Πανεπιστήμιο Αιγαίου

URL: http://www.aegean.gr

Εισαγωγή στις γλώσσες προγραμματισμού με τη γλώσσα C

Παναγιώτης Νάστου
Πανεπιστήμιο Αιγαίου
Τμήμα Μαθηματικών
832 00 Καρλόβασι
Σάμος

Κεφάλαιο 1

Εισαγωγή στις Γλώσσες Προγραμματισμού

Μια γλώσσα προγραμματισμού είναι μια τεχνητή γλώσσα μέσω της οποίας καθίσταται δυνατός ο έλεγχος της λειτουργίας μιας μηχανής, ενός υπολογιστή. Πιο συγκεκριμένα πρόκειται για μια συστηματική σημειογραφία με την οποία μπορεί να περιγραφεί ένα σύνολο βημάτων τα οποία μπορεί να εκτελέσει ένας υπολογιστής για την επίλυση ενός προβλήματος.

Οι φυσικές γλώσσες χρησιμοποιούνται μόνο για την αλληλεπίδραση μεταξύ ανθρώπων. Όταν δύο άνθρωποι επικοινωνούν μέσω μιας φυσικής γλώσσας οι ομιλητές ή οι γράφοντες μπορεί να είναι διφορούμενοι κάνοντας μικρά λάθη αλλά πάραυτα προσδοκούν ότι θα γίνουν κατανοητοί από τους συμμετέχοντες στην επικοινωνία. Δεν συμβαίνει όμως το ίδιο και με τους υπολογιστές. Ένας υπολογιστής κάνει ακριβώς αυτό που θα του πούμε να κάνει και δεν μπορεί να αντιληφθεί τι σκόπευε ο προγραμματιστής να γράψει σε περίπτωση που κάτι δεν γραφεί με σαφήνεια.

Ο προγραμματισμός είναι η δραστηριότητα υλοποίησης κατά τρόπο τυπικό αλγορίθμων σε υπολογιστές. Ένας αλγόριθμος διατυπωμένος σε μια γλώσσα κατανοητή από υπολογιστή ονομάζεται **πρόγραμμα**. Τι είναι όμως ο αλγόριθμος; Ο αλγόριθμος είναι μια πεπερασμένη ακολουθία συγκεκριμένων βημάτων τα οποία όταν εκτελεσθούν σε ένα υπολογιστή δίνουν τη λύση ενός υπολογιστικού προβλήματος. Έτσι ένας αλγόριθμος δέχεται κανένα ή περισσότερα δεδομένα εισόδου, εκτελεί μια πεπερασμένη ακολουθία καλά ορισμένων εντολών και παράγει τα δεδομένα εξόδου που επιλύουν το πρόβλημα για το οποίο αναπτύχθηκε ο αλγόριθμος (αλγόριθμος Ευκλείδη για την έρευση του ΜΚΔ, το κόσκινο του ερατοσθένη, αλγόριθμος εύρεσης τέλειων αριθμών κλπ).

Ένα πρόβλημα κατά την ανάπτυξη ενός αλγόριθμου είναι με ποιο τρόπο μπορεί να περιγραφεί πριν την υλοποίηση του με τη χρήση της κατάλληλης γλώσσας προγραμματισμού. Έμπειροι προγραμματιστές χρησιμοποιούν πολλές φορές οποιαδήποτε φυσική γλώσσα και στη συνέχεια προχωρούν στην υλοποίηση του. Ένας άλλος αποτελεσματικός τρόπος περιγραφής αλγορίθμων είναι τα **διαγράμματα ροής** τα οποία αποδίδουν κατά τρόπο λεπτομερή, κατανοητό και παραστατικό, την ακολουθία εντολών του αλγόριθμου. Ο πιο διαδεδομένος τρόπος περιγραφής αλγορίθμων είναι η χρήση μιας γλώσσας που βασίζεται σε εντολές μια δομημένης γλώσσας όπως η Pascal ή C που ονομάζεται **ψευδοκώδικας**. Στον ψευδο-κώδικα για την απόδοση λεπτομεριών του αλγορίθμου μπορεί να χρησιμοποιηθεί και μια οποιαδήποτε φυσική γλώσσα. Το πλεονέκτημα αυτής της μεθόδου έναντι αυτής του διαγράμματος ροής είναι ότι είναι πιο εύκολη η μεταφορά του αλγόριθμου σε μια γλώσσα προγραμματισμού και ότι ο αλγόριθμος καθίσταται περισσότερο κατναοητός.

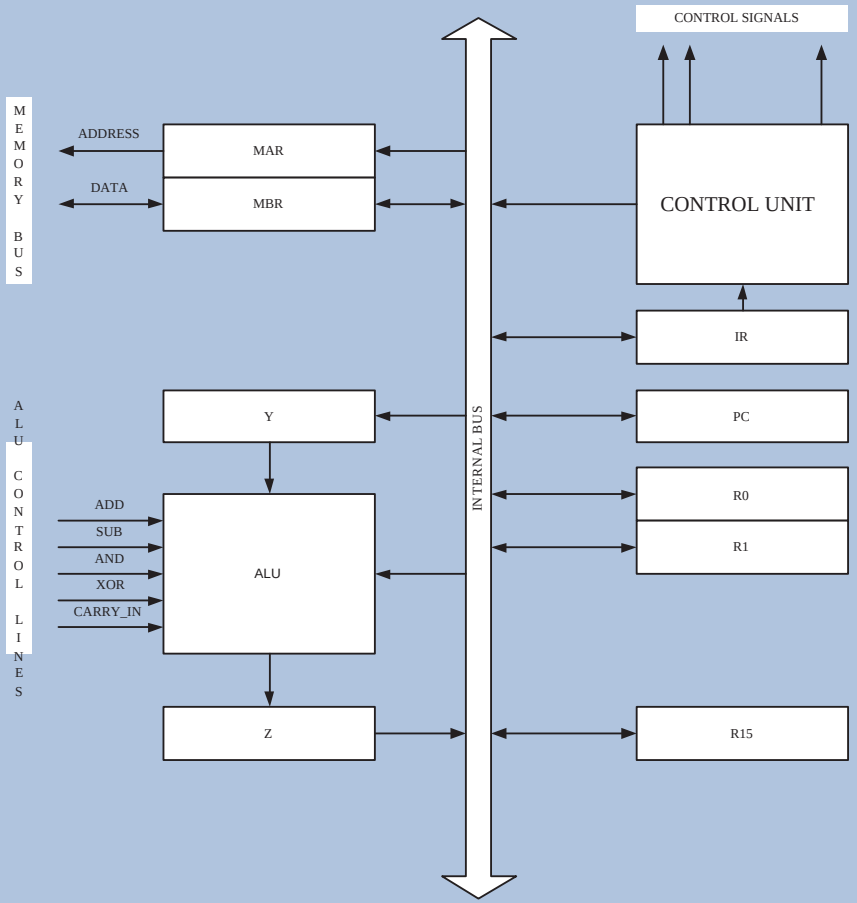
1.1. Γλώσσα Μηχανής και Συμβολικές Γλώσσες

Κάθε υπολογιστής διαθέτει ένα σύνολο πρωτογενών εντολών τις οποίες μπορεί να εκτελέσει απευθείας. Πρόκειται για το σύνολο εντολών του επεξεργαστή του, το οποίο είναι γνωστό ως **ρεπερτόριο εντολών** του επεξεργαστή. Το σύνολο αυτό των πρωτογενών εντολών

ονομάζεται γλώσσα μηχανής καθώς είναι η γλώσσα που κατανοεί ο υπολογιστής.

Ο προγραμματιστής που επιχειρεί να υλοποιήσει ένα αλγόριθμο στη γλώσσα μηχανής ενός υπολογιστή θα πρέπει να χρησιμοποιήσει εντολές της γλώσσας μηχανής του υπολογιστή αυτού. Είναι φανερό ότι ένα πρόγραμμα σε γλώσσα μηχανής ενός υπολογιστή δεν μπορεί να εκτελεσθεί σε υπολογιστή που διαθέτει διαφορετικό επεξεργαστή από αυτόν για τον οποίο αναπτύχθηκε το πρόγραμμα καθώς αυτός αντιλαμβάνεται διαφορετική γλώσσα μηχανής.

Μια εντολή μηχανής αποτελείται από δύο τμήματα: τον κώδικα λειτουργίας (operation code) και τους τελεστέους (operands) και είναι σε καθαρή δυαδική μορφή δηλαδή αποτελείται από 0 και 1. Ο κώδικας λειτουργίας καθορίζει ποιά θα είναι η λειτουργία που θα εκτελεσθεί ενώ οι τελεστέοι είναι τα δεδομένα πάνω στα οποία θα εκτελεσθεί η λειτουργία που ορίζει ο κώδικας λειτουργίας ή η διεύθυνση της μνήμης που θα αποθηκευθεί το αποτέλεσμα που θα προκύψει από την εκτέλεση της εντολής. Ένας τελεστέος μπορεί να είναι είτε δεδομένο είτε η διεύθυνση της μνήμης όπου βρίσκεται το δεδομένο ή που θα αποθηκευθεί το αποτέλεσμα της εντολής. Το πλήθος των τελεστέων σε μια εντολή μηχανής εξαρτάται από την αρχιτεκτονική του επεξεργαστή.



Σχήμα 1.1: Τυπική Δομή ενός επεξεργαστή.

Στο Σχήμα 1.1 δίνεται η τυπική δομή ενός επεξεργαστή προκειμένου να δώσουμε ένα παράδειγμα εντολών γλώσσας μηχανής. Ο επεξεργαστής αποτελείται:

1. Την αριθμητική και λογική μονάδα ALU που είναι η καρδιά ενός επεξεργαστή καθώς αυτή είναι που εκτελεί τις βασικές λειτουργίες. Στο Σχήμα διακρίνονται συνολικά τέσσερεις λειτουργίες: δύο αριθμητικές (πρόσθεσης) και δύο λογικές (ΚΑΙ και αποκλειστικό-Η). Για να προσδιοριστεί μια από τις τέσσερεις λειτουργίες απαιτούνται 2 bit καθώς μπορούν δώσουν $2^2 = 4$ συνδυασμούς έναν για κάθε λειτουργία όπως φάινεται στον Πίνακα 1.1,

2. τον καταχωρητή IR στον οποίο αποθηκεύεται ο κωδικός λειτουργίας μιας εντολής μηχανής,

3. τον καταχωρητή PC στον οποίο αποθηκεύεται η διεύθυνση της θέσης μνήμης στην οποία βρίσκεται η επόμενη εντολή μηχανής που πρόκειται να εκτελεστεί,

4. τους καταχωρητές $R_0 \ldots R_{15}$ οι οποίοι είναι διαθέσιμοι στον προγραμματιστή για την αποθήκευση τελεστέων,

5. τον καταχωρητή *Memory Address Register* (*MAR*) στον οποίο αποθηκεύεται η διεύθυνση της θέσης μνήμης από όπου πρόκειται να διαβασθεί/γραφεί δεδομένο ή να διαβασθεί εντολή μηχανής,

6. τον καταχωρητή *Memory Buffer Register* (*MBR*) στον οποίο βρίσκεται το δεδομένο ή η εντολή που έχει διαβασθεί από τη θέση μνήμης της οποίας η διεύθυνση βρίσκεται στον *MAR*, ή το δεδομένο που θα εγγραφεί στη θέση μνήμης της οποίας η διεύθυνση βρίσκεται στον *MAR*,

7. τη Μονάδα Ελέγχου CONTROL UNIT η οποία αποκωδικοποιεί τον κωδικό λειτουργίας που βρίσκεται στον καταχωρητή IR και στη συνέχεια μέσω των σημάτων ελέγχου CONTROL SIGNALS κατευθύνει τη λειτουργία της αριθμητικής και λογικής μονάδας καθώς και των υπολοίπων μονάδων όπως καταχωρητές και εξωτερική μνήμη.

Δεκαδικός	κωδικός	Λειτουργία	Μνημονικά
0	00	Πρόσθεση	ADD
1	01	Αφαίρεση	SUB
2	10	Λογικό-ΚΑΙ	AND
3	11	Αποκλειστικό-Η	XOR

Πίνακας 1.1: Κωδικοί λειτουργίας του επεξεργαστή του παραδείγματος

Κάθε πρόγραμμα είναι αποθηκευμένο στην κύρια μνήμη του υπολογιστή. Ο επεξεργαστής διαβάζει κάθε φορά μια εντολή (ή ένα σύνολο εντολών σε σύγχρονες αρχιτεκτονικές επεξεργαστών) την οποία και εκτελεί. Το αποτέλεσμα της εκτέλεσης αποθηκεύεται είτε σε καποιον από τους καταχωρητές του επεξεργαστή είτε στην κύρια μνήμη. Έστω για παράδειγμα ότι θέλουμε να προσθέσουμε δύο αριθμούς που βρίσκονται στους καταχωρητές R_2 και R_3 του επεξεργαστή και το αποτέλεσμα να αποθηκευθεί στον καταχωρητή $R1$. Έστω ότι οι εντολές της μηχανής μας, επιτρέπουν τρεις τελεστέους σε μια εντολή μηχανής και ότι η μορφή της μπορεί να είναι <κώδικας_λειτουργίας><διεύθυνση_προορισμού> <πηγαία_διεύθυνση1> <πηγαία_διεύθυνση2>.

Τότε η εντολή μηχανής για την πρόσθεση θα έχει τη μορφή:

00 0001 0010 0011

όπου τα πρώτα 2 bit 00 είναι ο κώδικας λειτουργίας που αντιστοιχεί στην πρόσθεση σύμφωνα με τον Πίνακα 1.1, τα επόμενα 4 bit 0001 = $(1)_{10}$ αντιστοιχούν στην διεύθυνση του καταχωρητή όπου θα αποθηκευθεί το αποτέλεσμα και που είναι ο R_1. Οι επόμενες δύο τετράδες από bit 0010 0011 αντιστοιχούν στις διευθύνσεις των καταχωρητών R_2 και R_3. Με άλλα λόγια, η παραπάνω εντολή μηχανής εκτελεί την πράξη $R_1 = R_2 + R_3$.

Είναι φανερό από το παραπάνω παράδειγμα ότι ο προγραμματισμός σε γλώσσα μηχανής είναι ιδιαίτερα επίπονος και χρονοβόρος και απαιτεί γνώση της αρχιτεκτονικής της μηχανής στην οποία πρόκειται να εκτελεσθεί το πρόγραμμα. Να τονισθεί, ότι όταν ανα-

πτυχθεί ένα πρόγραμμα στη γλώσσα ενός επεξεργαστή, η μεταφορά του προγράμματος σε άλλο επεξεργαστή δεν είναι άμεση, αλλά απαιτεί προσαρμογή στη νέα αρχιτεκτονική. Βέβαια, το πλεονέκτημα ενός προγράμματος σε γλώσσα μηχανής είναι ότι εκμεταλλεύεται πλήρως τις καινοτομίες του επεξεργαστή και κατ' επέκταση το πρόγραμμα είναι πιο γρήγορο.

Οι δυσκολίες στον προγραμματισμό σε γλώσσα μηχανής οδήγησε στο πρώτο βήμα ανάπτυξης των γλωσσών προγραμματισμού που ήταν οι **Συμβολικές Γλώσσες (Assembly Languages)**. Στις γλώσσες αυτές, οι κώδικες λειτουργίας αντικαταστάθηκαν από μνημόνικα (**mnemonics**) και οι απόλυτες διευθύνσεις από συμβολικά ονόματα. Έτσι για παράδειγμα η εντολή μηχανής που δόθηκε προηγουμένως μπορεί να γραφεί

$$ADD\ R_1,\ R_2,\ R_3$$

όπου πλέον έχουμε απαλλαγεί από τη δυαδική μορφή των εντολών μηχανής και πλέον οι εντολές της συμβολικής γλώσσας είναι περισσότερο κατανοητές και εύχρηστες από τον προγραμματιστή. Βέβαια ένα πρόγραμμα σε συμβολική γλώσσα δεν είναι άμεσα εκτελέσιμο από έναν υπολογιστή καθώς όπως αναφέραμε ένας υπολογιστής μπορεί να εκτελέσει μόνο εντολές μηχανής του επεξεργαστή του. Συνεπώς, είναι απαραίτητο ένα ειδικό πρόγραμμα ο **συμβολομεταφραστής (Assembler)** ο οποίος αναλαμβάνει να μεταφράσει ένα πρόγραμμα σε συμβολική γλώσσα σε ένα πρόγραμμα σε γλώσσα μηχανής. Είναι φανερό ότι μια συμβολική γλώσσα σχετίζεται άμεσα με τον επεξεργαστή καθώς δεν κάνει τίποτε άλλο από το να αποδίδει τις εντολές μηχανής του επεξεργαστή με συμβολικό τρόπο.

1.2. Γλώσσες Υψηλού Επιπέδου

Όπως αναφέρθηκε στο εδάφιο 1.1, ο προγραμματισμός σε γλώσσα μηχανής και σε συμβολική γλώσσα είναι ισχυρά εξαρτημένος από την αρχιτεκτονική της μηχανής. Επίσης, τα προγράμματα αποτελούνται από ατομικές εντολές ενώ δεν υπάρχει κάποιου είδους δομή

στον τρόπο υπολογισμού παραστάσεων, στον τρόπο αναπαράστασης των δεδομένων στη μνήμη και στις εντολές του προγράμματος.

Έτσι προέκυψε η ανάγκη για σχεδιασμό γλωσσών προγραμματισμού υψηλού επιπέδου, οι οποίες θα έπρεπε να είναι σε μεγάλο βαθμό ανεξάρτητες από την αρχιτεκτονική του υπολογιστή. Έτσι θα λύνονταν το πρόβλημα της φορητότητας καθώς λόγω του ότι ένα πρόγραμμα θα ήταν ανεξάρτητο από την αρχιτεκτονική του υπολογιστή θα ήταν πολύ εύκολο να εκτελεσθεί σε οποιοδήποτε υπολογιστή με μικρές αλλαγές και όχι με επανασχεδιασμό του.

Η εμφάνιση δομής στις γλώσσες προγραμματισμού υψηλού επιπέδου ξεκίνησε από τον τρόπο υπολογισμού των εκφράσεων και τον τρόπο αναπαράστασης δεδομένων. Έτσι μια εντολή ανάθεσης της μορφής $d = a + b \times c$ ολοκληρώνει τέσσερεις εντολές της συμβολικής γλώσσας (δύο για την ανάγνωση των δεδομένων b, c μια για την εκτέλεση του πολλαπλασιασμού και μια για την εκτέλεση της πρόσθεσης και αποθήκευσης του αποτελέσματος στη d). Επίσης σε ότι αφορά τα δεδομένα άρχισαν να εμφανίζονται οι πρώτες δομές δεδομένων που διευκόλυναν τους υπολογισμούς όπως αυτή του πίνακα δεδομένων όπου σε συνεχόμενες θέσεις μνήμης μπορούν να αποθηκεύονται δεδομένα και να εκτελούνται υπολογισμοί επι των στοιχείων του πίνακα κατά τρόπο δομημένο.

Στη συνέχεια σχεδιάσθηκαν τέσσερεις βασικές δομές εντολών μέσω των οποίων μπορεί να αναπτυχθεί οποιοδήποτε πρόγραμμα οι οποίες είναι οι εξής:

1. **ομάδα εντολών**, όπου μια ή περισσότερες εντολές σχηματίζουν μια ομάδα όπου η κάθε εντολή εκτελείται η μια μετά την άλλη,

2. **εντολές απόφασης**, με τις οποίες δύναται να αλλάξει η σειρά εκτέλεσης των εντολών ενός προγράμματος (ροή προγράμματος) ανάλογα με την τιμή που θα προκύψει από την αποτίμηση μιας έκφρασης,

3. **εντολές επανάληψης**, οι οποίες δίνουν τη δυνατότητα για επαναληπτική εκτέλεση μιας εντολής ή μιας ομάδας εντολών είτε για συγκεκριμένο αριθμό επαναλήψεων, είτε όσο ικανοποιείται μια λογική συνθήκη,

4. **συναρτήσεις και διαδικασίες**, όπου μια ομάδα εντολών μπορεί να αντιμετωπισθεί ως μια εντολή, δίνοντας τη δυνατότητα για μια **ιεραρχική δομή** στα προγράμματα.

Οι περισσότερες γλώσσες υψηλού επιπέδου διαθέτουν μια πρότυπη βιβλιοθήκη, η οποία περιλαμβάνει ορισμούς των πιο γνωστών αλγορίθμων, δομών δεδομένων καθώς και μηχανισμών εισόδου και εξόδου δεδομένων. Από τη μεριά του χρήστη, η βιβλιοθήκη αυτή αντιμετωπίζεται ως αναπόσπαστο στοιχείο της γλώσσας ενώ από τη μερια του σχεδιαστή αντιμετωπίζεται σαν μια ξεχωριστή οντότητα. Επιπρόσθετα, ο προγραμματιστής έχει τη δυνατότητα να δημιουργεί τις δικές του βιβλιοθήκες συμβάλλοντας στην επαναχρησιμοποίηση των προγραμμάτων που δημιουργεί (reusability).

Κατά τον σχεδιασμό των γλωσσών προγραμματισμού υψηλού επιπέδου ο προσανατολισμός ήταν η υπο σχεδίαση γλώσσα να επιλύει προβλήματα συγκεκριμένης κατηγορίας εφαρμογών. Έτσι αρχικά αναπτύχθηκαν η FORTRAN (FORmula TRANslation) η οποία αναπτύχθηκε αρχικά από την IBM σε μια προσπάθεια σχεδιασμού μιας γλώσσας για πολύπλοκες αριθμητικές εκφράσεις και βρίσκει εφαρμογή σε προβλήματα αριθμητικής ανάλυσης, η LISP (**LISt Processing**) η οποία έχοντας ως βασική δομή δεδομένων τη λίστα και υιοθετώντας ως βασικό μηχανισμό τη χρήση συναρτήσεων θεμελίωσε τον συναρτησιακό προγραμματισμό με πεδίο εφαρμογής την τεχνητή νοημοσύνη και η **COBOL** (**COmmon Business-Oriented Language**) η οποία χρησιμοποιείται ακόμη και σήμερα για την ανάπτυξη εμπορικών εφαρμογών.

Στη συνέχεια αναπτύχθηκε η γλώσσα **C**, η οποία αποτέλεσε τη βάση του λειτουργικού συστήματος UNIX αλλά και μεγάλου μέρους του υπάρχοντος λογισμικού συστημάτων. Οι περισσότερες ενσωματωμένες συσκευές σήμερα φέρουν λογισμικό που έχει αναπτυχθεί σε γλώσσα **C**. Είναι από τις πιο δημοφιλείς γλώσσες προγραμματισμού συστημάτων **system programming Language**. Την ίδια εποχή αναπτύχθηκε και η **Prolog** η οποία είναι η πρώτη γλώσσα λογικού προγραμματισμού με ευρύτατη εφαρμογή στην τεχνητή νοημοσύνη.

Ο αντικειμενοστρεφής προγραμματισμός (Object Oriented Programming OOP) έχει τις ρίζες του στη δεκαετία του 60 με την πρώτη αντικειμενοστρεφή γλώσσα την Small-

talk. Στις παραδοσιακές γλώσσες προγραμματισμού, ένα πρόγραμμα είναι ένα σύνολο από υπολογιστικές διεργασίες που πρέπει να εκτελεσθούν. Στις OOP γλώσσες, ένα πρόγραμμα είναι μια συλλογή από αντικείμενα τα οποία μπορούν να επικοινωνούν μεταξύ τους μέσω μηνυμάτων. Κάθε αντικείμενο είναι ικανό να λαμβάνει μηνύματα, να επεξεργάζεται δεδομένα και να στέλνει μηνύματα σε άλλα αντικείμενα. Πρόκειται για μικρές ανεξάρτητες μονάδες με συγκεκριμένο ρόλο και εμβέλεια. Παραδοσιακά ένα αντικείμενο συνίσταται από τα δεδομένα και ένα σύνολο ενεργειών που μπορούν να εκτελεσθούν πάνω σε αυτά όταν αυτό ζητηθεί μέσω κάποιου μηνύματος από κάποιο αντικείμενο. Στην κατηγορία των OOP γλωσσών ανήκουν η C++ η οποία επίσης ανήκει και στην κατηγορία των γλωσσών προγραμματισμού συστημάτων καθώς χρησιμοποιείται ευρύτατα στην ανάπτυξη λογισμικού συστημάτων και η Java.

Στη συνέχεια του βιβλίου θα αναλύσουμε τους μηχανισμούς των παραδοσιακών γλωσσών προγραμματισμού μέσα από την πιο δημοφιλή γλώσσα προγραμματισμού τη C.

1.3. Τυπικός ορισμός μιας Γλώσσας

Έστω $A = \{a, b, c, \ldots, x, y, z\}$ το σύνολο των γραμμάτων του λατινικού αλφάβητου. Κάθε λέξη n γραμμάτων που μπορεί να σχηματισθεί από γράμματα του A δεν είναι τίποτε άλλο από μια διατεταγμένη n-αδα. Για παράδειγμα η λέξη chapter είναι η 7-αδα ((((((c,h),a),p),t),e),r). Το σύνολο όλων των ακολουθιών n γραμμάτων συμβολίζεται με A^n, ενώ το σύνολο όλων των ακολουθιών με κανένα ή περισσότερα γράμματα συμβολίζεται με A^*.

Διευρύνουμε το σύνολο A με τα κεφαλαία γράμματα και με τους χαρακτήρες στίξης όπου ο χαρακτήρας _ αναπαριστά το κενό, δηλαδή:

$$A = \{a, b, c, \ldots, x, y, z, A, B, C, \ldots, X, Y, Z, ., , , :, ; , !, \_\}$$

Μετά την παραπάνω διεύρυνση μια πρόταση p στην Αγγλική Γλώσσα είναι μια ακολουθία γραμμάτων και χαρακτήρων στίξης, δηλαδή $p \in A^*$. Έτσι τυπικά ο ορισμός μιας

γλώσσας είναι ο ακόλουθος:

Ορισμός 1.3.1 *Αν A είναι ένα πεπερασμένο σύνολο, τότε μια γλώσσα επί του A είναι ένα υποσύνολο του συνόλου A^*. Το σύνολο A είναι το αλφάβητο της γλώσσας.*

Έστω για παράδειγμα B το σύνολο χαρακτήρων της γλώσσας προγραμματισμού C

$$\begin{aligned} B \quad = \quad & \{a, b, c, \ldots, x, y, z, A, B, C, \ldots, X, Y, Z, 0, 1, \ldots, 8, 9, \\ & +, -, /, \%, *, \tilde{}, \&, \hat{}, |, !, <, =, >, (,), \#, ?, :, ;, ., `, ', \{, \}, \_, , \backslash\} \end{aligned}$$

τότε κάθε πρόταση της γλώσσας προγραμματισμού C είναι στοιχείο υποσυνόλου του συνόλου B^*.

Το **συντακτικό** μιας οποιαδήποτε γλώσσας είτε φυσικής είτε τεχνητής είναι ένα σύνολο κανόνων που καθορίζει αν μια πρόταση έχει σχηματισθεί σωστά. Σύμφωνα με τον τυπικό ορισμό που δώσαμε για τις γλώσσες ότι δηλαδή μια γλώσσα είναι ένα σύνολο από ακολουθίες χαρακτήρων, θα μπορούσε να ορισθεί το συντακτικό μιας γλώσσας απλά ορίζοντας τις ιδιότητες των στοιχείων του συνόλου αυτού. Δοθείσας της περιγραφής μιας γλώσσας το ενδιαφέρον εστιάζεται στα εξής:

1. να σχηματισθούν κατά τρόπο εύκολο ακολουθίες χαρακτήρων της γλώσσας.
2. να αναγνωρισθεί εύκολα αν μια οποιαδήποτε ακολουθία χαρακτήρων ανήκει στη γλώσσα.

Για την επίλυση των δύο παραπάνω προβλημάτων, η περιγραφή των ιδιοτήτων των στοιχείων της γλώσσας με τον συνήθη τρόπο ορισμού των στοιχείων ενός συνόλου ακόμα και για μια τετριμμένη γλώσσα, καθίσταται πολύπλοκη. Για τον λόγο αυτό χρησιμοποιούνται οι **γραμματικές δομής φράσεως** οι οποίες είναι αποτέλεσμα μελετών των φυσικών γλωσσών.

Ορισμός 1.3.2 *Μια γραμματική δομής φράσεως αποτελείται από:*

1. *Ένα σύνολο τερματικών συμβόλων Τ: πρόκειται για σύμβολα που χρησιμοποιούνται για την δημιουργία των προτάσεων της γλώσσας,*

2. *Ένα σύνολο μη τερματικών συμβόλων Ν: πρόκειται για ενδιάμεσα σύμβολα που συμβάλλουν στην περιγραφή της δομής μιας πρότασης, ένα εκ των οποίων είναι το* **αρχικό σύμβολο**,

3. *Ένα σύνολο γραμματικών κανόνων Ρ που καθορίζουν τον τρόπο δημιουργίας μιας πρότασης.*

Για μια φυσική γλώσσα όπως η Ελληνική, το σύνολο των τερματικών συμβόλων είναι το σύνολο όλων των επιτρεπτών λέξεων της γλώσσας αυτής όπως της Ελληνικής {ουρανός, θάλλασα, γη, καλυπτεται, περιβάλλεται, η, . . .}.

Οι γραμματικοί κανόνες έχουν τη μορφή $x \rightarrow y$ όπου x και y είναι συμβολοσειρές αποτελούμενες από τερματικά και μη σύμβολα. Η εφαρμογή ενός γραμματικού κανόνα έχει ως αποτέλεσμα την αντικατάσταση σε μια συμβολοσειρά του x από το y.

Για την κατασκευή προτάσεων μιας γλώσσας που περιγράφεται από μια γραμματική δομής φράσεως ακολουθούνται οι παρακάτω κανόνες:

1. Αρχίζουμε από το αρχικό σύμβολο, το οποίο αντικαθίσταται από την δεξιά συμβολοσειρά του γραμματικού κανόνα που αφορά το αρχικό σύμβολο.

2. Στη συνέχεια αντικαθιστούμε κάθε μη τερματικό σύμβολο με την συμβολοσειρά που εμφανίζεται στα δεξιά ενός γραμματικού κανόνα,

3. Η συμβολοσειρά τερματικών συμβόλων που προκύπτει από την εφαρμογή των βημάτων 1 και 2 αποτελεί μια πρόταση της γλώσσας.

Ο μηχανισμός σχηματισμού αριθμητικών εκφράσεων όπως θα δούμε και στα επόμενα κεφάλαια είναι από τους ποιό σημαντικούς μηχανισμούς των γλωσσών προγραμματισμού. Ας ορίσουμε για παράδειγμα μια γραμματική δομής φράσεως για την περιγραφή των

αριθμητικών εκφράσεων που περιλαμβάνουν ονόματα μεταβλητών, τους αριθμητικούς τελεστές $+, *$, τον τελεστή $=$ και την δεξιά και αριστερή παρένθεση. Τότε το σύνολο των τερματικών συμβόλων είναι το σύνολο $T = \{A, B, C, +, *, (,), =\}$ και το σύνολο των μη τερματικών συμβόλων είναι το $N = \{asgn\_stat, exp, term, factor, id\}$ και το σύνολο των γραμματικών κανόνων αποτελείται από τους παρακάτω γραμματικούς κανόνες:

$$\begin{aligned} asgn\_stat &\rightarrow id = exp \\ exp &\rightarrow exp + term | term \\ term &\rightarrow term * factor | factor \\ factor &\rightarrow (exp) | id \\ id &\rightarrow A | B | C \end{aligned}$$

Ο τελευταίος γραμματικός κανόνας ορίζει ότι το μη τερματικό σύμβολο id μπορεί να είναι ένα από τα τερματικά σύμβολα-μεταβλητές. Ας δώσουμε ένα παράδειγμα χρήσης της παραπάνω γραμματικής για την παραγωγή μιας έκφρασης. Εφαρμόζοντας τους κανόνες που αναφέραμε παραπάνω έχουμε:

$$\begin{aligned} asgn\_stat &\Rightarrow id = exp && (1.1) \\ &\Rightarrow id = exp + term && (1.2) \\ &\Rightarrow id = exp + term * factor && (1.3) \\ &\Rightarrow id = exp + term * id && (1.4) \\ &\Rightarrow id = exp + term * C && (1.5) \\ &\Rightarrow id = exp + factor * C && (1.6) \\ &\Rightarrow id = exp + id * C && (1.7) \\ &\Rightarrow id = exp + B * C && (1.8) \\ & && (1.9) \end{aligned}$$

$$\Rightarrow \quad id = term + B * C \tag{1.10}$$
$$\Rightarrow \quad id = factor + B * C \tag{1.11}$$
$$\Rightarrow \quad id = id + B * C \tag{1.12}$$
$$\Rightarrow \quad id = A + B * C \tag{1.13}$$
$$\Rightarrow \quad A = A + B * C \tag{1.14}$$

Στην σχέση 1.1 ξεκινώντας από το αρχικό σύμβολο *asgn_stat* εφαρμόζουμε τον πρώτο γραμματικό κανόνα και προκύπτει η συμβολοσειρά *id = exp*. Στην συνέχεια αντικαθιστούμε αρχίζοντας από δεξιά το μη τερματικό σύμβολο *exp* με την συμβολοσειρά *exp+term* και προκύπτει η συμβολοσειρά της σχέσης 1.2. Οι αντικαταστάσεις μη τερματικών συμβόλων από δεξιά προς αριστερά συνεχίζεται μέχρι η τελική συμβολοσειρά να περιέχει μόνο τερματικά σύμβολα. Έτσι από το παραπάνω παράδειγμα προκύπτει η αριθμητική έκφραση 1.14 στην οποία η μεταβλητή A αυξάνεται κατά μια ποσότητα ίση με το γινόμενο των μεταβλητών B και C.

Στους γραμματικούς κανόνες της γραμματικής του παραδείγματος μας η αριστερή συμβολοσειρά είναι ένα μη τερματικό σύμβολο. Μια τέτοια γραμματική ονομάζεται **γραμματική τύπου 2**. Σχεδόν όλες οι γλώσσες προγραμματισμού ορίζονται από γραμματικές τύπου 2. Η γραμματική τύπου 2 ονομάζεται και γραμματική ελεύθερης συμφραζομένων (context-free grammar) διότι στα αριστερά των γραμματικών κανόνων εμφανίζεται μόνο μη τερματικό σύμβολο πράγμα που σημαίνει ότι σε μια οποιαδήποτε συμβολοσειρά κάθε μη τερματικό σύμβολο θα αντικατασταθεί με τη συμβολοσειρά που βρίσκεται στα δεξιά του γραμματικού κανόνα ανεξάρτητα από τα σύμβολα που το περιβάλλουν στη συμβολοσειρά.

Μέχρι τώρα είδαμε πως με τη χρήση των γραμματικών κανόνων μπορούν να σχηματισθούν προτάσεις που να ανήκουν στη γλώσσα. Για τις γλώσσες προγραμματισμού, ο έλεγχος του αν μια ακολουθία χαρακτήρων ανήκει στη γλώσσα παρουσιάζει ιδιαίτερο ενδιαφέρον από πρακτική πλευρά καθώς αποτελεί μέρος της διαδικασίας μεταγλώττισης

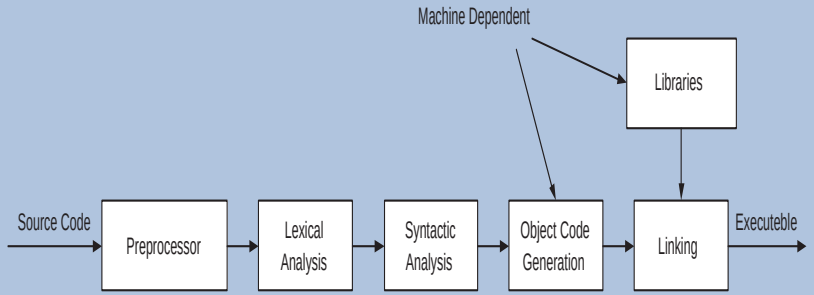
ενός προγράμματος καθώς πρώτα θα πρέπει να ελεγχθεί αν οι προτάσεις του προγράμματος ανήκουν στη χρησιμοποιούμενη γλώσσα προγραμματισμού και μετά να παραχθεί το κώδικας μηχανής που θα εκτελεσθεί τελικά στον υπολογιστή. Για τον έλεγχο του αν μια ακολουθία χαρακτήρων ανήκει στη γλώσσα αντι να χρησιμοποιηθεί η μέθοδος της εξαντλητικής αναζήτησης όλων των γραμματικών κανόνων η οποία είναι μια χρονοβόρα διαδικασία χρησιμοποιούνται διάφοροι αλγόριθμοι που δίνουν απάντηση στο πρόβλημα κατά τρόπο αποτελεσματικό.

Από τη στιγμή που θα ελεγχθεί συντακτικά μια ακολουθία λέξεων της γλώσσας ακολουθεί ο σημασιολογικός έλεγχος. Η σημασιολογία μιας γλώσσας είναι ένα σύνολο κανόνων που καθορίζει το νόημα μιας συντακτικά σωστής ακολουθίας λέξεων δηλαδή το τι επεξεργασία θα προκαλέσει μια ακολουθία στον υπολογιστή. Οι γλώσσες υψηλού επιπέδου διαθέτουν ένα **σύστημα τύπων (type system)** το οποίο κατηγοριοποιεί τις τιμές και τις εκφράσεις σε τύπους και καθορίζει πως μπορεί να επεξεργαστεί αυτούς τους τύπους και πως αυτοί αλληλεπιδρούν μεταξύ τους. Ο τομέας των μαθηματικών που πραγματεύεται τον σχεδιασμό κα τη μελέτη τέτοιων συστημάτων ονομάζεται **type theory**. Έτσι για κάθε τύπο καθορίζεται με σαφήνεια ποιες ακριβώς λειτουργίες μπορούν να εκτελεσθούν και υπονούνται ότι όλες οι άλλες δεν μπορούν. Στις περισσότερες γλώσσες οι λειτουργίες που δεν έχουν νόημα, όπως για παράδειγμα η διαίρεση ενός αλφαριθμητικού με ακέραιο αριθμό εντοπίζονται κατά την μεταγλώττιση του προγράμματος στον σημασιολογικό έλεγχο ενώ σε άλλες εντοπίζονται κατά την εκτέλεση του προγράμματος και γίνεται αντιληπτό μέσω των εξαιρέσεων κατά την εκτέλεση (runtime exceptions).

1.4. Μεταγλώττιση προγράμματος

Όπως προαναφέρθηκε κάθε υπολογιστής διαθέτει τη δική του γλώσσα μηχανής, εντολές της οποίας μπορεί να εκτελεί μόνο. Συνεπώς, ένα πρόγραμμα σε οποιαδήποτε γλώσσα προγραμματισμού υψηλού επιπέδου, όπως είναι η C, για να μπορέσει να εκτελεσθεί πρέπει πρώτα να μεταγλωττισθεί (compilation). Η μεταγλώττιση είναι μια διαδικασία κατά την

οποία ένα πρόγραμμα π.χ σε γλώσσα C, μεταφράζεται στη γλώσσα μηχανής του υπολογιστή στον οποίο πρόκειται να εκτελεσθεί. Ο μεταγλωττιστής (compiler) ο οποίος εκτελεί τη μεταγλώττιση ενός προγράμματος σε γλώσσσα υψηλού επιπέδου σε πρόγραμμα σε γλώσσα μηχανής είναι από τα πιο σημαντικά προγράμματα του λειτουργικού συστήματος ενός υπολογιστή.



Σχήμα 1.2: Φάσεις Μεταγλώττισης.

Στο Σχήμα 1.2 διακρίνονται οι φάσεις της διαδικασίας μεταγλώττισης ενός προγράμματος σε γλώσσα C. Στην πρώτη φάση εκτελούνται όλες οι εντολές του προεπεξεργαστή (preprocessor) και με την ολοκλήρωση της, ο κώδικας που προκύπτει είναι ο κώδικας που θα μεταγλωτισθεί. Στη φάση της λεκτικής ανάλυσης (Lexical Analysis) ο μεταγλωττιστής εντοπίζει όλα τα χρησιμοποιούμενα τερματικά σύμβολα **(token)** που χρησιμοποιούνται από το πρόγραμμα και ελέγχει αν τηρούνται οι κανόνες της γλώσσας για το σχηματισμό τους. Κατά τη συντακτική ανάλυση (Syntax Analysis) ελέγχεται η σύνταξη των προτάσεων-εντολών που χρησιμοποιούνται στο πρόγραμμα.

Στη φάση της δημιουργίας κώδικα μηχανής (object code generation) ο μεταγλωττιστής

εκτελεί τον σημασιολογικό έλεγχο και στη συνέχεια αντιστοιχεί σε κάθε εντολή του αρχικού προγράμματος μια ή περισσότερες εντολές μηχανής. Στην επόμενη φάση, αυτή της διασύνδεσης (Linking) αν το πρόγραμμα χρησιμοποιεί συναρτήσεις κάποιας βιβλιοθήκης (της πρότυπης ή οποιασδήποτε άλλης), τότε ο κώδικας που αντιστοιχεί στις συναρτήσεις αυτές ανασύρεται από την βιβλιοθήκη και διασυνδέεται με τον κώδικα που προέκυψε από τη φάση της δημιουργίας κώδικα μηχανής. Στο τέλος της φάσης διασύνδεσης, ο μεταγλωττιστής παράγει ένα σύνολο εντολών μηχανής οι οποίες μπορούν να εκτελεσθούν και να δώσουν τα αποτελέσματα τα οποία αποτελούν τη λύση του προβλήματος για το οποίο σχεδιάστηκε το αρχικό πρόγραμμα.

Κεφάλαιο 2

Βασικά στοιχεία της Γλώσσας C

2.1. Δομή ενός προγράμματος στη C

Ένα πρόγραμμα σε C συνίσταται από μια ή περισσότερες συναρτήσεις. Η μια από αυτές είναι η συνάρτηση *main* η οποία είναι η κύρια συνάρτηση ενός προγράμματος σε C που σημαίνει ότι η εκτέλεση του προγράμματος ξεκινά από τη συνάρτηση *main* και κάποιες ή όλες οι υπόλοιπες συναρτήσεις καλούνται μέσα από αυτή.

