισμα προσδιορίζει τι ακριβώς θέλουμε να κάνουμε με το συγκεκριμένο αρχείο, στο συγκεκριμένο παράδειγμα πρέπει να χρησιμοποιήσουμε ως δεύτερο όρισμα το “w” (αρχικό γράμμα της λέξης write) το οποίο σημαίνει πως δημιουργώ ένα νέο αρχείο το οποίο προορίζεται για εγγραφή δεδομένων. Προσοχή! Το όρισμα w θα δημιουργήσει στον σκληρό δίσκο του υπολογιστή το αρχείο με όνομα *arχειο.data*. Εάν ήδη υπάρχει αρχείο με αυτό το όνομα τα δεδομένα μέσα σε αυτό θα χαθούν. Κατά συνέπεια πρέπει να είμαστε πολύ προσεκτικοί.

- Για την εγγραφή δεδομένων μέσα στο αρχείο που δημιουργήσαμε χρησιμοποιούμε την συνάρτηση `fprintf()` όπως στη γραμμή:

```
fprintf(fp,"%f",x);
```

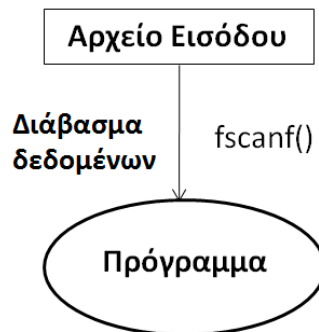
Η σύνταξη της συνάρτησης `fprintf()` είναι ακριβώς η ίδια με αυτή της `printf()` εκτός του ότι παίρνει ένα επί πλέον όρισμα (το πρώτο) το οποίο είναι ο δείκτης του αρχείου.

- Τέλος όταν έχουμε τελειώσει πλήρως με το αρχείο μας, πρέπει να το κλείσουμε με τη χρήση της συνάρτησης `fclose()` όπως στη γραμμή:

```
fclose(fp);
```

5.3 Άνοιγμα αρχείου και διάβασμα δεδομένων από αυτό.

Σε αυτή την παράγραφο θα ασχοληθούμε με το πώς μέσα από ένα πρόγραμμα μπορούμε να “ανοίξουμε” ένα αρχείο το οποίο προϋπάρχει στον σκληρό μας δίσκο αποθηκευμένο και να διαβάσουμε από αυτό δεδομένα. Θα ασχοληθούμε δηλαδή με το πρόβλημα που εικονίζεται στο σχήμα 5.3.



Σχήμα 5.3: Άνοιγμα του αρχείου εισόδου και διάβασμα δεδομένων από αυτό.

Ας υποθέσουμε πως στον σκληρό δίσκο του υπολογιστή μας προϋπάρχει το αρχείο δεδομένων με όνομα *dedomena.txt* και πως μέσα σε αυτό εμπεριέχεται ο αριθμός 7.689. Θέλουμε να αναπτύξουμε ένα πρόγραμμα το οποίο να “ανοίξει” το συγκεκριμένο αρχείο, να διαβάσει τον αριθμό, να τον αποθηκεύσει σε μια κατάλληλη μεταβλητή και να τον τυπώσει στην οθόνη του υπολογιστή μας. Η λύση φαίνεται παρακάτω:

```
#include<stdio.h>

int main(void){
    double y;          /* Δήλωση μεταβλητής y */
    FILE *pp;          /* Δήλωση δείκτη αρχείου με όνομα pp */
    pp=fopen("dedomena.txt ","r"); /* Άνοιγμα αρχείου dedomena.txt για διάβασμα */
    fscanf(pp,"%lf",&y);          /* Διάβασμα των δεδομένων και αποθήκευση στην y */
    printf("y=%f\n",y);          /* Εκτύπωση στην οθόνη της μεταβλητής y */
    fclose(pp);            /* Κλείσιμο του αρχείου */
    return 0;
}
```

Για το άνοιγμα του αρχείου dedomena.txt και το διάβασμα απαιτούνται τα εξής βήματα:

- Το πρώτο βήμα είναι η δήλωση ενός δείκτη αρχείου. Αυτό επιτυγχάνεται με την γραμμή:
FILE *pp;
 Ο δείκτης pp (μπορούμε να χρησιμοποιήσουμε εδώ όποιο όνομα θέλουμε σύμφωνα με τους κανόνες ονοματολογίας των μεταβλητών) είναι τύπου FILE, και μπορεί να διαχειρίζεται ένα αρχείο.
- Το “άνοιγμα” του αρχείου *dedomena.txt* γίνεται στην γραμμή:
pp=fopen("dedomena.txt ","r");
 Εδώ εμπλέκεται η συνάρτηση fopen() η οποία παίρνει δύο ορίσματα: Το πρώτο όρισμα είναι το όνομα του αρχείου από το οποίο θέλουμε να διαβάσουμε, στο συγκεκριμένο παράδειγμα το όνομα του αρχείου το οποίο προϋπάρχει στον σκληρό δίσκο του υπολογιστή μας είναι το *dedomena.txt*. Το δεύτερο όρισμα προσδιορίζει τι ακριβώς θέλουμε να κάνουμε με το συγκεκριμένο αρχείο, στο συγκεκριμένο παράδειγμα πρέπει να χρησιμοποιήσουμε ως δεύτερο όρισμα το “r” (αρχικό γράμμα της λέξης read) το οποίο σημαίνει πως το αρχείο προορίζεται για ανάγνωση δεδομένων. Προσοχή! Εάν το αρχείο με όνομα *dedomena.txt* δεν υπάρχει το πρόγραμμα θα τερματιστεί παράγοντας μήνυμα λάθους.
- Για το διάβασμα δεδομένων από το αρχείο *dedomena.txt* χρησιμοποιούμε την συνάρτηση fscanf() όπως στη γραμμή:
fscanf(pp,"%lf",&y);
 Η σύνταξη της συνάρτησης fscanf() είναι ακριβώς η ίδια με αυτή της scanf() εκτός του ότι παίρνει ένα επί πλέον όρισμα (το πρώτο) το οποίο είναι ο δείκτης του αρχείου.
- Τέλος όταν έχουμε τελειώσει πλήρως με το αρχείο μας πρέπει να το κλείσουμε με τη χρήση της συνάρτησης fclose() όπως στη γραμμή:
fclose(pp);

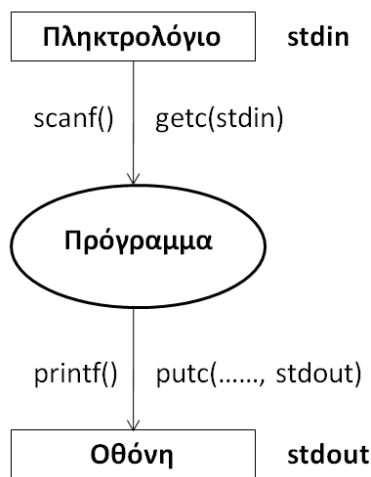
Εδώ συνοψίζοντας σχετικά με την συνάρτηση fopen(), το δεύτερο όρισμα έχει την ακόλουθη έννοια:

- w : Δημιουργία νέου αρχείου προς εγγραφή δεδομένων.
- r : “Άνοιγμα” αρχείου το οποίο προϋπάρχει προς ανάγνωση δεδομένων.

- a : “Ανοιγμα” αρχείου το οποίο προϋπάρχει προς πρόσθεση νέων δεδομένων μετά το τέλος του.

5.2 Οι συναρτήσεις putc() και getc().

Γενικά η είσοδος-έξοδος κειμένου στα προγράμματα είναι πολύ σημαντική και γιαυτό τον λόγο στη C υπάρχουν ειδικές συναρτήσεις εισόδου-εξόδου που αφορούν αποκλειστικά χαρακτήρες. Οι συναρτήσεις αυτές είναι οι **getc()** και **putc()**. Οι δύο αυτές συναρτήσεις μπορούν να χρησιμοποιηθούν είτε για είσοδο από το πληκτρολόγιο είτε για έξοδο στην οθόνη του υπολογιστή μας όπως φαίνεται στο σχήμα 5.4.



Σχήμα 5.4: Οι συναρτήσεις εισόδου-εξόδου χαρακτήρων getc() και putc().