Οι συναρτήσεις ενός προγράμματος διακρίνονται στις εξής κατηγορίες:

- σε αυτές της πρότυπης βιβλιοθήκης (standard lib) της ANSI C και
- σε αυτές που ο χρήστης δημιουργεί (user defined).

Για να γίνει χρήση κάποιων συναρτήσεων από βιβλιοθήκη του περιβάλλοντος προγραμματισμού σε ένα πρόγραμμα C, θα πρέπει να συμπεριληφθούν τα αρχεία επικεφαλίδες (τα οποία φέρουν την επέκταση .h) στα οποία είναι δηλωμένες οι συναρτήσεις αυτές. Αυτό πραγματοποιείται με την εντολή #include του προεπεξεργαστή (preprocessor) της C.

Η εντολές συμπερίληψης αρχείων επικεφαλίδας σε ένα πρόγραμμα C εμφανίζονται στην αρχή ενός προγράμματος C. Στον πίνακα 2.1 παρουσιάζεται ένα μικρό σύνολο αρχείων επικεφαλίδας που διαθέτει η ANSI C καθώς και περιγραφή του περιεχομένου τους.

Αρχείο	**Περιγραφή Συναρτήσεων**
stdarg.h	Διαχείρισης Μεταβλητών Λιστών Ορισμάτων
io.h	Διαχείρισης εισόδου / εξόδου σε χαμηλό επίπεδο
limits.h & float.h	Ορίζει τα όρια των ακεραίων και των αριθμών κινητής υποδιαστολής.
math.h	Μαθηματικές Συναρτήσεις
mem.h	Διαχείριση Δυναμικής Μνήμης
stdio.h	Διαχείριση εισόδου και εξόδου σε υψηλό επίπεδο
stdlib.h	Συχνά χρησιμοποιούμενες συναρτήσεις
string.h & ctype.h	Συναρτήσεις Επεξεργασίας αλφαριθμητικών και χαρακτήρων
time.h	Διαχείρισης ώρας και ημερομηνίας και επεξεργασίας χρόνου

Πίνακας 2.1: Βασικά Αρχεία Επικεφαλίδας

Στο Σχήμα 2.1 δίνεται ένα πρόγραμμα σε C το οποίο όταν εκτελεσθεί τυπώνεται στην οθόνη το κείμενο *This is my first code!!!*. Το πρόγραμμα αυτό αποτελείται μόνο από την συνάρτηση *main* το σώμα της οποίας ορίζεται από το αριστερό άγκιστρο και ολοκληρώνεται με το δεξί άγκιστρο. Στο εσωτερικό της *main* γίνεται κλήση της συνάρτησης *printf* της πρότυπης βιβλιοθήκης με όρισμα το κείμενο "This is my first code!!!" (γραμμή με αριθμό 3 του προγράμματος). Η συνάρτηση *printf* της πρότυπης βιβλιοθήκης για να χρησιμοποιηθεί στο εν λόγω πρόγραμμα πρέπει να προηγηθεί η ενσωμάτωση του αρχείου επικεφαλίδα *stdio.h* στο οποίο είναι δηλωμένη (γραμμή με αριθμό 0 στο πρόγραμμα) και

το οποίο περιβάλλεται από τα σύμβολα <>.

```
0:  #include <stdio.h>
1:
2:  int main(){
3:      printf("This is my first code!!!\n");
4:      return 0;
5:  }
```

Σχήμα 2.1: Ο πρώτος μου κώδικας.

Το πρόγραμμα του Σχήματος 2.1 αποθηκεύεται σε αρχείο με επέκταση .c π.χ. myFirstCode.c. Θεωρώντας ότι το λειτουργικό σύστημα της μηχανής μας είναι το UNIX, ο βασικός μεταγλωττιστής είναι ο cc και για να μεταγλωττισθεί το πρόγραμμα του Σχήματος 2.1 μπορούμε να γράψουμε σε περιβάλλον γραμμής εντολής *#cc myfirstCode.c* με το αποτέλεσμα της μεταγλώττισης να είναι το δυαδικό αρχείο myfirstCode το οποίο είναι άμεσα εκτελέσιμο από τη μηχανή.

2.2. Βασικοί τύποι δεδομένων

Η C διαθέτει τέσσερεις βασικούς τύπους δεδομένων:

- **char**: πρόκειται για ακεραίους μεγέθους ενός byte που σημαίνει ότι μπορεί να αναπαρασταθούν το πολύ 256 χαρακτήρες. Ο τύπος αυτός είναι ικανός να αναπαραστήσει τους ASCII χαρακτήρες που στο σύνολο τους είναι 128.

- **int**: πρόκειται για ακεραίους μεγέθους μεγαλύτερου του ενός byte και μικρότερου αυτού που μπορεί να αναπαρασταθεί από το μηχάνημα.

- **float**: αριθμοί κινητής υποδιαστολής απλής ακρίβειας
- **double**: αριθμοί κινητής υποδιαστολής διπλής ακρίβειας

Μια σημαντική παράμετρος, όπως θα δούμε στα επόμενα κεφάλαια, ενός τύπου δεδομένων στη C είναι το μέγεθος του που είναι το πλήθος των byte που καταλαμβάνει ένα δεδομένο αυτού του τύπου. Το μέγεθος σε byte για κάθε τύπο μπορούμε να το υπολογίσουμε με τη βοήθεια του τελεστή μεγέθους sizeof τυλίγοντας το όνομα του τύπου με παρενθέσεις δηλαδή sizeof(<όνομα_τύπου>). Για παράδειγμα για να υπολογίσουμε το μέγεθος του τύπου float γράφουμε sizeof(float). Τα μεγέθη των βασικών τύπων ορίζονται ως σταθερές στα αρχεία < *limits.h* > και < *float.h* >. Συνεπώς, σε κάθε πρόγραμμα στο οποίο πρόκειται να χρησιμοποιήσουμε τις σταθερές αυτές πρέπει στην αρχή να συμπεριλάβουμε τα παραπάνω αρχεία επικεφαλίδες δηλαδή:

#include < *limits.h* >
#include < *float.h* >

Η C διαθέτει ένα σύνολο προσδιοριστών οι οποίοι εφαρμόζονται στους βασικούς τύπους. Έτσι για τον τύπο int υπάρχουν οι προσδιοριστές short και long. Για παράδειγμα γράφοντας short int ή short ορίζεται ένας μικρού μεγέθους ακέραιος και long int ή long ορίζεται ένας μεγάλου μεγέθους ακέραιος. Σε κάθε υπολογιστικό σύστημα αυτό που ισχύει είναι $sizeof(short) \leq sizeof(int) \leq sizeof(long)$. Στους περισσότερους προσωπικούς υπολογιστές το μέγεθος του int και long είναι 4 byte (32-bit) ενώ του short είναι 2 byte (16-bit).

Επίσης στους περισσότερους προσωπικούς υπολογιστές το μέγεθος ενός αριθμού κινητής υποδιαστολής απλής ακρίβειας (float) είναι 4 byte (32-bit) και ενός διπλής ακρίβειας (double) είναι 8 byte (64-bit). Το πρότυπο που χρησιμοποιείται για την αναπαράσταση των αριθμών κινητής υποδιαστολής είναι το 754 της IEEE (Institute of Electrical and Electronics Engineers). Μπορεί να χρησιμοποιηθεί ο προσδιοριστής long για τον τύπο double, δηλαδή έχουμε έναν τύπο long double, προκειμένου να αυξήσουμε το μέγεθος

και κατ' επέκταση την ακρίβεια ενός αριθμού κινητής υποδιαστολής. Βέβαια θα πρέπει να υποστηρίζει και η μηχανή μεγαλύτερη ακρίβεια διαφορετικά ο τύπος long double εκφυλίζεται ουσιαστικά στον double.

Δύο ακόμα προσδιοριστές είναι οι signed (προσημασμένος) unsigned (μη προσημασμένος) οι οποίοι μπορούν να εφαρμοστούν μόνο στον τύπο char και σε όλους τους ακεραίους αφού σύμφωνα με το πρότυπο 754 της IEEE όλοι οι αριθμοί κινητής υποδιαστολής είναι προσημασμένοι. Οι προσδιοριστές αυτοί καθορίζουν αν ένας αριθμός είναι προσημασμένος ή όχι. Ο τύπος char είναι ένας προσημασμένος ακέραιος των 8-bit πράγμα που σημαίνει ότι το πιο σημαντικό bit αναπαριστά το πρόσημο του, το 1 αντιστοιχεί στο − και το 0 στο +. Για παράδειγμα ο 8-bit ακέραιος **10000001** αναπαριστά το -1 (το περισσότερο σημαντικό ψηφίο δεν λογίζεται στον υπολογισμό του μέτρου του αριθμού) και με τον προσδιοριστή unsigned αναπαριστά το 128 αφού το περισσότερο σημαντικό ψηφίο λογίζεται πλέον στο μέτρο του αριθμού, δηλαδή $1\times2^7+1\times2^0=128$. Με άλλα λόγια ο τύπος unsigned char καλύπτει τους ακέραιους από $0\ldots255$ ενώ ο προσημασμένος από $-128\ldots127$. Οι προσδιοριστές αυτοί εφαρμόζονται με τον ίδιο τρόπο και σε όλες τις εκδόσεις του τύπου int.

Στη C αν θέλουμε να μετονομάσουμε έναν ήδη υπάρχοντα τύπο δεδομένων μπορούμε να χρησιμοποιήσουμε τη λέξη κλειδί **typedef** ως εξής:

$$typedef \ < newTypeName > < oldTypename >;$$

Στο τέλος της παραπάνω εντολής μετονομασίας τίθεται ο χαρακτήρας **;** που στη C είναι ο χαρακτήρας που δηλώνει το τέλος μιας εντολής. Για παράδειγμα αν θέλουμε να μετονομάσουμε το τύπο int σε integer γράφουμε:

$$typedef \ integer \ int;$$

ενώ για δημιουργήσουμε τον τύπο byte για την αναπαράσταση των ακεραίων από 0 . . . 255 γράφουμε:

typedef byte unsigned char;

2.3. Μεταβλητές

Η C είναι μια case-sensitive γλώσσα που σημαίνει ότι διακρίνει τα πεζά από τα κεφαλαία γράμματα. Το όνομα μιας μεταβλητής πρέπει να αρχίζει πάντα με γράμμα και μπορούν να χρησιμοποιηθούν: αλφαριθμητικοί χαρακτήρες καθώς και ο χαρακτήρας υπογράμμισης _ ο οποίος καλό είναι να αποφεύγεται στην αρχή του ονόματος μιας μεταβλητής καθώς χρησιμοποιείται από συναρτήσεις της πρότυπης βιβλιοθήκης.

Μια καλή τακτική στη C είναι να χρησιμοποιούνται πεζά γράμματα για τις μεταβλητές και το όνομα τους να είναι σχετικό με το σκοπό για τον οποίο προορίζονται. Ένας ακόμα περιορισμός στο σχηματισμό ονομάτων των μεταβλητών είναι ότι δεν επιτρέπεται η χρήση των λέξεων κλειδιών του Πίνακα 2.2 οι οποίες ονομάζονται δεσμευμένες λέξεις (reserved words).

Λέξεις - Κλειδιά	**Περιγραφή**
int, char, double, float	Ορίζουν τους βασικούς τύπους Δεδομένων
long, short, signed, unsigned, register, static, extern, auto, void	Τροποποιούν το μέγεθος και το είδος των δεδομένων
const, volatile	Καθορίζουν αν ένα δεδομένο μπορεί να μεταβληθεί ή είναι σταθερά
break, case, continue, switch, default, do, while, for, if, else, return, goto	Εντολές
typedef, enum	Δημιουργούν νέα ονόματα για υπάρχοντες τύπους δεδομένων
struct, union	Ομαδοποιούν μεταβλητές διαφορετικών τύπων σε ένα σύνολο

Πίνακας 2.2: Λέξεις - Κλειδιά που είναι δεσμευμένες

Κάθε μεταβλητή πριν τη χρήση της πρέπει πρώτα να δηλωθεί και προαιρετικά να αρχικοποιηθεί με μια συγκεκριμένη τιμή. Έτσι τυπικά η δήλωση μιας μεταβλητής έχει ως εξής:

$$< typename > < variable\_name >=< value >; \quad (2.1)$$

όπου $< typename >$ είναι ο τύπος της μεταβλητής και $< variable\_name >$ είναι το όνομα της μεταβλητής σύμφωνα με τους κανόνες που προαναφέρθηκαν και $< value >$ είναι η αρχική τιμή που θέτουμε στη μεταβλητή. Στη δήλωση 2.1 εμφανίζεται ο τελεστής ανάθεσης = όπου στα αριστερά του βρίσκεται η μεταβλητή στην οποία θα ανατεθεί η τιμή

που βρίσκεται στα δεξιά του τελεστή. Στο τέλος κάθε δήλωσης τίθεται ο χαρακτήρας **;** που στη C, όπως προαναφέρθηκε είναι ο χαρακτήρας που δηλώνει το τέλος μιας εντολής.

Για παράδειγμα ο μεταγλωττιστής κατά τη μεταγλώττιση μιας δήλωσης της μορφής:

$$int\ count = 5;$$

δεσμεύει τέσσερα byte μνήμης (μέγεθος ενός ακεραίου τύπου int) για τη μεταβλητή count στα οποία αναθέτει την αρχική τιμή 00000000 00000000 00000000 00000101 που είναι ο δεκαδικός αριθμός 5. Σε μια δήλωση μπορούμε να δηλώσουμε περισσότερες από μια μεταβλητές όπως:

$$double\ percent = 28.5,\ rate = 5.6;$$

όπου δηλώνονται οι μεταβλητές κινητής υποδιαστολής διπλής ακρίβειας percent και rate με αρχικές τιμές 28.5 και 5.6 αντίστοιχα.

Μια μεταβλητή από τη στιγμή που θα δηλωθεί μπορεί να χρησιμοποιηθεί στο πρόγραμμα απλά χρησιμοποιώντας το όνομα της σε εντολές και σε αριθμητικές και λογικές παραστάσεις. Η απλούστερη εντολή είναι αυτή της ανάθεσης τιμής σε μεταβλητή. Για παράδειγμα η εκτέλεση της εντολής ανάθεσης

$$count = 5;$$

θα έχει ως αποτέλεσμα την ανάθεση του ακεραίου 5 στη μεταβλητή count.

2.4. Σταθερές

Στη C, όπως και σε κάθε γλώσσα προγραμματισμού, ο προγραμματιστής μπορεί να ορίσει και σταθερές η τιμή των οποίων δεν αλλάζει καθ' όλη τη διάρκεια εκτέλεσης του προγράμματος. Ακολουθούνται οι ίδιοι κανόνες ονοματολογίας που αναφέρθηκαν και για

τις μεταβλητές με τη διαφορά ότι το όνομα μιας σταθεράς συντίθεται μόνο από κεφαλαία γράμματα. Υπάρχουν δύο κατηγορίες σταθερών στη C:

- οι **literal**:στην εντολή ανάθεσης $\pi = 3.14159$ η literal σταθερά είναι ο αριθμός κινητής υποδιαστολής 3.14159.
- οι **συμβολικές**, όπου στη literal σταθερά 3.14159 αντιστοιχούμε το συμβολικό όνομα PI το οποίο και χρησιμοποιούμε στο πρόγραμμα. Υπάρχουν 2 τρόποι στη C για να ορισθεί μια τέτοια αντιστοιχία:

  1. με τον προσδιοριστή **const**: *const float PI* = 3.14159;
  2. Μέσω της εντολής του προεπεξεργαστή **#define**:

     #define PI 3.14159

     Στις αριθμητικές παραστάσεις χρησιμοποιείται πλέον το συμβολικό όνομα PI. Όταν είναι επιθυμητή η αναίρεση μιας συμβολικής σταθεράς αυτό μπορεί να γίνει με τη χρήση της εντολής του προεπεξεργαστή **#undef**. Έτσι για να αναιρέσουμε την σταθερά PI γράφουμε:

     #undef PI

Μια ακέραια σταθερά εξ ορισμού είναι τύπου int και προσημασμένη (signed) εκτός και αν είναι μεγάλος ακέραιος αριθμός οπότε εκλαμβάνεται ως **long**. Αν είναι επιθυμητό για τις ανάγκες του κώδικα μια μικρή ακέραια σταθερά να εκληφθεί από τον μεταγλωττιστή ως long ή μη προσημασμένη θα πρέπει να έχει ως κατάληξη το L ή U, π.χ. 2048L ή 2048U ή 2048UL. Μια ακέραια σταθερά επίσης μπορεί να αναπαρασταθεί στο οκταδικό ή στο δεκαεξαδικό χρησιμοποιώντας το πρόθεμα 0 και 0x ή 0X αντίστοιχα, π.χ. $077 = (63)_{10}$ και $0x2f = (63)_{10}$.

Οι σταθερές κινητής υποδιαστολής περιέχουν μια υποδιαστολή (π.χ. 23.45) ή μια δύναμη (2e-3) (που είναι ο 2×10^{-3}) ή και τα δύο και ο τύπος τους είναι double εκτός και αν υπάρχει η κατάληξη:

- F ή f οπότε υποδηλώνεται μια σταθερά float
- L ή l οπότε υποδηλώνεται μια σταθερά double.

Μια σταθερά τύπου χαρακτήρα είναι ένας χαρακτήρας μέσα σε μονά εισαγωγικά π.χ. ο $'x'$. Κατά βάση, η σταθερά χαρακτήρα είναι ο $8-bit$ ASCII αριθμός που αντιστοιχεί στον χαρακτήρα που εμπεριέχεται στα μονά εισαγωγικά δηλαδή γράφοντας $'x'$ ουσιαστικά υποννοείται ο ακέραιος 120 που αντιστοιχεί στον χαρακτήρα x στον ASCII πίνακα. Αυτό σημαίνει ότι οι σταθερές χαρακτήρα όπως και οι μεταβλητές χαρακτήρα είναι ακέραιοι αριθμοί και κατ' επέκταση μπορούν να λάβουν μέρος σε αριθμητικές πράξεις όπως θα δούμε στη συνέχεια.

Στον Πίνακα 2.3 εμφανίζεται μέρος του ASCII πίνακα που περιλαμβάνει ένα υποσύνολο των εκτυπώσιμων χαρακτήρων. Στη C, ο χαρακτήρας \ λέγεται χαρακτήρας διαφυγής και έχει έναν ειδικό ρόλο. Θα εξηγήσουμε την λειτουργία του με ένα παράδειγμα. Η σταθερά $'n'$ αντιστοιχεί στο λατινικό γράμμα n ενώ η σταθερά $'\n'$ αντιστοιχεί στον χαρακτήρα αλλαγής γραμμής του πίνακα ASCII με ASCII αριθμό 10. Με άλλα λόγια ο χαρακτήρας \ όταν προηγείται ενός χαρακτήρα αλλάζει την ερμηνεία του χαρακτήρα αυτού. Ένα άλλο παράδειγμα είναι οι χαρακτήρες $'\t'$ και $'\ '$ που αντιστοιχούν στον οριζόντιο στηλογνώμονα και στον κενό χαρακτήρα του πίνακα ASCII αντίστοιχα. Τέλος, ένας άλλος πολύ σημαντικός χαρακτήρας είναι ο $'\0'$ που είναι ο χαρακτήρας τερματισμού αλφαριθμητικού. Ο κενός χαρακτήρας, ο χαρακτήρας αλλαγής γραμμής, οι στηλογνώμονες και ο χαρακτήρας τερματισμού αλφαριθμητικού ονομάζονται **λευκοί χαρακτήρες**.

Μια αλφαριθμητική σταθερά (string constant) είναι μια ακολουθία από μηδέν ή περισσότερους χαρακτήρες εντός διπλών εισαγωγικών, π.χ. "Hello World" ή "" που είναι το κενό string. Στο τέλος ενός αλφαριθμητικού υπάρχει ο χαρακτήρας $'\0'$ που δηλώνει το

τέλος του, δηλαδή η εσωτερική αναπαράσταση του “” είναι “\0” και του “Hello World” είναι η “Hello World\0”.

Πεζά Γράμματα	Κεφαλαία Γράμματα	Δεκαδικά Ψηφία
97(a)	65(A)	48(0)
98(b)	66(B)	49(1)
99(c)	67(C)	50(2)
100(d)	68(D)	51(3)
101(e)	69(E)	52(4)
102(f)	70(F)	53(5)
103(g)	71(G)	54(6)
104(h)	72(H)	55(7)
105(i)	73(I)	56(8)
106(j)	74(J)	57(9)
107(k)	75(K)	
108(l)	76(L)	
109(m)	77(M)	
110(n)	78(N)	
111(o)	79(O)	
112(p)	80(P)	
113(q)	81(Q)	
114(r)	82(R)	
115(s)	83(S)	
116(t)	84(T)	
117(u)	85(U)	
118(v)	86(V)	
119(w)	87(W)	
120(x)	88(X)	
121(y)	89(Y)	
122(z)	90(Z)	

Πίνακας 2.3: ASCII κωδικοί για τα γράμματα του λατινικού αλφαβήτου και των δεκαδικών ψηφίων.

2.5. Τελεστές

Στο παρόν εδάφιο θα αναλύσουμε την λειτουργία των τελεστών που διαθέτει η C καθώς και την χρήση τους στον σχηματισμό εκφράσεων.

2.5.1. Αριθμητικοί Τελεστές

Η C διαθέτει επτά συνολικά αριθμητικούς τελεστές με τους οποίους μπορεί να σχηματισθούν αριθμητικές εκφράσεις. Από αυτούς δύο είναι μοναδιαίοι τελεστές: $++, --$ οι οποίοι μπορεί να βρίσκονται είτε δεξιά είτε αριστερά μιας μεταβλητής. Έτσι όταν εκτελείται μια έκφραση της μορφής $x++$ αυτό που πρακτικά γίνεται είναι να αυξηθεί η μεταβλητή x κατά 1 δηλαδή οι εκφράσεις $x++$ και $++x$ είναι ισοδύναμες με την $x = x + 1$. Όμοια και οι $x--$ και $--x$ είναι ισοδύναμες με την $x = x - 1$.

Είναι εύλογο να αναρωτηθεί κανείς ποιά η διαφορά μεταξύ των εκφράσεων $x++$ και της $++x$. Η διαφορά είναι ότι όταν εκτελείται η $x++$ πρώτα επιστρέφεται η τρέχουσα τιμή της x και μετά αυξάνεται η μεταβλητή κατά ένα. Ενώ όταν εκτελείται η $++x$ πρώτα αυξάνεται η μεταβλητή κατά 1 και στη συνέχεια επιστρέφεται η νέα τιμή της μεταβλητής. Για να το κατανοήσουμε καλύτερα ας δούμε τον παρακάτω κώδικα:

$$int\ z, y,\ x = 5;$$

$$y = ++x;$$
$$z = x++;$$

Μετά την εκτέλεση των δύο εκφράσεων η μεταβλητή y θα έχει την τιμή 6, η z την τιμή 6 και η x θα έχει την τιμή 7.

Οι υπόλοιποι πέντε τελεστές: +, -, /, *, % είναι δυαδικοί τελεστές. Αξίζει να τονισθεί ότι ο τελεστής / ορίζει την πράξη της ακέραιας διαίρεσης ενώ ο τελεστής % ορίζει την πράξη modulo δηλαδή η έκφραση $x\%y$ επιστρέφει το υπόλοιπο της διαίρεσης της μεταβλητής x με τη μεταβλητή y.

Σε μια αριθμητική έκφραση οι μοναδιαίοι τελεστές έχουν τη μεγαλύτερη προτεραιότητα. Ακολουθούν οι τελεστές *,/,% με την ίδια προτεραιότητα ενώ τελευταίοι είναι οι τελεστές + και -. Σε περίπτωση που σε μια έκφραση εμπλέκονται περισσότεροι από έναν τελεστές με την ίδια προτεραιότητα τότε η εκτέλεση των πράξεων γίνεται από αριστερά προς τα δεξιά. Οι παραπάνω προτεραιότητες μπορούν να αλλάξουν με την εισαγωγή παρενθέσεων.

Ας δούμε μερικά παραδείγματα:

- $8\%3*6$. Δεδομένου ότι ο % και ο * έχουν την ίδια προτεραιότητα, ο υπολογισμός θα γίνει από αριστερά προς τα δεξιά, δηλαδή το αποτέλεσμα εκτέλεσης της έκφρασης είναι το 12.
- 3+4*6. Επειδή ο πολλαπλασιασμός έχει μεγαλύτερη προτεραιότητα από αυτόν της πρόσθεσης η έκφραση επιστρέφει 27.
- (3+4)*6. Με την εισαγωγή των παρενθέσεων αλλάζει η προτεραιότητα, οπότε πρώτα εκτελούνται οι πράξεις εντός των παρενθέσεων και μετά ο πολλαπλασιασμός δηλαδή η έκφραση αυτή επιστρέφει 42.
- $w*x/y*z$. Επειδή οι τελεστές * και / έχουν την ίδια προτεραιότητα, οι πράξεις θα εκτελεσθούν από αριστερά προς τα δεξιά, δηλαδή $((w*x)/y)*y$.
- $w*x/y+z/y$. Η έκφραση αυτή αποτελείται από δύο υπο εκφράσεις. Η μια είναι αυτή που βρίσκεται αριστερά του τελεστή + που έχει την μικρότερη προτεραιότητα και η δεύτερη είναι αυτή που βρίσκεται στα δεξιά του. Επειδή και στις δύο υποεκφράσεις οι τελεστές έχουν την ίδια προτεραιότητα τότε η εκτέλεση των πράξεων θα γίνει από αριστερά προς τα δεξιά. Δηλαδή, η ισοδύναμη έκφραση είναι η $((w*x)/y)+(z/y)$.

2.5.2. Τελεστές Μετατόπισης

Η C διαθέτει δύο τελεστές μετατόπισης, $\ll, \gg$ που μετατοπίζουν τα bit μιας ακέραιας μεταβλητής αριστερά ή δεξιά ορισμένο αριθμό θέσεων. Η έκφραση $x \gg n$ ($x \ll n$) μετατοπίζει τα bit της μεταβλητής x, n θέσεις δεξιά (αριστερά) θέτοντας 0 στις n θέσεις αριστερά (δεξιά) και επιστρέφεται το αποτέλεσμα χωρίς να αλλάξει η μεταβλητή x. Η προτεραιότητα των τελεστών μετατόπισης είναι μικρότερη από αυτή των αριθμητικών τελεστών.

Έστω ότι $x = (12)_{10} = (00001100)_2$ και $n = 2$ τότε η έκφραση $x \gg n$ (δεξιά μετατόπιση) μετατοπίζει την τιμή της μεταβλητής x δύο θέσεις δεξιά θέτοντας 0 στις δύο περισσότερο σημαντικές θέσεις και επιστρέφει $(00000011)_2 = (3)_{10}$. Αν τώρα $n = 3$ τότε η έκφραση $x \ll n$ μετατοπίζει την τιμή της μεταβλητής x δύο θέσεις αριστερά θέτοντας 0 στις δύο λιγότερο σημαντικές θέσεις και επιστρέφει $(01100000)_2 = (96)_{10}$ ενώ η έκφραση $x \gg n$ επιστρέφει $(00000001)_2 = (1)_{10}$. Να τονισθεί ότι η μεταβλητή x δεν αλλάζει τιμή, δηλαδή εξακολουθεί η τιμή της να είναι $(12)_{10} = (00001100)_2$.

Κάτω από ειδικές συνθήκες, οι τελεστές μετατόπισης μπορούν να χρησιμοποιηθούν για τον πολλαπλασιασμό και την διαίρεση ενός ακεραίου με δυνάμεις του 2. Αν κατά την αριστερή μετατόπιση κατά n θέσεις, δεν χάνονται bit από το περισσότερο σημαντικό μέρος του αριθμού τότε ο ακέραιος έχει πολλαπλασιασθεί με το 2^n. Η δεξιά μετατόπιση κατά n θέσεις είναι η ακέραια διαίρεση, όπου το κλασματικό μέρος αποκόπτεται. Για παράδειγμα μετατοπίζοντας το 7 μια θέση δεξιά προκύπτει το 3 που είναι το αποτέλεσμα της ακέραιας διαίρεσης με το 2.

Τι γίνεται όμως όταν οι ακέραιοι είναι προσημασμένοι, σε μια δεξιά και σε μια αριστερή μετατόπιση; Υπάρχουν δύο κατηγορίες μετατοπίσεων:

- Η **λογική μετατόπιση** όπου οι αριθμοί αντιμετωπίζονται ως μη προσημασμένοι.
- Η **αριθμητική μετατόπιση**, όπου σε κάθε αριστερή μετατόπιση το bit προσήμου διατηρείται και σε κάθε δεξιά μετατόπιση στις περισσότερο σημαντικές θέσεις εισάγεται 0 (για θετικούς) ή 1 (για αρνητικούς) ανάλογα με την τιμή του bit προσήμου.

2.5.3. Συσχετιστικοί Τελεστές

Για τη σύγκριση των αποτελεσμάτων αριθμητικών και λογικών εκφράσεων, η C διαθέτει έξι τελεστές συγκρισης οι οποίοι εμφανίζονται στον Πίνακα 2.4. Για κάθε ένα από αυτούς το αποτέλεσμα της έκφρασης που βρίσκεται στα αριστερά του τελεστή σύγκρισης συγκρίνεται με το αποτέλεσμα της έκφρασης που βρίσκεται στα δεξιά του και επιστρέφεται το αποτέλεσμα της σύγκρισης: 1 αν ικανοποιείται η σχέση και 0 αν δεν ικανοποιείται.

Τελεστής	Σύμβολο	Υποβαλλόμενη ερώτηση	Παράδειγμα
Ισότητα	==	Είναι η αριστερή έκφραση ίση με την δεξιά?	x==y
Μεγαλύτερο	>	Είναι η αριστερή έκφραση μεγαλύτερη από τη δεξιά?	x>y
Μικρότερο	<	Είναι η αριστερή έκφραση μικρότερη από τη δεξιά?	x<y
Μεγαλύτερο ή ίσο	>=	Είναι η αριστερή έκφραση μεγαλύτερη ή ίση από τη δεξιά?	x>=y
Μικρότερο ή ίσο	<=	Είναι η αριστερή έκφραση μικρότερη ή ίση από τη δεξιά?	x<=y
Διάφορο	! =	Είναι η αριστερή έκφραση διαφορετική από τη δεξιά?	x! =y

Πίνακας 2.4: Συσχετιστικοί Τελεστές

Οι τελεστές $<, <=, >, >=$ έχουν μεγαλύτερη προτεραιότητα έναντι των ! = και ==. Για παράδειγμα η έκφραση $x == y > z$ είναι ισοδύναμη με την $x == (y > z)$. Επίσης, οι συσχετιστικοί τελεστές έχουν μικρότερη προτεραιότητα έναντι των αριθμητικών τελεστών και των τελεστών μετατόπισης. Για παράδειγμα, η έκφραση $x + 6 > y$ είναι ισοδύναμη με

την έκφραση $(x + 6) > y$.

Επίπεδο	Τελεστής
1	() [ ] → .
2	! ~ + + − − ∗(τελεστής έμμεσης αναφοράς) (*cast*) &(διεύθυνση αντικειμένου) sizeof +(μοναδιαίος) -(μοναδιαίος)
3	∗(πολλαπλασιασμός) / %
4	+ −
5	≪ ≫
6	> < >= <=
7	== ! =
8	& (Λογικό ΚΑΙ)
9	^
10	|
11	&&
12	||
13	? :
14	= + = − = ∗ = / = % = &=^= | = ≪= ≫=
15	,

Πίνακας 2.5: Προτεραιότητες Τελεστών

2.5.4. Λογικοί Τελεστές bit

Για την επεξεργασία μεμονωμένων bit σε ακεραίους, η C διαθέτει τρεις δυαδικούς λογικούς τελεστές bit:

- & : Σύζευξη

- | : Διάζευξη
- ^: Αποκλειστικό Ή

Έστω οι ακέραιοι $(240)_{10} = (11110000)_2$ και $(85)_{10} = (01010101)_2$ και ότι θέλουμε να σχηματίσουμε εκφράσεις με την χρήση των παραπάνω τελεστών, τότε

1. 11110000 & 01010101 = 01010000 (240 & 85 = 80)
2. 11110000 | 01010101 = 11110101 (240 | 85 = 245)
3. 11110000^01010101 = 10100101 (240^85 = 165)

Τέλος, η C διαθέτει ένα μοναδιαίο λογικό τελεστή bit τον ~ που δεν είναι άλλος από τον τελεστή του συμπληρώματος δηλαδή χρησιμοποιείται για να αντιστρέψει τα bit της μεταβλητής στην οποία εφαρμόζεται. Για παράδειγμα αν $n = (254)_{10} = (11111110)_2$, τότε η έκφραση $\sim n$ επιστρέφει $(1)_{10} = (00000001)_2$.

Ο τελεστής ~ έχει τη μεγαλύτερη προτεραιότητα και μάλιστα έχει την ίδια προτεραιότητα με τους μοναδιαίους αριθμητικούς τελεστές. Ο & έχει μεγαλύτερη προτεραιότητα από τον^ο οποίος έχει μεγαλύτερη προτεραιότητα από τον |. Και οι τρεις έχουν μικρότερη από αυτήν των αριθμητικών τελεστών, των τελεστών μετατόπισης και των συσχετιστικών τελεστών. Στον Πίνακα 2.5 διακρίνονται τα επίπεδα προτεραιότητας των τελεστών στη C.

2.5.5. Λογικοί Τελεστές

Για τον σχηματισμό λογικών εκφράσεων, η C διαθέτει τρεις λογικούς τελεστές οι οποίοι παρουσιάζονται στον Πίνακα 2.6. Εξ ορισμού, η αριθμητική τιμή μιας συσχετιστικής ή λογικής έκφρασης είναι 1 όταν αυτή είναι αληθής και 0 όταν είναι ψευδής. Στο επόμενο κεφάλαιο θα εξετάσουμε την χρήση των τελεστών αυτών στον σχηματισμό λογικών εκφράσεων στις εντολές επανάληψεις και απόφασης.

Ο τελεστής ! έχει προτεραιότητα ίση με αυτή των μοναδιαίων αριθμητικών τελεστών. Ο τελεστής && έχει μεγαλύτερη προτεραιότητα έναντι του ||. Και οι δύο τελεστές έχουν μικρότερη προτεραιότητα έναντι των αριθμητικών, των τελεστών μετατόπισης, των λογικών τελεστών bit και των συσχετιστικών τελεστών (επίπεδα 11 και 12 στον Πίνακα 2.5). Η λογική έκφραση για παράδειγμα $a < b \,||\, b < c \,\&\&\, c < d$ είναι ισοδύναμη με την έκφραση $(a < b) \,||\, ((b < c) \,\&\&\, (c < d))$.

Τελεστής	Σύμβολο	Παράδειγμα
ΚΑΙ	&&	expr1 && expr2
Ή	||	expr1 || expr2
ΟΧΙ	!	!expr

Πίνακας 2.6: Λογικοί Τελεστές

2.5.6. Εκφράσεις υπο συνθήκη

Κατά την εκτέλεση υπολογισμών είναι συχνό το φαινόμενο να θέλουμε να υπολογίσουμε δύο διαφορετικές εκφράσεις ανάλογα με το αν ικανοποιείται η όχι μια συνθήκη. Η C διαθέτει τον τριαδικό τελεστή ? : **(τελεστής συνθήκη)** για τον υπο συνθήκη υπολογισμό εκφράσεων.

Η χρήση του τελεστή συνθήκη για την υπο συνθήκη εκτέλεση εκφράσεων έχει ως εξής:

$$< \textbf{cond} >? < \textbf{expr1} >:< \textbf{expr2} >$$

όπου αν ικανοποιείται η συνθήκη $< cond >$ τότε υπολογίζεται η $< expr1 >$ διαφορετικά υπολογίζεται η $< expr2 >$. Ένα πολύ απλό παράδειγμα είναι η ανάθεση στη μεταβλητή z της μεγαλύτερης εκ των μεταβλητών a και b, δηλαδή $z = (a > b) \; ? \; a \; : \; b$, όπου όταν

ικανοποιείται η συνθήκη ($a > b$) η τιμή της μεταβλητής a ανατίθεται στη μεταβλητή z διαφορετικά ανατίθεται η τιμή της b. Η προτεραιότητα του τελεστή αυτού όπως φαίνεται και στον Πίνακα 2.5 είναι μικρότερη από όλους τους προαναφερθέντες τελεστές.

2.5.7. Τελεστές Αντικατάστασης και ο Τελεστής κόμμα

Στην C, υπάρχει μια ειδική κατηγορία τελεστών ανάθεσης αυτής των **τελεστών αντικατάστασης**. Ας συμβολίσουμε τους αριθμητικούς και λογικούς τελεστές bit και τους τελεστές μετατόπισης με το σύμβολο *op*. Μια εντολή της μορφής:

$$< variable >=< variable > op < expr >;$$

είναι ισοδύναμη με την

$$< variable > op =< expr >;$$

Ο τελεστής op=, όπου op είναι ένας εκ των δυαδικών αριθμητικών τελεστών, των τελεστών μετατόπισης και λογικών τελεστών bit, ονομάζεται **τελεστής αντικατάστασης**. Έστω για παράδειγμα η παρακάτω εντολή ανάθεσης:

$$x = x + (8 * b + a);$$

που σημαίνει ότι αυτό που θέλουμε να κάνουμε είναι να αυξήσουμε την τιμή της μεταβλητής x κατά την τιμή που θα προκύψει από τον υπολογισμό της έκφρασης $(8 * b + a)$. Χρησιμοποιώντας τον τελεστή αντικατάστασης + = η παραπάνω έκφραση μπορεί ισοδύναμα να γραφεί ως εξής:

$$x+ = (8 * b + a);$$

Οι τελεστές αντικατάστασης έχουν τη μικρότερη προτεραιότητα από όλους τους άλλους τελεστές, πλήν του τελεστή κόμμα. Ο τελεστής κόμμα, είναι ο τελεστής με την μικρότερη προτεραιότητα. Ένα σύνολο εκφράσεων που χωρίζονται με , υπολογίζονται από αριστερά προς τα δεξιά και ο τύπος και η τιμή της όλης έκφρασης είναι αυτά της δεξιότερης έκφρασης. Έστω για παραδείγμα η εντολή ανάθεσης

$$x = (a++,\ b++);$$

αν a=0 και b=4, τότε η τιμή που θα ανατεθεί στην x είναι το 4 ενώ αν η εντολή ανάθεσης ήταν η

$$x = (a++,\ ++b);$$

τότε θα ανατίθονταν στη μεταβλητή x η τιμή 5 (γιατί?).

2.6. Μετατροπή Τύπων

Όταν ένας δυαδικός τελεστής εφαρμόζεται σε δεδομένα διαφορετικών τύπων, τότε αυτά μετατρέπονται σε έναν ενιαίο τύπο με βάση κάποιους κανόνες. Υπάρχουν δύο κατηγορίες μετατροπών των τύπων των ορισμάτων των τελεστών κατά την εκτέλεση της πράξης που ορίζουν:

- Οι αυτόματες, όπου ο μικρότερος τύπος μετατρέπεται στον μεγαλύτερο χωρίς απώλεια πληροφορίας.
- Οι ρητές με τη χρήση του μοναδιαίου τελεστή προσαρμογής (cast) ο οποίος έχει την ίδια προτεραιότητα με όλους τους άλλους μοναδιαίους τελεστές.

Έστω το παρακάτω κομμάτι κώδικα

```
int w = 4, x = 3;
float y;
```

$$y = (float)\ w/x; \tag{2.2}$$

Στη εντολή ανάθεσης 2.2, ο τελεστής (*float*) επειδή έχει τη μεγαλύτερη προτεραιότητα εφαρμόζεται πρώτος που σημαίνει ότι μετατρέπει την τιμή της μεταβλητής w στην έκφραση από ακέραια σε κινητής υποδιαστολής απλής ακρίβειας. Τότε, αυτομάτως και η τιμή της μεταβλητής x μετατρέπεται σε κινητής υποδιαστολής απλής ακρίβειας, με αποτέλεσμα η εκτελούμενη διαίρεση να είναι μεταξύ αριθμών κινητής υποδιαστολής απλής ακρίβειας. Η τιμή που ανατίθεται στη μεταβλητή y είναι το 1.333.... Στον Πίνακα 2.7 εμφανίζεται η διαδικασία μετατροπής τύπων από την οποία εύκολα μπορεί να διακρίνει κανείς τη διάταξη μεταξύ των βασικών τύπων δεδομένων της C.

Επίπεδο	Κανόνας Μετατροπής
1	Αν ένα από τα ορίσματα είναι long double τότε και το άλλο μετατρέπεται long double.
2	διαφορετικά αν ένα από τα ορίσματα είναι double τότε και το άλλο μετατρέπεται double.
3	Αν ένα από τα ορίσματα είναι float τότε και το άλλο μετατρέπεται float.
4	διαφορετικά τα ορίσματα τύπου char και short int μετατρέπονται σε int αν μπορεί να αναπαρασταθεί η αρχική τους τιμή αλλιώς σε unsigned int.
5	Αν ένα από τα ορίσματα είναι unsigned long τότε και το άλλο μετατρέπεται unsigned long.
6	Αν τα δύο ορίσματα είναι unsigned int και long, και ο long μπορεί να αναπαραστήσει όλες τις τιμές του τύπου unsigned int τότε ο κοινός τύπος είναι ο long αλλιώς είναι ο unsigned long.
7	Αν το ένα από τα ορίσματα είναι long, μετατρέπεται και το άλλο σε long.
8	Αν το ένα από τα ορίσματα είναι unsigned int, μετατρέπεται και το άλλο σε unsigned int.
9	Φτάνοντας στο βήμα αυτό και τα δύο ορίσματα πρέπει να είναι int.

Πίνακας 2.7: Βήματα Μετατροπής Τύπων

Στη C, όπως προαναφέρθηκε, ένας χαρακτήρας στην ουσία είναι ένας μικρός ακέραιος και δεν καθορίζεται αν οι μεταβλητές χαρακτήρα είναι προσημασμένες ή απρόσημες ποσότητες. Τι θα συμβεί αν γίνει μετατροπή ενός char σε int; Η C τουλάχιστον εγγυάται ότι οι εκτυπώσιμοι χαρακτήρες θα είναι πάντα θετικές ποσότητες. Για λόγους φορητότη-

τας καλό είναι να καθορίζονται αν οι μεταβλητές τύπου char είναι προσημασμένες ή όχι (signed ή unsigned).

2.7. Βασικές συναρτήσεις Εισόδου και Εξόδου

Η πρότυπη βιβλιοθήκη της C διαθέτει δύο βασικές συναρτήσεις εισόδου και εξόδου δεδομένων:

- printf()
- scanf()

2.7.1. Η συνάρτηση printf()

Η συνάρτηση printf() είναι η βασική συνάρτηση εξόδου δεδομένων. Η δήλωση της στο αρχείο επικεφαλίδα *stdio.h* έχει ως εξής:

$$int\ printf(char\ *format,\ arg1,\ arg2, \ldots);$$

Το πρώτο όρισμα αποτελεί τη φόρμα, βάση της οποίας θα γίνει η μορφοποιημένη εκτύπωση των δεδομένων στην οθόνη του υπολογιστή ενώ η συνάρτηση επιστρέφει το πλήθος των δεδομένων που έχουν τυπωθεί επιτυχώς. Στο κεφάλαιο 7 θα αναλύσουμε τον μηχανισμό με τον οποίο υλοποιείται στη C η εκτύπωση στην οθόνη. Ο προγραμματιστής μέσω της φόρμας αυτής καθορίζει το πλήθος των δεδομένων που θα τυπωθούν αλλά και τη μορφή με την οποία θα τυπωθούν (μορφοποιημένη έξοδος). Η φόρμα αυτή ως αλφαριθμητικό περιλαμβάνει εντός διπλών εισαγωγικών τις εξής κατηγορίες χαρακτήρων:

- κοινούς χαρακτήρες οι οποίοι τυπώνονται όπως είναι στην οθόνη

- και χαρακτήρες μετατροπής.

Ένας μετατροπέας αποτελείται από τον χαρακτήρα % ακολουθούμενο από έναν χαρακτήρα του Πίνακα 2.8. Έστω για παράδειγμα η εξής κλήση της συνάρτησης printf:

printf("The result is:%d", v);

όπου το πρώτο όρισμα "The result is:%d" είναι η φόρμα εκτύπωσης και καθορίζει ότι αυτό που θα τυπωθεί είναι το The result is: ακολουθούμενο από την τιμή της μεταβλητής v τυπωμένη ως ακέραιο. Το ότι το δεδομένο θα τυπωθεί ως ακέραιος καθορίζεται από τον μετατροπέα %d. Δηλαδή αν η μεταβλητή είχε την τιμή 3 αυτό που θα τυπώνονταν στην οθόνη θα ήταν το The result is:3. Επειδή ακριβώς η φόρμα εκτύπωσης ορίζει ότι ένα δεδομένο θα τυπωθεί, ακολουθεί ένα μόνο όρισμα αυτό της μεταβλητής v. Έτσι σε κάθε κλήση της συνάρτησης printf ακολουθούν τόσα ορίσματα όσα είναι τα δεδομένα που η φόρμα καθορίζει ότι πρέπει να τυπωθούν. Τα ορίσματα μπορεί να είναι μεταβλητές, σταθερές και αριθμητικές ή λογικές εκφράσεις.

Ένα άλλο παράδειγμα κλήσης τη συνάρτησης printf είναι το ακόλουθο:

printf("%d %f", v, p);

όπου η συνάρτηση printf θα τυπώσει τις τιμές των μεταβλητών v και p ως ακέραιο και κινητής υποδιαστολής απλής ακρίβειας αντίστοιχα αφήνοντας ένα κενό ανάμεσα τους και θα επιστρέψει την τιμή 2 καθώς δύο δεδομένα θα έχουν τυπωθεί επιτυχώς. Στον Πίνακα 2.8 εμφανίζονται οι κυριότεροι μετατροπείς που μπορούν να χρησιμοποιηθούν στη φορμα εκτύπωσης της συνάρτησης printf.

Προσδιοριστής	Περιγραφή
d,i	Προσημασμένος ακέραιος αριθμός
u	Μη Προσημασμένος ακέραιος αριθμός
c	απλός χαρακτήρας
x	Μη προσημασμένος ακέραιος σε δεκαεξαδική μορφή όπου τα a, b, c, d, e, f τυπώνονται πεζά
X	Μη προσημασμένος ακέραιος σε δεκαεξαδική μορφή όπου τα a, b, c, d, e, f τυπώνονται κεφαλαία
o	Μη προσημασμένος ακέραιος σε οκταδική μορφή
f	Αριθμός κινητής υποδιαστολής της μορφής $-m.dddddd$ όπου το πλήθος των d καθορίζεται από την ακρίβεια η οποία εξ' ορισμού είναι 6 και μπορεί να αλλάξει από τον προγραμματιστή στη φόρμα μετατροπής
e, E	Αριθμός κινητής υποδιαστολής της μορφής $-m.ddddddde+/-xx$ ή $-m.ddddddE+/-xx$ όπου το πλήθος των d καθορίζεται από την ακρίβεια η οποία εξ' ορισμού είναι 6 και μπορεί να αλλάξει από τον προγραμματιστή στο φόρμα μετατροπής
g, G	όταν ο εκθέτης είναι μικρότερος του -4 ή μεγαλύτερος ή ίσος από την ακρίβεια τότε χρησιμοποιείται το %e/%E διαφορετικά το %f
s	Αλφαριθμητικό
p	το ορισμά που αντιστοιχεί στον προσδιοριστή αυτό τυπώνεται ως δείκτης

Πίνακας 2.8: Προσδιοριστές Μετατροπής της printf

Μεταξύ του χαρακτήρα % και ενός χαρακτήρα μετατροπής, μπορεί να υπάρχουν και άλλοι χαρακτήρες που διαμορφώνουν τόσο το πεδίο εκτύπωσης του αντίστοιχου ορίσματος όσο και τον τρόπο με τον οποίο θα τυπωθεί το δεδομένο εντός του πεδίου. Οι χαρακτήρες αυτοί με την σειρά εμφάνισης τους είναι οι εξής:

1. -, ο οποίος καθορίζει αριστερή στοίχιση του ορίσματος εντός του πεδίου του.
2. +, ο οποίος καθορίζει ότι ο εκτυπούμενος αριθμός θα είναι πάντα με πρόσημο.
3. κενό διάστημα, όπου σε περίπτωση απουσίας προσήμου από το όρισμα τυπώνεται το κενό διάστημα.
4. 0, ο οποίος καθορίζει ότι αν ένα όρισμα έχει λιγότερους χαρακτήρες από το μήκος του πεδίου του τότε το πεδίο συμπληρώνεται με 0. Εξ ορισμού η συμπλήρωση ενός πεδίου γίνεται με κενά διαστήματα.
5. # ο οποίος για κάθε έναν από τους μετατροπείς του Πίνακα 2.8 καθορίζει μια εναλλακτική μορφή εκτύπωσης:
   - για ο τυπώνει το 0 ως πρώτο χαρακτήρα.
   - για x ή X τυπώνει 0x ή 0X.
   - e, E και f η έξοδος θα έχει πάντα υποδιαστολή.
   - g και G, η έξοδος θα έχει πάντα υποδιαστολή και τα μη-σημαντικά μηδενικά δεν αφαιρούνται από το τέλος.
6. ένας αριθμός που καθορίζει το ελάχιστο πλάτος του πεδίου εκτύπωσης σε χαρακτήρες. Αν το όρισμα είναι μικρότερο από το πλάτος του πεδίου τότε το πεδίο συμπληρώνεται είτε με 0 είτε με κενά διαστήματα.
7. τελεία, που χωρίζει το πλάτος του πεδίου από την ακρίβεια.

8. ένας αριθμός που καθορίζει την ακρίβεια που είναι:

   - για αλφαριθμητικά, ο μέγιστος αριθμός χαρακτήρων που θα τυπωθούν,
   - για ακεραίους, ο ελάχιστος αριθμός ψηφίων που θα τυπωθούν και σε περίπτωση που το πλάτος του πεδίου είναι μεγαλύτερο τότε πριν από τα ψηφία του ακέραιου θα τυπωθούν 0,
   - για αριθμούς κινητής υποδιαστολής, ο αριθμός των ψηφίων που θα εμφανιστούν μετά την υποδιαστολή για μετατροπές (e, E και f) ή ο αριθμός των σημαντικών ψηφίων για μετατροπές (g και G).

9. h δηλώνοντας ότι το αντίστοιχο όρισμα θα τυπωθεί ως short int, l δηλώνοντας ότι το αντίστοιχο όρισμα θα τυπωθεί ως long int και L δηλώνοντας ότι το όρισμα θα τυπωθεί ως long double.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  #define LOWLET "abcdefghijklmnopqrstuvwxyz"
 4:
 5:  int main()
 6:  {
 7:     printf("%20.1s\n", LOWLET);
 8:     printf("%#X\n", 30);
 9:     printf("%6.2f\n", 80.756);
10:
11:   return 0;
12:  }
```

Σχήμα 2.2: Κλήσεις της printf.

Στο Σχήμα 2.2 δίνεται ένα πρόγραμμα σε C όπου η κύρια συνάρτηση περιέχει τρεις κλήσεις της συνάρτησης printf. Στη γραμμή 3 του προγράμματος δηλώνεται η αλφαριθμητική σταθερά LOWLET που περιλαμβάνει όλους τους πεζούς λατινικούς χαρακτήρες. Η εκτέλεση της συνάρτησης στη γραμμή 7 έχει ως αποτέλεσμα τη εκτύπωση του χαρακτήρα a της σταθεράς σε ένα πεδίο 20 χαρακτήρων με τους υπολοιπους χαρακτήρες του πεδίου να είναι κενά διαστήματα. Επίσης, η εκτέλεση της συνάρτησης στη γραμμή 8 έχει ως αποτέλεσμα την εκτύπωση του 0X1E που είναι ο αριθμός 30 στη δεκαεξαδική του μορφή ενώ το αποτέλεσμα της εκτέλεσης της συνάρτησης στη γραμμή 9 έχει ως αποτέλεσμα την εκτύπωση του αριθμού 80.75 δηλαδή δεν τυπώνεται το τρίτο δεκαδικό ψηφίο της literal σταθεράς 80.756. Η εκτύπωση του χαρακτήρα \n έχει ως αποτέλεσμα την αλλαγή γραμμής (newline χαρακτήρας) με το πέρας της εκτύπωσης.

2.7.2. Η συνάρτηση scanf()

Η συνάρτηση scanf() είναι η βασική συνάρτηση εισόδου δεδομένων. Η δήλωση της στο αρχείο επικεφαλίδα *stdio.h* έχει ως εξής:

$$int\ scanf(char\ *format,\quad var1,\ var2,\ldots);$$

Το πρώτο όρισμα αποτελεί τη φόρμα, βάση της οποίας θα γίνει η ανάγνωση των δεδομένων από το πληκτρολόγιο του υπολογιστή, ενώ η συνάρτηση επιστρέφει το πλήθος των δεδομένων που έχουν διαβασθεί επιτυχώς. Στο κεφάλαιο 7 θα αναλύσουμε τον μηχανισμό με τον οποίο υλοποιείται στη C η ανάγνωση δεδομένων. Ο προγραμματιστής μέσω της φόρμας αυτής καθορίζει το πλήθος των δεδομένων που θα διαβασθούν αλλά και με ποια μορφή θα διαβασθούν (μορφοποιημένη είσοδος). Έστω για παράδειγμα η εξής κλήση της συνάρτησης scanf:

scanf("%d", &v);

όπου το πρώτο όρισμα "%d" είναι η φόρμα ανάγνωσης και καθορίζει ότι αυτό που θα διαβασθεί είναι ένας ακέραιος αριθμός ο οποίος στη συνέχεια θα αποθηκευθεί στη

μεταβλητή v. Αξίζει να σημειωθεί ότι ως όρισμα στη λιστα ορισμάτων εμφανίζεται το όνομα της μεταβλητής με τον τελεστή & να προηγείται. Η χρήση του τελεστή & πριν από το όνομα μιας μεταβλητής, την οποία θα εξηγήσουμε αναλυτικά στο κεφάλαιο 7, στη συνάρτηση scanf έχει ως συνέπεια τη χρήση της διεύθυνσης της μεταβλητής.

Η φόρμα ανάγνωσης ως αλφαριθμητικό μπορεί να περιλαμβάνει εντός διπλών εισαγωγικών:

1. Κενά ή στηλογνώμονες που κατά την ανάγνωση αγνοούνται

2. Κοινούς χαρακτήρες που κατά την ανάγνωση θα πρέπει να ταιριάζουν με τους εισαγόμενους χαρακτήρες

3. οδηγίες μετατροπής, που αποτελούνται από τον ειδικό χαρακτήρα % ακολουθούμενο προαιρετικά από τον χαρακτήρα * προκειμένου να παρεμποδιστεί η ανάθεση του δεδομένου στο όρισμα, έναν προαιρετικό αριθμό που καθορίζει το μέγιστο πλάτος του πεδίου εισόδου , ένα προαιρετικό h, l, L που καθορίζει το μέγεθος του ορίσματος προορισμού και έναν από τους χαρακτήρες του Πίνακα 2.9.

Προσδιοριστής	Περιγραφή
d	Προσημασμένος ακέραιος αριθμός
i	Προσημασμένος ακέραιος αριθμός, ο οποίος μπορεί να δοθεί και σε οκταδική μορφή με το 0 να προηγείται του ακέραιου ή σε δεκαεξαδική με το 0x ή 0X να προηγείται του ακέραιου
u	Μη Προσημασμένος ακέραιος αριθμός
c	απλός χαρακτήρας
x	δεκαεξαδικός ακέραιος με ή χωρίς 0x ή 0X να προηγείται του ακέραιου
o	οκταδικός ακέραιος με ή χωρίς 0 να προηγείται του ακέραιου
f, e, g	Αριθμός κινητής υποδιαστολής, η μορφή που μπορεί να έχει είναι μια σειρά ψηφίων που μπορεί να περιλαμβάνει την υποδιαστολή, ακολοθούμενη από τον χαρακτήρα E και ένα προσημασμένο ακέραιο αριθμό
s	Αλφαριθμητικό

Πίνακας 2.9: Προσδιοριστές Μετατροπής της scanf

Το πεδίο εισόδου που αναφέρθηκε παραπάνω εξ ορισμού αποτελείται από μη λευκούς χαρακτήρες (δηλαδή δεν είναι χαρακτήρας αλλαγής γραμμής, κενό, στηλογνώμονες). Συνεπώς, η ανάγνωση ενός πεδίου εισόδου θα τερματίσει όταν συναντηθεί λευκός χαρακτήρας ή καλυφθεί το οριζόμενο από τον προγραμματιστή πλάτος του πεδίου εισόδου.

Στο Σχήμα 2.3 δίνονται απλά παραδείγματα κλήσεων της συνάρτησης scanf. Κατά την εκτέλεση του προγράμματος αυτού, η scanf της γραμμής 7 αναμένει είσοδο από τον χρήστη. Ο χρήστης θα πρέπει να πληκτρολογήσει τους χαρακτήρες Arithmos ακολου-

θούμενους από έναν αριθμό κινητής υποδιαστολής των 5 ψηφίων συμπεριλαμβανομένου και της υποδιαστολής. Αν ο χρήστης πληκτρολογήσει για παράδειγμα Arithmos543.645 τότε η scanf θα αναθέσει στην μεταβλητή κινητής υποδιαστολής y τον αριθμό 543.6 δεδομένου ότι η ανάγνωση του πεδίου εισόδου θα τερματίσει όταν αναγνωσθούν 5 χαρακτήρες από την είσοδο όπως ορίζεται από την φόρμα ανάγνωσης της scanf στη γραμμή 7. Οι υπόλοιποι χαρακτήρες, δηλαδή το 45, θα αναγνωσθούν από την scanf της γραμμής 9 αλλά λόγω του ότι στη φόρμα ανάγνωσης έχει εισαχθεί ο χαρακτήρας ∗ μεταξύ του % και του f δεν θα εκχωρηθούν στη μεταβλητή y.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  int main()
 4:  {
 5:     float y;
 6:
 7:     scanf("Arithmos%5f", &y);
 8:     printf("%f\n",  y);
 9:     scanf("%*f",&y);
10:     printf("%f\n",  y);
11:
12:   return 0;
13:  }
```

Σχήμα 2.3: Κλήσεις της scanf.

2.8. Ασκήσεις

Άσκηση 2.8.1 *Τι θα τυπωθεί στην οθόνη όταν εκτελεσθεί ο παρακάτω κώδικας; Γράψτε ένα πρόγραμμα για να ελέγξετε τις απαντήσεις σας;*

```
int x, y = 6, z = 8;
x = y++ + ++z;
printf("%d %d %d\n", x, y, z);
x = ++y + z++;
printf("%d %d %d\n", x, y, z);
x = ++y + ++z;
printf("%d %d %d\n", x, y, z);
x = --y + z--;
printf("%d %d %d\n", x, y, z);
```

Άσκηση 2.8.2 *Ποιό είναι το αποτέλεσμα της εκτέλεσης της παρακάτω εντολής ανάθεσης;*

```
d = (a = 4) * (c = 8);
```

Τι θα συμβεί αν αφαιρέσουμε όλες τις παρενθέσεις;

Άσκηση 2.8.3 *Συμπληρώστε τον πίνακα που ακολουθεί. Στη συνέχεια, γράψτε ένα πρόγραμμα για να επιβεβαιώσετε τις τιμές που έχετε υπολογίσει.*

```
int x = 5, y = -6, z = 8, w = -7, u = 11;
```

Έκφραση	Τιμή
`x/y/z`	
`9 + z / --w * u`	
`4 * x % -y + z - 2`	
`23 / - --u - +12 % z`	
`x += y *= z -= 4 * 8`	

Άσκηση 2.8.4 *Γράψτε ένα πρόγραμμα με τις παρακάτω εντολές κώδικα. Τι θα τυπωθεί στην οθόνη και γιατί;*

int $u = -3$;
unsigned int $v = -3$;
if $(u \gg 1 == v \gg 1)$ *printf*("*Eiserxontai mhdenika kata thn metatopish*\n");
else printf("*To bit proshmou eiserxetai kata thn metatopish*\n");

Άσκηση 2.8.5 *Συμπληρώστε του πίνακα που ακολουθεί.*

int $x = 3,\ w = 6,\ u = 4,\ v = 2$;
float $y = 2.0$;

Έκφραση	Τιμή
$x > w$ && $u > v$	
$x <!w \| \| !x$	
$x + w <!u + v$	
$x - y \parallel w/u$ && $w * x$	

Κεφάλαιο 3

Εντολές Ελέγχου Ροής Προγράμματος

Όπως αναφέρθηκε και στο κεφάλαιο 1, για την επίλυση ενός προβλήματος αποκλειστικός στόχος αρχικά είναι η ανεύρεση ενός αποδοτικού αλγόριθμου που να δίνει τα επιδιωκόμενα αποτελέσματα σε ικανοποιητικό χρόνο και με την καλύτερη δυνατή αξιοποίηση των πόρων του υπολογιστή όπως είναι η μνήμη του. Στη φάση λοιπόν της σχεδίασης και ανάλυσης του αλγόριθμου που θα λύνει το πρόβλημα μπορεί κανείς να αποδώσει τον αλγόριθμο σε ψευδο-γλώσσα.

Στη συνέχεια, ο ψευδο-κώδικας θα πρέπει να επανοκωδικοποιηθεί χρησιμοποιώντας μια πραγματική γλώσσα προγραμματισμού. Το πρόγραμμα που προκύπτει αποτελείται από ένα σύνολο εντολών της γλώσσας προγραμματισμού. Μια εντολή είναι οι ακριβείς οδηγίες προς τον υπολογιστή προκειμένου να διεξάγει συγκεκριμένο έργο. Κατά την εκτέλεση του προγράμματος από τον υπολογιστή, μια συγκεκριμένη ακολουθία εντολών η οποία καθορίζεται από τα δεδομένα εισόδου εκτελείται. Η ακολουθία εντολών που

εκτελείται κάθε φορά ονομάζεται ροή εκτέλεσης προγράμματος. Εξ ορισμού, όταν ολοκληρώνεται η εκτέλεση μιας εντολής, η αμέσως επόμενη εντολή προγράμματος είναι αυτή που πρόκειται να εκτελεσθεί. Σε κάθε γλώσσα προγραμματισμού υπάρχουν εντολές που δίνουν την δυνατότητα στον προγραμματιστή να αλλάξει τη ροή εκτέλεσης του προγράμματος, όπως για παράδειγμα αντί να εκτελεσθεί η αμέσως επόμενη εντολή να εκτελεσθεί μια εντολή που βρίσκεται αρκετές εντολές είτε πριν είτε μετά.

Στο παρόν κεφάλαιο αρχικά ορίζουμε την έννοια της ομάδας εντολών στη C και στη συνέχεια αναλύουμε τις εντολές ελέγχου ροής προγράμματος της C.

3.1. Ομάδες Εντολών

Τυπικά στη C, μια οποιαδήποτε έκφραση ακολουθούμενη από ; ορίζει μια εντολή. Μέχρι τώρα είδαμε εντολές του προεπεξεργαστή όπως η συμπερίληψη αρχείων επικεφαλίδας και ο ορισμός σταθερών (οι εντολές του προεπεξεργαστή χωρίς ;), και εντολές προγράμματος όπως οι δηλώσεις μεταβλητών και οι εντολές ανάθεσης ή καταχώρησης.

Πολλές εντολές μαζί μπορούν να ενταχθούν σε μια ομάδα (block) εντολών με τη χρήση των άγκιστρων { } η οποία εκλαμβάνεται ως μια σύνθετη εντολή (compound-statement) όπως φαίνεται παρακάτω:

```
{
  δηλώσεις_μεταβλητών;
  εντολή-1;
  . . .
  εντολή-n ;
}
```

Σε κάθε σύνθετη εντολή, οι δηλώσεις των μεταβλητών προηγούνται όλων των υπόλοιπων εντολών. Η κύρια ομάδα εντολών στη C είναι αυτή της κύριας συνάρτησης main.

3.2. Η Εντολή Απόφασης If

Στο κεφάλαιο 2 παρουσιάστηκε ο τελεστής συνθήκη ? : ο οποίος δίνει τη δυνατότητα για την υπό συνθήκη εκτέλεση εκφράσεων, όπου όταν μια συνθήκη ικανοποιείται τότε εκτελείται μια έκφραση διαφορετικά εκτελείται μια άλλη έκφραση.

Στη C μια εντολή ή ένα σύνολο εντολών μπορεί να εκτελεστεί υπό συνθήκη με τη χρήση της εντολής απόφασης **if** η σύνταξη της οποίας έχει ως εξής:

if (συνθήκη) εντολή;
if (συνθήκη) σύνθετη-εντολή

όπου στη θέση της συνθήκης μπορεί να είναι μια λογική έκφραση ή μια οποιαδήποτε έκφραση καθώς όπως αναφέραμε κάθε έκφραση επιστρέφει μια τιμή η οποία αν είναι 0 εκλαμβάνεται από την if ως ψευδής διαφορετικά εκλαμβάνεται ως αληθής. Όπως φαίνεται από την παραπάνω σύνταξη μια απλή εντολή ή μια σύνθετη εντολή μπορεί να εκτελεσθεί μόνο αν είναι αληθής η συνθήκη. Μετά την εκτέλεση της εντολής απόφασης, η ροή του προγράμματος συνεχίζεται με την επόμενη της εντολής if.

Με την σύνταξη της εντολής απόφασης που ακολουθεί είναι δυνατή η προσθήκη εναλλακτικής εντολής για την περίπτωση που δεν ικανοποιείται η συνθήκη.

if (συνθήκη) εντολή-1 ;
else εντολή-2 ;

if (συνθήκη) σύνθετη-εντολή-1
else σύνθετη-εντολή-2

Κατά συνέπεια, όταν η συνθήκη είναι αληθής εκτελείται η εντολή-1/σύνθετη-εντολή-1 διαφορετικά εκτελείται η εντολή-2/σύνθετη-εντολή-2. Και πάλι με την ολοκλήρωση της εκτέλεσης της εντολής απόφασης **if else** η ροή του προγράμματος συνεχίζεται με την επόμενη της εντολής απόφασης.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  int main()
 4:  {
 5:
 6:      int a, b;
 7:
 8:      printf("\nGive an Integer Value for a: ");
 9:      scanf("%d", &a);
10:
11:      printf("\n Give an Integer Value for b: ");
12:      scanf("%d", &b);
13:
14:      if (a == b)
15:          printf("\n a is equal to b\n");
16:      if (a > b)
17:          printf("\n a is greater than b\n");
18:      if (a < b)
19:          printf("\n a is smaller than b\n");
20:
21:      return 0;
22:  }
```

Σχήμα 3.1: Εντολή Απόφασης if για τη σύγκριση ακεραίων

Στο Σχήμα 3.1 δίνεται ένα ενδεικτικό παράδειγμα χρήσης της εντολής απόφασης if. Αρχικά ζητείται από τον χρήστη μέσω των εντολών των γραμμών 8 και 9 να δώσει έναν ακέραιο ο οποίος αποθηκεύεται στη μεταβλητή a. Στη συνέχεια, μέσω των εντολών των γραμμών 11 και 12 ζητείται άλλος ένας ακέραιος, ο οποίος καταχωρείται στη μεταβλητή b. Η εκτέλεση του προγράμματος συνεχίζεται με την εντολή απόφασης της γραμμής 14. Αν η τιμή της μεταβλητής a είναι ίση με αυτή της b τυπώνεται το μήνυμα που αφορά

την ισότητα. Στη συνέχεια, εκτελείται η εντολή απόφασης 16 στην οποία ελέγχεται αν η μεταβλητή *a* είναι μεγαλύτερη από την *b*. Ενώ το πρόγραμμα ολοκληρώνεται με την εκτέλεση της εντολής απόφασης της γραμμής 18, όπου τώρα ελέγχεται αν η μεταβλητή *a* είναι μικρότερη από τη *b*. Γίνεται φανερό ότι το μειονέκτημα του προγράμματος είναι ότι γίνονται πάντα τρεις έλεγχοι ακόμα και όταν η σχέση των μεταβλητών έχει προσδιοριστεί με τον πρώτο έλεγχο. Για να γραφεί ένας πιο αποδοτικός κώδικας μπορούμε να αξιοποιήσουμε το γεγονός ότι η σύνθετη εντολή σε μια εντολή απόφασης μπορεί να είναι επίσης μια εντολή απόφασης. Η σύνταξη της εντολής απόφασης στην περίπτωση αυτή είναι η εξής:

if (συνθήκη-1) σύνθετη-εντολή-1
else if (συνθήκη-2) σύνθετη-εντολή-2
else σύνθετη-εντολή-3
επόμενη-εντολή;

Αξιοποιώντας τη δυνατότητα αυτή, το πρόγραμμα μας που εξετάζει τη σχέση των τιμών των μεταβλητών *a* και *b* παίρνει τη μορφή που δίνεται στο Σχήμα 3.2.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  int main()
 4:  {
 5:
 6:     int a, b;
 7:
 8:     printf("\nGive an Integer Value for a: ");
 9:     scanf("%d", &a);
10:
11:     printf("\n Give an Integer Value for b: ");
12:     scanf("%d", &b);
13:
14:     if (a == b)
15:        printf("\n a is equal to b\n");
16:     else if (a > b)
17:        printf("\n a is greater than b\n");
18:     else printf("\n a is smaller than b\n");
19:
20:     return 0;
21:  }
```

Σχήμα 3.2: Εντολή Απόφασης if/else για τη σύγκριση ακεραίων

3.3. Η Εντολή Επιλογής switch

Είναι πολλές φορές που κατά την επίλυση ενός προβλήματος χρειάζεται να επιλεγεί ένα σύνολο λειτουργιών, το οποίο είναι συνδεδεμένο με μια σταθερή ακέραια τιμή, ανάλογα με το αποτέλεσμα της αποτίμησης μιας έκφρασης. Για παράδειγμα, αν σε ένα αλγόριθμο πρέπει να εκτελεσθούν άλλες λειτουργίες όταν η υπολογισθείσα τιμή είναι το 1 και άλλες

όταν η υπολογισθείσα τιμή είναι το 2 κλπ τότε αυτό μπορεί να υλοποιηθεί με την χρήση πολλαπλών εντολών απόφασης if-else. Για τις περιπτώσεις που οι επιλογές είναι πολλές όπου η χρήση πολλαπλών if-else είναι αναποτελεσματική,η C διαθέτει την εντολή switch για την αποτελεσματική υλοποίηση τέτοιων απαιτήσεων.

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2:
 3: main()
 4: {
 5:    int key;
 6:
 7:    printf("Enter a number between 1 and 5:");
 8:    scanf("%d", &key);
 9:    switch (key)
10:    {
11:       case 1: printf("You entered 1.\n");
12:       case 2: printf("You entered 2.\n");
13:       case 3: printf("You entered 3.\n");
14:       case 4: printf("You entered 4.\n");
15:       case 5: printf("You entered 5.\n");
16:       default: printf("Out of range, try again.\n");
17:    }
18:    return 0;
19: }
```

Σχήμα 3.3: Χρήση της εντολής επιλογής switch με προβλήματα

Η εντολή switch αποτελείται από μια έκφραση και μια ομάδα εντολών. Στην ομάδα εντολών της switch υπάρχει ένα σύνολο σταθερών ακέραιων τιμών όπου σε καθε μία

από αυτές αντιστοιχεί ένα σύνολο εντολών. Αρχικά αποτιμάται η έκφραση της switch και εκτελούνται οι εντολές που αντιστοιχούν στην ακέραια τιμή που επιστρέφεται από την αποτίμηση της έκφρασης. Για την περίπτωση που η τιμή που επιστρέφεται από την αποτίμηση της έκφρασης δεν εμπίπτει σε καμία από τις τιμές του συνόλου των σταθερών ακέραιων τιμών η switch διαθέτει την επιλογή default στην οποία αντιστοιχούν οι εντολές που θα εκτελεσθούν στην περίπτωση αυτή. Η σύνταξη της έχει ως εξής:

```
switch(έκφραση) {
        case σταθερή-έκφραση-1: εντολές;
        case σταθερή-έκφραση-2: εντολές;
        . . .
        default: εντολές;
}
```

Οι σταθερές εκφράσεις πρέπει να είναι διαφορετικές και οι τιμές τους ακέραιοι. Υπολογίζεται αρχικά η έκφραση, και στη συνέχεια αναζητείται η **case** με ετικέτα ισότιμη με την τιμή που υπολογίσθηκε. Η εκτέλεση συνεχίζεται από την **case** που βρέθηκε στο προηγούμενο βήμα μέχρι τέλος της **switch**. Η **default** επιλογή, εκτελείται όταν καμία από τις **case** δεν ταυτίζεται με την έκφραση. Μπορεί να παραληφθεί, όμως είναι καλή τακτική να εισάγεται.

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2:
 3: main()
 4: {
 5:    int key;
 6:
 7:    printf("Enter a number between 1 and 5:");
 8:    scanf("%d", &key);
 9:    switch (key)
10:    {
11:       case 1: printf("You entered 1.\n");
12:               break;
13:       case 2: printf("You entered 2.\n");
14:               break;
15:       case 3: printf("You entered 3.\n");
16:               break;
17:       case 4: printf("You entered 4.\n");
18:               break;
19:       case 5: printf("You entered 5.\n");
20:               break;
21:       default:printf("Out of range, try again.\n");
22:               break;
23:    }
24:    return 0;
25: }
```

Σχήμα 3.4: Σωστή χρήση της εντολής επιλογής switch

Στο Σχήμα 3.3 δίνεται ένα πολύ απλό παράδειγμα χρήσης της εντολής switch που όμως είναι προβληματικό. Τι θα συμβεί όταν εκτελεστεί το πρόγραμμα αυτό και δώσουμε σαν είσοδο το 2? Αυτό που θα συμβεί είναι να τυπωθούν τα παρακάτω μηνύματα:

You entered 2.

```
You entered 3.
You entered 4.
You entered 5.
Out of range, try again.
```

κάτι το οποίο δεν είναι και το επιδιωκόμενο. Το πρόβλημα λύνεται με τη χρήση της εντολής **break** της C η οποία διακόπτει την εκτέλεση της τρέχουσας εντολής.

Στο Σχήμα 3.4 δίνεται το διορθωμένο πρόγραμμα με την προσθήκη της εντολής break της C στο τέλος κάθε επιλογής case. Έτσι κατά την εκτέλεση του προγράμματος όταν δοθεί το 2 ως είσοδος θα εκτελεσθεί η συνάρτηση printf που αντιστοιχεί στην τιμή και στη συνέχεια θα εκτελεσθεί η εντολή break η οποία θα διακόψει στο σημείο αυτό την εκτέλεση της switch, ενώ η ροή του προγράμματος θα συνεχιστεί με την εντολή που ακολουθεί την switch. Συνεπώς, αυτό που θα τυπωθεί είναι το:

```
You entered 2.
```

3.4. Εντολές Επανάληψης

Μέχρι τώρα είδαμε την ακολουθιακή εκτέλεση των εντολών ενός προγράμματος με τις εντολές του να εκτελούνται μια φορά, η μια μετά την άλλη, με μοναδική εξαίρεση τις εντολές απόφασης και επιλογής οι οποίες επιλέγουν με βάση κάποια συνθήκη το ποιές εντολές θα εκτελεσθούν κάθε φορά. Πολλές φορές οι αλγόριθμοι για την αποτελεσματική επίλυση ενός προβλήματος απαιτούν την επαναληπτική εκτέλεση μιας ομάδας εντολών είτε συγκεκριμένο αριθμό επαναλήψεων είτε όσο ισχύει κάποια συνθήκη. Η C διαθέτει τις εντολές for, while και do-while που επιτρέπουν στον προγραμματιστή να εκτελεί ομάδα εντολών είτε συγκεκριμένο αριθμό επαναλήψεων είτε όσο ισχύει μια συνθήκη.

3.4.1. Η εντολή while

Η εντολή **while** επιτρέπει την επαναληπτική εκτέλεση μιας ή περισσότερων εντολών όσο μια έκφραση δίνει μη μηδενική τιμή. Η σύνταξη της έχει ως εξής:

while (έκφραση) εντολή;
while (έκφραση) σύνθετη-εντολή

Αρχικά υπολογίζεται η έκφραση. Αν το αποτέλεσμα που επιστρέφεται με το πέρας του υπολογισμού είναι μη μηδενικό τότε εκτελείται η εντολή. Με την ολοκλήρωση της εκτέλεσης της, ο έλεγχος του προγράμματος μεταφέρεται στον υπολογισμό της έκφρασης της while. Η εντολή θα εκτελείται μέχρι το αποτέλεσμα υπολογισμού της έκφρασης γίνει μηδέν. Η εκτελούμενη εντολή μπορεί να είναι είτε απλή είτε σύνθετη.

Η εκτέλεση της εντολής στη while θα πρέπει να επηρεάζει την τιμή των μεταβλητών που συμμετέχουν στην έκφραση της while, προκειμένου κάποιος υπολογισμός της έκφρασης να δώσει μηδενικό αποτέλεσμα και να τερματίστει η while διαφορετικά θα εκτελείται ατέρμονα.

Έστω οι παρακάτω εντολές ενός προγράμματος

c = *getchar*();
while(*c* ==′ ′|| *c* ==′ *n*′|| *c* ==′ *t*′) *c* = *getchar*();

Στην εντολή ανάθεσης *c* = *getchar*() εμφανίζεται η συνάρτηση *getchar*() η οποία είναι μια συνάρτηση της προτύπου βιβλιοθήκης της C (αρχείο επικεφαλίδα *stdio.h*) η οποία διαβάζει έναν χαρακτήρα από την είσοδο τον οποίο και επιστρέφει. Συνεπώς, η εκτέλεση της εντολής ανάθεσης έχει ως αποτέλεσμα η μεταβλητή c να λάβει τον χαρακτήρα που έχει διαβάσθει από την *getchar*().

Η έκφραση της while που καθορίζει τον αριθμό των επαναλήψεων είναι η (*c* ==′ ′||*c* ==′ \n′||*c* ==′ \t′) ενώ το τμήμα εντολής της while είναι η εντολή ανάθεσης *c* =

getchar(). Η εκτέλεση του προγράμματος συνεχίζεται με την αποτίμηση της έκφρασης εξετάζοντας αν ο χαρακτήρας που περιέχεται στη μεταβλητή c είναι ένας εκ των ′ ′ , ′\n′ και ′\t′ δηλαδή αν είναι ένας από τους λευκούς χαρακτήρες. Σε περίπτωση που είναι ένας από αυτούς, τότε διαβάζεται από την είσοδο ο επόμενος χαρακτήρας, ο οποίος ανατίθεται και πάλι στη μεταβλητή c και ο έλεγχος του προγράμματος επανέρχεται στη αποτίμηση της έκφρασης της while. Η εντολή while θα τερματίσει όταν διαβασθεί ένας μη λευκός χαρακτήρας. Με άλλα λόγια με την παραπάνω εντολή while αγνοούνται όλοι οι λευκοί χαρακτήρες της εισόδου.

Το ίδιο αποτέλεσμα μπορεί να επιτευχθεί πιο απλά με την παρακάτω σύνταξη της while

while((*c* = *getchar*()) ==′ ′|| *c* ==′ \n′|| *c* ==′ \t′);

όπου η εντολή ανάθεσης αποτελεί μέρος της έκφρασης της while ενώ το τμήμα εντολής της είναι κενό που σημαίνει ότι για κάποιο αριθμό επαναλήψεων που καθορίζεται από την παραπάνω έκφραση δεν θα εκτελεσθεί καμία εντολή. Δηλαδή όσο θα διαβάζονται λευκοί χαρακτήρες από την είσοδο καμία εντολή δεν θα εκτελείται. Με την ανάγνωση του πρώτου μη λευκού χαρακτήρα η while τερματίζει και η επόμενη εντολή είναι αυτή που ακολουθεί την while.

3.4.2. Η εντολή for

Για την επαναληπτική εκτέλεση ομάδας εντολών για πεπερασμένο αριθμό επαναλήψεων, χρησιμοποιείται η εντολή for της C. Η σύνταξη της έχει ως εξής:

for (εντολές_ανάθεσης; συνθήκη; έκφραση-τροποποίησης) εντολή;
for (εντολές_ανάθεσης; συνθήκη; έκφραση-τροποποίησης) σύνθετη-εντολή

Κατά την εκτέλεση της εντολής for αρχικά εκτελούνται οι εντολές ανάθεσης οι οποίες αρχικοποιούν μια ή περισσότερες μεταβλητές. Όταν έχουμε περισσότερες από μια εντολές

ανάθεσης τότε αυτές χωρίζονται με τον τελεστή κόμμα. Στη συνέχεια εξετάζεται η συνθήκη και στην περίπτωση που είναι αληθής τότε εκτελείται η εντολή ή σύνθετη-εντολή διαφορετικά ο έλεγχος του προγράμματος μεταφέρεται στην εντολή που έπεται της εντολής for. Όταν η εντολή ή η σύνθετη εντολή εκτελεσθεί τότε εκτελείται η έκφραση-τροποποίησης η οποία αυξάνει ή μειώνει τις μεταβλητές που αρχικοποιήθηκαν με την εντολή ανάθεσης και επανεξετάζεται η συνθήκη.

Έστω για παράδειγμα η εντολή $for(i = 0; i < n; i + +)$; όπου αρχικά ανατίθεται η τιμή 0 στη μεταβλητή i, ελέγχεται η συνθήκη $i < n$ και σε περίπτωση που είναι αληθής επειδή το τμήμα εντολής είναι κενό δεν εκτελείται καμία εντολή. Στη συνέχεια αυξάνεται η μεταβλητή i κατά 1 και επανεξετάζεται η συνθήκη $i < n$. Ουσιαστικά η εκτέλεση της εντολής for δεν κάνει τίποτε άλλο από το εισάγει μια καθυστέρηση n βημάτων μεταξύ της εντολής που προηγείται της for και αυτής που έπεται.

Μπορούμε να κατασκευάσουμε μια ατέρμονη επανάληψη εντολών (ατέρμνος βρόχος) απλά παραλείποντας την συνθήκη, δηλαδή:

for (εντολές_ανάθεσης; ; έκφραση-τροποποίησης) εντολή;
for (εντολές_ανάθεσης; ; έκφραση-τροποποίησης) σύνθετη-εντολή

Για να διακοπεί ένας ατέρμονος βρόχος, διαφορετικά το πρόγραμμα δεν θα τερματίζει ποτέ, θα πρέπει στην εκτελούμενη εντολή να έχουμε προβλέψει μια συνθήκη που όταν ικανοποιείται θα εκτελείται η εντολή break η οποία θα τερματίζει την εκτέλεση του βρόχου.

Στο Σχήμα 3.5 δίνεται το τμήμα ενός προγράμματος που υλοποιεί την μετατροπή ενός αλφαριθμητικού σε ακέραιο. Στη C ένα αλφαριθμητικό αναπαρίσταται από έναν μονοδιάστατο πίνακα του οποίου τα στοιχεία είναι χαρακτήρες. Η δήλωση της γραμμής 1 ορίζει τον πίνακα s ως έναν πίνακα 15 χαρακτήρων. Όταν είναι επιθυμητό να επεξεργαστούμε τον i-στο χαρακτήρα του πίνακα s, γράφουμε $s[i]$.

```
 0:  int i, n, sign;
 1:  char s[15];
 2:
 3:  for (i = 0; isspace(s[i]); i++);
 4:
 5:  sign = (s[i] == '-') ? -1 : 1;
 6:
 7:  if (s[i] == '+' || s[i] == '-') i++;
 8:
 9:  for (n = 0; isdigit(s[i]); i++) n = 10 * n + (s[i] - '0');
10:
11:  n *= sign;
```

Σχήμα 3.5: Κώδικας Μετατροπής Αλφαριθμητικού σε Ακέραιο.