Η σύνταξη των συναρτήσεων getc() και putc() εικονίζεται στις παρακάτω γραμμές τόσο για την είσοδο των χαρακτήρων από το πληκτρολόγιο όσο και για την έξοδο στην οθόνη. Για να γίνει κατανοητή η χρήση τους εικονίζεται δίπλα και ο αντίστοιχος κώδικας με τις συναρτήσεις scanf() και printf().

Είσοδος χαρακτήρων από το πληκτρολόγιο

Συνάρτηση getc()

```
char c;  
c=getc(stdin);
```

⇔

Συνάρτηση scanf()

```
char c;  
scanf("%c",&c);
```

Έξοδος χαρακτήρων στην οθόνη

Συνάρτηση putc()

```
char c;  
c='k';  
putc(c,stdout);
```

⇔

Συνάρτηση printf()

```
char c;  
c='k';  
printf("%c",c);
```

Οι συναρτήσεις `getc()` και `putc()` μπορούν να ανακατευθύνουν χαρακτήρες και σε κατάλληλα αρχεία εισόδου-εξόδου, αρκεί στην θέση των `stdin` και `stdout` να χρησιμοποιήσουμε τους αντίστοιχους δείκτες αρχείων. Ακολουθεί η σύνταξη των συναρτήσεων `getc()` και `putc()` τόσο για την είσοδο χαρακτήρων από αρχείο δεδομένων όσο και για την έξοδο.

Είσοδος χαρακτήρων από αρχείο δεδομένων

```
char c;          /* Δήλωση μεταβλητής τύπου char */
FILE *fp;        /* Δήλωση δείκτη αρχείου με όνομα fp */
fp=fopen("dedomena.txt","r"); /* Άνοιγμα αρχείου dedomena.txt για διάβασμα */
c=getc(fp);      /* Διάβασμα ενός χαρακτήρα από το αρχείο και αποθήκευση στην c */
fclose(fp);      /* Κλείσιμο του αρχείου */
```

Έξοδος χαρακτήρων σε αρχείο δεδομένων

```
char c;          /* Δήλωση μεταβλητής τύπου char */
FILE *fp;        /* Δήλωση δείκτη αρχείου με όνομα fp */
c='k';           /* Αρχικοποίηση της μεταβλητής c */
fp=fopen("arxeio.data","w"); /* Δημιουργία αρχείου arxeio.data για εγγραφή */
putc(c, fp);     /* Γράψιμο του χαρακτήρα στο αρχείο */
fclose(fp);      /* Κλείσιμο του αρχείου */
```

Σημειώνουμε εδώ πως στην περίπτωση ανακατεύθυνσης χαρακτήρων σε αρχεία εισόδου-εξόδου η C είναι εφοδιασμένη και με τις συναρτήσεις **`fgetc()`** και **`fputc()`** οι οποίες είναι ακριβώς ισοδύναμες με τις `getc()` και `putc()`.

Τέλος κλείνοντας αυτή την παράγραφο θέλουμε να επισημάνουμε κάτι πολύ χρήσιμο για τα προγράμματά μας. Όταν διαβάζουμε έναν-έναν του χαρακτήρες ενός αρχείου, για να αντιληφθούμε πως έχουμε φτάσει στο τέλος του, πρέπει να ελέγχουμε για έναν ειδικό χαρακτήρα ο οποίος σημαίνει το τέλος ενός αρχείου. Ο χαρακτήρας αυτός είναι ο **EOF** (από τα αρχικά των αγγλικών λέξεων End Of File). Για παράδειγμα στις επόμενες γραμμές δίνουμε έναν εύκολο τρόπο για να διαβάσουμε όλους τους χαρακτήρες ενός αρχείου έως το τέλος του:

```
char c;          /* Δήλωση μεταβλητής τύπου char */
FILE *fp;        /* Δήλωση δείκτη αρχείου με όνομα fp */
fp=fopen("dedomena.txt","r"); /* Άνοιγμα αρχείου dedomena.txt για διάβασμα */

while((c=getc(fp))!=EOF){
    .....
    .....
}

fclose(fp);      /* Κλείσιμο του αρχείου */
```


Στην γραμμή: `while((c=getc(fp))!=EOF){ }` διαβάζουμε πρώτα έναν-έναν τους χαρακτήρες του κειμένου. Η εντολή `while` εκτελείται όσο ο χαρακτήρας που διαβάζουμε από το αρχείο δεν είναι ίσος με τον EOF.