Στη συνθήκη της εντολής for της γραμμής 3 εμφανίζεται η *isspace*(*s*[*i*]) η οποία είναι συνάρτηση της πρότυπης βιβλιοθήκης της C η δήλωση της οποίας βρίσκεται στο αρχείο επικεφαλίδα (< *ctype.h* >). Η συνάρτηση αυτή δέχεται ως όρισμα έναν χαρακτήρα και επιστρέφει 1 αν είναι λευκός χαρακτήρας και 0 διαφορετικά. Έτσι η συγκεκριμένη εντολή for αρχίζοντας από τον πρώτο χαρακτήρα του αλφαριθμητικού που είναι το *s*[0] διατρέχει το αλφαριθμητικό όσο συναντά λευκούς χαρακτήρες. Συνεπώς, με την ολοκλήρωση της for ο χαρακτήρας *s*[*i*] είναι ο πρώτος μη λευκός χαρακτήρας.

Στη συνέχεια ο χαρακτήρας αυτός εξετάζεται αν είναι το αρνητικό πρόσημο '-' και η μεταβλητή *sign* παίρνει την τιμή −1 σε περίπτωση που είναι διαφορετικά την τιμή 1. Στην εντολή της γραμμής 7 αυξάνεται η τιμή της μεταβλητής *i* δείκτη κατά ένα στην περίπτωση που ο τρέχον χαρακτήρας είναι ένας εκ των '+' ή '-'.

Στη συνθήκη της εντολής for της γραμμής 9 εμφανίζεται η *isdigit*(*s*[*i*]) η οποία είναι συνάρτηση της πρότυπης βιβλιοθήκης της C η δήλωση της οποίας βρίσκεται στο αρχείο επικεφαλίδα (< *ctype.h* >). Η συνάρτηση αυτή δέχεται ως όρισμα έναν χαρακτήρα και

επιστρέφει 1 αν είναι ψηφίο και 0 διαφορετικά. Με την έναρξη εκτέλεσης της εντολής for αρχικοποιείται η μεταβλητή $n = 0$, εξετάζεται ο χαρακτήρας $s[i]$ αν είναι ψηφίο και στην περίπτωση που είναι επανακαθορίζεται η τιμή της μεταβλητής n. Στη συνέχεια αυξάνεται η μεταβλητή i κατά 1 και η εκτέλεση της εντολής for συνεχίζεται με τον επόμενο χαρακτήρα. Έστω για παράδειγμα ότι το αλφαριθμητικό s είναι "-45" τότε όταν η ροή του προγράμματος βρίσκεται στην εντολή της γραμμής 9, οι μεταβλητές του προγράμματος έχουν τις εξής τιμές:

$$\begin{aligned} sign &= -1; \\ i &= 1; \\ n &= 0; \end{aligned}$$

Εξετάζεται ο χαρακτήρας $s[1]$ που είναι ο $'4'$ και επειδή είναι ψηφίο εκτελείται η έκφραση $n = n*10+(s[1]-'0')$ όπου το γινόμενο είναι 0 και η διαφορά $s[1]-'0'$ ισούται με 4 αφού ο ASCII αριθμός του $'4'$ είναι ο 52 και ο ASCII αριθμός του $'0'$ είναι ο 48. Τελικά, στο βήμα αυτό η μεταβλητή n παίρνει την τιμή 4 και η i την τιμή 2. Επαναξετάζεται το $s[2]$ και επειδή είναι το ψηφίο $'5'$ εκτελείται η έκφραση $n = n * 10 + (s[2] - '0')$ όπου στο δεξί μέρος η μεταβλητή n έχει την τιμή 4 οπότε αντικαθιστώντας τις τιμές στην έκφραση έχουμε $n = 4 * 10 + (53 - 48)$ όπου το 53 είναι ο ASCII αριθμός του $'5'$, δηλαδή $n = 45$. Αυξάνεται η μεταβλητή i κατά ένα ($i = 3$) και εξετάζεται ο χαρακτήρας $s[3]$ του αλφαριθμητικού s. Επειδή δεν είναι ψηφίο ($s[3] = '\0'$) η εντολή for τερματίζει και η ροή του προγράμματος περνά στην εντολή της γραμμής 11 όπου η τιμή της μεταβλητής n μετατρέπεται από θετική σε αρνητική αφού $sign = -1$.

Στο Σχήμα 3.6 δίνονται δύο ισοδύναμες εντολές for οι οποίες αντιστρέφουν ένα αλφαριθμητικό, δηλαδή το "hello" γίνεται "olleh". Στο τμήμα αρχικοποίησης της for που βρίσκεται στο πάνω μέρος του σχήματος εμφανίζεται η συνάρτηση της πρότυπης βιβλιοθήκης (αρχείο επικεφαλίδα $<$ *string.h* $>$) *strlen*() η οποία δέχεται ως όρισμα ένα αλφαριθμητικό και επιστρέφει το πλήθος των χαρακτήρων του αλφαριθμητικού. Επιπρόσθετα, με τη

χρήση του τελεστή κόμμα τόσο στο τμήμα αρχικοποίησης όσο και στο τμήμα αύξησης των δεικτών *i* και *j* δίνεται η δυνατότητα εκτέλεσης δύο ή περισσοτέρων εντολών. Η σύνθετη εντολή του βρόχου αντιμεταθέτει τον *i*-στο χαρακτήρα του αλφαριθμητικού με τον *j*-στο.

Επίσης, στη for στο κάτω μέρος του σχήματος έχουμε απλά μετατρέψει τις εντολές αντιμετάθεσης της ομάδας εντολών της for του πρώτου παραδείγματος από ξεχωριστές εντολές σε μια εντολή αποτελούμενη από εντολές αναθέσης χωρισμένες με τον τελεστή κόμμα.

```
 0:  int c, i, j;
 1:
 2:  //Version 1
 3:  for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
 4:    c = s[i];
 5:    s[i] = s[j];
 6:    s[j] = c;
 7:  }
 8:
 9:  // Version 2
10:  for (i = 0, j = strlen(s)-1; i < j; i++, j--)
11:    c = s[i], s[i] = s[j], s[j] = c;
```

Σχήμα 3.6: Αντιστροφή Αλφαριθμητικού. Χρήση του τελεστή κόμμα

3.4.3. Η εντολή do-while

Μέχρι τώρα είδαμε ότι τόσο στη for όσο και στη while η αποτίμηση της έκφρασης που καθορίζει το αν θα εκτελεστεί η όχι η εντολή του βρόχου γίνεται πάντα στην αρχή. Η C διαθέτει μια ακόμα εντολή επανάληψης την **do-while**, όπου ο έλεγχος λαμβάνει χώρα αφού εκτελεσθεί η εντολή του βρόχου μια φορά, δηλαδή στο τέλος.

Η σύνταξη της εντολής έχει ως εξής:

do εντολή; **while** έκφραση;
do σύνθετη-εντολή **while** έκφραση;

Στο Σχήμα 3.7 δίνεται τμήμα προγράμματος που μετατρέπει ένα ακέραιο στο αντίστοιχο αλφαριθμητικό. Ας εξετάσουμε τις εντολές διεξοδικά. Με τις δηλώσεις των γραμμών 0 και 1 ορίζονται οι μεταβλητές i, n, sign τύπου int και ο πίνακας s 15 στοιχείων τύπου χαρακτήρα. Με την εντολή 3, ένας αρνητικός αριθμός μετατρέπεται στον αντίστοιχο θετικό.

```
0:  int i=0, n, sign;
1:  char s[15];
2:
3:  if ((sign = n) < 0) n = -n;
4:
5:  do {
6:      s[i++] = n % 10 + '0';
7:  } while ((n /= 10) > 0);
8:  if (sign < 0) s[i++] = '-';
9:  s[i] = '\0';
```

Σχήμα 3.7: Κώδικας Μετατροπής Ακεραίου σε αλφαριθμητικό.

Στις γραμμές 5 – 7 παρατίθενται οι εντολές της do-while. Εκτελείται πρώτα η εντολή της γραμμής 6 όπου αρχικά υπολογίζεται το υπόλοιπο της διαίρεσης του ακεραίου n με το 10 και στη συνέχεια αυτο προστίθεται στον ASCII αριθμό που αντιστοιχεί στο '0' υπολογίζοντας κατά αυτόν τον τρόπο τον ASCII αριθμό που αντιστοιχεί στο υπόλοιπο. Ο ASCII αριθμός του υπολοίπου αποθηκεύεται στο στοιχείο i του πίνακα s και ο δείκτης i αυξάνεται κατά 1 αμέσως μετά. Μετά την εκτέλεση της εντολής αυτής εκτελείται η εντολή

αντικατάστασης $n/ = 10$ και στη συνέχεια ελέγχεται αν το αποτέλεσμα της είναι αριθμός μεγαλύτερος του 0. Αν είναι τότε επαναλαμβάνεται η εκτέλεση της εντολής της γραμμής 6 διαφορετικά η ροή του προγράμματος συνεχίζεται με την εντολή της γραμμής 8 όπου στην περίπτωση που ο ακέραιος είναι αρνητικός ανατίθεται στο στοιχείο i του πίνακα s ο χαρακτήρας −. Τέλος, με την εντολή της γραμμής 9 τερματίζεται το αλφαριθμητικό s με τον χαρακτήρα $'\backslash 0'$.

Για παράδειγμα η εκτέλεση του παραπάνω κώδικα για $n = -45$ έχει ως αποτέλεσμα τα στοιχεία του πίνακα s να έχουν τις εξής τιμές:

$$\begin{aligned} s[0] &= '5' \\ s[1] &= '4' \\ s[2] &= '-' \\ s[3] &= '\backslash 0' \end{aligned}$$

3.4.4. Η εντολή continue

Μέχρι τώρα είδαμε πως για να διακοπεί η εκτέλεση της ομάδας εντολών μια εντολής επανάληψης πρόωρα, δηλαδή πριν η συνθήκη που επιτρέπει την επαναληπτική εκτέλεση της ομάδας εντολών γίνει ψευδής, μπορούσε να επιτευχθεί με την εντολή break.

Στο Σχήμα 3.8 δίνεται ένα παράδειγμα χρήσης της εντολής break. Η εντολή η οποία επαναλαμβάνεται στη for είναι μια εντολή απόφασης if. Στην i-στη επανάληψη ο i-στος χαρακτήρας του αλφαριθμητικού s εξετάζεται αν είναι λευκός χαρακτήρας. Αν ποτέ δεν ικανοποιηθεί η συνθήκη της if τότε η εντολή for θα τερματίσει όταν εξαντληθούν όλοι οι χαρακτήρες του αλφαριθμητικού. Αν όμως κάποιος χαρακτήρας είναι λευκός τότε θα εκτελεσθεί η break οπότε θα τερματίσει την εκτέλεση της for.

```
 0:  \\Break Command
 1:
 2:  for (n = strlen(s)-1; n >= 0; n--)
 3:      if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n') break;
 4:
 5:  s[n+1] = '\0';
 6:
 7:  \\Continue Command
 8:
 9:  for (i = 0; i < n; i++){
10:   if (a[i] < 0) continue;
11:    ...
12:  }
```

Σχήμα 3.8: Χρήση των εντολών break και continue.

Όπως αναφέρθηκε και στο εδάφιο 3.4.2 η εντολή που επαναλαμβάνεται στη for μπορεί να είναι και σύνθετη εντολή. Η εντολή continue επιτρέπει την πρόωρη έναρξη της επόμενης επανάληψης σε μια εντολή επανάληψης, δηλαδή δεν εκτελούνται οι εντολές που ακολουθούν την continue. Έστω ότι καλούμαστε να επεξεργαστούμε μόνο τους θετικούς αριθμούς από ένα πίνακα ακεραίων a. Στο Σχήμα 3.8 στο κάτω μέρος δίνεται η εντολή for στην οποία γίνεται χρήση της εντολής continue προκειμένου να μην εκτελεστούν οι εντολές που την ακολουθούν για τους αρνητικούς ακεραίους του πίνακα a. Η εντολή continue μπορεί να χρησιμοποιηθεί και στις εντολές επανάληψης while και do-while.

3.5. Παραδείγματα

Στο εδάφιο αυτό παρουσιάζονται ενδεικτικά προγράμματα στα οποία εφαρμόζονται όλα όσα αναφέρθηκαν στα κεφάλαια 2 και 3.

3.5.1. Υπολογισμός Πολικών Συντεταγμένων σημείου

Έστω ότι θέλουμε να αναπτύξουμε ένα πρόγραμμα που θα διαβάζει τις καρτεσιανές συντεταγμένες ενός σημείου στο επίπεδο στη μορφή (x, y), θα υπολογίζει και θα τυπώνει τις πολικές συντεταγμένες του σημείου στη μορφή (r, θ) με την γωνία θ στο διάστημα $[0, 2\pi)$. Το πρόγραμμα δεν θα υπολογίζει τις πολικές συντεταγμένες για τα σημεία που βρίσκονται στον άξονα y, δηλαδή όταν $x = 0$.

Η είσοδος του προγράμματος μας είναι οι καρτεσιανές συντεταγμένες του σημείου x, y και η έξοδος του οι πολικές συντεταγμένες. Δεδομένου ότι οι καρτεσιανές συντεταγμένες θα πρέπει να διαβάζονται με την μορφή (x, y) το αλφαριθμητικό μορφοποίησης της συνάρτησης scanf θα έχει τη μορφή που παρουσιάζεται στο Σχήμα 3.9 στη γραμμή 10. Δεδομένου ότι δεν είναι επιθυμητό να υπολογισθούν οι πολικές συντεταγμένες για τα σημεία του άξονα y καθίσταται αναγκαίος η εξέταση της τιμής της μεταβλητής x όπως φαίνεται με την εντολή απόφασης if-else των γραμμών 12 – 18. Εάν το σημείο ανήκει στον άξονα y απλά τυπώνεται το μήνυμα της γραμμής 18 διαφορετικά εκτελείται η ομάδα εντολών της εντολής if.

```
 0:  #include <stdio.h>
 1:  #include <math.h>
 2:
 3:  #define PI 3.14159
 4:
 5:  void main()
 6:  {
 7:     float x=0, y=0, r, angle;
 8:
 9:     printf("Dvste tis Cartesian syntetagmenes\n");
10:     scanf("(%g, %g)", &x, &y);
11:
12:     if (x){
13:        r = sqrt((pow(x,2) + pow(y,2)));
14:        angle = atan2(y, x);
15:        angle += (x<0) ? PI : ((y>=0)? 0 : 2*PI);
16:         printf("(%g, %g)\n", r, angle);
17:     }
18:     else printf("To shmeio anhkei sto axona y");
19:
20:  }
```

Σχήμα 3.9: Υπολογισμός Πολικών Συντεταγμένων σημείου

Αρχικά πρέπει να υπολογισθεί η απόσταση $r = \sqrt{x^2 + y^2}$ του σημείου από την αρχή των αξόνων. Για τον υπολογισμό της χρειάζονται δύο συναρτήσεις της πρότυπης μαθηματικής βιβλιοθήκης με αρχείο επικεφαλίδα $< math.h >$ το οποίο συμπεριλαμβάνεται στην αρχή του προγράμματος η sqrt() και η pow(). Η sqrt() δέχεται ως όρισμα έναν αριθμό κινητής υποδιαστολής διπλής ακρίβειας και επιστρέφει την τεταγωνική ρίζα του ορίσματος ενώ δέχεται δύο ορίσματα x και y όπου και τα δύο είναι αριθμοί κινητής υποδιαστολής διπλής ακρίβειας και επιστρέφει το αποτέλεσμα του υπολογισμού x^y. Ο υπολογισμός της απόστασης με την χρήση των παραπάνω συναρτήσεων φαίνεται στην γραμμή 13.

Για τον υπολογισμό της γωνίας θ χρησιμοποιείται η παρακάτω εξίσωση:

$$\theta = \begin{cases} \arctan(\frac{y}{x}) & if\ x > 0\ and\ y \geq 0 \\ \arctan(\frac{y}{x}) + 2\pi & if\ x > 0\ and\ y < 0 \\ \arctan(\frac{y}{x}) + \pi & if\ x < 0 \end{cases}$$

Για την πραγματοποίηση του παραπάνω υπολογισμού χρησιμοποιούμε τη συνάρτηση atan2() της πρότυπης μαθηματικής βιβλιοθήκης η οποία δέχεται δύο ορίσματα κινητής υποδιαστολής διπλής ακρίβειας y, x και επιστρέφει το τόξο εφαπτομένης $\frac{y}{x}$. Στη γραμμή 14 φαίνεται η εφαρμογή της συνάρτησης αυτής. Στη γραμμή 15 ολοκληρώνεται ο υπολογισμός της γωνίας με τη βοήθεια του τελεστή συνθήκη με τον οποίο υλοποιούνται και οι τρεις επιλογές της παραπάνω εξίσωσης.

3.5.2. Κωδικοποίηση Αλφαριθμητικού

Έστω ότι θέλουμε να αναπτύξουμε ένα πρόγραμμα που θα κωδικοποιεί την ακολουθία χαρακτήρων (plaintext string) “Attack at Dawn” σύμφωνα με την συνάρτηση $f(x) = 65 + (5x + 4)\ mod\ 26$ όπου x είναι ο ASCII κωδικός κάθε χαρακτήρα της ακολουθίας και θα τυπώνει την κωδικοποιημένη ακολουθία (ciphered string). Δεν θα κωδικοποιούνται οι λευκοί χαρακτήρες και δεν θα εμφανίζονται στην έξοδο.

```
 0: #include <stdio.h>
 1:
 2: main()
 3: {
 4:     char p[]="Attack At Dawn";
 5:     int i;
 6:
 7:     for(i=0; p[i]!='\0'; i++){
 8:         char x;
 9:
10:         switch(p[i]){
11:             case '\n':
12:             case '\t':
13:             case ' ': break;
14:             default:
15:                 x=65 + (5*p[i]+4) % 26;
16:                 printf("%c", x);
17:                 break;
18:         }
19:     }
20:     printf("\n");
21: }
```

Σχήμα 3.10: Κωδικοποίηση Αλφαριθμητικού

Πριν προχωρήσουμε στην ανάλυση του προγράμματος του Σχήματος 3.10 ας αναλύσουμε τη συνάρτηση $f(x)$ όπου x χαρακτήρας. Όπως αναφέραμε και στο κεφάλαιο

2 η τιμή μιας μεταβλητής τύπου χαρακτήρα είναι ο ASCII αριθμός που αντιστοιχεί στο χαρακτήρα αυτό. Έτσι για παράδειγμα για τον χαρακτήρα $'D'$ του αλφαριθμητικού μας η μεταβλητή x θα έχει την τιμή 68. Συνεπώς, αν αντικαταστήσουμε την μεταβλητή x στο τμήμα της συνάρτησης $(5\times x+4)$ *mod* 26 θα υπολογισθεί ο αριθμός $(5\times 68+4)$ *mod* $26 = 6$ δηλαδή το έβδομο γράμμα του λατινικού αλφαβήτου που είναι το $'G'$ (το έβδομο γιατί η αρίθμηση αρχίζει από το 0). Για να πάρουμε όμως τον ASCII αριθμό που αντιστοιχεί στον χαρακτήρα αυτό προσθέτουμε τον αριθμό 65 που αντιστοιχεί στον χαρακτήρα $'A'$. Με άλλα λόγια όλοι οι χαρακτήρες του αλφαριθμητικού θα κωδικοποιηθούν με κεφαλαία γράμματα.

Επιπρόσθετα από τις προδιαγραφές του προβλήματος δεν πρέπει να κωδικοποιηθούν οι λευκοί χαρακτήρες. Για να επιτευχθεί αυτό μπορούμε να ελέγξουμε κάθε χαρακτήρα του αλφαριθμητικού με την εντολή switch όπως φαίνεται στις γραμμές 10 – 18 και αν είναι λευκός χαρακτήρας να μην κωδικοποιηθεί διαφορετικά να υπολογισθεί ο ο ASCII αριθμός του νέου χαρακτήρα που προκύπτει.

Συνεπώς, το πρόβλημα μας λύνεται με μια εντολή for αυτή των γραμμών 7 – 19 η οποία σαρώνει όλους τους χαρακτήρες του αλφαριθμητικού και για κάθε έναν από αυτούς εκτελεί την εντολή switch των γραμμών 10 – 18. Αξιοσημείωτο είναι η δήλωση εντός της σύνθετης εντολής της for στη γραμμή 8 καθώς σε κάθε ομάδα εντολών μπορούν στη αρχή να δηλωθούν κάποιες μεταβλητές που θα χρησιμοποιηθούν εντός της ομάδας εντολών της σύνθετης εντολής.

3.5.3. Περιστροφή των bit ακεραίου

Έστω ότι θέλουμε να αναπτύξουμε ένα πρόγραμμα που θα δέχεται ως είσοδο δύο μη προσημασμένους ακέραιους αριθμούς x και n, θα περιστρέφει τα bits του αριθμού x κατά n θέσεις αριστερά και θα τυπώνει τον ακέραιο x και τον αριθμό που προκύπτει από την εν λόγω περιστροφή των bits σε δεκαεξαδική μορφή. Κατά την αριστερή περιστροφή των bits ενός ακέραιου αριθμού κατά n θέσεις, ο αριθμός μετατοπίζεται αριστερά κατά

n θέσεις και στα n λιγότερα σημαντικά bits που ελευθερώνονται κατά τη μετατόπιση εισέρχονται τα n περισσότερα σημαντικά bits που εξέρχονται κατά την μετατόπιση.

```
 0:  #include <stdio.h>
 1:
 2:  void main()
 3:  {
 4:     unsigned int x, n, y;
 5:
 6:     printf("Dvste ton akeraio kai to plhthos tvn thesevn\n");
 7:     scanf("%u %u", &x, &n);
 8:
 9:     y = (x<<n)|(x >> (sizeof(int)*8 - n));
10:     printf("%0x %0x\n", x, y);
11:  }
12:
```

Σχήμα 3.11: Περιστροφή των bit ακεραίου

Η αριστερή μετατόπιση κατά n θέσεις της τιμής της μεταβλητής x μπορεί να επιτευχθεί με την έκφραση $x \ll n$. Έστω για παράδειγμα ότι $n = 2$ και $x = 11001010$ τότε το αποτέλεσμα της αριστερής μετατόπισης κατά δύο θέσεις θα είναι το 0010100. Για να ολοκληρωθεί η αριστερή περιστροφή του αριθμού x θα πρέπει τα n περισσότερο σημαντικά bit που εξέρχονται κατά την μετατόπιση να εισέλθουν στις n λιγότερο σημαντικές θέσεις. Για να γίνει αυτό πολύ απλά μετατοπίζουμε δεξιά κατά $8 \times sizeof(int) - n$ την τιμή της μεταβλητής x (όπου $8 \times sizeof(int)$ είναι το πλήθος των bit ενός ακεραίου τύπου int), δηλαδή $x \gg (8 \times sizeof(int) - n)$ και στη συνέχεια παίρνουμε το λογικό | των εκφράσεων $x \ll n$ και $x \gg (8 \times sizeof(int) - n)$, δηλαδή $x \ll n | x \gg (8 \times sizeof(int) - n)$ όπως φαίνεται στην εντολή της γραμμής.

3.5.4. Υπολογισμός συμπληρώματος ως προς 2 ακεραίου

Στο αριθμητικό σύστημα αναπαράστασης n-bit ακέραιων αριθμών υπο μορφή συμπληρώματος ως προς 2, οι θετικοί αριθμοί $[0 \ldots 2^{n-1}]$ αναπαριστάνονται ως μη προσημασμένοι ακέραιοι και οι αρνητικοί αριθμοί αναπαριστάνονται με το συμπλήρωμα ως προς 2 του αντίστοιχου θετικού αριθμού.

```
 0: #include <stdio.h>
 1:
 2: #define MASK8  0xff /*255*/
 3: #define LEASTMASK 0x01
 4:
 5:
 6: void main()
 7: {
 8:    unsigned char x;
 9:
10:    printf("Positive Numbers\t Negative Numbers\n");
11:    printf("\t (%d) %0x \t\t\t \n", 0, 0);
12:    for(x=1; x<128; x++){
13:       unsigned char i, y;
14:
15:       i=1;
16:       while (!((x >> (i-1)) & LEASTMASK)) i++;
17:       y = x ^ (MASK8 << i);
18:       printf("\t (%d) %0x \t\t\t (-%d) %x\n", x, x, x, y);
19:    }
20:    printf("\t \t \t\t\t (-%d) %x\n", x, x);
21: }
```

Σχήμα 3.12: Υπολογισμός συμπληρώματος ως προς 2 ακεραίου

Ένας πρακτικός αλγόριθμος για την εύρεση της αναπαράστασης συμπληρώματος ως

προς 2 ενός ακέραιου αριθμού είναι η εύρεση αρχικά του πρώτου μη μηδενικού bit ξεκινώντας την αναζήτηση από το λιγότερο σημαντικό bit, και στη συνέχεια η αντιστροφή των bits του αριθμού που βρίσκονται μετά από το πρώτο αυτό μη μηδενικό bit.

Στο Σχήμα 3.12 παρουσιαζεται ένα πρόγραμμα που τυπώνει σε μορφή συμπληρώματος ως προς 2 σε μια στήλη τους θετικούς αριθμούς των 8-bit και σε μια δεύτερη τους αρνητικούς. Για κάθε αριθμό εμφανίζεται εντός παρενθέσεως η δεκαδική τιμή του αριθμού και δίπλα σε δεκαεξαδική μορφή, π.χ. $(-1)FF$.

Στην εντολή for για κάθε θετικό ακέραιο από $1 \ldots 128$ υπολογίζεται το συμπλήρωμα του ως προς 2 με τον αλγόριθμο που αναλύθηκε στην προηγούμενη παράγραφο. Με την εντολή while της γραμμής 16 εντοπίζεται η θέση του πρώτου μη μηδενικού bit του αριθμού μετατοπίζοντας τον κάθε φορά μια θέση δεξιά και εξετάζοντας (πέρνωντας το λογικό ΚΑΙ του μετατοπισμένου αριθμού και της σταθεράς LEFTMASK που έχει 1 στο λιγότερο σημαντικό bit) αν το λιγότερο σημαντικό bit είναι μη μηδενικό. Όταν τερματισθεί η εντολή while η μεταβλητή i έχει τη θέση του πρώτου μη μηδενικού bit. Απομένει να διατηρήσουμε τα i λιγότερο σημαντικά bit και αντιστρέψουμε τα υπόλοιπα. Για το σκοπό αυτό έχει ορισθεί η σταθερά MASK8 που έχει 1 και στα 8-bit και κάθε φορά μετατοπίζεται αριστερά i θέσεις δίνοντας μια μάσκα της μορφής:

$$11 \ldots 1 \underbrace{0 \ldots 0}_{i}$$

Στη συνέχεια εφαρμόζεται η πράξη αποκλειστικό-Η μεταξύ της παραπάνω μάσκας και του αριθμού. Τα i λιγότερο σημαντικά bit παραμένουν αναλοίωτα καθώς για μια δυαδική μεταβλητή $x \in \{0, 1\}$ ισχύει $x \oplus 0 = x$ ενώ τα υπόλοιπα αντιστρέφονται καθώς $x \oplus 1 = x'$. Συνεπώς η εντολή της γραμμής 17 υπολογίζει το συμπλήρωμα ως προς 2 του αριθμού.

3.6. Ασκήσεις

Άσκηση 3.6.1 *Να γραφούν τρία προγράμματα που θα διαβάζουν την τιμή της ακέραιας μεταβλητής x και ανάλογα με την τιμή της θα εκτελούνται οι εντολές s1, s2, s3 σύμφωνα με τον παρακάτω πίνακα:*

x	Πρόγραμμα 1	Πρόγραμμα 2	Πρόγραμμα 3
1	s1	s2, s3	s3
2	s3	s1	s2
3	s1	s2	s1, s2

Στα προγράμματα αυτά όταν για να δείξετε ότι εκτελείτε για παράδειγμα η εντολή s3 για κάποια τιμή της μεταβλητής x θα γράψετε

printf("Ekteleite h entolh s3");

Άσκηση 3.6.2 *Να γραφεί πρόγραμμα το οποίο θα περιέχει την παρακάτω εντολή επανάληψης*

```
while(scanf("%lf", &capital) == 1){
...
}
```

Στο σώμα της εντολής επανάληψης θα υπολογίζεται η απόδοση της κατάθεσης του κεφαλαίου capital στο τέλος του χρόνου όταν το ετήσιο επιτόκιο είναι 8% λαμβάνοντας υπόψη ότι οι τόκοι φορολογούνται με 1% και θα τυπώνει το κεφάλαιο, του τόκο, του φόρο και το υπόλοιπο του λογαριασμού.

Άσκηση 3.6.3 *Έστω w ένας θετικός πραγματικός αριθμός και η ακολουθία πραγματικών αριθμών a_i η οποία ορίζεται από τις παρακάτω σχέσεις:*

$$a_0 = 1$$

και

$$a_{i+1} = \frac{1}{2}(a_i + \frac{w}{a_i})$$

Σύμφωνα με την μέθοδο Newton-Raphson όταν $i \to \infty$ τότε $a_i \to \sqrt{w}$.

Να γραφεί πρόγραμμα που θα διαβάζει τον αριθμό w θα κάνει έλεγχο αν είναι θετικός αριθμός και σε περίπτωση που είναι θα υπολογίζει και θα τυπώνει τους όρους της παραπάνω ακολουθίας καθώς και την ακρίβεια $e_i = w - a_i^2$. Στη συνέχεια να τροποποιήσετε τον κώδικα έτσι ώστε να διαβάζει και την ακρίβεια υπολογισμού e και το πρόγραμμα θα τερματίζει όταν $e_i \leq e$.

Άσκηση 3.6.4 *Να γραφεί πρόγραμμα που θα διαβάζει τον συντελεστές a, b, c του δευτεροβάθμιου πολυωνύμου*

$$ax^2 + bx + c$$

θα υπολογίζει και θα τυπώνει τις ρίζες του. Μαζί με τις ρίζες του πολυωνύμου θα τυπώνουται οι τιμές των συντελεστών καθώς και η τιμή της διακρίνουσας.

Άσκηση 3.6.5 *Τα τμήματα κώδικα του Σχήματος 3.13 να γραφούν με τέτοιο τρόπο έτσι ώστε να αποφευχθούν οι εντολές break και continue.*

```
0:  while (c=getchar()){
1:      if (isdigit(c)) break;
2:      ++cnt;
3:      if (islower(c)) printf("%c", toupper(c));
4:  }
5:
6:  j=-10;
7:  n=65; sum=0;
8:  while (j<n) {
9:      j++;
10:     if ((j%2)==0) continue;
11:     sum += j;
12:     printf("j= %d kai athroisma = %d\n", j, sum);
13: }
```

Σχήμα 3.13: Κώδικας άσκησης 3.6.5

Κεφάλαιο 4

Συναρτήσεις, Αρθρωτός Προγραμματισμός

Στα παραδείγματα που είδαμε μέχρι τώρα, το πρόγραμμα μας αποτελούνταν από μια μόνο συνάρτηση την κύρια συνάρτηση main στο σώμα της οποίας καλούνταν κάποιες συναρτήσεις της πρότυπης βιβλιοθήκης οι οποίες υλοποιούσαν κάποια πολύ συγκεκριμένα καθήκοντα. Για παραδειγμα για την είσοδο και έξοδο δεδομένων καλούνταν οι συναρτήσεις scanf() και printf() ενώ στο παράδειγμα της μετατροπής των καρτεσιανών συντεταγμένων ενός σημείου στις αντίστοιχες πολικές χρησιμοποιήθηκαν οι μαθηματικές συναρτήσεις sqrt(), pow() και atan2().

Απλά προβλήματα μπορούν να λυθούν με την ανάπτυξη του προγράμματος σε μια συνάρτηση αυτής της κυρίας συνάρτησης καλώντας κάθε φορά τις κατάλληλες συναρτήσεις της πρότυπης βιβλιοθήκης (επίπεδος προγραμματισμός). Το ίδιο μπορεί να συμβεί και με σύνθετα προβλήματα με τη διαφορά ότι στη περίπτωση αυτή καθίσταται επώδυνη η αποσφαλμάτωση του προγράμματος καθώς και η συντήρηση και επέκταση του. Είναι

δυνατό για μια πολύ μικρή αλλαγή στο πρόγραμμα να προκληθεί ριζική αναδιοργάνωση στο πρόγραμμα η οποία στη συνέχεια μπορεί να οδηγήσει σε μια επώδυνη διαδικασία αποσφαλμάτωσης.

Έτσι η επίλυση σύνθετων προβλήματων μπορεί να επιτευχθεί με αναγωγή σε μικρότερα απλά προβλήματα τα αποτελέσματα των οποίων στο τέλος όταν συνδυαστούν δίνουν τη λύση στο αρχικό πρόβλημα. Η διαδικασία αυτή της αναγωγής ενός σύνθετου προβλήματος σε μικρότερα και απλούστερα επιμέρους προβλήματα ονομάζεται **αποσύνθεση (decomposition)**. Με ποιά μέθοδο μπορεί κανείς να ξεκινήσει την αποσύνθεση ενός σύνθετου προβλήματος; Η πιο γνωστή μέθοδος σχεδιασμού ενός προγράμματος που επιλύει ένα πρόβλημα είναι αυτή της **από επάνω προς τα κάτω σχεδίασης (top down design)**. Ξεκινάμε από το κυρίως πρόγραμμα και επιχειρούμε να εντοπίσουμε τα συστατικά του μέρη του. Στη συνέχεια αυτά τα συστατικά μέρη του προγράμματος που ενδεχομένως να είναι πολύπλοκα μπορούν να υποδιαιρεθούν περαιτέρω. Η υποδιαίρεση συνεχίζεται μέχρι να φτάσουμε στα μικρότερα δυνατά μέρη.

Η C διαθέτει τις συναρτήσεις και τον μηχανισμό κλήσης συναρτήσεων με τα οποία μπορεί εύκολα κανείς να δομήσει ένα τελικό πρόγραμμα υλοποιώντας τα επιμέρους συστατικά μέρη τα οποία προέκυψαν άπο τη φάση της αποσύνθεσης με τη μέθοδο σχεδίασης από πάνω προς τα κάτω. Όπως αναφέρθηκε στο κεφάλαιο 2, οι συναρτήσεις ενός προγράμματος C μπορούν να βρίσκονται είτε σε ένα πηγαίο αρχείο (source file) είτε σε ένα σύνολο από πηγαία αρχεία. Ο διαμοιρασμός του πηγαίου κώδικα σε περισσότερα από ένα αρχεία και η οργάνωση του έτσι ώστε οι οριζόμενες συναρτήσεις να μπορούν επαναχρησιμοποιηθούν από άλλα προγράμματα ονομάζεται **Αρθρωτός Προγραμματισμός (Modular Programming)**. Το κάθενα από αυτά τα αρχεία ονομάζεται module και το αρχείο που περιέχει την κύρια συνάρτηση main ονομάζεται main module.

Έστω ότι στα πλαίσια μιας εφαρμογής κρυπτογραφίας κατασκευάσαμε ένα σύνολο συναρτήσεων για αριθμητική των 128-bit (16 byte) τις οποίες τοποθετήσαμε σε ένα module με όνομα *math_128.c* και τη συνάρτηση main μαζί με τις υπόλοιπες συναρτήσεις της εφαρμογής στο *module crypto.c*. Κατά τη μετάφραση του αρθρωτού προγράμματος

σε command line γράφουμε: *cc crypto.c math_128.c*. Στη συνέχεια στα πλαίσια κάποιας άλλης εφαρμογής μπορεί να είναι επιθυμητή η κλήση συναρτήσεων για αριθμητική των 128-bit (16 byte) τις οποίες ήδη υλοποιήσαμε για την κρυπτογραφική εφαρμογή. Το module math_128.c που ήδη αναπτύχθηκε μπορεί να χρησιμοποιηθεί από τη νέα εφαρμογή αρκεί να γίνουν στα module της εφραμογής οι κατάλληλες δηλώσεις των συναρτήσεων του module math_128.c. Κατά αυτόν τρόπο κτίζουμε την δική μας βιβλιοθήκη για μελλοντική χρήση.

Στα επόμενα εδάφια του κεφάλαιου αυτού θα αναλυθούν ο τρόπος σύνταξης των συναρτήσεων, ο μηχανισμός κλήσεων συναρτήσεων στη C καθώς και τον μηχανισμό οργάνωσης αρθρωτών προγραμμάτων.

4.1. Ορισμός Συνάρτησης

Μέχρι τώρα είχαμε περιοριστεί στη κύρια συνάρτηση ενός προγράμματος σε C τη main καθώς και σε κλήσεις συναρτήσεων της πρότυπης βιβλιοθήκης της C χωρίς να αναφερθούμε σε λεπτομέρειες που αφορούν τη δομή τους. Όπως προαναφέρθηκε ένας αλγόριθμος μπορεί να υλοποιηθεί με μια συνάρτηση αυτή της main και με τη χρήση κάποιων ίσως συναρτήσεων της πρότυπης βιβλιοθήκης. Σύνθετα όμως προβλήματα επιλύονται αποτελεσματικότερα με τη μέθοδο της αποσύνθεσης στην οποία τα επιμέρους υποπροβλήματα στα οποία διασπάται το βασικό πρόβλημα μπορεί να υλοποιηθεί από συναρτήσεις που μπορεί να ορίσει ο προγραμματιστής.

Η συνάρτηση μπορεί να θεωρηθεί ως μια αυτόνομη μονάδα προγράμματος η οποία δέχεται ως είσοδο ένα σύνολο δεδομένων, εκτελεί ένα σύνολο εντολών και επιστρέφει στη συνάρτηση που την κάλεσε μια τιμή. Στη C μια συνάρτηση αποτελείται από δύο μέρη: την επικεφαλίδα και το σώμα της το οποίο δεν είναι τίποτε άλλο παρά μια ομάδα εντολών όπως αυτή ορίσθηκε στο εδάφιο 3.1:

<ονομα_τύπου><όνομα_συνάρτησης> (<λίστα τυπικών παραμέτρων>)

```
{
    <δηλώσεις_μεταβλητών>;
    <εντολή-1>;
    ...
    <εντολή-n >;
    return(<έκφραση>);
}
```

Η επικεφαλίδα μιας συνάρτησης είναι η διεπαφή της συνάρτησης με το υπόλοιπο πρόγραμμα και είναι αυτό το οποίο χρειάζεται ένας προγραμματιστής για να την χρησιμοποιήσει. Με την επικεφαλίδα ορίζεται το όνομα της συνάρτησης, ο τύπος της τιμής την οποία επιστρέφει όταν κληθεί καθώς και μια λίστα παραμέτρων οι οποίες καλούνται **τυπικές παράμετροι** της συνάρτησης μέσω των οποίων η συνάρτηση δέχεται ως είσοδο δεδομένα, επιστρέφει δεδομένα ή και τα δύο. Αν παραληφθεί το <ονομα_τύπου> τότε υποννοείται ο int.

Η καλούμενη συνάρτηση επιστρέφει στην καλούσα μια τιμή μέσω ενός μηχανισμού που υλοποιείται με την εντολή return, η σύνταξη της οποίας έχει ως εξής: return(<έκφραση>); Το αποτέλεσμα υπολογισμού της έκφρασης μετατρέπεται αν χρειασθεί στον τύπο επιστροφής. Η ύπαρξη της έκφρασης είναι προαιρετική και μπορεί να περικλείεται σε παρενθέσεις. Ο έλεγχος του προγράμματος επιστρέφει στην καλούσα συνάρτηση με την εκτέλεση της εντολής return.

Στην ANSI C χρησιμοποιείται η λέξη κλειδί **void** για να δηλωθεί:

1. η απουσία τυπικών παραμέτρων σε μια συνάρτηση,
2. το ότι μια συνάρτηση δεν επιστρέφει τιμή,
3. ένας γενικός δείκτης με τον οποίο θα ασχοληθούμε στο επόμενο κεφάλαιο.

Μια συνάρτηση που δεν επιστρέφει κάποια τιμή ονομάζεται **διαδικασία**. Στην περίπτωση αυτή ο ορισμός της συνάρτησης έχει ως εξής:

```
void<όνομα_συνάρτησης> (<λίστα τυπικών παραμέτρων>)
{
    <δηλώσεις_μεταβλητών>;
    <εντολή-1>;
    ...
    <εντολή-n >;
    return(<έκφραση>);
}
```

Η επιστροφή του ελέγχου στην καλούσα συνάρτηση στην περίπτωση μιας διαδικασίας μπορεί να γίνει και χωρίς τη χρήση της εντολής return, απλά φτάνοντας στο δεξί άγκιστρο }.

Οι τυπικές παράμετροι στη λίστα παραμέτρων είναι ουσιαστικά οι μεταβλητές εισόδου της συνάρτησης. Η λίστα τυπικών παραμέτρων έχει τυπικά την μορφή:

(<τύπος_1><παράμετρος_1>, <τύπος_2><παράμετρος_2, . . . , <τύπος_i ><παράμετρος_i)

όπου τύπος_i είναι ο τύπος της i-οστης παραμέτρου. Στο σώμα της συνάρτησης οι τυπικές παραμέτρους μπορούν χρησιμοποιηθούν ως μεταβλητές. Είναι λοιπόν φανερό ότι οι παράμετροι μιας συνάρτησης υλοποιούν την επικοινωνία της συνάρτησης με τις υπόλοιπες συναρτήσεις του προγράμματος. Σε περίπτωση που ο σχεδιασμός μιας συνάρτησης δεν απαιτεί παραμέτρους, τότε στον ορισμό της εντός των παρενθέσεων εισάγεται η λέξη κλειδί void. Μια συνάρτηση, λοιπόν επικοινωνεί με τις υπόλοιπες συναρτήσεις μέσω των παραμέτρων και της επιστρεφόμενης τιμής.

Έστω για παράδειγμα ότι θέλουμε να γράψουμε μια συνάρτηση που θα υπολογίζει το άθροισμα $\sum_{i=0}^{k} \frac{x^i}{i!}$ όπου το x είναι ένας αριθμός κινητής υποδιαστολής διπλής ακρίβειας

και k είναι ένας ακέραιος απλής ακρίβειας μεγαλύτερος της μονάδας. Από την περιγραφή του προβλήματος, είναι φανερό ότι η συνάρτηση που θέλουμε να σχεδιάσουμε θα επιστρέφει έναν αριθμό κινητής υποδιαστολής διπλής ακρίβειας που είναι το αποτέλεσμα από τον υπολογισμό του αθροίσματος. Επιπρόσθετα, για να υπολογισθεί το άθροισμα απαιτούνται δύο τυπικές παράμετροι μια για τον ακέραιο k και μια για τη μεταβλητή x του αθροίσματος.

```
 0:  double calcSeries(int k, double x)
 1:  {
 2:     int i, fact;
 3:     double result, x_pow;
 4:
 5:     x_pow = 1;
 6:     fact = 1;
 7:     result = 1;
 8:     for(i=1; i<=k; i++){
 9:        x_pow *= x;
10:        fact *= i;
11:        result += (x_pow/fact);
12:     }
13:     return result;
14:  }
```

Σχήμα 4.1: Συνάρτηση Υπολογισμού της σειράς $\sum_{i=0}^{k} \frac{x^i}{i!}$

Στο Σχήμα 4.1 δίνεται η συνάρτηση με όνομα calcSeries που υπολογίζει το ζητούμενο άθροισμα. Η επικεφαλίδα της έχει δύο τυπικές παραμέτρους την k που είναι τύπου *int* και την x που είναι τύπου *double* και επιστρέφει μια τιμή που είναι τύπου *double*. Στο σώμα της συνάρτησης υπολογίζεται το άθροισμα. Για τις ανάγκες υπολογισμού του αθροίσματος χρειάζεται να ορίσουμε στο σώμα της συνάρτησης δύο ακέραιες μεταβλητές

και δύο μεταβλητές κινητής υποδιαστολής διπλής ακρίβειας τη χρήση των οποίων θα εξηγήσουμε στη συνέχεια. Οι μεταβλητές που δηλώνονται στο σώμα μιας συνάρτησης ονομάζονται **τοπικές μεταβλητές** της συνάρτησης και ισχύουν μόνο εντός του σώματος της συνάρτησης.

Καταρχήν για τον υπολογισμό του αθροίσματος πρέπει να υπολογίσουμε τους k όρους του αθροίσματος και στη συνέχεια να υπολογίσουμε το άθροισμα τους. Αυτό σημαίνει ότι χρειαζόμαστε μια εντολή επανάληψης που στην προκειμένη περίπτωση, που ο αριθμός των βημάτων που θα εκτελεσθούν είναι γνωστός και μάλιστα παράμετρος της συνάρτησης, η καταλληλότερη εντολή επανάληψης είναι η *for*. Χρειαζόμαστε λοιπόν μια μεταβλητή την *i* για τα βήματα στη *for* και μια μεταβλητή τη *result* για το αποτέλεσμα υπολογισμού του αθροίσματος.

Θα μπορούσαμε βέβαια σε κάθε βήμα της εντολής επανάληψης να υπολογίζουμε το $k!$ και το x^k αλλά προκειμένου να κάνουμε τη συνάρτηση μας πιο γρήγορη παρατηρούμε ότι ο κάθε όρος μπορεί να υπολογισθεί αρκεί να πολλαπλασιάσουμε το αριθμητή με x και τον παρονομαστή με την τιμή i του βήματος επανάληψης. Μετά και αυτές τις παρατηρήσεις προκύπτει η εντολή επανάληψης των γραμμών 8-12 του Σχήματος 4.1 όπου προέκυψε η ανάγκη για δύο επιπλέον μεταβλητές την *fact* και την *x_pow* για τον υπολογισμό του αριθμητή και του παρονομαστή κάθε όρου. Με την ολοκλήρωση της εντολής επανάληψης εκτελείται η εντολή *return* η οποία επιστρέφει την τιμή της μεταβλητής *result* που είναι το ζητούμενο άθροισμα.

4.2. Δήλωση πρωτοτύπου συνάρτησης

Στο προηγούμενο εδάφιο περιγράψαμε πως ορίζουμε μια συνάρτηση. Όπως αναφέραμε και στο κεφάλαιο 1, ένα πρόγραμμα σε C είναι ένα σύνολο συνεργαζόμενων συναρτήσεων. Επίσης, αναφέραμε ότι όλες οι συναρτήσεις μπορεί να βρίσκονται είτε στο ίδιο πηγαίο αρχείο είτε σε διαφορετικά. Όταν βρίσκονται σε διαφορετικά αρχεία, με ποιό τρόπο οι συναρτήσεις ενός αρχείου γνωρίζουν τους ορισμούς των συναρτήσεων που βρίσκονται σε

άλλα αρχεία; Όταν οι συναρτήσεις έχουν ορισθεί σε ένα αρχείο μπορούν να βρίσκονται είτε πριν την κύρια συνάρτηση main είτε μετά. Στη C οι συναρτήσεις που έχουν ορισθεί πριν την main μπορούν να χρησιμοποιηθούν στο σώμα της main ενώ τόσο αυτές που έχουν ορισθεί μετά τη main ή βρίσκονται σε άλλο αρχείο δεν μπορούν καθώς είναι άγνωστες για την main. Για το λόγο αυτό το πρωτότυπο μιας συνάρτησης πρέπει να δηλωθεί πριν από το σημείο της main από το οποίο πρόκειται να κληθεί.

Γενικεύοντας, μια συνάρτηση Α, που έχει ορισθεί σε άλλο αρχείο από αυτό στο οποίο έχει ορισθεί μια συνάρτηση Β ή που έχει ορισθεί στο ίδιο αρχείο αλλά μετά τη συνάρτηση Β, για να χρησιμοποιηθεί από τη Β χρειάζεται να δηλωθεί το πρωτότυπο της συνάρτησης πριν από το σημείο της συνάρτησης Β από το οποίο πρόκειται να κληθεί.

Όμως τι είναι το πρωτότυπο συνάρτησης; Στο προηγούμενο εδάφιο αναφέραμε ότι η επικεφαλίδα μιας συνάρτησης είναι ουσιαστικά η διεπαφή της συνάρτησης με τις υπόλοιπες συναρτήσεις. Η δήλωση του πρωτοτύπου μιας συνάρτησης δεν είναι τίποτε άλλο παρά η δήλωση της επικεφαλίδας χωρίς όμως το σώμα της. Στην ANSI C, τυπικά μια δήλωση πρωτοτύπου συνάρτησης έχει ως εξής:

<τύπος><όνομα_συνάρτησης> (<τύπος_1> <όνομα_παραμετρου_1>, ...);

και

<τύπος><όνομα_συνάρτησης> (*void*);

όταν η συνάρτηση δεν έχει παραμέτρους. Να τονισθεί ότι τα ονόματα των παραμέτρων στη δήλωση των πρωτοτύπων συναρτήσεων είναι πιο πολύ περιγραφικά και μπορούν να παραληφθούν. Αυτό όμως που προέχει είναι να υπάρχει αντιστοιχία μεταξύ των τύπων που εμφανίζονται στον ορισμό της συνάρτησης και στη δήλωση πρωτοτύπου συνάρτησης.

Μετά τον ορισμό του πρωτοτύπου συνάρτησης, επανερχόμαστε στο πρόβλημα που θέσαμε προηγουμένως για τις συναρτήσεις *A* και *B*. Με βάση αυτά που αναφέραμε παραπάνω η λύση στο πρόβλημα μας είναι η εξής:

```
<ονομα_τύπου> Α(<λίστα τυπικών παραμέτρων>);
<επικεφαλίδα συνάρτησης Β> {
        <δηλώσεις μεταβλητών>;
        <εντολή-1>;
        . . .
        <κλήση συνάρτησης A >;
        return(έκφραση);
}
```

Μια καλή προγραμματιστική τεχνική είναι οι δηλώσεις πρωτοτύπων των συναρτήσεων να τίθενται αμέσως μετά τις εντολές του προεπεξεργαστή και πριν την παράθεση των ορισμών των συναρτήσεων του πηγαίου αρχείου ή στο τμήμα δηλώσεων του σώματος της συνάρτησης που πρόκειται να χρησιμοποιηθούν. Οι δηλώσεις πρωτοτύπων για τις συναρτήσεις της πρότυπης βιβλιοθήκης βρίσκονται στα αρχεία επικεφαλίδας. Αυτός άλλωστε είναι και ο λόγος που συμπεριλαμβάνουμε τα αρχεία επικεφαλίδας στο πρόγραμμα μας όποτε πρόκειται να χρησιμοποιήσουμε μια συνάρτηση. Έτσι για παράδειγμα όταν θέλουμε να χρησιμοποιήσουμε τη συνάρτηση printf() πρέπει να συμπεριλάβουμε το αρχείο επικεφαλίδα < *stdio.h* > με τη χρήση της εντολής *#include* του προεπεξεργαστή.

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2:
 3: int isPerfect(int);
 4: int combinations(int n, int k);
 5: double calcSeries(int k, double x);
 6:
 7: int main()
 8: {
 9:    int n, k, choice, i;
10:    double x=0;
11:
12:    printf("Dvse ton arithmo ths askhshs:");
13:    scanf("%d", &choice);
14:    printf("\n");
15:    switch(choice){
16:      case 1:   printf("Dvse to x: ");
17:                scanf("%lf", &x);
18:                for(i=1; i<=10; i++)
19:                  printf("k0=%d --> Seira= %g\n", i, calcSeries(i,x));
20:                break;
21:      case 2:   for(i=1; i<=9999; i++)
22:                 if (isPerfect(i)) printf("%d\n", i);
23:                break;
24:      case 3:   printf("Dvse 2 akeraious: ");
25:                scanf("%d, %d", &n, &k);
26:                printf("C(%d,%d)=%d\n", n, k, combinations(n,k));
27:                break;
28:      default: break;
29:    }
30: }
31:
```

Σχήμα 4.2: Κυρίως Πρόγραμμα που καλεί τις συναρτήσεις των εδαφίων

Στο προηγούμενο εδάφιο σχεδιάσαμε την συνάρτηση *calcSeries* που υπολογίζει το άθροισμα $\sum_{i=0}^{k} \frac{x^i}{i!}$. Έστω ότι έχει τοποθετηθεί στο πηγαίο αρχείο *myMath.c* και έστω ότι η κύρια συνάρτηση *main* του Σχήματος 4.2 βρίσκεται στο πηγαίο αρχείο *myCode.c*. Στη γραμμή 5 του πηγαίου αρχείου *myCode.c* όπως φαίνεται στο σχήμα βρίσκεται η δήλωση πρωτοτύπου της συνάρτησης *calcSeries* ακριβώς πριν τον ορισμό της συνάρτησης *main* στην οποία καλείται στη γραμμή 19. Το πως καλείται μια συνάρτηση και τι συμβαίνει κατά την κλήση της θα το αναλύσουμε στο επόμενο εδάφιο.

4.3. Μηχανισμός κλήσης συνάρτησης

Θα ολοκληρώσουμε τα βασικά θέματα των συναρτήσεων παρουσιάζοντας τον μηχανισμό κλήσεων συναρτήσεων της C. Καταρχήν μια συνάρτηση μπορεί να κληθεί από οποιοδήποτε σημείο άλλης συνάρτησης καθώς και της ίδιας συνάρτησης (αναδρομή). Για να κληθεί μια συνάρτηση πρέπει να γραφεί το όνομα της ακολουθούμενο από μια λίστα ορισμάτων εντός παρενθέσεως τα οποία είναι τόσα όσες είναι και οι δηλωθείσες τυπικές παράμετροι στη δήλωση πρωτοτύπου της συνάρτησης. Στη γραμμή 19 του προγράμματος στο Σχήμα 4.2 αφού διαβασθεί από την scanf ένας αριθμός κινητής υποδιαστολής διπλής ακρίβειας και αποθηκευθεί στη μεταβλητή *x* εκτελείται μια εντολή επανάληψης για 10 βήματα όπου σε κάθε βήμα εκτελείται η συνάρτηση printf η οποία έχει ως όρισμα το αποτέλεσμα που θα επιστρέψει η συνάρτηση calcSeries. Βλέπουμε εδώ έναν πολύ βολικό τρόπο κλήσης συναρτήσεων, στη θέση ενός ορίσματος, όταν δεν μας ενδιαφέρει να αποθηκεύσουμε αυτό που επιστρέφει η συνάρτηση. Στην προκειμένη περίπτωση απλά θέλουμε να τυπώσουμε την τιμή που επιστρέφει.

Κατά την κλήση μιας συνάρτησης εκτελούνται τα παρακάτω βήματα:

1. Στην καλούσα συνάρτηση υπολογίζονται οι τιμές όλων των ορισμάτων της καλούμενης συνάρτησης. Στο παράδειγμα μας σε κάθε βήμα της εντολής επανάληψης τα ορίσματα που χρειάζεται η συνάρτηση calcSeries είναι η τιμή της μεταβλητής

x η οποία ορίσθηκε από την scanf και δεν αλλάζει σε κάθε βήμα και η τιμή της μεταβλητής i η οποία καθορίζεται αμέσως μετά την εκτέλεση της έκφρασης $i++$ της εντολής *for*.

2. Στη συνέχεια, η τιμή κάθε ορίσματος αντιγράφεται στην αντίστοιχη τυπική παράμετρο, κάνοντας αν χρειάζεται μετατροπή τύπου. Έτσι η τιμή της μεταβλητής i αντιγράφεται στην τυπική παράμετρο k και η τιμή της μεταβλητής x αντιγράφεται στην τυπική παράμετρο x η οποία παρόλο που έχει το ίδιο όνομα με τη μεταβλητή της συνάρτησης main είναι διαφορετική από αυτή.

3. Ο έλεγχος του προγράμματος μεταφέρεται στο σώμα της καλούμενης συνάρτησης. Στο παράδειγμα μας ο έλεγχος μεταφέρεται στην συνάρτηση calcSeries. Στο σημείο αυτό θα πρέπει να αναφέρουμε ότι στο σώμα της συνάρτησης οι τυπικές παράμετροι μπορούν να συμμετέχουν σε οποιαδήποτε έκφραση και εντολή. Μπορεί το περιεχόμενο των τυπικών παραμέτρων να αλλάξει αλλά αυτό δεν σημαίνει ότι οι αλλαγές αυτές θα μεταφερθούν και στα ορίσματα της καλούσας συνάρτησης και αυτό γιατί όπως προαναφέραμε οι τυπικές παράμετροι μιας συνάρτησης είναι αντίγραφα των τιμών των ορισμάτων της καλούσας συνάρτησης. Ο τρόπος αυτός περάσματος τιμών σε συναρτήση ονομάζεται **κλήση με τιμή (call by value)**. Στο επόμενο κεφάλαιο που θα εισάγουμε την έννοια του δείκτη θα δούμε έναν ακόμα τρόπο περάσματος ορισμάτων σε συνάρτηση που ονομάζεται **κλήση με αναφορά (call by reference)**.

4. Το λειτουργικό σύστημα δεσμεύει χώρο μνήμης από την στοίβα του χρήστη για την αποθήκευση των τιμών των τυπικών παραμέτρων και των τοπικών μεταβλητών της συνάρτησης ο οποίος ονομάζεται **πλαίσιο στοίβας (stack frame)**. Το μέγεθος σε byte του πλαισίου στοίβας ισούται με το άθροισμα των μεγεθών των παραμέτρων και των τοπικών μεταβλητών. Για παράδειγμα το μέγεθος πλαισίου στοίβας της συνάρτησης calcSeries ισούται με 36 byte καθώς έχουμε τρεις αριθμούς κινητής υποδιαστολής διπλής ακρίβειας (μια παράμετρος και δύο τοπικές μεταβλητές) καθένας από αυτούς όπως αναφέραμε στο κεφάλαιο 2 απαιτεί 8 byte και τρεις ακεραίους (μια

παράμετρος και δύο τοπικές μεταβλητές) καθένας από αυτούς όπως αναφέραμε στο κεφάλαιο 2 απαιτεί 4 byte.

5. Εκτελούνται όλες οι εντολές του σώματος της συνάρτησης μέχρι την εντολή return ή απουσία της return μέχρι το δεξί άγκιστρο }.

6. Ο έλεγχος επιστρέφει στην καλούσα συνάρτηση όπου η κλήση της συνάρτησης αντικαθίσταται από την τιμή που έχει επιστραφεί (αν επιστρέφει τιμή). Στο παράδειγμα μας, η κλήση της συνάρτησης calcSeries στη γραμμή 19 αντικαθίσταται κατά την επιστροφή της από την τιμή που επιστρέφει και συνεπώς η τιμή αυτή γίνεται όρισμα της printf και η εκτέλεση του προγράμματος συνεχίζει με την εκτέλεση της printf η οποία τυπώνει τον δείκτη i του όρου της σειράς και την τιμή του όρου. Επίσης, με την επιστροφή στην καλούσα συνάρτηση, το πλαίσιο στοίβας της συνάρτησης απελευθερώνεται που σημαίνει ότι όλος ο χώρος μνήμης που δεσμεύθηκε για το πλαίσο στοίβας επιστρέφεται πίσω στο λειτουργικό σύστημα με αποτέλεσμα όλες οι τοπικές μεταβλητές της συνάρτησης να χάσουν την τιμή που είχαν. Αυτό σημαίνει ότι όταν επανακληθεί η συνάρτηση το λειτουργικό σύστημα θα δεσμεύσει εκ νέου μνήμη από τη στοίβα για τη συνάρτηση και οι τοπικές μεταβλητές της συνάρτησης θα αρχικοποιηθούν.

Τελικά το αποτέλεσμα της εκτέλεσης των εντολών της case 1 στις γραμμές 16-20 είναι για κάποιο αριθμό κινητής υποδιαστολής x να υπολογίζονται και να τυπώνονται στην οθόνη οι όροι $S_k(x) = \sum_{i=0}^{k} \frac{x^i}{i!}$ για $k = 1, 2, \ldots 10$.

4.4. Κατηγορηματικές συναρτήσεις

Η C διαθέτει ένα σύνολο συναρτήσεων οι οποίες εξετάζουν αν συγκεκριμένες προτάσεις είναι αληθείς ή ψευδείς επιστρέφοντας ακέραια τιμή διάφορη του μηδενός αν είναι αληθής *TRUE* και μηδέν αν είναι ψευδής *FALSE*. Οι δηλώσεις των πρωτοτύπων τους βρίσκονται

στο αρχείο επικεφαλίδα $< ctype.h >$. Στον Πίνακα 4.1 παρουσιάζεται ένα υποσύνολο κατηγορηματικών συναρτήσεων της πρότυπης βιβλιοθήκης της *C*.

Συνάρτηση	Περιγραφή
islower(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι πεζό γράμμα.
isupper(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι κεφαλαίο γράμμα.
isalnum(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι γράμμα ή ψηφίο.
isalpha(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι γράμμα.
isdigit(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι ψηφίο.
ispunct(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι σημείο στίξης.
isspace(*ch*)	Επιστρέφει $\neq 0$ αν ο χαρακτήρας *ch* είναι λευκός.

Πίνακας 4.1: Κατηγορηματικές συναρτήσεις της πρότυπης βιβλιοθήκης της *C*.

Εκτός των παραπάνω κατηγορηματικών συναρτήσεων που διαθέτει η *C*, ο προγραμματιστής μπορεί να κατασκευάσει τις δικές του κατηγορηματικές συναρτήσεις. Πριν όμως προχωρήσουμε σε ορισμένα παραδείγματα κατηγορηματικών συναρτήσεων θα κάνουμε μια παρένθεση για να παρουσιάσουμε τον μηχανισμό απαρίθμησης που διαθέτει η *C* ο οποίος θα μας φανεί χρήσιμος στην δημιουργία νέων τύπων δεδομένων όπως ο τύπος **bool** που θα χρειαστούμε για τις κατηγορηματικές συναρτήσεις.

```
0:  bool IsEven(int k)
1:  {
2:     return(k % 2 == 0);
3:  }
4:
5:  bool IsLeapYear(int year)
6:  {
7:   return(((year%4 == 0) && (year % 100 != 0)) || (year % 400 == 0) );
8:  }
9:
10: bool IsVowel(char ch)
11: {
12:    switch (tolower(ch)) {
13:       case 'a': case 'e': case 'i': case 'o': case 'u':
14:          return(TRUE);
15:       default: return(FALSE);
16:    }
17: }
```

Σχήμα 4.3: Παραδείγματα Κατηγορηματικών Συναρτήσεων

Στο κεφάλαιο 2 παρουσιάστηκαν όλοι οι βασικοί τύποι δεδομένων της *C*. Ένα πεπερασμένο σύνολο τιμών μπορεί να ορισθεί ως τύπος δεδομένων με τον μηχανισμό της απαρίθμησης που διαθέτει η *C*. Απαρίθμηση είναι η διαδικασία της παράθεσης όλων των δεδομένων ενός συνόλου υπο μορφή λίστας. Ένας τύπος που ορίζεται με την παράθεση όλων των στοιχείων του ονομάζεται τύπος απαρίθμησης (*enumeration type*). Τυπικά ένας τύπος απαρίθμησης ορίζεται ως εξής:

enum <όνομα_τύπου> {<τιμή_1>, <τιμή_2>, . . . , <τιμή_n >};

Επειδή οι τιμές ενός τύπου απαρίθμησης πρέπει να είναι μοναδικές για κάθε τύπο απαρίθμησης δεν μπορεί αυτές οι τιμές να είναι ακέραιοι ή πραγματικοί αριθμοί. Ας

δούμε ένα παράδειγμα. Έστω ότι θέλουμε να φτιάξουμε τον τύπο **bool**, ο οποίος θα συγκροτείται από τις τιμές *TRUE* και *FALSE*. Τότε μπορούμε να γράψουμε:

enum bool {**FALSE**, **TRUE**}·

και για να δηλώσουμε μεταβλητές τύπου *Bool* γράφουμε:

enum bool temp;

και οι τιμές που μπορεί να πάρει η μεταβλητή *temp* είναι **FALSE** και **TRUE**. Για να αποφύγουμε τη λέξη **enum** κάθε φορά που θα θέλουμε να δηλώσουμε μια μεταβλητή τύπου **bool** μπορούμε να χρησιμοποιήσουμε τη λέξη κλειδί *typedef* που παρουσιάσαμε στο κεφάλαιο 2 ως εξής:

typedef enum {<λίστα_στοιχείων>} <όνομα_τύπου>;

όπου <λίστα_στοιχείων> είναι μια λίστα ονομάτων χωρισμένα με κόμμα. Τώρα οι δηλώσεις του παραπάνω παραδείγματος μας έχουν ως εξής:

typedef enum {*FALSE, TRUE*} *bool*;
bool temp;

Ένα ακόμα παράδειγμα είναι η δημιουργία του τύπου *months*, ο οποίος θα συγκροτείται από τις τιμές January, February, March, April, May, June, July, August, September, October, November, December. Τότε η δήλωση έχεις ως εξής:

typedef enum {January, February, March, April, May, June, July, August, September, October, November, December} months;

Είναι φανερό ότι ο μηχανισμός τύπων απαρίθμησης μας δίνει τη δυνατότητα να εργαζόμαστε με τιμές που είναι πιο φυσικές προς εμάς από το να χρησιμοποιούμε αριθμητικές τιμές.

Στις παραπάνω δηλώσεις, η παράθεση των ονομάτων σηματοδοτεί την εξ ορισμού αντιστοίχιση αυτών με ακεραίους αρχίζοντας από το 0 και αυξάνοντας κατά 1. Για παράδειγμα στις προηγούμενες δηλώσεις των τύπων *bool* και *months* προκύπτουν οι εξής αντιστοιχίσεις:

FALSE → 0, *TRUE* → 1

και

January → 0, *February* → 1, *March* → 2, *April* → 3, *May* → 4, *June* → 5, *July* → 6, *August* → 7, *September* → 8, *October* → 9, *November* → 10, *December* → 11

Σε περίπτωση που είναι επιθυμητή διαφορετική αρίθμηση τότε η δήλωση του τύπου *months* για παράδειγμα θα έχει ως εξής:

typedef enum {January=1, February=2, March=3, April=4, May=5, June=6, July=7, August=8, September=9, October=10, November=11, December=12} months;

ή

typedef enum {January=1, February, March, April, May, June, July, August, September, October, November, December} months;

Δεδομένου ότι στις τιμές ενός τύπου απαρίθμησης αντιστοιχούν αριθμητικές τιμές τότε οι μεταβλητές αυτού του τύπου μπορούν να συμμετέχουν σε αριθμητικές εκφράσεις όπως για παράδειγμα η μεταβλητή *month* μπορεί να συμμετέχει στην ακόλουθη έκφραση: *month* = (*month* + 1)%12. Όλη η αριθμητική των τύπων απαρίθμησης λειτουργεί με τον ίδιο τρόπο που λειτουργεί και στους ακεραίους.

Επανερχόμαστε στις κατηγορηματικές συναρτήσεις παρουσιάζοντας τις συναρτήσεις του Σχήματος 4.3. Η πρώτη συνάρτηση *IsEven* έχει ως τυπική παράμετρο έναν ακέραιο *k* και επιστρέφει 1 ή 0 ανάλογα με το αν ο το υπόλοιπο της διαίρεσης του ακέραιου *k* με το 2 είναι 0 ή όχι δηλαδή αν είναι άρτιος ή όχι. Η καλούσα συνάρτηση επειδή η επιστραφείσα τιμή είναι τύπου *bool* μπορεί να χρησιμοποιεί τις τιμές *FALSE* και *TRUE*.

Η δεύτερη συνάρτηση *IsLeapYear* έχει ως τυπική παράμετρο έναν ακέραιο *year* και επιστρέφει το αποτέλεσμα της λογικής έκφρασης που μπορεί να είναι 1 ή 0 ανάλογα με

το αν ο το έτος είναι δίσεκτο ή όχι. Και πάλι η καλούσα συνάρτηση επειδή η επιστραφείσα τιμή είναι τύπου *bool* μπορεί να χρησιμοποιεί τις τιμές *FALSE* και *TRUE*. Τέλος, η τρίτη συνάρτηση *IsVowel* έχει ως τυπική παράμετρο μια μεταβλητή τύπου χαρακτήρα (*ch*) και αφού μετατρέψει τον χαρακτήρα σε πεζό με τη χρήση της συνάρτησης *tolower*() επιστρέφει *TRUE* ή *FALSE* ανάλογα με το αν είναι αγγλικό φωνήεν ή όχι.

```
 0:  bool isPerfect(int n)
 1:  {
 2:     int i, sum;
 3:
 4:     sum=0;
 5:     for(i=1; i<n; i++){
 6:        if ((n%i)==0) sum+=i;
 7:        if (sum>n) return FALSE;
 8:     }
 9:     if (sum == n) return TRUE;
10:     else return FALSE;
11:  }
```

Σχήμα 4.4: Συνάρτηση εύρεσης Τέλειων Αριθμών

Ας δούμε ένα πιο σύνθετο παράδειγμα κατηγορηματικής συνάρτησης. Γνήσιος διαιρέτης ενός ακέραιου αριθμού n είναι οποιοσδήποτε διαιρέτης που είναι μικρότερος του ίδιου του n. Ένας ακέραιος αριθμός ο οποίος είναι ίσος με το άθροισμα των γνησίων διαιρετών του ονομάζεται **τέλειος αριθμός**. Θέλουμε να αναπτύξουμε την κατηγορηματική συνάρτηση *isPerfect*(n) που θα δέχεται ως τυπική παράμετρο έναν ακέραιο n και θα επιστρέφει *TRUE* ή *FALSE* ανάλογα με το αν ο αριθμός είναι τέλειος ή όχι.

Στο σώμα της συνάρτησης αρχικά ορίζεται η ακέραια μεταβλητή *sum* η οποία θα κρατά το άθροισμα των διαιρετών της τυπικής παραμέτρου n. Από τον ορισμό του τέλειου αριθμού ο πιο απλός τρόπος να εξετάσουμε αν ο n είναι τέλειος ή όχι είναι να εξετάσουμε

όλους τους ακέραιους από 1 έως $n-1$ αν διαιρούν ακριβώς το n και να πάρουμε το άθροισμα τους. Για την γρήγορη λήψη απόφασης στη γραμμή 7 εξετάζεται για κάθε εξεταζόμενο ακέραιο αν το άθροισμα που προέκυψε μέχρι τώρα είναι μεγαλύτερο από το n. Σε περίπτωση που είναι μεγαλύτερο τερματίζει η συνάρτηση και επιστρέφεται η τιμή *FALSE* διαφορετικά ολοκληρώνεται η εντολή επαναληψης και στο τέλος της συνάρτησης ελέγχεται αν το άθροισμα των διαιρετών είναι ίσο με το n και σε περίπτωση που είναι αληθές επιστρέφεται η τιμή *TRUE* διαφορετικά *FALSE*.

Χρησιμοποιώντας την παραπάνω συνάρτηση αν εξετάσουμε όλους τους ακέραιους στο διάστημα [1 . . . 9999] θα διαπιστώσουμε ότι μόνο 4 τέλειοι αριθμοί υπάρχουν στο διάστημα αυτό.

4.5. Κλάσεις Αποθήκευσης και Εμβέλεια Μεταβλητών

Μέχρι τώρα έχουμε αναφέρει ότι μια συνάρτηση επικοινωνεί με τις υπόλοιπες συναρτήσεις ενός προγράμματος μέσω των τυπικών της παραμέτρων και της τιμής που επιστρέφει. Επίσης, έχουμε χρησιμοποιήσει δηλώσεις μεταβλητών μόνο εντός μιας ομάδας εντολών και μάλιστα μόνο στην αρχή της ομάδας καθώς και στο σώμα μιας συνάρτησης που και αυτό όπως αναφέραμε στην αρχή του κεφαλαίου δεν είναι τίποτε άλλο παρά μια ομάδα εντολών. Ο χώρος δράσης των τυπικών παραμέτρων και των μεταβλητών που ορίζονται στο σώμα μιας συνάρτησης ή μιας ομάδας εντολών που όπως είπαμε ονομάζονται **τοπικές μεταβλητές** περιορίζεται εντός της συνάρτησης ή της ομάδας στην οποία ορίζονται και έχουν διάρκεια όσο διαρκεί η εκτέλεση της συνάρτησης ή της ομάδας εντολών. Με τον όρο **εμβέλεια ονόματος** εννοούμε το τμήμα του προγράμματος στο οποίο το όνομα αυτό μπορεί να χρησιμοποιηθεί. Αποτέλεσμα των παραπάνω είναι μεταβλητές συναρτήσεων με το ίδιο όνομα σε διαφορετικές συναρτήσεις να είναι διαφορετικές μεταβλητές.

Η επικοινωνία μέσω των τυπικών παραμέτρων καθίσταται αναποτελεσματική όταν το πλήθος των τυπικών παραμέτρων γίνεται μεγάλο. Και αυτό γιατί όπως προαναφέραμε όταν καλείται μια συνάρτηση, η καλούσα συνάρτηση αντιγράφει τις τιμές των ορισμάτων

της στις τυπικές παραμέτρους της καλούμενης συνάρτησης πράγμα εξαιρετικά χρονοβόρο και δυσκίνητο για ένα πρόγραμμα. Ένας άτυπος κανόνας είναι το πλήθος των τυπικών παραμέτρων μιας συνάρτησης κατά τον σχεδιασμό της να διατηρείται μικρότερο από 3. Αν πάραυτα ο αριθμός των παραμέτρων είναι απαραίτητα μεγάλος υπάρχει ένας εναλλακτικός τρόπος επικοινωνίας των συναρτήσεων. Πρόκειται για τις **εξωτερικές ή καθολικές μεταβλητές** οι οποίες ορίζονται και δηλώνονται εξωτερικά κάθε συνάρτησης και είναι εν δυνάμει διαθέσιμες σε κάθε συνάρτηση. Έτσι κάποιες από τις τυπικές παραμέτρους μπορούν να γίνουν εξωτερικές μειώνοντας τον αριθμό των παραμέτρων σημαντικά.

Οι εξωτερικές μεταβλητές έχουν μεγαλύτερη εμβέλεια και διάρκεια ζωής έναντι των τοπικών μεταβλητών που προαναφέραμε. Οι εξωτερικές μεταβλητές είναι μόνιμες, σε αντίθεση με τις τοπικές που χάνουν τις τιμές τους όταν η καλούμενη συνάρτηση επιστρέφει στην καλούσα συνάρτηση. Η εμβέλεια μιας εξωτερικής μεταβλητής ξεκινά από το σημείο στο οποίο ορίζεται ή δηλώνεται μέχρι το τέλος του αρχείου.

Όπως αναφέρθηκε στην αρχή του κεφαλαίου οι συναρτήσεις ενός προγράμματος μπορούν να βρίσκονται σε περισσότερα από ένα αρχεία (αρθρωτός προγραμματισμός). Στην περίπτωση αυτή μια εξωτερική μεταβλητή ορίζεται σε ένα μόνο από αυτά τα αρχεία. Στα υπόλοιπα αρχεία αν η ορισθείσα εξωτερική μεταβλητή χρησιμοποιείται από κάποιες συναρτήσεις των αρχείων αυτών πρέπει απλά να δηλωθεί με τη χρήση της λέξης κλειδί *extern*. Αρχικοποίηση μιας εξωτερικής μεταβλητής γίνεται μόνο κατά τον ορισμό της. Είναι φανερό ότι αν το πρόγραμμα μας περιορίζεται σε ένα αρχείο δεν χρειάζεται να την δηλώσουμε χρειάζεται μόνο να την ορίσουμε.

Έστω για παράδειγμα ότι θέλουμε να ορίζουμε δύο εξωτερικές μεταβλητές την *sp* και την *val* οι οποίες είναι τύπου *int* και μονοδιάστατος πίνακας 400 αριθμών κινητής υποδιαστολής διπλής ακρίβειας αντίστοιχα. Τότε σε ένα από τα πηγαία αρχεία του προγράμματος γράφουμε:

```
int sp;
double val[400];
```

ενώ στα υπόλοιπα πηγαία αρχεία που χρησιμοποιούνται οι παραπάνω εξωτερικές μεταβλητές γράφουμε:

extern int sp;
extern double val[];

που όπως βλέπουμε για τον πίνακα *val* δεν χρειάζεται να γράψουμε το πλήθος των στοιχείων του. Όμως ποιά η διαφορά μιας δήλωσης *extern* από τον ορισμό της εξωτερικής μεταβλητής; Η διαφορά έγκειται στο ότι κατά τη μεταγλώττιση όταν ο μεταγλωττιστής βρίσκει τον ορισμό μιας εξωτερικής μεταβλητής σε κάποιο από τα πηγαία αρχεία δεσμεύει αμέσως ένα σύνολο από *byte* που απαιτούνται για τον τύπο της μεταβλητής ενώ όταν εντοπίζει σε άλλο πηγαίο αρχείο μια δήλωση *extern* τότε αναζητεί τον ορισμό της σε κάποια από τα πηγαία αρχεία του υπο μεταγλώττιση προγράμματος και θέτει μια αναφορά στον ορισμό αυτό.

Έτσι στο παραπάνω παράδειγμα μας, ο μεταγλωτιστής δεσμεύει αρχικά 4 byte για την *sp* (όσα απαιτούνται για τον τύπο *int*) και 400×8 byte για τον πίνακα *val* (θυμίζουμε ότι απαιτούνται 8 byte για τον τύπο double) και όταν συναντά τις δηλώσεις *extern* απλά θέτει αναφορές στις μεταβλητές που ορίσθηκαν προηγούμενα. Αυτός είναι και ο λόγος που δεν βάλαμε πλήθος στοιχείων στη δήλωση *extern* για τον πίνακα καθώς όπως αναφέραμε όταν συναντάται δήλωση δημιουργείται μια απλή αναφορά στο ορισμό που προηγήθηκε. Λόγω ακριβώς του γεγονότος ότι ο μεταγλωττιστής δεσμεύει στατικά μνήμη για τις εξωτερικές μεταβλητές οι οποίες ισχύουν από την αρχή της εκτέλεσης του προγράμματος μέχρι το τέλος της εκτέλεσης και επιστρέφεται η μνήμη στο λειτουργικό μόνο εφόσον τερματίσει το πρόγραμμα καλό είναι ο προγραμματιστής να χρησιμοποιεί τις εξωτερικές μεταβλητές με φειδώ καθώς δεν είναι πανάκεια και μπορεί η αλόγιστη χρήση τους να οδηγήσει σε αστάθεια συστήματος. Χαρακτηριστικά αναφέρουμε την περίπτωση των ενσωματωμένων συστημάτων όπως ασύρματα σημεία πρόσβασης, υπολογιστές παλάμης, κινητά και ένα σωρό άλλες συσκευές που διαθέτουν περιορισμένη μνήμη.

Μια τοπική μεταβλητή μιας συνάρτησης όπως προαναφέρθηκε, διατηρεί την τιμή της

όσο διαρκεί η εκτέλεση της συνάρτησης. Είναι κατά κάποιο τρόπο μια προσωρινή μεταβλητή καθώς με την επιστροφή του ελέγχου στην καλούσα συνάρτηση, η τιμή της χάνεται. Δηλώνοντας μια τοπική μεταβλητή ως **στατική** με τη λέξη κλειδί *static* μια συνηθισμένη τοπική μεταβλητή καθίσταται μόνιμη, δηλαδή δεν χάνει την τιμή της όταν η συνάρτηση επιστρέψει τον έλεγχο στην καλούσα συνάρτηση. Ο τυπικός ορισμός μιας μεταβλητής ως στατική έχει ως εξής:

static int stackPointer;

Μια εξωτερική μεταβλητή όταν ορίζεται ως *static* τότε η εμβέλεια της περιορίζεται από το σημείο στο οποίο ορίζεται μέχρι το τέλος του αρχείου που είναι ορισμένη. Στα υπόλοιπα αρχεία είναι αόρατη. Καθίσταται **ιδιωτική μεταβλητή**. Τα υπόλοιπα αρχεία μπορούν να χρησιμοποιήσουν το όνομα μιας μεταβλητής που έχει οριστεί ως *static* σε ένα άλλο αρχείο κώδικα. Με την χρήση της λέξης κλειδί *static*, και μια συνάρτηση μπορεί να καταστεί ιδιωτική συνάρτηση με ισχύ μόνο μέσα σε ένα αρχείο.

Στο Σχήμα 1.1 του κεφαλαίου 1 παρουσιάστηκε η αρχιτεκτονική ενός επεξεργαστή. Βασικό στοιχείο στην οργάνωση του επεξεργαστή είναι οι καταχωρητές οι οποίοι είναι στοιχεία μνήμης μεγαλύτερης ταχύτητας από την κύρια εξωτερική μνήμη με την οποία επικοινωνεί ο επεξεργαστής και στην οποία βρίσκονται τα προγράμματα τα οποία εκτελούνται από τον επεξεργαστή. Στη C μια δήλωση μεταβλητής ως μεταβλητής **καταχωρητή** (*register*) δηλώνει στον μεταγλωττιστή ότι η μεταβλητή αυτή πρόκειται να χρησιμοποιηθεί εκτεταμένα και κατά συνέπεια ο μεταγλωττιστής για την επιτάχυνση του προγράμματος επιχειρεί να τοποθετήσει τη μεταβλητή σε έναν από τους καταχωρητές του επεξεργαστή. Μόνο οι τοπικές μεταβλητές και τα ορίσματα των συναρτήσεων μπορούν να δηλωθούν ως καταχωρητή. Για παράδειγμα ο ορισμός μιας συνάρτησης έχει ως εξής:

```
G(register int x, register long y) {
register int j;
. . .
}
```

Μόνο ορισμένος αριθμός και τύπος μεταβλητών μπορεί να ορισθούν ως καταχωρητή και αυτό εξαρτάται από την μηχανή. Η διεύθυνση των μεταβλητών καταχωρητή, δεν είναι διαθέσιμη στον προγραμματιστή, πράγμα που σημαίνει ότι δεν μπορούν να χρησιμοποιηθούν ούτε στην *scanf*() ούτε στην αριθμητική δεικτών.

4.6. Αναδρομικές Συναρτήσεις

Στην αρχή του κεφαλαίου είδαμε ότι η επίλυση σύνθετων προβλήματων μπορεί να επιτευχθεί με αναγωγή σε μικρότερα απλά προβλήματα τα αποτελέσματα των οποίων στο τέλος όταν συνδυαστούν δίνουν τη λύση στο αρχικό πρόβλημα. Η διαδικασία αυτή της αναγωγής ενός σύνθετου προβλήματος σε μικρότερα και απλούστερα επιμέρους προβλήματα ονομάζεται **αποσύνθεση (decomposition)**. Μια ειδική περίπτωση της αποσύνθεσης είναι η **αναδρομή (recursion)** με την οποία ένα σύνθετο πρόβλημα ανάγεται σε μικρότερα προβλήματα της **ίδιας ακριβώς μορφής**.

Για να γίνει κατανοητό αυτό ας εξετάσουμε το παράδειγμα υπολογισμού των διωνυμικών συντελεστών. Οι διωνυμικοί συντελεστές είναι οι συντελεστές των όρων του διωνυμικού αναπτύγματος 4.1

$$(1+x)^n = \sum_{k=0}^{n} C(n,k) \times x^k \tag{4.1}$$

Οι διωνυμικοί συντελεστές $C(n,k)$ είναι οι γνωστοί μας συνδυασμοί από τα διακριτά μαθηματικά οι οποίοι μπορούν να υπολογισθούν από τη σχέση 4.2 η οποία βασίζεται στον υπολογισμό τριων παραγοντικών.

$$C(n,k) = \frac{n!}{k! \times (n-k)!} \tag{4.2}$$

Ένας εναλλακτικός τρόπος υπολογισμού των $C(n,k)$ είναι ο αναδρομικός. Ο αριθμός $C(n,k)$ εκφράζει το πλήθος των τρόπων να επιλέξουμε k αντικείμενα από μια συλλογή n διακριτών αντικειμένων ο οποίος μπορεί να υπολογισθεί αναδρομικά. Είναι φανερό ότι με έναν τρόπο μπορούμε να επιλέξουμε 0 αντικείμενα από n, δηλαδή $C(n,0) = 1$. Επίσης είναι φανερό ότι δεν υπάρχει τρόπος να επιλέξουμε k αντικείμενα από n όταν $k > n$, δηλαδή $C(n,k) = 0$ όταν $k > n$.

Για την γενική περίπτωση σκεπτόμαστε ως εξής, αφαιρούμε ένα οποιοδήποτε αντικείμενο από τα n και υπολογίζουμε:

1. το πλήθος των τρόπων να επιλέξουμε $k-1$ αντικείμενα από τα εναπομείναντα $n-1$, δηλαδή $C(n-1,k-1)$ που ισοδυναμεί με το πλήθος των τρόπων να επιλέξουμε k από τα n όπου όμως στα k να περιλαμβάνεται το αντικείμενο που αφαιρέσαμε,

2. το πλήθος των τρόπων να επιλέξουμε k αντικείμενα από τα εναπομείναντα $n-1$, δηλαδή $C(n-1,k)$ που ισοδυναμεί με το πλήθος των τρόπων να επιλέξουμε k από τα n όπου όμως στα k αντικείμενα δεν περιλαμβάνεται το αντικείμενο που αφαιρέσαμε.

δηλαδή $C(n,k) = C(n-1,k-1) + C(n-1,k)$ που σημαίνει ότι αναγάγαμαι τον υπολογισμό του $C(n,k)$ στον υπολογισμό των $C(n-1,k-1)$ και $C(n-1,k)$. Με άλλα λόγια αναγάγαμαι την επίλυση του αρχικού μας προβλήματος σε δύο μικρότερα μεγέθους προβλήματα αλλά της ίδιας ακριβώς μορφής με το αρχικό μας πρόβλημα. Συνεχίζοντας, με τον ίδιο τρόπο τον υπολογισμό των $C(n-1,k-1)$ και $C(n-1,k)$ καταλήγουμε στον υπολογισμό των τετριμμένων περιπτώσεων $C(n,0)$ και $C(n,k)$ για $k > n$ των οποίων η επίλυση είναι άμεση όπως προαναφέραμε.

Οι διωνυμικοί συντελεστές ορίζονται από τον ακόλουθο αναδρομικό τύπο:

$$\begin{aligned} C(n,0) &= 1 \\ C(n,k) &= 0 \text{ για } k > n \\ C(n,k) &= C(n-1,k-1) + C(n-1,k) \text{ για } 0 < k \leq n \end{aligned} \tag{4.3}$$

Βλέπουμε λοιπόν ότι το ίδιο πρόβλημα μπορεί να λυθεί με δύο τρόπους: τον επαναληπτικό με τη σχέση 4.2 και τον αναδρομικό με τη σχέση 4.3.

```
0:  int combinations(int n, int k)
1:  {
2:     if (n<k) return(0);
3:     else if (k==0) return(1);
4:     else return(combinations(n-1,k-1)+combinations(n-1,k));
5:  }
```

Σχήμα 4.5: Συνάρτηση υπολογισμού διωνυμικών συντελεστών

Πως όμως μπορούμε να υλοποιήσουμε την αναδρομική λύση ενός προβλήματος με μια γλώσσα προγραμματισμού; Στις περισσότερες γλώσσες προγραμματισμού, μια συνάρτηση που στο σώμα της καλεί έμμεσα ή άμεσα τον εαυτό της ονομάζεται **αναδρομική συνάρτηση**. Στο Σχήμα 4.5 δίνεται η αναδρομική συνάρτηση η οποία υλοποιεί την αναδρομική σχέση 4.3. Η συνάρτηση ονομάζεται *combinations* έχει δύο τυπικές παραμέτρους n και k και επιστρέφει μια ακέραια τιμή που είναι ο $C(n,k)$. Οι γραμμές του προγράμματος 2-3 τερματίζουν την αναδρομή και ουσιαστικά υλοποιούν τις τετριμμένες περιπτώσεις του υπολογισμού μας ενώ στη γραμμή 4 η συνάρτηση καλεί τον εαυτό της 2 φορές αλλά με μειωμένες τιμές στις τυπικές παραμέτρους. Στο Σχήμα 4.2 δίνεται η κύρια συνάρτηση όπου στην επιλογή 3 καλείται η συνάρτηση *combinations*.

Κατά τον σχεδιασμό μιας αναδρομικής συνάρτησης θα πρέπει να έχουμε υπόψη μας:

1. Την επιλογή των σωστών τετριμμένων περιπτώσεων που η επίλυση τους είναι μη αναδρομική και τερματίζουν την αναδρομική συνάρτηση. Κακή επιλογή των τετριμμένων περιπτώσεων μπορεί να οδηγήσει σε ατέρμονες κλήσεις της συνάρτησης γεγονός που μπορεί να εισάγει τον υπολογιστή σε κατάσταση αστάθειας.

2. Τον καθορισμό με σαφήνεια των κανόνων με τους οποίους θα καλεί η συνάρτηση τον εαυτό της.

Η δομή μιας αναδρομικής συνάρτησης έχει την παρακάτω μορφή:

if (έλεγχος τετριμμένης περίπτωσης) *return*(μη αναδρομική λύση)
else {
  return(κλήση της ίδιας συνάρτησης με τροποποιημένες τις τιμές
  των τυπικών παραμέτρων.)
}

Προς εφαρμογή των παραπάνω ας δούμε την αναδρομική συνάρτηση που υπολογίζει τους όρους της ακολουθίας *Fibonacci*. Οι όροι της ακολουθίας *Fibonacci* υπολογίζονται βάσει του αναδρομικού τύπου 4.4.

$$
\begin{aligned}
fib(0) &= 0 \\
fib(1) &= 1 \\
fib(n) &= fib(n-1) + fib(n-2)
\end{aligned}
\tag{4.4}
$$

Οι τετριμμένες περιπτώσεις είναι φανερό ότι είναι ο υπολογισμός των δύο πρώτων όρων της ακολουθίας ο υπολογισμός των οποίων είναι άμεσος και μη αναδρομικός ενώ οι υπόλοιποι όροι υπολογίζονται αναδρομικά. Στο Σχήμα 4.6 δίνεται η αναδρομική συνάρτηση που υλοποιεί τον τύπο 4.4 η οποία έχει τη δομή που προαναφέρθηκε καθώς και η κύρια συνάρτηση η οποία καλεί την συνάρτηση αυτή.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  int fib(int);
 4:
 5:  int main()
 6:  {
 7:   int n;
 8:
 9:   printf("Dvste ton oro ths akolouthias Fibonacci: ");
10:   scanf("%d", &n);
11:
12:   printf("Fib(%d)=%d\n", n, fib(n));
13:  }
14:
15:  int fib(int n)
16:  {
17:     if (n<=0) return n;
18:     else return (fib(n-1)+fib(n-2));
19:  }
```

Σχήμα 4.6: Συνάρτηση υπολογισμού των όρων της ακολουθίας Fibonacci.

Τα περισσότερα προβλήματα έχουν επαναληπτική και αναδρομική λύση. Στο Σχήμα 4.7 παρουσιάζονται δύο συναρτήσεις για τον υπολογισμό του Μέγιστου Κοινού Διαιρέτη δύο ακεραίων, την *e_gcd*() που τον υπολογίζει με επαναληπτικό τρόπο και την *recusiveGCD*() με αναδρομικό τρόπο. Η αναδρομική λύση είναι πιο εύκολο να γραφεί και υλοποιηθεί και απαιτεί συνήθως λιγότερες μεταβλητές από την επαναληπτική. Πάραυτα, υπάρχουν κάποια μειονεκτήματα που καθιστούν την αναδρομική λύση αναποτελεσματική. Για να κατανοήσουμε αυτά τα μειονεκτήματα θα επιχειρήσουμε να αναλύσουμε τι συμβαίνει όταν καλείται μια αναδρομική συνάρτηση έχοντας ως παράδειγμα τον υπολογισμό των διωνυμικών συντελεστών που περιγράψαμε προηγουμένως.

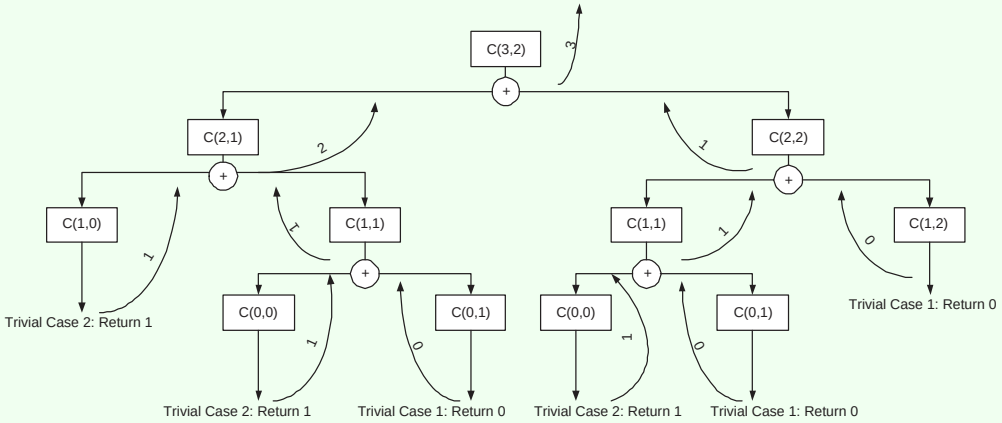
```
0:  #include <stdio.h>
1:  #include <stdlib.h>
2:
3:  #define TRUE 1
4:
5:  int recursiveGCD(int x, int y);
6:  int e_gcd(int x, int y);
7:
8:  int main()
9:  {
10:    int x, y;
11:
12:   printf("Dvse tous arithmous x kai y:");
13:   scanf("%d %d", &x, &y);
14:   printf("%d %d\n", e_gcd(x,y), recursiveGCD(x, y));
15:   return 0;
16: }
17:
```

```
18: int e_gcd(int x, int y)
19: {
20:    int r;
21:
22:    while (TRUE){
23:       r = x%y;
24:       if (r==0) break;
25:       x=y;
26:       y=r;
27:    }
28:    return y;
29: }
30:
31: int recursiveGCD(int x, int y)
32: {
33:    int r;
34:
35:    if ((r=x%y)==0) return y;
36:    else return recursiveGCD(y, r);
37: }
```

Πρώτα από όλα στην επαναληπτική λύση η συνάρτηση καλείται μια μόνο φορά ενώ στην περίπτωση της αναδρομικής λύσης η συνάρτηση καλείται περισσότερες από μια φορά και όσο το μέγεθος του προβλήματος μεγαλώνει ο αριθμός των κλήσεων μεγαλώνει υπερβολικά. Στον Πίνακα 4.2 δίνεται ο αριθμός των κλήσεων της αναδρομικής συνάρτησης *combinations*() για $k = 2$ και για διάφορες τιμές του n. Βλέπουμε ότι για $n = 10$ το πλήθος των κλήσεων υπερβαίνει τις 100. Μπορεί να αναλογιστεί κανείς τι θα συμβεί με το πλήθος των κλήσεων της συνάρτησης όταν οι τιμές του n και k γίνουν ρεαλιστικές. Μερικές φορές ο υπολογιστής μπορεί να οδηγηθεί σε αστάθεια για τον λόγο που θα εξηγήσουμε παρακάτω.

n	Αριθμός κλήσεων
3	11
4	19
5	29
6	41
7	55
8	71
9	89
10	109

Πίνακας 4.2: Αριθμός κλήσεων της συνάρτησης *combinations*() σαν συνάρτηση του n για $k = 2$.



Σχήμα 4.8: Κλήσεις της συνάρτησης combinations για τον υπολογισμό $C(3,2)$.

Όπως αναφέραμε στον μηχανισμό κλήσης συνάρτησης, κάθε φορά που καλείται μια συνάρτηση το λειτουργικό σύστημα δεσμεύει χώρο μνήμης από την στοίβα του χρήστη για το πλαίσιο στοίβας της συνάρτησης. Το μέγεθος σε byte του πλαισίου στοίβας ισούται με το άθροισμα των μεγεθών των παραμέτρων και των τοπικών μεταβλητών. Στο Σχήμα 4.8 δίνεται το δένδρο κλήσεων της συνάρτησης για τον υπολογισμό του $C(3,2)$. Η πρώτη κλήση είναι αυτή για τον συντελεστή $C(3,2)$, άρα ένα πλαίσιο στοίβας, όπου επειδή δεν ικανοποιείται κανένα κριτήριο τερματισμού της αναδρομής καλείται αρχικά η συνάρτηση για τον υπολογισμό του $C(2,1)$ όπου και πάλι δεν ικανοποιείται κανένα κριτήριο τερματισμού και καλείται η συνάρτηση για τον υπολογισμό του $C(1,0)$ όπου τώρα ικανοποιείται η τετριμμένη λύση 2 και επιστρέφεται στην $C(2,1)$ η τιμή 1. Μέχρι να φτάσουμε σε κλήση όπου ικανοποιήθηκε κάποιο από τα κριτήρια τερματισμού χρειάστηκε το λειτουργικό σύ-

στημα να συντηρήσει τρια πλαίσια στοίβας γιατί έλαβαν χώρα τρεις συνεχόμενες κλήσεις της συνάρτησης. Με την επιστροφή στην $C(2,1)$ από την $C(1,0)$ καλείται η $C(1,1)$ η οποία με τη σειρά της καλεί πρώτα την $C(0,0)$ η οποία τερματίζει και επιστρέφει την τιμή 1. Μέχρι το σημείο επιστροφής το λειτουργικό σύστημα συντηρεί τέσσερα πλαίσια στοίβας. Ο μηχανισμός που περιγράφηκε λειτουργεί με τον ίδιο τρόπο και για τις υπόλοιπες κλήσεις μέχρι τελικά να υπολογιστεί ο $C(3,2)$ με τον μέγιστο αριθμό πλαισίων στοίβας την ίδια χρονική στιγμή να φτάνει τα τέσσερα πλαίσια. Να τονισθεί ότι η επιστροφή από συνάρτηση σε συνάρτηση είναι μια ιδιαίτερα χρονοβόρα διαδικασία καθώς λαμβάνουν χώρα από την μεριά του λειτουργικού συστήματος αρκετές λειτουργίες για την επαναφορά του περιβάλλοντος εργασίας μιας συνάρτησης.

Από την παραπάνω ανάλυση γίνεται αντιληπτό ότι η επιβάρυνση σε μνήμη και χρόνο που εισάγεται από μια αναδρομική υλοποίηση είναι μεγάλη ιδιαίτερα όταν το μέγεθος του προβλήματος είναι μεγάλο. Για μεγάλα n και k ο αριθμός των πλαισίων που συντηρούνται από το λειτουργικό σύστημα είναι μεγάλος και όταν το μέγεθος του προβλήματος υπερβεί ένα κατώφλι τότε το σύστημα μπορεί να οδηγηθεί σε αστάθεια καθώς θα έχει εξαντληθεί η στοίβα του συστήματος. Η χρήση της αναδρομής καλό είναι να αποφεύγεται σε ενσωματωμένα συστήματα όπου οι πόροι του συστήματος είναι ιδιαίτερα περιορισμένοι.

4.7. Ο Προεπεξεργαστής της C

Μέχρι τώρα εξετάσαμε τις βασικές εντολές του προπεξεργαστή της C *#include* και *#define*. Με την *#include* γίνεται συμπερίληψη των δηλώσεων και των ορισμών που βρίσκονται στα αρχεία επικεφαλίδας ενώ με την *#define* ορίζονται οι σταθερές του προγράμματος μας. Όταν πρόκειται να συμπεριλάβουμε αρχεία επικεφαλίδας της πρότυπης βιβλιοθήκης το όνομα του αρχείου πλαισιώνεται από τα σύμβολα <> όπως για παράδειγμα *#include* < *stdio.h* >. Ο προεπεξεργαστής όταν συναντά την εντολή *#include* εισάγει όλες τις δηλώσεις του αρχείου επικεφαλίδας στο πρόγραμμά μας ενώ όταν συναντά την εντολή *#define* αντικαθιστά στο πρόγραμμα τη συμβολική σταθερά όπου αυτή εμφανίζεται με την τιμή

της.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  #define POWER2(x) (1L << (x))
 4:  #define BitSet(a,pos) ((a) | (1L << (pos)))
 5:  #define BitClr(a,pos) ((a) & ~(1L << (pos)))
 6:
 7:  int main()
 8:  {
 9:     int i, x;
10:
11:     x=0xFF;
12:   for(i=0; i<6; i++) {
13:       x = BitClr(x,i);
14:       printf("%d %x\n", POWER2(i), x);
15:   }
16:  }
```

Σχήμα 4.9: Παράδειγμα μακροεντολής.

Είναι φορές που το σώμα μιας συνάρτησης μπορεί να περιορισθεί στον υπολογισμό μιας απλής έκφρασης της μιας γραμμής. Με την εντολή *#define*, μπορούμε να ορίσουμε μια συνάρτηση η οποία ονομάζεται **μακροεντολή macro** ως εξής:

#define <όνομα_μακροεντολής (<λίστα_ορισμάτων>) (<έκφραση>)

όπου στη <λίστα_ορισμάτων> παρατίθενται οι μεταβλητές που χρησιμοποιούνται στην έκφραση που ακολουθεί χωρίς τη δήλωση του τύπου τους. Στο Σχήμα 3.11 παρουσιάσθηκε το πρόγραμμα που περιστρέφει κατα συγκεκριμένο αριθμό θέσεων τα bit ενός ακεραίου. Επειδή η λειτουργία της περιστροφής είναι μια πολλή βασική λειτουργία θα θέλαμε να

την υλοποιήσουμε ως συνάρτηση. Επειδή όμως το σώμα της είναι μια απλή έκφραση μπορούμε αντι του ορισμού της συνάρτησης να ορίσουμε μια μακροεντολή. Η μακροεντολή που περιστρέφει τα bit ενός ακέραιου αριθμού x κατά n θέσεις ορίζεται ως εξής:

$$\#define\ left\_rotate(x,n)(((x) \ll (n))|((x) \gg (sizeof(int) * 8 - (n))))$$

Κατά τον ορισμό μιας μακροεντολής, ιδιαίτερη προσοχή πρέπει να δοθεί στα ορίσματα, τα οποία στο τμήμα της τιμής της *#define* θα πρέπει να περικλείονται από παρενθέσεις. Έτσι στο πρόγραμμα μας μπορούμε να χρησιμοποιήσουμε την μακροεντολή $left\_rotate(x,n)$ για να περιστρέψουμε τον ακέραιο x κατά n θέσεις. Βέβαια ο προεπεξεργαστής κατά τη μεταγλώττιση θα αντικαταστήσει όλες τις εμφανίσεις της μακροεντολής $left\_rotate(x,n)$ με την τιμή στον ορισμό της. Κατά αυτόν τον τρόπο έχουμε τα θετικά που προσδίδει μια συνάρτηση χωρίς τις επιβαρύνσεις που εισάγονται από τον μηχανισμό κλήσης συναρτήσεων. Η χρήση των μακροεντολών προσδίδουν ταχύτητα στην εκτέλεση του πρόγραμματός μας.

Στο Σχήμα 4.9 παρουσιάζονται τρεις ακόμα μακροεντολές. Όπως αναφέραμε και στο κεφαλαιο 2 η δυνάμεις του 2 μπορούν να υπολογισθούν με την χρήση του τελεστή μετατόπισης αριστερά. Αυτό μπορούμε να το εκμεταλευτούμε για να ορίσουμε την μακρονετολή $POWER2(x)$. Επίσης, σε πολλές εφαρμογές όπως για παράδειγμα κρυπτογραφίας είναι συχνές οι λειτουργίες θέσης και καθαρισμού συγκεκριμένου bit ενός ακεραίου. Μπορούμε πολύ εύκολα να ορίσουμε τις μακροεντολές του $BitSet(a,pos)$ και $BitClr(a,pos)$ όπου a είναι ο ακέραιος και pos είναι η θέση του bit που θέλουμε να επεξεργαστούμε.

4.8. Ασκήσεις

Άσκηση 4.8.1 *Τι θα τυπωθεί όταν εκτελεσθεί ο κώδικας του Σχήματος 4.11; Αρχικά δοκιμάστε να βρείτε το αποτέλεσμα που τυπώνεται χωρίς να τρέξετε τον κώδικα στον υπολογιστή*

και στη συνέχεια ελέγξτε τις απαντήσεις σας τρέχοντας ένα πρόγραμμα που περιέχει τον κώδικα του σχήματος.

```
 0:  int x=2, y=4, z=8;
 1:
 2:  y -= x *= z--;
 3:
 4:  printf("%6d%6d%6d\n", x, y, z);
 5:  {
 6:     double y = 3.0, x;
 7:
 8:     x += z = 5 * y;
 9:
10:     printf("%6.2f%6.1f%6d\n", x, y, z);
11:  }
12:  printf("%6d%6d%6d\n", x, y, z);
```

Σχήμα 4.10: Κώδικας άσκησης 4.8.1

Άσκηση 4.8.2 *Χρησιμοποιώντας την κατηγορηματική συνάρτηση isPerfect που παρουσιάστηκε στο παρόν κεφάλαιο, να γράψετε την κατηγορηματική συνάρτηση isFibPerfect η οποία θα έχει μια τυπική παράμετρο n τύπου int και θα επιστρέφει αν ο n-στος όρος της ακολουθίας fibonacci είναι τελειος αριθμός διαφορετικά FALSE. Για τον υπολογισμό των όρων της ακολουθίας fibonacci μπορείτε να χρησιμοποιήσετε την αναδρομική εκδοχή που δόθηκε στο κεφάλαιο αυτό.* **Υπόδειξη**

Άσκηση 4.8.3 *Περιγράψτε την λειτουργία του προγράμματος του Σχήματος 4.12.*

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  #define FOREVER 1
 4:  #define END 500
 5:
 6:  main(void){
 7:      int k=5;
 8:      void proc(int);
 9:
10:      while(FOREVER) proc(k);
11:  }
12:
13:  void proc(int j){
14:      static int x=0;
15:
16:      printf("%d\n", x+=k);
17:      if (x==END) exit(0);
18:  }
```

Σχήμα 4.11: Κώδικας άσκησης 4.8.3

Άσκηση 4.8.4 *Δεδομένων δύο ακεραίων αριθμών m και n για τους οποίους ισχύει $m \leq n$ να γραφεί μια αναδρομική συνάρτηση που να υπολογίζει το άθροισμα όλων των ακεραίων μεταξύ των m και n.*

Κεφάλαιο 5

Πίνακες και Δείκτες

Στα προηγούμενα κεφάλαια αναλύθηκαν οι βασικοί τύποι δεδομένων, θέματα που αφορούσαν τις μεταβλητές και σταθερές της γλώσσας, οι βασικές εντολές ελέγχου ροής του προγράμματος, οι συναρτήσεις και η χρήση του προεπεξεργαστή για την ανάπτυξη αρθρωτών προγραμμάτων. Στο Κεφάλαιο αυτό θα ασχοληθούμε με τους πίνακες δεδομένων (arrays) καθώς και με το δείκτη σε δεδομένα που καθιστά τη *C* μοναδική στη σχεδίαση και ανάπτυξη λογισμικού για συστήματα.

5.1. Διεύθυνση Δεδομένου

Πριν προχωρήσουμε στην παρουσίαση των πινάκων και των δεικτών θα αναλύσουμε την έννοια της διεύθυνσης δεδομένου προκειμένου να γίνει περισσότερο κατανοητή η παρουσίαση των παραπάνω δομών δεδομένων.

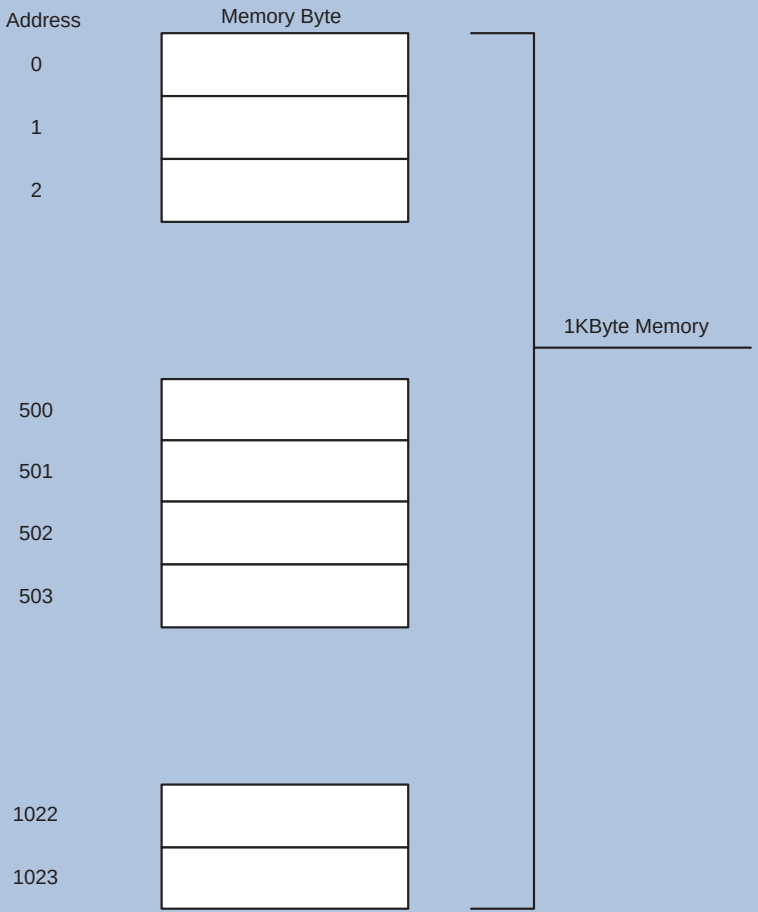
Το μικρότερο στοιχείο μνήμης ενός υπολογιστή είναι το *bit*, στο οποίο μπορεί να είναι αποθηκευμένο το 0 ή το 1. Για την αποθήκευση των τύπων δεδομένων όπως χαρακτή-

ρες και αριθμοί, τα μεμονωμένα *bit* οργανώνονται σε σύνολα από *bit*. Τα σύνολα αυτά αντιμετωπίζονται ως ενιαίες μονάδες αποθήκευσης. Ένα σύνολο από οκτώ *bit* συνθέτουν ένα *byte*. Η κύρια μνήμη του υπολογιστή είναι ένα σύνολο από *byte* και το μέγεθος της μετριέται σε *byte*:

1. $1024 Bytes = 1 KB$

2. $2048 Bytes = 2 KB$

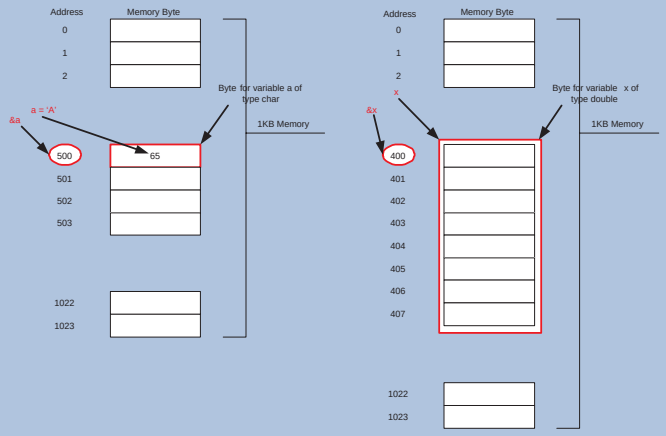
3. $1.048.576 Bytes = 1024 \times 1024 = 1 MB$



Σχήμα 5.1: Οργάνωση Μνήμης σε byte.

Στο Σχήμα 5.1 δίνεται η οργάνωση μιας μνήμης μεγέθους $1KB$, η οποία αποτελείται από 1024 *byte*. Σε κάθε *byte* μιας μνήμης μεγέθους N αντιστοιχεί ένας μοναδικός αριθμός που προκύπτει από την διαδοχική αρίθμηση των *byte* αρχίζοντας από το 0 μέχρι το $N-1$. Ο αριθμός αυτός ονομάζεται **διεύθυνση του** *byte* **(***address***)**. Στο παράδειγμα μας, δεδομένου ότι έχουμε 1024 *byte*, οι διευθύνσεις των *byte* βρίσκονται εντός του διαστήματος $0 \ldots 1023$. Για να λάβουμε το δεδομένο που είναι αποθηκευμένο σε κάποιο *byte* αρκεί να γνωρίζουμε τη διεύθυνση του *byte*, οπότε χρησιμοποιώντας τη διεύθυνση μπορούμε να το επιλέξουμε και να διαβάσουμε το δεδομένο που βρίσκεται αποθηκευμένο σε αυτό. Για να το κατανοήσουμε καλύτερα μπορούμε να φαντασθούμε τα *byte* ως ένα σύνολο από συρτάρια τα οποία τα έχουμε αριθμήσει. Έτσι όταν θέλουμε να αναζητήσουμε ένα δεδομένο αρκεί να γνωρίζουμε τον αριθμό του συρταριού.

Η δήλωση μιας μεταβλητής σηματοδοτεί την έναρξη της διαδικασίας κατανομής χώρου μνήμης του μεταγλωτιστή (*memory allocation*) κατά την οποία δεσμεύεται επαρκής αριθμός από *byte* για την αποθήκευση της τιμής της μεταβλητής. Για παράδειγμα μια δήλωση της μορφής *char* $a = 'A'$; σηματοδοτεί τη δέσμευση ενός *byte* για την αποθήκευση της τιμής της μεταβλητής a αφού για τον τύπο *char* όπως αναφέραμε στο Κεφάλαιο 2 απαιτείται ένα *byte* στο οποίο στη συνέχεια ανατίθεται ως αρχική τιμή ο αριθμός 65 που δεν είναι άλλος από τον ASCII κωδικό του χαρακτήρα $'A'$. Στη συνέχεια η διεύθυνση του *byte* συσχετίζεται με τη μεταβλητή a που στο εξής θα την ονομάζουμε διεύθυνση της μεταβλητής a παρόλο που κατα βάση πρόκειται για τη διεύθυνση του *byte* που έχει κατανεμηθεί στη μεταβλητή a. Στο Σχήμα 5.2 η διεύθυνση της μεταβλητής a είναι το 500 και στο αντίστοιχο *byte* ανατέθηκε ο ASCII κωδικός του χαρακτήρα $'A'$. Η *C* διαθέτει τον μοναδιαίο τελεστή & , τον οποίο είδαμε χωρίς περαιτέρω ανάλυση να προηγείται των μεταβλητών στη λίστα των ορισμάτων της *scanf*, με τον οποίο μπορούμε να έχουμε πρόσβαση στη διεύθυνση μιας μεταβλητής. Έτσι γράφοντας &a λαμβάνουμε τη διεύθυνση του *byte* που έχει αντιστοιχιθεί στη μεταβλητή a και a λαμβάνουμε την τιμή της μεταβλητής.



Σχήμα 5.2: Κατανομή *byte* σε μεταβλητές τύπου *char* και *double*.

Για τους υπόλοιπους τύπους δεδομένων που απαιτούνται περισσότερα *byte* όπως *int, double*, τα *byte* οργανώνονται σε μεγαλύτερες ομάδες που ονομάζονται λέξεις (*words*). Στην περίπτωση αυτή η λέξη είναι προσβάσιμη μέσω ενός μοναδικού αριθμού που ονομάζεται **διεύθυνση της λέξης** και η οποία είναι η διεύθυνση του πρώτου *byte* της ομάδας από *byte*. Για παράδειγμα για την μεταβλητή *double* x; ο μεταγλωτιστής δεσμεύει 8 συνεχόμενα *byte* για τη μεταβλητή x και η διεύθυνση του πρώτου *byte* αντιστοιχίζεται στη μεταβλητή x. Έτσι η έκφραση $\&x$ δίνει τη διεύθυνση της λέξης μνήμης που έχει αντιστοιχηθεί στη μεταβλητή x. Στο Σχήμα 5.2 για τη μεταβλητή x έχουν δεσμευθεί τα *byte* με διευθύνσεις $400, 401, \ldots, 407$ και έχει αντιστοιχηθεί η διεύθυνση 400 του πρώτου *byte* της ομάδας αυτής στη μεταβλητή x που σημαίνει ότι η έκφραση $\&x$ επιστρέφει τη διεύθυνση 400. Επιπρόσθετα, με την κλήση της συνάρτησης *scanf*(”%*f*”, $\&x$) ο αριθμός

κινητής υποδιαστολής που θα διαβασθεί θα αποθηκευθεί στη μεταβλητή με διεύθυνση &*x*.

Για τις τοπικές μεταβλητές κάθε φορά που καλείται μια συνάρτηση ανατίθεται ένα πλαίσιο στοίβας στη συνάρτηση στο οποίο κατανέμονται οι μεταβλητές της με τον τρόπο που περιγράψαμε προηγουμένως και η κατανομή αυτή διαρκεί όσο χρόνο εκτελείται η συνάρτηση. Όταν η συνάρτηση επιστρέφει τον έλεγχο στην καλούσα συνάρτηση το πλαίσιο στοίβας αποδεσμεύεται για να είναι διαθέσιμο για τις υπόλοιπες συναρτήσεις του προγράμματος με αποτέλεσμα οι τοπικές μεταβλητές μετά την αποδέσμευση να μην αντιστοιχούν σε *byte* μνήμης. Για τις καθολικές ή εξωτερικές μεταβλητές η κατανομή ξεκινά με την εκτέλεση του προγράμματος και οι μεταβλητές διατηρούν τη δεσμευμένη μνήμη μέχρι τον τερματισμό.

5.2. Πίνακας Δεδομένων

5.2.1. Μονοδιάστατος Πίνακας Δεδομένων

Ο πίνακας (*array*) είναι μια συλλογή τιμών δεδομένων η οποία έχει τα εξής χαρακτηριστικά:

1. Είναι **Διατεταγμένη**, δηλαδή τα στοιχεία του πίνακα ξεχωρίζουν από τη σειρά τους: πρώτο, δεύτερο κλπ.

2. Είναι **Ομοιογενής**: οι τιμές σε έναν πίνακα είναι του ιδίου τύπου.

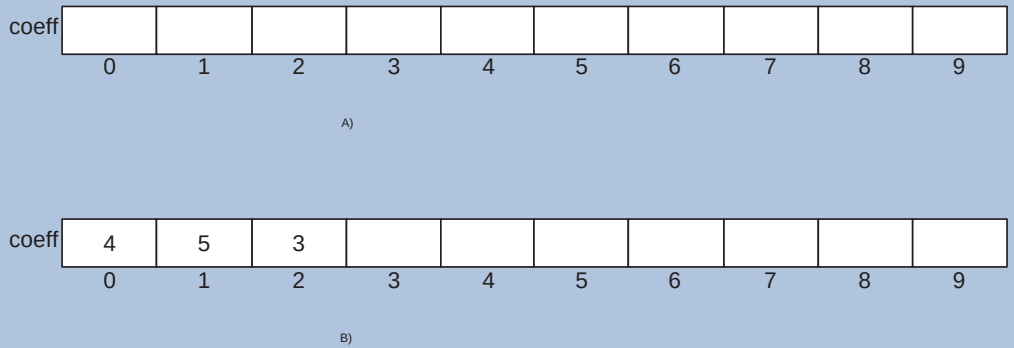
Κάθε μια από τις τιμές του πίνακα ονομάζεται στοιχείο του πίνακα (*element*). Για τον ορισμό στη *C* ενός μονοδιάστατου πίνακα απαιτούνται ο ορισμός του τύπου των στοιχείων του πίνακα και ο καθορισμός του πλήθους των στοιχείων του. Μια καλή προγραμματιστική τεχνική είναι το μέγεθος ενός πίνακα να ορίζεται μέσω μιας συμβολικής σταθεράς καθώς η τροποποίηση του είναι πιο εύκολη και το πρόγραμμα είναι περισσότερο ευανάγνωστο. Τυπικά ο ορισμός ενός πίνακα έχει ως εξής:

<τύπος_στοιχείου> όνομα_πίνακα[πλήθος_στοιχείων];

Για παράδειγμα, για να ορίσουμε τον μονοδιάστατο πίνακα *coeff* δέκα ακεραίων γράφουμε:

```
int coeff[10];
```

Κάθε στοιχείο του πίνακα προσδιορίζεται από μια αριθμητική τιμή που ονομάζεται αριθμοδείκτης (*index*) του στοιχείου. Οι αριθμοδείκτες ξεκινούν από 0 και φτάνουν μέχρι το μέγεθος του πίνακα μειωμένο κατά ένα. Κατά τον ορισμό ενός πίνακα τα στοιχεία του δεν λαμβάνουν συγκεκριμένες τιμές. Ο μονοδιάστατος πίνακας *coeff* που ορίσθηκε προηγουμένως έχει τη μορφή που φαίνεται στο Σχήμα 5.3 Α) όπου στα στοιχεία δεν έχουν ανατεθεί αρχικές τιμές. Για τον λόγο αυτό πριν τα στοιχεία χρησιμοποιηθούν σε εκφράσεις μέσα σε συναρτήσεις πρέπει πρώτα να λάβουν τιμές.



Σχήμα 5.3: Α) η οργάνωση του μονοδιάστατου πίνακα *coeff* πριν τις αναθέσεις και Β) μετά από αναθέσεις τιμών στα τρια πρώτα στοιχεία του πίνακα .

Η αναφορά σε ένα στοιχείο του πίνακα επιτυγχάνεται με τον προσδιορισμό: του ονόματος του πίνακα και του αριθμοδείκτη που αντιστοιχεί στο στοιχείο του πίνακα. Τυπικά γράφουμε:

Όνομα_πίνακα[αριθμοδείκτης]

Σε ένα πρόγραμμα μια αναφορά σε ένα στοιχείο του πίνακα λειτουργεί όπως και μια μεταβλητή. Δηλαδή, οι παρακάτω εντολές αναθέτουν τιμές στα στοιχεία του πίνακα που αντιστοιχούν στους αναφερόμενους αριθμοδείκτες:

coeff[0] = 4;
coeff[1] = 5;
coeff[2] = 3;

Μετά τις παραπάνω αναθέσεις, ο πίνακας *coeff* αποκτά τη μορφή που εμφανίζεται στο Σχήμα 5.3 Β). Επίσης, σε μια έκφραση, ο αριθμοδείκτης μπορεί να είναι μεταβλητή όπως στη παρακάτω εντολή επανάληψης:

```
for(i=0; i<=N; i++) printf("%d", coeff[i]);
```

Η αρχικοποίηση ενός πίνακα μπορεί να επιτευχθεί στην αρχή της εκτέλεσης του προγράμματος με τη στατική ανάθεση τιμών στα στοιχεία ενός πίνακα όπως φαίνεται στο παρακάτω παράδειγμα:

$$int\ decimalDigits[10] = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\};$$

Με τη στατική ανάθεση τιμών στα στοιχεία του πίνακα μπορεί να ορισθεί το μέγεθος του πίνακα αυτόματα χωρίς την αναγραφή του πλήθους των στοιχείων εντός των αγκυλών στους μονοδιάστατους πίνακες όπως φαίνεται στο παρακάτω παράδειγμα:

$$int\ decimalDigits[] = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\};$$

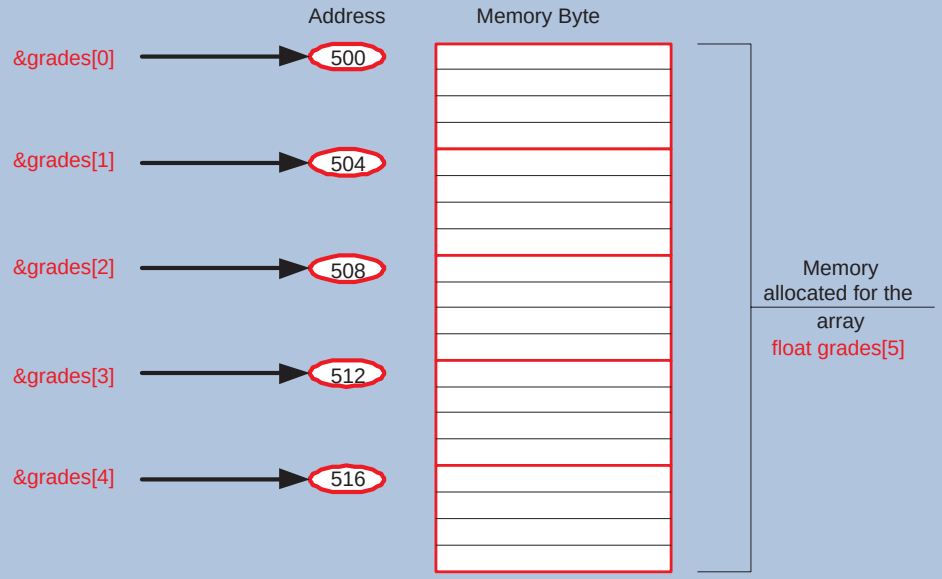
Στην περίπτωση του αυτόματου προσδιορισμού του μεγέθους ενός πίνακα, ο προγραμματιστής μπορεί να υπολογίσει το μέγεθος του πίνακα με τη βοήθεια του τελεστή *sizeof* όπως φαίνεται στην έκφραση:

$$X = sizeof(decimalDigits)/sizeof(decimalDigits[0]);$$

όπου η έκφραση $sizeof(decimalDigits[0])$ επιστρέφει το πλήθος των *byte* που καταλαμβάνει το πρώτο στοιχείο του πίνακα *decimalDigits*[0] (και κατ' επέκταση κάθε στοιχείου του πίνακα) και η έκφραση $sizeof(decimalDigits)$ επιστρέφει το πλήθος των *byte* που καταλαμβάνει ο πίνακας στο σύνολο του.

5.2.2. Κατανομή byte σε πίνακα

Μια δήλωση μονοδιάστατου πίνακα στη C *float grades*[5] σημαίνει ότι ο μεταγλωττιστής θα κατανέμει στον πίνακα *grades* συνολικά $5 * 4 = 20$ *byte* οργανωμένα σε ομάδες των 4 *byte* καθώς κάθε αριθμός κινητής υποδιαστολής απλής ακρίβειας απαιτεί 4 *byte* (*sizeof*(*float*)) και ο πίνακας είναι 5 στοιχείων. Η διεύθυνση του πρώτου στοιχείου του πίνακα ονομάζεται **διεύθυνση βάσης** του πίνακα και αντιστοιχεί στο όνομα του πίνακα. Στο Σχήμα 5.4 δίνεται ένα παράδειγμα κατανομής *byte* στον πίνακα *grade* που ορίσαμε παραπάνω. Η διεύθυνση βάσης του πίνακα είναι η 500.



Σχήμα 5.4: Κατανομή byte στον πίνακα grades πέντε στοιχείων που καθένας είναι αριθμός κινητής υποδιαστολής απλής ακρίβειας.

Μια παράσταση αναφοράς στο στοιχείο *grades*[*i*] οδηγεί τον μεταγλωτιστή στον υπολογισμό της διεύθυνσης &*grade*[*i*] του στοιχείου αυτού βάσει του τύπου:

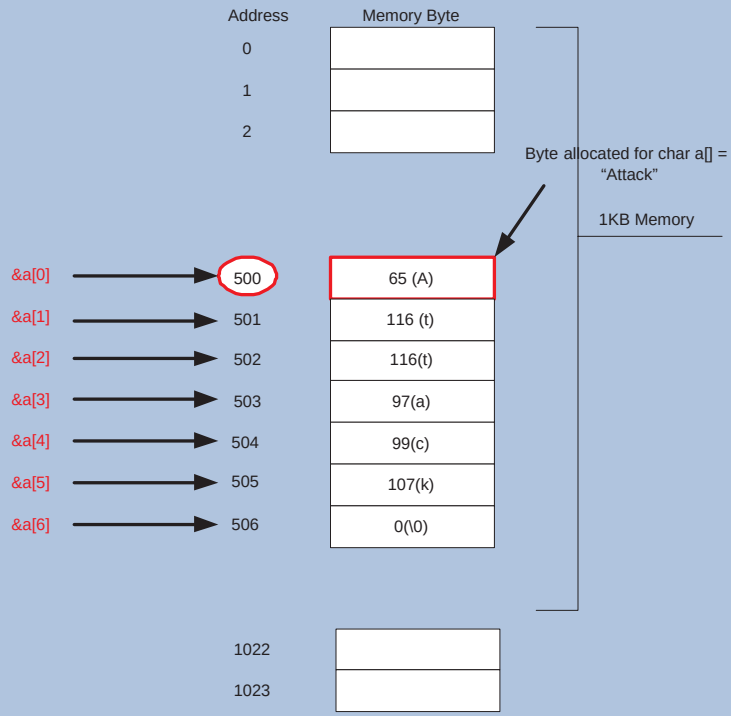
$$\&grade[i] = sizeof(float) * i + \text{<διεύθυνση\_βάσης\_πίνακα>}$$

όπου το γινόμενο $sizeof(float) * i$ ονομάζεται σχετική απόσταση (*offset*) του στοιχείου *grades*[*i*] από τη διευθυνση βάσης του πίνακα. Στο παράδειγμα του Σχήματος 5.4 ο τύπος υπολογισμού έχει ως εξής:

$$\&grade[i] = 500 + 4 * i$$

Προσοχή χρειάζεται για τις τιμές του αριθμοδείκτη καθώς μια λανθασμένη τιμή σε μια έκφραση μπορεί να οδηγήσει σε αναφορά εκτός ορίων του πίνακα. Μια καλή προγραμματιστική τακτική είναι να ελέγχεται η τιμή του σε σχέση με τη μέγιστη που μπορεί να πάρει. Όταν ο αριθμοδείκτης είναι παράμετρος σε συνάρτηση να ελέγχεται η τιμή του εντός της συνάρτησης.

5.2.3. Αλφαριθμητικά



Σχήμα 5.5: Κατανομή byte στο αλφαριθμητικό char a[] ="Attack".

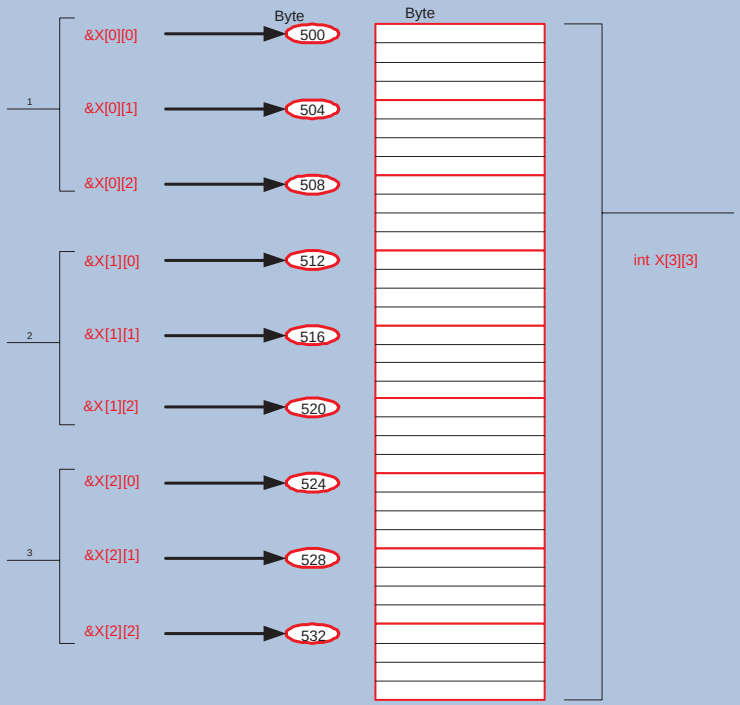
Στη *C*, ένα αλφαριθμητικό (*string*) μπορεί να μοντελοποιηθεί μέσω ενός μονοδιάστατου πίνακα. Το μέγεθος του char a[]="Attack" σύμφωνα με αυτά που προαναφέραμε ισούται με $sizeof(a) = 7$, παρόλο που οι ορατοί χαρακτήρες είναι 6 διότι ο μεταγλωτιστής δεσμεύει ένα επιπλεόν *byte* για τον χαρακτήρα τερματισμού του αλφαριθμητικού $'\backslash 0'$. Στο Σχήμα 5.5 δίνεται η κατανομή *byte* για το παραπάνω αλφαριθμητικό όπου σε κάθε δεσμευμένο *byte* διακρίνεται τόσο ο ASCII κωδικός που είναι αποθηκευμένος στο *byte* όσο και ο χαρακτήρας που αντιστοιχεί σε αυτόν. Η διεύθυνση βάσης του πίνακα που αντιστοιχεί στο όνομα του πίνακα είναι η διεύθυνση του πρώτου στοιχείου $\&a[0] = 500$.

Η επεξεργασία καθενός από τους παραπάνω χαρακτήρες γίνεται πολύ απλά χρησιμοποιώντας την παράσταση αναφοράς σε στοιχείο του πίνακα, όπως για παράδειγμα

$a[i] = 65 + (5 * a[i] + 6) \ mod \ 26;$
$for(i = 0; a[i]! =' \backslash 0'; i + +)$ <έκφραση>;
Ή
$for(i = 0; i < sizeof(a); i + +)$ <έκφραση>;

Στην πρώτη έκφραση επεξεργαζόμαστε κάθε χαρακτήρα του πίνακα όπως θα επεξεργαζόμασταν και έναν ακέραιο καθώς όπως αναφέραμε η τιμή μιας μεταβλητής τύπου χαρακτήρα είναι ο ASCII κωδικός που αντιστοιχεί στον χαρακτήρα. Στην πρώτη εντολή επανάληψης *for*, στην έκφραση ελέγχου εξετάζουμε αν ο τρέχον χαρακτήρας του πίνακα είναι ο τερματικός χαρακτήρας $'\backslash 0'$ ενώ στη δεύτερη περίπτωση ελέγχουμε αν η τρέχουσα τιμή του αριθμοδείκτη i είναι μικρότερη από το μέγεθος του πίνακα. Και οι δύο εντολές επανάληψης είναι ισοδύναμες.

5.2.4. Πολυδιάστατοι Πίνακες



Σχήμα 5.6: Κατανομή byte στον πίνακα *int X*[3][3]

Μέχρι τώρα, είδαμε μονοδιάστατους πίνακες όπου κάθε στοιχείο του πίνακα ήταν του ίδιου απλού τύπου. Τα στοιχεία ενός πίνακα όμως μπορεί να είναι και αυτά πίνακες ορίζοντας κατά αυτόν τον τρόπο πολυδιάστατους πίνακες. Για να ορίσουμε στη *C* ένα διδιάστατο πίνακα ακεραίων *X*, των 3 γραμμών και 3 στηλών γράφουμε *int X*[3][3]; όπου ο πρώτος αριθμοδείκτης καθορίζει το πλήθος των γραμμών και ο δεύτερος το πλήθος των στηλών. Ο μεταγλωτιστής όταν συναντά έναν τέτοιο ορισμό κατανέμει *byte* στον πίνακα με τον τρόπο που φαίνεται στο Σχήμα 5.6. Συγκεκριμένα, απαιτείται χώρος μνήμης για συνολικά $3 \times 3 = 9$ ακεραίους. Δεδομένου ότι για τον τύπο *int* απαιτούνται $sizeof(int) = 4$ *byte* δεσμεύονται για τον πίνακα *X* συνολικά $9 \times 4 = 36$ *byte*. Αυτά οργανώνονται σε τετράδες καθώς $sizeof(int) = 4$ *byte* οι οποίες κατανέμονται ως εξής: οι πρώτες τρεις τετράδες κατανέμονται στα στοιχεία της πρώτης γραμμής, οι επόμενες τρεις στα στοιχεία της δεύτερης γραμμής και οι τελευταίες τρεις στα στοιχεία της τρίτης γραμμής. Με άλλα λόγια ο πίνακας αποθηκεύεται κατά γραμμές. Έτσι η διεύθυνση του δεύτερου στοιχείου της πρώτης γραμμής &*X*[0][1] (προσοχή η αρίθμηση των στοιχείων και των γραμμών του πίνακα ξεκινά από το 0)είναι σύμφωνα με το Σχήμα 5.6 η 504.

Η αρχικοποίηση ενός διδιάστατου πίνακα μπορεί να γίνει με στατική ανάθεση τιμών στα στοιχεία του πίνακα όπως φαίνεται στο παρακάτω παράδειγμα.

$$int\ identity2D[][3] = \{\{1,0,0\},\{0,1,0\},\{0,0,1\}\};$$

Επιπρόσθετα, με την στατική ανάθεση τιμών σε πίνακα όπως προαναφέραμε μπορεί να ορισθεί το μέγεθος του πίνακα αυτόματα χωρίς την αναγραφή του πρώτου αριθμοδείκτη στους πολυδιάστατους. Τότε για να υπολογίζουμε το πλήθος των γραμμών του πίνακα χρησιμοποιούμε τον τύπο

$$lines = sizeof(identity2D)/(sizeof(int) * 3);$$

όπου στον παρονομαστή έχουμε το πλήθος των *byte* για κάθε γραμμή και στον αριθμητή το πλήθος των *byte* για τον πίνακα συνολικά. Για να αθροίσουμε τα στοιχεία του πίνακα *X* που ορίσαμε ανα γραμμή γράφουμε:

```
for(i = 0; i < 3; i++)
    for(j = 0;j < 3;j++)a[i]+ = x[i][j];
```

όπου κάθε βήμα της εντολής επανάληψης με δείκτη i περιλαμβάνει την εκτέλεση τριών αθροίσεων αυτών που προκύπτουν από την εκτέλεση της εντολής επανάληψης με δείκτη j. Για να υπολογίσουμε το άθροισμα του πίνακα ανα στήλη γράφουμε:

```
for(j = 0;j < 3;j++)
    for(i = 0; i < 3; i++)a[j]+ = x[i][j];
```

5.2.5. Ο πίνακας ως τυπική παράμετρος συνάρτησης

Κατά τον ορισμό ενός στατικού πίνακα καθορίζεται το μέγεθος του είτε άμεσα είτε έμμεσα (μέσω της ανάθεσης συγκεκριμένων τιμών). Στον ορισμό του πίνακα τίθεται ένα μέγιστο μέγεθος το οποίο ικανοποιεί στην πράξη όλες τις δυνατές περιπτώσεις της εφαρμογής. Ο μεταγλωττιστής κατανέμει τόσα *byte* όσα καθορίζει το μέγεθος του πίνακα το οποίο ονομάζεται **κατανεμημένο μέγεθος (***allocated size***)**. Στην πράξη βέβαια χρησιμοποιείται μέρος του κατανεμημένου πίνακα (σπατάλη χώρου) το μέγεθος του οποίου ονομάζεται **αποτελεσματικό μέγεθος (***effective size***)**. Συνεπώς, όταν ένας στατικός πίνακας είναι τυπική παράμετρος μιας συνάρτησης, τότε για να είναι αποτελεσματική η εκτέλεση της συνάρτησης πρέπει να δηλώνεται και μια δεύτερη παράμετρος που δεν είναι άλλη από αυτή του αποτελεσματικού μεγέθους.

Η δήλωση ενός στατικού πίνακα ως παραμέτρου μιας συνάρτησης είναι ίδια με την απλή δήλωση του με τη διαφορά ότι:

1. στους μονοδιάστατους μπορεί να μην δηλωθεί το μέγεθος του πίνακα, όπως για παράδειγμα *int copyString*(*char* s[], *char* t[]);,
2. ενώ στους πολυδιάστατους μπορεί να μην δηλωθεί ο πρώτος αριθμοδείκτης του πίνακα, όπως για παράδειγμα *int multiplyMatrices*(*int* s[][4], *int* t[][3]);

Στο κεφάλαιο 4 αναφέραμε ότι κατά την κλήση μια συνάρτησης η καλούσα συνάρτηση αντιγράφει το πραγματικό όρισμα στην παράμετρο της καλούμενης συνάρτησης (κλήση με τιμή, *call by value*). Επειδή λοιπόν η καλούμενη συνάρτηση επεξεργάζεται αντίγραφο του πραγματικού ορίσματος, το πραγματικό όρισμα παραμένει ως ήταν. Τα πράγματα είναι διαφορετικά όταν η παράμετρος της συνάρτησης είναι πίνακας:

1. Κατά την κλήση της συνάρτησης, η καλούσα συνάρτηση αντιγράφει στο πλαίσιο στοίβας της καλούμενης συνάρτησης τη διεύθυνση βάσης του πίνακα και όχι όλον τον πίνακα. Ο τρόπος αυτός περάσματος παραμέτρων ονομάζεται **κλήση με αναφορά (***call by reference***)**.

2. Η συνάρτηση στη συνέχεια μπορεί να επεξεργάζεται τα πραγματικά δεδομένα του πίνακα και όχι αντίγραφα όπως συμβαίνει με την περίπτωση απλών μεταβλητών. Με άλλα λόγια τόσο η καλούσα όσο και η καλούμενη συνάρτηση χρησιμοποιούν τον ίδιο χώρο αποθήκευσης.

Από τα παραπάνω γίνεται αντιληπτός ένας ακόμα τρόπος επιστροφής τιμών στην καλούσα συνάρτησης εκτός του συνήθους με την εντολή *return*. Συγκεκριμένα, η καλούσα συνάρτηση περνά στην καλούμενη συνάρτηση έναν πίνακα είτε κενό είτε με συγκεκριμένες τιμές και η καλούμενη συνάρτηση όταν επιστρέφει τον έλεγχο στην καλούσα έχει τροποποιήσει τα δεδομένα του αρχικού πίνακα.

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2: #include <ctype.h>
 3: #include <string.h>
 4:
 5: #define N 20
 6:
 7: void ConvertToUpperCase(char s[], char d[]);
 8:
 9: int main()
10: {
11:  char s[N+1], d[N+1], c;
12:  int i=0;
13:
14:  printf("Plhktrologhste alfarhthmitiko mhkous <= 20\n");
15:
16:  while((c=getchar())!='\n' && i<N) {
17:      s[i] = c;
18:      i++;
19:    }
20:  s[i]='\0';
21:  printf("Arxiko String: %s\n", s);
22:  ConvertToUpperCase(s,d);
23:  printf("Antigrafo: %s\n", d);
24:  return 0;
25: }
26:
27: void ConvertToUpperCase(char source[], char dest[])
28: {
29:    int i;
30:
31:    for(i=0; source[i]!='\0'; i++){
32:       if (isalpha(source[i]) && islower(source[i])) {
33:          dest[i] = toupper(source[i]);
34:       }
35:       else dest[i]=source[i];
36:    }
37:    dest[i]='\0';
38: }
```

Σχήμα 5.7: Συνάρτηση μετατροπής αλφαριθμητικού.

Για να το κατανοήσουμε ας δούμε ένα παράδειγμα. Έστω ότι να υλοποιησουμε τη συνάρτηση *ConvertToUpperCase* η οποία θα δέχεται σαν όρισμα ένα αλφαριθμητικό *s* και θα επιστρέφει ένα αντίγραφο του αλφαριθμητικού στο οποίο όλοι οι αλφαβητικοί χαρακτήρες θα έχουν μετατραπεί σε κεφαλαίους. Στο Σχήμα 5.7 δίνεται μια υλοποίηση της συνάρτησης αυτής αλλά και της κύριας συνάρτησης που την καλεί. Η κύρια συνάρτηση του προγράμματος διαβάζει το πολύ 20 αλφαριθμητικούς χαρακτήρες, καλεί την παραπάνω συνάρτηση και στη συνέχεια τυπώνει τόσο το αλφαριθμητικό που διαβάστηκε αρχικά όσο και το αυτό που επιστράφηκε από την *ConvertToUpperCase*. Η συνάρτηση *ConvertToUpperCase* δέχεται ως πρώτη παράμετρο τον πηγαίο πίνακα *source* και ως δεύτερη παράμετρο το πίνακα προορισμού όπου θα αποθηκευθεί το τροποιημένο αντίγραφο. Στο σώμα της συνάρτησης εξετάζεται κάθε χαρακτήρας του πηγαίου αλφαριθμητικού μέχρι να συναντηθεί ο χαρακτήρας τερματισμού αλφαριθμητικού '\0'. Με την χρήση των κατηγορηματικών συναρτήσεων *isalpha* και *islower*() που είναι δηλωμένες στο αρχείο επικεφαλίδα < *ctype.h* > εξετάζεται ο χαρακτήρας είναι γράμμα και πεζός. Σε περίπτωση που η απάντηση είναι θετική τότε καλείται η συνάρτηση της πρότυπης βιβλιοθήκης (αρχείο επικεφαλίδα < *string.h* >) *toupper*() η οποία μετατρέπει τον πεζό χαρακτήρα σε κεφαλαίο και το αποτέλεσμα αποθηκεύεται στην αντίστοιχη θέση του πίνακα προορισμού. Στη γραμμή 37 με την ολοκλήρωση της μετατροπής βάζουμε στο τέλος του αλφαριθμητικού προορισμού τον χαρακτήρα τερματισμού '\0'.

```
0:  #include <stdio.h>
1:  #include <stdlib.h>
2:
3:  #define N 5
4:
5:  float evaluate(int p[], float x)
6:  {
7:     int i;
8:     float temp, result;
9:
10:    result = x*p[N];
11:    for(i=N-1; i>0; i--)
12:    {
13:       temp = p[i] + result;
14:       result = x*temp;
15:    }
16:    return (p[0]+result);
17: }
18:
19: void addPolyonyms(int p[], int q[], int w[])
20: {
21:    int i;
22:
23:    for(i=0; i<=N; i++) w[i] = p[i]+q[i];
24: }
25:
26: void printPolyonym(int p[])
27: {
28:    int i;
29:
30:    for(i=N; i>0; i--)
31:    {
32:       if (p[i]!=0) printf("%d x^%d + ", p[i], i);
33:    }
34:    printf("%d",p[0]);
35:    printf("\n");
36: }
```

Σχήμα 5.8: Βιβλιοθήκη συναρτήσεων που επεξεργάζονται ακέραια πολυώνυμα

Ένα άλλο παράδειγμα που ο πίνακας είναι μια βολική δομή δεδομένων για μοντελοποίηση είναι αυτό της ανάπτυξης μιας βιβλιοθήκης συναρτήσεων που μοντελοποιούν τις λειτουργίες των ακέραιων πολυωνύμων. Ένα ακέραιο πολυώνυμο p μιας πραγματικής μεταβλητής x βαθμού το πολύ N και πραγματικούς συντελεστές $a_0, a_1, \ldots, a_N$ μπορεί να αναπαρασταθεί από ένα μονοδιάστατο πίνακα μεγέθους $(N+1)$ στον οποίο αποθηκεύονται οι συντελεστές του πολυωνύμου p. Η τιμή ενός πολυωνύμου p για κάποιο x μπορεί πολύ εύκολα να υπολογισθεί επαναληπτικά με τη χρήση του κανόνα του Horner:

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{N-1} + x(a_N))))$$

Το πολυώνυμο που προκύπτει από την πρόσθεση δύο πολυωνύμων p και q βαθμού το πολύ N μπορεί να υπολογισθεί απλά προσθέτοντας τους συντελεστές των αντίστοιχων

δυνάμεων των επιμέρους πολυωνύμων:

$$\begin{aligned} d(x) &= p(x) + q(x) \\ &= (a_0 + b_0) + (a_0 + b_0)x + \ldots + (a_n + b_n)x^N \end{aligned} \tag{5.1}$$

Στο Σχήμα 5.8 δίνονται οι εξής συναρτήσεις:

1. η *evaluate*() η οποία έχει ως τυπικές παραμέτρους έναν πίνακα με τους συντελεστές ενός πολυωνύμου p και ένα αριθμό κινητής υποδιαστολής απλής ακρίβειας x. Υπολογίζει την τιμή πολυωνύμου $p(x)$ και επιστρέφει την τιμή αυτή. Η χρήση του κανόνα του Horner για τον υπολογισμό της τιμής του πολυωνύμου καθιστά τη συνάρτηση αποτελεσματική καθώς η μόνη χρονοβόρα πράξη που απαιτείται είναι ο πολλαπλασιασμός ενώ αποφεύγεται η χρήση συναρτήσεων υπολογισμού δυνάμεων του x. Η εφαρμογή του κανόνα είναι απλή καθώς στη γραμμή 10 υπολογίζεται αρχικά το $x \times p[n]$ και στη συνέχεια εκτελείται η εντολή επανάληψης η οποία σε κάθε βήμα προσθέτει τον τρέχοντα συντελεστή του πολυωνύμου στο αποτέλεσμα του προηγούμενου βήματος.

2. η *addPolyonyms*() η οποία έχει ως τυπικές παραμέτρους τρεις πίνακες p, q, w όπου p και q είναι τα πολυώνυμα εισόδου και w είναι το πολυώνυμο που προκύπτει από το άθροισμα των πολυωνύμων p και q.

3. η *printPolyonym*() η οποία έχει ως τυπική παράμετρο έναν πίνακα με τους συντελεστές ενός πολυωνύμου p και η εκτέλεση της έχει ως αποτέλεσμα την εκτύπωση του πολυωνύμου.

Οι συνάρτησεις *addPolyonyms*() και *printPolyonym* είναι διαδικασίες καθώς δεν επιστρέφουν τιμή μέσω της εντολής *return*. Στο Σχήμα 5.9 δίνεται το κυρίως πρόγραμμα το οποίο διαβάζει τους συντελεστές $a_0, a_1, \ldots, a_N$ ενός πολυωνύμου p, τους αποθηκεύει σε

έναν μονοδιάστατο πίνακα που ορίζει το πολυώνυμο p. Στη συνέχεια διαβάζει την τιμή του x, καλεί τη συνάρτηση υπολογισμού της τιμής του πολυωνύμου για το δοθέν x και τυπώνει το αποτέλεσμα στη οθόνη. Τέλος, διαβάζει τους συντελεστές $b_0, b_1, \ldots, b_N$ ενός δεύτερου πολυωνύμου q, τους αποθηκεύει σε έναν μονοδιάστατο πίνακα που ορίζει το πολυώνυμο q, καλεί τη συνάρτηση που αθροίζει τα πολυώνυμα p και q και τυπώνει τους συντελεστές των πολυωνύμων p και q όσο και του αθροίσματος. Παρατηρείστε ότι στις δηλώσεις των πινάκων το μέγεθος τους ισούται με μια συμβολική σταθερά N που συμβολίζει τον βαθμό του πολυωνύμου προσαυξημένη κατά ένα.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  #define N 5
 4:
 5:  float evaluate(int p[], float x);
 6:  void addPolyonyms(int p[], int q[], int w[]);
 7:  void printPolyonym(int p[]);
 8:
 9:  int main()
10:  {
11:   int p[N+1], q[N+1], w[N+1], i;
12:   float x;
13:
14:      printf("Eisagvgh sytelestvn polyonymou q\n");
15:      for(i=N; i>=0; i--)
16:      {
17:         printf("Dvse ton synetlesth tou x^%d=", i);
18:         scanf("%d", &p[i]);
19:         printf("\n");
20:      }
21:
22:      printf("Dvse timh sth metavlhth x=");
23:      scanf("%f", &x);
24:      printf("\n");
25:
26:      printf("p(%f) = %f\n", x, evaluate(p, x));
27:
28:      printf("Eisagvgh syntelestvn polyonymou q\n");
29:      for(i=N; i>=0; i--)
30:      {
31:         printf("Dvse ton synetlesth tou x^%d=", i);
32:         scanf("%d", &q[i]);
33:         printf("\n");
34:      }
35:      addPolyonyms(p, q, w);
36:      printPolyonym(p);
37:      printPolyonym(q);
38:      printPolyonym(w);
39:
40:   return 0;
41:  }
```

Σχήμα 5.9: Πρόγραμμα που καλεί συναρτήσεις της βιβλιοθήκης του Σχήματος 5.8

5.3. Δείκτες

Στο εδάφιο 5.1 ορίσαμε τη διεύθυνση μιας μεταβλητής x ως τη διεύθυνση της ομάδας των *byte* που δεσμεύεται για την αποθήκευση της τιμής της μεταβλητής x. Έστω ότι η διεύθυνση μιας ομάδας 8 *byte* που κατανέμεται για τη μεταβλητή x τύπου *double* είναι ο ακέραιος αριθμός 500 όπως φαίνεται στο Σχήμα 5.2. Όπως προαναφέραφε τη διεύθυνση της μεταβλητής x μπορούμε να τη βρούμε με τη βοήθεια του τελεστή &, δηλαδή $\&x$. Στη *C* τη διεύθυνση μιας ομάδας από *byte* μπορούμε να τη χειριστούμε ως ένα δεδομένο. Αυτό επιτυγχάνεται με τις μεταβλητές τύπου **δείκτη (pointer)** οι τιμές των οποίων είναι διευθύνσεις ομάδων από *byte*. Μια μεταβλητή τύπου δείκτη δηλώνεται ως εξής:

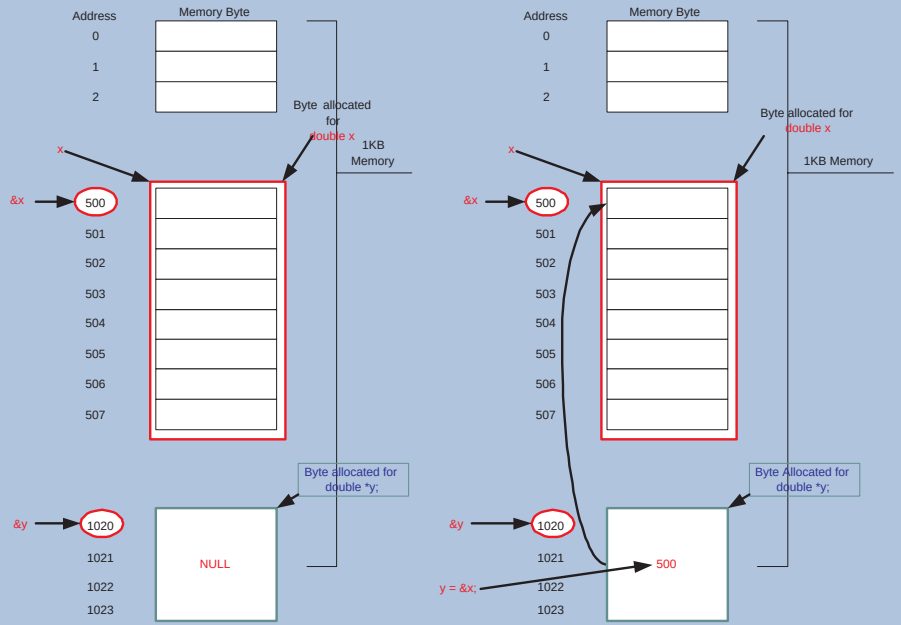
<τύπος_δεδομένου> *<όνομα_μεταβλητής>;

Η τιμή μιας μεταβλητής τύπου δείκτη δεν είναι τίποτε άλλο παρά ένας ακέραιος. Ας δούμε ένα παράδειγμα για να το καταλάβουμε. Έστω οι παρακάτω δηλώσεις:

double x;
double $*\, y$;

Ο μεταγλωτιστής όταν συναντά μια δήλωση της μορφής *double* $*\, y$ κατανέμει 4 *byte* στη μεταβλητή y (όσα χρειάζονται για μια μεταβλητή τύπου *int*) και της δίνει ως αρχική τιμή τη σταθερά *NULL* (τιμή 0) η οποία είναι δηλωμένη στο αρχείο επικεφαλίδα *stdlib.h*. Αυτό σημαίνει ότι το περιεχόμενο της δεν είναι μια έγκυρη διεύθυνση και ως εκ τούτου δεν δείχνει σε έγκυρα δεδομένα. Η μεταβλητή τύπου δείκτη y έχει ασφαλώς διεύθυνση $\&y$. Για τη μεταβλητή x του παραδείγματος μας δεσμεύεται μια ομάδα των 8 *byte* (όσα απαιτούνται για ένα αριθμό κινητής υποδιαστολής διπλής ακρίβειας) η διεύθυνση της οποία είναι η $\&x$. Στο αριστερό τμήμα τους Σχήματος 5.10 η διεύθυνση της μεταβλητής x είναι η $\&x = 500$ ενώ η διεύθυνση της μεταβλητής y είναι η $\&y = 1020$ καθώς έχουν δεσμευθεί τα τελευταία 4 *byte* για τη μεταβλητή αυτή τα οποία έχουν πάρει την αρχική τιμή *NULL*. Όταν αναφερόμαστε στη μεταβλητή y θα λέμε ότι είναι ένας δείκτης σε

αριθμό κινητής υποδιαστολή διπλής ακρίβειας διότι η τιμή της κατά κάποιο τρόπο δείχνει σε αριθμό κινητής υποδιαστολής διπλής ακρίβειας καθώς είναι η διεύθυνση του πρώτου *byte* του αριθμού.



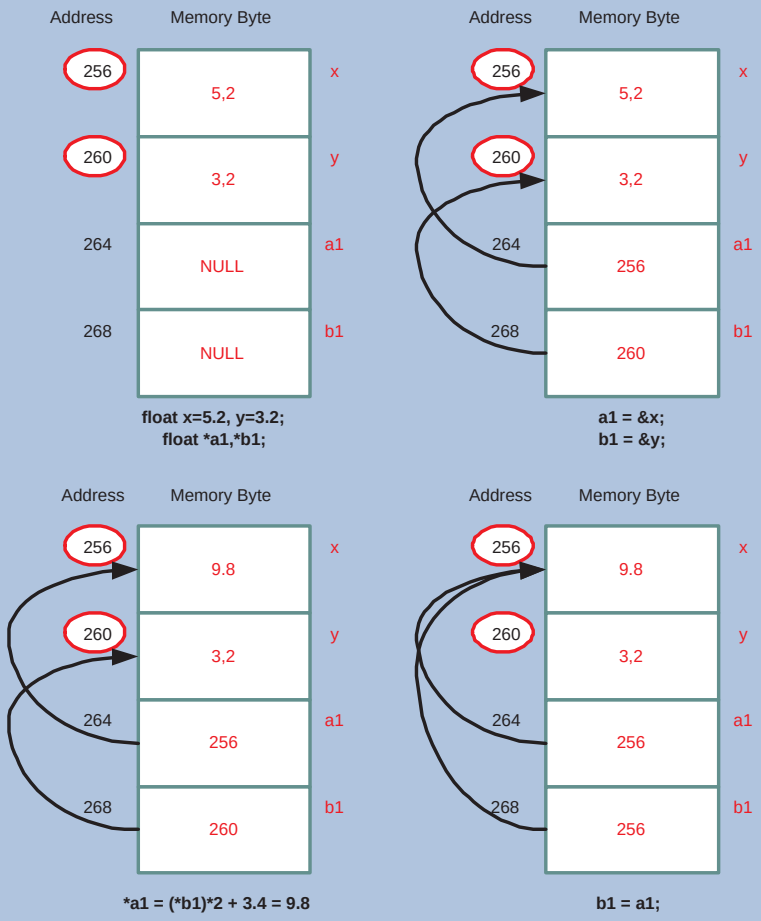
Σχήμα 5.10: Η διεύθυνση ως δεδομένο

Στο σημείο αυτό αν χρησιμοποιηθεί μια μεταβλητή τύπου δείκτη που έχει τιμή *NULL* είναι πολύ πιθανό το πρόγραμμα να καταρρεύσει χωρίς να γίνει αντιληπτό από τον προγραμματιστή ο λόγος κατάρρευσης. Μια ανάθεση του τύπου

$y = \&x;$

έχει ως αποτέλεσμα η μεταβλητή y να αποκτήση ως τιμή τη διεύθυνση της μεταβλητής x δηλαδή $y = 500$. Στο σημείο αυτό η μεταβλητή τύπου δείκτη y δείχνει σε έγκυρα δεδομένα όπως φαίνεται στο δεξί μέρος του Σχήματος 5.10. Μετά την ανάθεση αυτή μπορούμε να ανακτήσουμε την τιμή της x μέσω της μεταβλητής y. Όμως με ποιο τρόπο μπορούμε να ανακτήσουμε το περιεχόμενο της x μέσω της y? Η πρόσβαση στη τιμή της x μέσω της y επιτυγχάνεται μέ τον μοναδιαίο τελεστή * γράφοντας $*y$. Έτσι, οι δύο παρακάτω εκφράσεις είναι ισοδύναμες, δίνουν δηλαδή το ίδιο αποτέλεσμα:

$$((*y) * 5 + 3)/10$$
$$(x * 5 + 3)/10$$



Σχήμα 5.11: Παράδειγμα εκφράσεων με δείκτες

Ας δούμε ακόμα ένα παράδειγμα. Έστω οι παρακάτω δηλώσεις:

float $x = 5.2,\ y = 3.2;$
float $*a1,\ *b1;$

όπου ορίζονται δύο μεταβλητές x και y κινητής υποδιαστολής απλής ακρίβειας (*float*) με αρχικές τιμές 5.2 και 3.2 αντίστοιχα και δύο δείκτες σε αριθμούς κινητής υποδιαστολής $a1$ και $b1$ με αρχική τιμή *NULL*. Το αποτέλεσμα των παραπάνω δηλώσεων κατά τη μεταγλώττιση φαίνεται στο Σχήμα 5.11 στο πάνω αριστερά στιγμιότυπο της μνήμης του προγράμματος μας. Επίσης, διακρίνουμε ότι οι διευθύνσεις 256 και 260 έχουν ανατεθεί στις μεταβλητές x και y αντίστοιχα ενώ στις μεταβλητές $a1$ και $b1$ έχουν ανατεθεί οι διευθύνσεις 264 και 268. Με τις παρακάτω αναθέσεις τιμών στους δείκτες $a1$ και $b1$:

$a1 = \&x;$
$b1 = \&y;$

αρχικοποιούνται οι δείκτες $a1$ και $b1$ στο να δείχνουν στις τιμές των x και y αντίστοιχα. Το αποτέλεσμα της εκτέλεσης των παραπάνω εντολών διακρίνεται στο πάνω δεξιά στιγμιότυπο της μνήμης του προγράμματος μας.

Στη συνέχεια με την ακόλουθη εντολής ανάθεσης

$$*a1 \quad = \quad (*b1) * 2 + 3.4; \tag{5.2}$$

η τιμή της έκφρασης στο δεξί μέρος της εντολής ανάθεσης που βασίζεται στην τιμή της y λόγω του δείκτη $b1$ που δείχνει στην τιμή της y αποτελεί τη νέα τιμή της μεταβλητής x καθώς το $*a1$ σημαίνει το περιεχόμενο της ομάδας από *byte* με διεύθυνση $a1$ που από την προηγούμενη ανάθεση είναι η διεύθυνση της μεταβλητής x. Στο κάτω αριστερά στιγμιότυπο της μνήμης του προγράμματός μας του Σχήματος 5.11 διακρίνεται το αποτέλεσμα αποτίμησης της έκφρασης 5.2 όπου η μεταβλητή x πήρε την τιμή 9.8.

Τέλος, με την εντολή ανάθεσης

$b1 = a1;$

ο δείκτης $b1$ δείχνει πλέον και αυτός στη μεταβλητή x όπως φαίνεται και στο κάτω δεξιά στιγμιότυπο της μνήμης του προγράμματος μας στο Σχήμα 5.11.

5.4. Ο Δείκτης ως τυπική παράμετρος συνάρτησης

Στο εδάφιο 5.2.5 εισάγαμε για πρώτη φορά τη κλήση συνάρτησης με αναφορά όπου ως τυπική παράμετρος μιας συνάρτησης χρησιμοποιούνταν πίνακες. Όταν καλείται μια συνάρτηση με τυπική παράμετρο πίνακα, η καλούσα συνάρτηση αντιγράφει τη διεύθυνση βάσης του πίνακα και όχι τα στοιχεία του πίνακα στη τυπική παράμετρο. Κατά αυτόν τον τρόπο η καλούμενη συνάρτηση έχει πρόσβαση στα στοιχεία του πίνακα τα οποία μπορεί και επεξεργάζεται. Η κλήση της συνάρτησης με αναφορά δίνει επίσης τη δυνατότητα να επιστρέφονται περισσότερες από μια τιμές από την καλούμενη συνάρτηση στην καλούσα καθώς με την εντολή *return* μόνο μια τιμή κάθε φορά μπορεί να επιστραφεί.

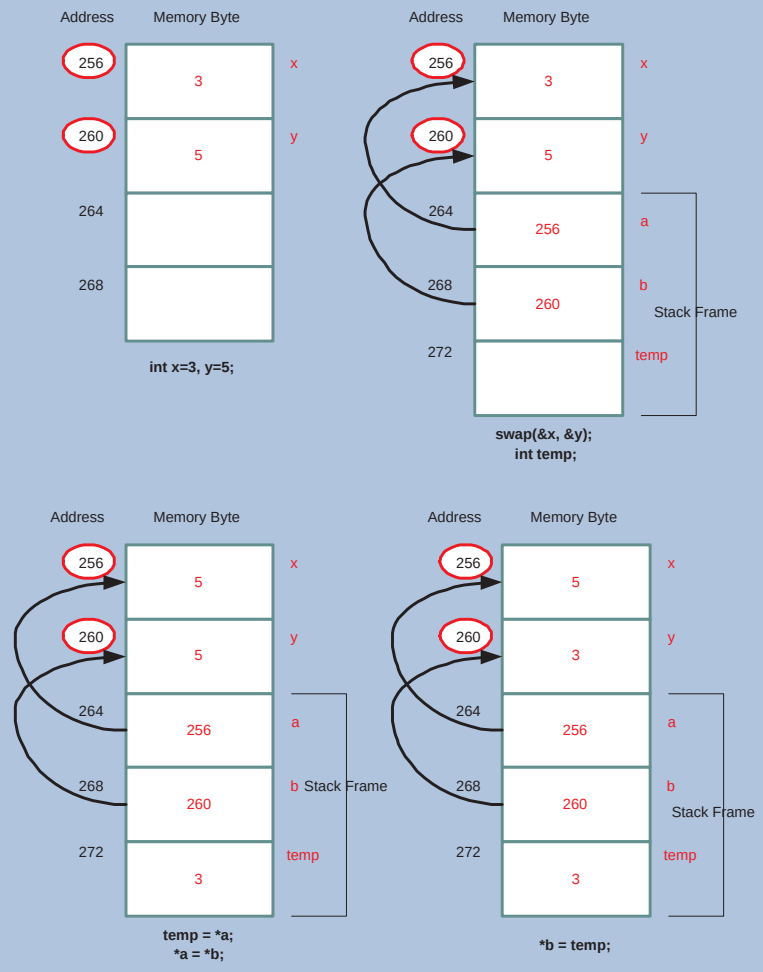
```
0:  #include <stdio.h>
1:
2:  int swap(int *, int *);
3:
4:  main(void)
5:  {
6:   int x=3,y=5;
7:
8:   printf("x=%d, y=%d\n", x, y);
9:   swap(&x, &y);
10:  printf("x=%d, y=%d\n", x, y);
11: }
12:
```

```
13:  int swap(int *a, int *b)
14:{
15:    int temp;
16:temp = *a;
17:*a = *b;
18:*b = temp;
19:}
```

Σχήμα 5.12: Κλήση της συνάρτησης *swap* με αναφορά.

Οι δείκτες διευρύνουν την κλήση συνάρτησης με αναφορά καθώς πλέον όπως θα δούμε μπορούμε να δουλεύουμε με οποιαδήποτε μεταβλητή όχι απαραίτητα πίνακες. Ας δούμε ένα παράδειγμα χρήση των δεικτών στην κλήση συνάρτησης με αναφορά. Στο Σχήμα 5.12 παρουσιάζεται το πρόγραμμα που αντιμεταθέτει τις τιμές των μεταβλητών x και y. Οι μεταβλητές x και y παίρνουν στην αρχή του προγράμματος τις τιμές 3 και 5

αντίστοιχα και τυπώνονται στην οθόνη. Στο πάνω αριστερά στιγμιότυπο της μνήμης του προγράμματος μας του Σχήματος 5.13 φαίνεται το αποτέλεσμα εκτέλεσης των παραπάνω αρχικοποιήσεων.



Σχήμα 5.13: Στιγμιότυπα της μνήμης κατά την κλήση με αναφορά της *swap*

Πριν αναφερθούμε στην κλήση της συνάρτησης *swap* ας εξετάσουμε την επικεφαλίδα της συνάρτησης αυτής. Η συνάρτηση δέχεται ως τυπικές παραμέτρους δύο δείκτες *a* και *b* σε ακεραίους. Στη δήλωση του πρωτοτύπου της συνάρτησης στη γραμμή 2 βλέπουμε στη θέση των τυπικών παραμέτρων να αναγράφεται ο τύπος, δηλαδή *int*, ακολουθούμενος από τον μοναδιαίο τελεστή * χωρίς να είναι απαραίτητη η αναγραφή του ονόματος της παραμέτρου, που όπως είπαμε είναι πιο πολύ περιγραφική η παρουσία του ονόματος της παραμέτρου. Όταν καλείται η συνάρτηση *swap* δεσμεύεται χώρος μνήμης για το πλαίσιο στοίβας της συνάρτησης που θα πρέπει να καλύπτει τις τυπικές παραμέτρους και την προσωρινή μεταβλητή *temp* του σώματος της συνάρτησης όπως φαίνεται στο πάνω δεξιά στιγμιότυπο της μνήμης στο Σχήμα 5.13. Στη συνέχεια, οι διευθύνσεις των μεταβλητών *x* και *y* (&*x* και &*y* αντίστοιχα) αντιγράφονται στις μεταβλητές *a* και *b* που όπως είπαμε είναι δείκτες σε ακεραίους. Έτσι στο Σχήμα 5.13 βλέπουμε οι τιμές των δεικτών *a* και *b* να είναι 256 και 260 αντίστοιχα που είναι οι διευθύνσεις των μεταβλητών *x* και *y*.

Στο σώμα της συνάρτησης η προσωρινή μεταβλητή *temp* δέχεται ως τιμή των ακέραιο στον οποίο δείχνει ο δείκτης *a*. Στη συνέχεια ο ακέραιος στον οποίο δείχνει ο δείκτης *b*, που είναι η τιμή της *y*, αποθηκεύεται στη θέση που δείχνει ο δείκτης *a*, που είναι η μεταβλητή *x*. Το αποτέλεσμα εκτέλεσης των εντολών στις γραμμές 16 και 17 φαίνεται στο κάτω αριστερά στιγμιότυπο της μνήμης στο Σχήμα 5.13 όπου η μεταβλητή *x* πήρε την τιμή της *y*. Η τελευταία εντολή της συνάρτησης η τιμή της *temp* αποθηκεύεται στη θέση που δείχνει ο δείκτης *b*, που είναι η μεταβλητή *y*, ολοκληρώνοντας κατά αυτόν τον τρόπο την αντιμετάθεση. Το αποτέλεσμα εκτέλεσης της εντολής αυτής φαίνεται στο κάτω δεξιά στιγμιότυπο της μνήμης στο Σχήμα 5.13 όπου η μεταβλητή *y* πήρε την τιμή της *x*.

5.5. Δείκτες και πίνακες

Στο εδάφιο 5.2 αναφέραμε ότι το όνομα ενός πίνακα αντιπροσωπεύει τη διεύθυνση βάσης του πίνακα που δεν είναι άλλη από τη διεύθυνση του πρώτου στοιχείου του πίνακα. Στο σημείο αυτό θα εξετάσουμε πως μπορούμε να αξιοποιήσουμε τους δείκτες που ορίσαμε

στο προηγούμενο εδάφιο προκειμένου να επεξεργαστούμε τα στοιχεία ενός πίνακα.

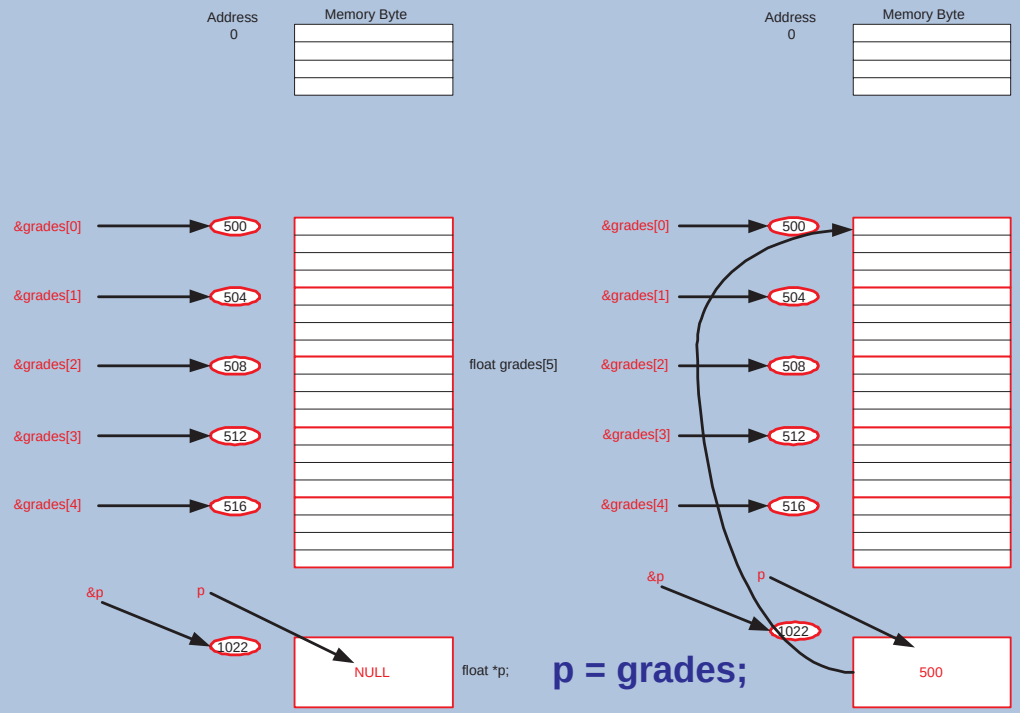
Για παράδειγμα, έστω οι παρακάτω δηλώσεις μεταβλητών:

float grades[5];
float $*$ *p*;

Με την πρώτη δήλωση ορίζεται ένας πίνακας 5 στοιχείων τύπου *float* στον οποίο ανατίθενται μια ομάδα 20 *byte* με διεύθυνση πρώτου στοιχείου &*grade*[0] το 500 δεύτερου στοιχείου &*grade*[1] το 504, τρίτου στοιχείου &*grade*[2] το 508, τέταρτου στοιχείου &*grade*[3] το 512 και πέμπτου &*grade*[4] το 516. Όπως προαναφέραμε, για οποιονδήποτε πίνακα στη *C* ισχύει

grades ≡ &*grades*[0]

δηλαδή το όνομα του πίνακα αντιπροσωπεύει τη διεύθυνση του πρώτου στοιχείου του πίνακα που είναι η διεύθυνση βάσης του πίνακα. Με τη δεύτερη δήλωση ορίζεται ένας δείκτης *p* σε *float* του οποίου η διεύθυνση &*p* είναι το 1022 και ο οποίος αρχικοποιείται με τη σταθερά *NULL*. Στο Σχήμα 5.14 στο αριστερό στιγμιότυπο της μνήμης φαίνεται το αποτέλεσμα των παραπάνω δηλώσεων.



Σχήμα 5.14: Στιγμιότυπα της μνήμης κατά την ανάθεση της διεύθυνσης βάσης πίνακα σε δείκτη

Η ανάθεση της μορφής

$p = grades$;

αναθέτει τη διεύθυνση βάσης του πίνακα *grades* στο δείκτη p όπως φαίνεται στο δεξί στιγμιότυπο της μνήμης του Σχήματος 5.14. Κατά κάποιο τρόπο ο δείκτης p και το όνομα του πίνακα *grades* μπορούν να χρησιμοποιηθούν εναλλακτικά. Με άλλα λόγια, τα στοιχεία του πίνακα είναι πλέον προσβάσιμα και μέσω του δείκτη p γράφοντας απλά για παράδειγμα για το j-στο στοιχείο $p[j]$.

Οι τελεστές + και - όταν εφαρμοστούν σε δείκτες ορίζουν την **αριθμητική δεικτών**. Έτσι στο παράδειγμα μας, η έκφραση $p + j$ προσδιορίζει τη διεύθυνση του στοιχείου του πίνακα $grades[j]$. Συγκεκριμένα, για κάθε μονάδα που προστίθεται στον δείκτη p, εσωτερικά ισοδυναμεί με αύξηση κατά *sizeof*(*float*) δεδομένου ότι τα στοιχεία του πίνακα είναι *float* δηλαδή η εσωτερική τιμή που προστίθεται κάθε φορά εξαρτάται από τον τύπο των στοιχείων του πίνακα. Για να επεξεργαστούμε το στοιχείο $grades[j]$ του πίνακα μέσω του δείκτη p γράφουμε $*(p + j)$.

Ανακεφαλαιώνοντας, μετά την ανάθεση $p = grades$ μπορούμε να επεξεργαστούμε τα στοιχεία του πίνακα μέσω του δείκτη p με δύο τρόπους:

1. ως στοιχείο πίνακα $p[j]$

2. με αριθμητική δεικτών $*(p + j)$

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2: #include <ctype.h>
 3: #include <string.h>
 4:
 5: #define N 20
 6:
 7: void ConvertToUpperCase(char *s, char *d);
 8:
 9: int main()
10: {
11:  char s[N+1], d[N+1], c;
12:  int i=0;
13:
14:  printf("Plhktrologhste alfarhthmitiko mhkous <= 20\n");
15:
16:  while((c=getchar())!='\n' && i<N) {
17:     s[i] = c;
18:     i++;
19:    }
20:  s[i]='\0';
21:  printf("Arxiko String: %s\n", s);
22:  ConvertToUpperCase(s,d);
23:  printf("Antigrafo: %s\n", d);
24:  return 0;
25: }
26:
27: void ConvertToUpperCase(char *source, char *dest)
28: {
29:    int i;
30:
31:    for(i=0; source[i]!='\0'; i++){
32:       if (isalpha(source[i]) && islower(source[i])) {
33:          dest[i] = toupper(source[i]);
34:       }
35:       else dest[i]=source[i];
36:    }
37:    dest[i]='\0';
38: }
```

Σχήμα 5.15: Συνάρτηση μετατροπής αλφαριθμητικού με τη χρήση δεικτών ως τυπικές παραμέτρους

Στο Σχήμα 5.7 δόθηκε το πρόγραμμα μετατροπής των χαρακτήρων ενός αλφαριθμητικού σε κεφαλαία γράμματα όπου στον ορισμό της *ConvertToUpperCase* χρησιμοποιήθηκαν πίνακες ως τυπικές παράμετροι. Το αποτέλεσμα θα είναι το ίδιο αν χρησιμοποιήσουμε δείκτες αντί για πίνακες τόσο στον ορισμό όσο και στη δήλωση του πρωτοτύπου της συνάρτησης. Στο Σχήμα 5.15 δίνεται η νέα έκδοση του προγράμματος όπου στη λίστα των τυπικών παραμέτρων της συνάρτησης *ConvertToUpperCase* περιλαμβάνονται οι δείκτες *source* και *dest*. Αξιοσημείωτο είναι ότι το σώμα της συνάρτησης όπως και ο τρόπος κλήσης της δεν άλλαξε. Έτσι όταν θέλουμε να καλέσουμε την συνάρτηση αρκεί η αναγραφή των ονομάτων των πινάκων όπως φαίνεται και στο παράδειγμα.

Η μοναδική πράξη που ορίζεται μεταξύ δεικτών είναι αυτή της αφαίρεσης που δίνει το πλήθος των στοιχείων μεταξύ των δύο δεικτών. Για παράδειγμα αν ο δείκτης $p1$ είναι η διεύθυνση του *grades*[2] και ο $p2$ είναι η διεύθυνση του *grades*[0] τότε η έκφραση $p1 - p2$ θα δώσει 2 καθώς μεταξύ των διευθύνσεων των 2 δεικτών υπάρχουν 2 στοιχεία του πίνακα. Μια έκφραση της μορφής $*p++$ σημαίνει ότι αρχικά με τον τελεστή * ανακτάται το περιεχόμενο της θέσης που δείχνει ο δείκτης p και μετά ο δείκτης p αυξάνεται κατά ένα και είναι ισοδύναμη με την $*(p++)$.

Στο Κεφάλαιο 4 αναφέρθηκε ότι μια από τις χρήσεις της λέξης κλειδί void είναι ότι ορίζει ένα δείκτη σε μη προσδιορισμένο τύπο δεδομένων. Δηλαδή μια δήλωση της μορφής

void $*$ p;

ορίζει έναν δείκτη p σε απροσδιόριστο τύπο δεδομένου. Σε ένα γενικό δείκτη μπορεί να εκχωρηθεί η τιμή ενός δείκτη σε οποιοδήποτε τύπο δεδομένων. Αν q είναι ένας δείκτης σε *int* και w είναι ένας δείκτης σε *double*, η ανάθεση $p = q$ είναι επιτρεπτή και σημαίνει πλέον ότι ο p δείχνει σε ακέραιο ενώ η ανάθεση $p = w$ σημαίνει ότι ο p δείχνει σε *double*.

5.6. Δυναμική Κατανομή Μνήμης και Δυναμικοί Πίνακες

Μέχρι τώρα είδαμε δύο μηχανισμούς κατανομής μνήμης στις μεταβλητές ενός προγράμματος:

1. **Στατική Κατανομή**: Με την έναρξη εκτέλεσης ενός προγράμματος δεσμεύονται θέσεις μνήμης για τις καθολικές ή εξωτερικές μεταβλητές και οι οποίες διατηρούνται μέχρι την ολοκλήρωση του προγράμματος.

2. **Αυτόματη Κατανομή**: Για τις τοπικές μεταβλητές στο σώμα των συναρτήσεων, η μνήμη κατανέμεται από τη στοίβα (*stack*) του συστήματος που έχει διατεθεί για το προς εκτέλεση πρόγραμμα και το τμήμα της στοίβας που κατανέμεται στη συνάρτηση ονομάζεται πλαίσιο στοίβας.

Η *C* διαθέτει μια διεργασία κατανομής μνήμης κατά απαίτηση η οποία ονομάζεται **δυναμική κατανομή**. Όταν απαιτείται για την λειτουργία του προγράμματος η κατανομή χώρου μνήμης σε μεταβλητή του προγράμματος, τότε διατίθεται στη μεταβλητή αυτή ο απαραίτητος χώρος από το σωρό (*heap*). Για την υποστήριξη της δυναμικής κατανομής η πρότυπη βιβλιοθήκη της *C* διαθέτει δύο συνάρτησεις τα πρότυπο των οποίων είναι δηλωμένο στο αρχείο επικεφαλίδα *stdlib.h*:

1. **void *calloc(size_t, size_t)** η οποία έχει ως τυπικές παραμέτρους το πλήθος των στοιχείων (πρώτη παράμετρος) και το μέγεθος κάθε στοιχείου σε *byte* (δεύτερο όρισμα). Δεσμεύει τόσα *byte* όσο είναι το γινόμενο του μεγέθους κάθε στοιχείου επί το πλήθος των στοιχείων, τα αρχικοποιεί με 0 και επιστρέφει έναν γενικό δείκτη. Σε περίπτωση σφάλματος, επιστρέφει *NULL*.

2. **void *malloc(size_t)**: η οποία έχει ως τυπική παράμετρο το πλήθος των *byte* που καλείται να δεσμεύσει. Αν η κλήση είναι επιτυχής επιστρέφει έναν γενικό δείκτη διαφορετικά *NULL*.

Κατά βάση η χρήση των συναρτήσεων *malloc* και *calloc* είναι ισοδύναμη. Η διαφορά τους είναι ότι η *calloc* αρχικοποιεί τον χώρο που δεσμεύεται. Αν δεν είναι απαραίτητη η αρχικοποίηση των θέσεων μνήμης που δεσμεύονται, καλό είναι να χρησιμοποιείται η *malloc* για μεγαλύτερη ταχύτητα καθώς αποφεύγεται ο χρόνος αρχικοποίησης, ειδικά όταν το πλήθος των στοιχείων είναι μεγάλο.

Ο γενικός δείκτης που επιστρέφεται από τις *malloc* και *calloc* μπορεί να καταχωρηθεί σε οποιαδήποτε μεταβλητή τύπου δείκτη είτε απευθείας είτε μετά από ρητή μετατροπή τύπου (*cast*). Για παράδειγμα οι παρακάτω κλήσεις της συνάρτησης *calloc*:

```
int * a, n;
a = calloc(n, sizeof(int));
a = (int *) calloc(n, sizeof(int));
```

καθώς στην πρώτη ο γενικός δείκτης που επιστρέφει η συνάρτηση μετατρέπεται σε δείκτη σε *int* αυτόματα κατά την ανάθεση ενώ στη δεύτερη περίπτωση πρώτα γίνεται η μετατροπή με τη βοήθεια του τελεστή *cast* και μετά γίνεται η ανάθεση. Το αποτέλεσμα όμως είναι το ίδιο.

Όταν στο σωρό δεν υπάρχει χώρος για να ικανοποιηθεί η κλήση της *malloc* ή *calloc* τότε οι συναρτήσεις επιστρέφουν *NULL*. Επειδή όπως αναφέραμε η χρήση ενός δείκτη που είναι *NULL* μπορεί να οδηγήσει σε αστάθεια του συστήματος για τον λόγο αυτό αμέσως μετά την καταχώρηση σε μια μεταβλητή τύπου δείκτη του γενικού δείκτη που επιστρέφεται από τις *malloc* και *calloc* καλό είναι να ελέγχεται αν είναι *NULL* ή όχι. Ενδεικτικά μπορεί να γραφούν οι παρακάτω γραμμές προγράμματος αμέσως μετά την παραπάνω κλήση της *calloc*:

```
if(a==NULL){
  printf("Πρόβλημα μνήμης:Δεν ήταν δυνατή η δέσμευση byte για τον δείκτη a");
  exit(1);
}
```

όπου όταν διαπιστωθεί ότι ο δείκτης a είναι *NULL* τυπώνεται το παραπάνω μήνυμα και στη συνέχεια εκτελείται η εντολή *exit*(1) η οποία τερματίζει την εκτέλεση του προγράμματος. Κατά αυτόν τρόπο μπορεί πολύ εύκολα να διαπιστωθεί η ύπαρξη προβλημάτων κατανομής μνήμης και να επιλυθούν, διευκολύνοντας την αποσφαλμάτωση των προγραμμάτων.

Όταν η μνήμη που έχει δυναμικά δεσμευθεί (από τον προγραμματιστή κατά την εκτέλεση του προγράμματος) δεν χρειάζεται μπορεί να αποδεσμευθεί. Η αποδέσμευση γίνεται με τη συνάρτηση free η οποία έχει ως τυπική παράμετρο δείκτη. Κατά την κλήση της free ο δείκτης με τον οποίο θα κληθεί θα πρέπει να είναι αυτός που επιστράφηκε αρχικά από τις *malloc* και *calloc*. Η συνάρτηση free επιστρέφει στο σωρό (heap) το τμήμα μνήμης που αντιστοιχεί στο δείκτη της τυπικής παραμέτρου της. Το τμήμα αυτό μπορεί να χρησιμοποιηθεί από άλλο τμήμα του προγράμματος που ενδεχομένως να ζητήσει μέσω της malloc ή calloc. Αν η επιστροφή των συναρτήσεων δεν συνοδεύεται από απελευθέρωση του χώρου μνήμης που δυναμικά έχει εκχωρηθεί σε τοπικές μεταβλητές τους τότε υπάρχει το ενδεχόμενο να εξαντληθεί ο σωρός και να μην είναι δυνατή η περαιτέρω δυναμική εκχώρηση μνήμης σε μεταβλητές του προγράμματος. Για αυτό πριν την επιστροφή πρέπει να καλείται η free για την αποδέσμευση των χώρων μνήμης που έχουν κατανεμηθεί σε μεταβλητές της.

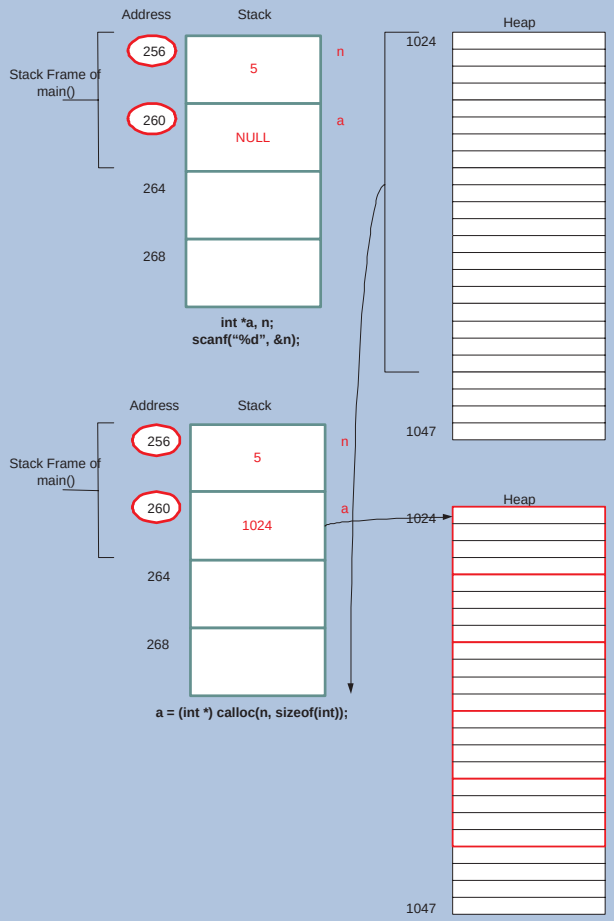
```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  void fillData(int *a, int n);
 4:
 5:  main(void)
 6:  {
 7:     int *a, n;
 8:
 9:     printf("Plhthos tvn stoixeivn tou pinaka:");
10:     scanf("%d", &n);
11:
12:     a = (int *) calloc(n, sizeof(int));
13:     fillData(a, n);
14:
15:     free(a);
16:  }
17:
```

```
18:  void fillData(int *b, int m)
19:  {
20:     int i;
21:
22:     for(i=0; i<m; i++) b[i] = rand() % 16;
23:  }
```

Σχήμα 5.16: Απλό παράδειγμα δημιουργίας δυναμικού πίνακα

Στην αρχή του κεφαλαίου είδαμε ότι το μέγεθος ενός πίνακα μπορεί να ορισθεί άμεσα με τον προσδιορισμό του μέγιστου αριθμού των στοιχείων του στη δήλωση του. Κατά την εκτέλεση του προγράμματος όμως μόνο ένα μέρος του πίνακα χρησιμοποιείται πράγμα που οδηγεί σε σπατάλη πόρων μνήμης. Στα ενσωματωμένα συστήματα (embedded systems) ειδικά, που οι πόροι τους είναι περιορισμένοι αυτό είναι πρόβλημα. Οι συναρτήσεις δυναμικής ανάθεσης μνήμης σε μεταβλητές είναι η λύση του προβλήματος. Ανάλογα με τις απαιτήσεις της εφαρμογής μας κάθε φορά καλείται η συνάρτηση malloc ή calloc με συγκεκριμένο αριθμό στοιχείων. Κατά αυτό τον τρόπο μπορούμε να δημιουργήσουμε **δυναμικούς πίνακες**. Στο Σχήμα 5.16 δίνεται ένα πολύ απλό πρόγραμμα που κάνει χρήση του μηχανισμού που περιγράφηκε. Συγκεκριμένα, με την scanf στην αρχή

του προγράμματος διαβάζεται το πλήθος των στοιχείων του πίνακα μας n. Στο πάνω μέρος του Σχήματος 5.17 διακρίνεται το στιγμιότυπο της μνήμης του προγράμματός μας μετά την εκτέλεση των εντολών των γραμμών 9 – 10, όπου η μεταβλητή n πήρε την τιμή 5 και ο δείκτης a έχει αρχικοποιηθεί με την *NULL*.



Σχήμα 5.17: Στιγμιότυπα της μνήμης κατά την δημιουργία δυναμικού πίνακα

Στη γραμμή 12 του προγράμματος του Σχήματος 5.16 γίνεται κλήση της calloc ζητώντας $n = 5$ στοιχεία μεγέθους $sizeof(int) = 4$ *byte*, δηλαδή 20 *byte* τα οποία δεσμεύονται από τον σωρό heap και επιστρέφονται στην main και με τη μετατροπή τύπου μορφοποιούνται σε 5 ομάδες των 4 *byte*. Στο κάτω μέρος του Σχήματος 5.17 βλέπουμε ότι η διεύθυνση του πρώτου από τα 20 *byte* του σωρού που δευσμεύθηκαν με την κλήση της calloc αποτελεί την τιμή του δείκτη *a*. Στη γραμμή 13 καλείται η συνάρτηση *fillData* με ορίσματα τον δείκτη *a* και το πλήθος *n* των στοιχείων του δυναμικού πίνακα που δημιουργήθηκε με την calloc. Στο σώμα της συνάρτησης *fillData* στον πίνακα *a* αποθηκεύονται τυχαίοι αριθμοί στο διάστημα 0 . . . 15 καλώντας επαναληπτικά τη συνάρτηση *rand* η οποία είναι συνάρτηση της πρότυπης βιβλιοθήκης με το πρωτότυπό της δηλωμένο στο αρχείο επικεφαλίδα *stdlib.h*. Με την ολοκλήρωση του προγράμματος μας, που σηματοδοτείται με την ολοκλήρωση της συνάρτησης main, απελευθερώνεται η δεσμευμένη μνήμη με την κλήση της συνάρτησης free.

Ας δούμε ακόμα ένα παράδειγμα. Θέλουμε να αναπτύξουμε μια συνάρτηση που

- θα κατασκευάζει ένα δυναμικό πίνακα αριθμών κινητής υποδιαστολής απλής ακρίβειας μεγέθους *N* το οποίο θα είναι μια από τις παραμέτρους της συνάρτησης,
- θα διαβάζει *N* αριθμούς κινητής υποδιαστολής απλής ακρίβειας και θα τους καταχωρεί στον πίνακα αυτό,
- θα υπολογίζει την ελάχιστη τιμή (*min*), τη μέγιστη τιμή (*max*) και την μέση τιμή (*mean*) των αριθμών κινητής υποδιαστολής τις οποίες θα επιστρέφει στο κυρίως πρόγραμμα μέσω δεικτών στη λίστα παραμέτρων της συνάρτησης και
- θα αποδεσμεύει τον χώρο που δέσμευσε για τον πίνακα και θα επιστρέφει στο κυρίως πρόγραμμα.

Στο Σχήμα 5.18 εμφανίζεται η συνάρτηση *statistics* η οποία έχει τέσσερεις τυπικές παραμέτρους: το πλήθος των αριθμών κινητής υποδιαστολής απλής ακρίβειας *N* που θα

διαβασθούν και θα επεξεργασθούν, και τους δείκτες *mean*, *min*, *max* σε αριθμούς κινητής υποδιαστολής απλής ακρίβειας (*float*). Με την ανάθεση της γραμμής 9 δημιουργείται ένας πίνακας από *N* αριθμούς κινητής υποδιαστολής απλής ακρίβειας ενώ μέσω επαναληπτικών κλήσεων της *scanf* στις γραμμές 11 – 16 τα στοιχεία του πίνακα αποκτούν τιμές

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:  #include <mem.h>
 3:
 4:  void statistics(int N, float *mean, float *min, float *max)
 5:  {
 6:     int i;
 7:     float *p;
 8:
 9:     p = (float *) malloc(N*sizeof(float));
10:
11:     for(i=0; i<N; i++)
12:     {
13:        printf("Dvse ton %d o arithmo=", i+1);
14:        scanf("%f", &p[i]);
15:        printf("\n");
16:     }
17:
18:     for(i=0; i<N; i++)   *mean += p[i];
19:     *mean /= N;
20:     *min=p[0]; *max=p[0];
21:
22:    for(i=0; i<N; i++)
23:    {
24:       *min = (*min <= p[i]) ? *min : p[i];
25:       *max = (*max >= p[i]) ? *max : p[i];
26:    }
27:    free(p);
28: }
```

Σχήμα 5.18: Παράδειγμα Δυναμικής Κατανομής Μνήμης

Στη συνέχεια με τις εντολές των γραμμών 17 – 22 υπολογίζεται η μέση τιμή των αριθμών που διαβάσθηκαν ενώ με τις εντολές των γραμμών 24 – 30 υπολογίζονται η ελάχιστη και η μέγιστη τιμή. Αξίζει να σημειωθεί ότι στο σώμα της συνάρτησης χρησιμοποιούνται

οι τυπικές παράμετροι της συνάρτησης για την επιστροφή των τιμών που θέλουμε να υπολογίσουμε. Επειδή είναι δείκτες, για να επεξεργαστούμε τον αριθμό στον οποίο δείχνουν χρησιμοποιούμε τον τελεστή *. Να θυμίσουμε ότι όταν κληθεί η συνάρτηση, η καλούσα θα αντιγράψει στις εν λόγω παραμέτρους τις διευνθύσεις των ορισμάτων της και όχι τις τιμές αυτών. Συνεπώς, στο σώμα της συνάρτησης μπορούμε εύκολα να επεξεργαστούμε τις τιμές μέσω του τελεστή *. Στη γραμμή 31 απελευθερώνεται ο χώρος που δεσμεύθηκε με τη malloc. Μια προφανής βελτίωση του κώδικα του Σχήματος 5.18 είναι η συγχώνευση των επαναληπτικών εντολών των γραμμών 18 – 21 και 26 – 30 βελτιώνοντας αρκετά τον χρόνο εκτέλεσης της συνάρτησης ειδικά για μεγάλο N.

5.7. Επικοινωνία Προγράμματος με Λειτουργικό Σύστημα

Μέχρι τώρα η κύρια συνάρτηση main των προγραμμάτων που παρουσιάσαμε δεν είχε τυπικές παραμέτρους ενώ όλες οι άλλες διέθεταν. Θα συμπεράνε κανείς λανθασμένα ότι η main δεν απαιτεί παραμέτρους. Στη *C* η κύρια συνάρτηση *main* διαθέτει δύο τυπικές παραμέτρους:

- *int argc*: δίνει το πλήθος των ορισμάτων της κλήσης του πρόγραμματος συμπεριλαμβανομένου και του ονόματος του προγράμματος.
- *char *argv[]*: Πίνακας δεικτών σε αλφαριθμητικά strings τα οποία είναι τα ορίσματα που το πρόγραμμα χρειάζεται για να λειτουργήσει σωστά.

μέσω των οποίων ένα πρόγραμμα μπορεί να επικοινωνήσει με το λειτουργικό σύστημα αλλά και άλλα προγράμματα. Επειδή τα ορίσματα είναι υπο μορφή ακολουθίας χαρακτήρων, ο προγραμματιστής οφείλει να τα μετασχηματίσει στη μορφή που θέλει.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  int main(int argc, char *argv[])
 4:  {
 5:     int i;
 6:
 7:     if (argc > 1) {
 8:        if (!strcmp(argv[1], "-d"))
 9:           printf("The integer is:%d\n", atoi(argv[2]));
10:        else if (!strcmp(argv[1], "-f"))
11:           printf("The floating point number is:%f\n", atof(argv[2]));
12:         else {
13:               printf("Bad Command\n");
14:               return 1;
15:     }
16:     else{
17:        printf("Give the option -d <number> or -f <number>");
18:         return 1;
19:     }
20:
21:     return 0;
22:  }
```

Σχήμα 5.19: Πρόγραμμα που υλοποιεί την εντολή *convertStringToNumber*

Για να το κατανοήσουμε ας δούμε ένα παράδειγμα. Έστω ότι θέλουμε να υλοποιήσουμε μια καινούργια εντολή για το λειτουργικό σύστημα όπως είναι οι εντολή *Dir* στο *Dos* και η *ls* στο *Unix* η οποία θα δέχεται μια ακολουθία ψηφίων την οποία θα μετατρέπει σε ακέραιο ή αριθμό κινητής υποδιαστολής απλής ακρίβειας αν δοθεί η επιλογή $-d$ ή $-f$ αντίστοιχα. Ας ονομάσουμε την εντολή αυτή *convertStringToNumber* η σωστή σύνταξη της οποίας σύμφωνα με τις παραπάνω προδιαγραφές είναι:

1. *convertStringToNumber* $-d$ $<$ *string* $>$

2. *convertStringToNumber* $-f$ $<$ *string* $>$

Βλέπουμε ότι το πλήθος των ορισμάτων της εντολής αυτής είναι 3: το όνομα της convertStringToNumber, η επιλογή $-d$ ή $-f$ και το αλφαριθμητικό $<$ *string* $>$. Δημιουργούμε ένα αρχείο με όνομα convertStringToNumber.c στο οποίο γράφουμε το πρόγραμμα που εμφανίζεται στο Σχήμα 5.19. Η συνάρτηση main πλέον είναι πλήρης με τις δύο τυπικές παραμέτρους της. Όπως αναφέραμε η παράμετρος *argc* έχει ως τιμή το πλήθος των ορισμάτων της κλήσης του προγράμματος. Αρχικά στη γραμμή 7 γίνεται ο έλεγχος του πλήθους των ορισμάτων ο οποίος αν είναι μικρότερος ή ίσος του 1 τυπώνεται μήνυμα που διευκρινίζει στον χρήστη τη σύνταξη της εντολής και το πρόγραμμα επιστρέφει την τιμή 1 στο λειτουργικό σύστημα. Αν είναι μεγαλύτερος από ένα τότε εξετάζουμε τα στοιχεία του πίνακα *argv*. Δεδομένου ότι *argv*[0] =“convertStringToNumber” αυτό που πρέπει να ελέγξουμε είναι αν το *argv*[1] περιέχει το “-d” ή το “-f”. Σε περίπτωση που ικανοποιείται μια από τις παραπάνω συνθήκες τότε καλείται μια εκ των συναρτήσεων *atoi* ή *atof* ανάλογα με το αν η τιμή του *argv*[1] είναι “-d” ή “-f” αντίστοιχα. Αν δεν ικανοποιείται καμία από τις συνθήκες αυτές τότε τυπώνεται μήνυμα λάθους. Δοκιμάστε να εκτελέσετε σε *command* περιβάλλον (σε περιβάλλον Windows επιλέξτε **έναρξη** και μετά **εκτέλεση**, τρέξτε την εντολή **cmd** και μετά μετακινηθείτε στο φάκελο που βρίσκεται το εκτελέσιμο που προέκυψε από τη μεταγλώττιση) τις εξής εντολές:

1. *convertStringToNumber*

2. *convertStringToNumber* $-f$ 534.2

3. *convertStringToNumber* $-d$ 534.2

5.8. Συναρτήσεις για αλφαριθμητικά

Στο Κεφάλαιο 2 ορίσαμε ότι ένα αλφαριθμητικό είναι (string constant) μια ακολουθία από μηδέν ή περισσότερους χαρακτήρες εντός διπλών εισαγωγικών ενώ στο εδάφιο 5.2.3 είδαμε ότι ένα αλφαριθμητικό στη *C* μοντελοποιείται με μονοδιάστατο πίνακα. Μετά και τον ορισμό του δείκτη μπορούμε τώρα να πούμε ότι ένας δείκτης σε μια ακολουθία χαρακτήρων είναι ένα αλφαριθμητικό.

Συνάρτηση
*size_t strlen(const char * s);*
*char * strcat(char * s1, const char * s2);*
*char * strncat(char * s1, const char * s2, int n);*
*int strcmp(const char * s1, const char * s2);*
*int strncmp(const char * s1, const char * s2, int n);*
*char strcpy(char * s1, char const * s2);*
*char strncpy(char * s1, char const * s2, int n);*

Πίνακας 5.1: Συναρτήσεις αλφαριθμητικών της πρότυπης βιβλιοθήκης της *C*.

Η ANSI C παρέχει στον προγραμματιστή ένα σύνολο συναρτήσεων διαχείρισης αλφαριθμητικών. Στον Πίνακα 5.1 δίνονται οι δηλώσεις των πρωτοτύπων των σημαντικότερων συναρτήσεων αλφαριθμητικών που εμφανίζονται στο αρχείο επικεφαλίδα *string.h*.

Η συνάρτηση *strlen* έχει ως τυπική παράμετρο μια αλφαριθμητική σταθερά και επιστρέφει το πλήθος των χαρακτήρων του αλφαριθμητικού. Στο Σχήμα 5.20 δίνεται το σώμα της συνάρτησης *strlen* το οποίο αποτελείται από μια επαναληπτική εντολή στη οποία απαριθμούνται όλοι οι χαρακτήρες εκτός του χαρακτήρα τερματισμού του αλφαριθμητικού. Αξίζει να δούμε τη χρήση της αριθμητικής δεικτών όπου με την έκφραση $s++$ έχουμε πρόσβαση σε όλους τους χαρακτήρες του αλφαριθμητικού.

```
 0:  size_t strlen(const char *s)
 1:  {
 2:      size_t n;
 3:
 4:      for (n=0; *s!='\0'; s++) ++n;
 5:
 6:      return n;
 7:  }
 8:
 9:  char *strcpy(char *s1, const char *s2)
10:  {
11:      char *temp=s1;
12:
13:      while (*temp++ = *s2++);
14:
15:      return s1;
16:  }
17:
```

```
18:  char *strcat(char *s1, const char *s2)
19:  {
20:      char *temp=s1;
21:
22:      while (*temp) ++temp;
23:      while (*temp++=*s2++)
24:
25:      return s1;
26:  }
```

Σχήμα 5.20: Το σώμα μερικών συναρτήσεων του αρχείου επικεφαλίδα *string.h*

Οι συναρτήσεις *strcat* και *strncat* εκτελούν την ίδια λειτουργία με μια μικρή διαφορά: η *strcat* τοποθετεί όλους τους χαρακτήρες του αλφαριθμητικού $s2$ αμέσως μετά τον τελευταίο χαρακτήρα του αλφαριθμητικού $s1$, ενώ η *strncat* τοποθετεί μόνο τους n πρώτους χαρακτήρες του $s2$. Στο Σχήμα 5.20 δίνεται το σώμα της συνάρτησης *strcat* στο οποίο χρησιμοποιείται μια προσωρινή μεταβλητή *temp* για να εντοπίσει τη θέση του χαρακτήρα τερματισμού στο αλφαριθμητικό $s1$ (γραμμή 22) και στη συνέχεια από το σημείο αυτό και μετά να αντιγράψει τους χαρακτήρες του αλφαριθμητικού $s2$ (γραμμή 23). Επιστρέφει τον δείκτη $s1$.

Η συνάρτηση *strcmp* συγκρίνει το αλφαριθμητικό $s1$ με το αλφαριθμητικό $s2$ και

επιστρέφει 0 σε περίπτωση ισότητας, > 0 όταν το $s1$ είναι λεξικογραφικά μεγαλύτερο από το $s2$ και < 0 όταν είναι λεξικογραφικά μικρότερο. Η *strncmp* συγκρίνει τους n πρώτους χαρακτήρες των αλφαριθμητικών $s1$ και $s2$.

Τέλος, οι συναρτήσεις *strcpy* και *strncpy* εκτελούν την ίδια λειτουργία με μια μικρή διαφορά: η *strcpy* αντιγράφει όλους τους χαρακτήρες του αλφαριθμητικού $s2$ στο $s1$ ενώ η *strncat* αντιγράφει μόνο τους n πρώτους χαρακτήρες του $s2$. Στο Σχήμα 5.20 δίνεται το σώμα της συνάρτησης *strcpy* στο οποίο χρησιμοποιείται μια προσωρινή μεταβλητή *temp* η οποία αρχικοποιείται με την τιμή του δείκτη $s1$ (γραμμή 11) ενώ στη συνέχεια (γραμμή 13) αντιγράφονται οι χαρακτήρες του $s2$ μέσω της *temp* στο $s1$. Επιστρέφει τον δείκτη $s1$. Εδώ θα πρέπει να αναφέρουμε ότι στην καλούσα συνάρτηση θα πρέπει να έχει προβλεφθεί το μέγεθος του $s1$ να είναι αρκετό για να φιλοξενήσει τους χαρακτήρες τους $s2$.

5.9. Ασκήσεις

Άσκηση 5.9.1 *Όταν εκτελεσθεί ο παρακάτω κώδικας τυπώνονται τέσσερεις τιμές. Χωρίς να εκτελέσετε τον κώδικα εξηγείστε τι θα τυπωθεί στην οθόνη ?*

$int\ k = 3;$
$int * u = \&k;$
$printf(\text{“}\%p\ \%d\ \%d\ \%d\ \backslash n\text{”}, u, *u + 4, 4 * **\&u + 2, 3 * (u - (u - 1)));$

Άσκηση 5.9.2 *Να εξηγήσετε τη λειτουργία του προγράμματος του Σχήματος 5.21 η κλήση της calculate(). Ποιό είναι το αποτέλεσμα που θα τυπωθεί στην οθόνη και γιατί?*

```
 1:#  include <stdio.h>
 2:#  include <stdlib.h>
 3:
 4:  int calculate(int, int *);
 5:
 6:  int main()
 7:{
 8:     int i, x;
 9:     int *p;
10:
11:     p=&i;
12:     i=5;
13:    x=6;
14:
15:    calculate(x, p);
16:    printf(   "%d %d\n", x, *p);
17:    return 0;
18:}
19:
```

```
20:  int calculate(int x, int *p)
21:{
22:     int a=4;
23:
24:     x+=a;
25:     *p*=4;
26:
27:     return 1;
28:}
```

Σχήμα 5.21: Κώδικας άσκησης 5.9.2

Άσκηση 5.9.3 *Να γραφεί η συνάρτηση reverse η οποία έχει τυπική παράμετρο ένα αλφαριθμητικό και επιστρέφει το ίδιο αλφαριθμητικό αντεστραμμένο. Δηλαδή αν η τυπική παράμετρος έχει ως τιμή το "Hello World" θα επιστρέψει το "ldroW olleH". Ελέγξτε τη συνάρτηση που γράψατε καλώντας μέσα από την main, τις εξής εντολές:*

char input[] = "*abcdefghijklmnopqrstuvwxyz*";
printf("%s\n", *reverse*(*input*));

Άσκηση 5.9.4 *Να γραφούν οι συναρτήσεις mean() και stdDeviation() οι οποίες έχουν δύο τυπικές παραμέτρους έναν πίνακα με στοιχεία τύπου double και το μέγεθος του πίνακα και επιστρέφουν την μέση τιμή των στοιχείων του πίνακα και την τυπική απόκλιση των στοιχείων*

αυτών από τη μέση τιμή αντίστοιχα. Επιβεβαιώστε την ορθή λειτουργία των συναρτήσεων αυτών γράφοντας ένα πρόγραμμα που θα διαβάζει το πλήθος των στοιχείων του πίνακα, θα δεσμεύει δυναμικά μνήμη για την αποθήκευση των στοιχείων, θα διαβάζει τα στοιχεία και θα τα αποθηκεύει στον πίνακα, θα καλεί τις συναρτήσεις αυτές και θα τυπώνει τις υπολογισθείσες τιμές.

Άσκηση 5.9.5 Να γραφεί πρόγραμμα που θα διαβάζει και θα τυπώνει τα στοιχεία ενός $N \times N$ πίνακα αριθμών κινητής υποδιαστολής διπλής ακρίβειας. Στη συνέχεια να γραφούν και να κληθούν μέσα από το πρόγραμμα αυτό καθεμιά από τις παρακάτω συναρτήσεις:

1. μια συνάρτηση η οποία ελέγχει αν ένας τετραγωνικός πίνακας είναι συμμετρικός και επιστρέφει TRUE ή FALSE ανάλογα με το αποτέλεσμα του ελέγχου.

2. μια συνάρτηση η οποία βρίσκει το μέγιστο κατά απόλυτη τιμή στοιχείο μιας συγκεκριμένης γραμμής του πίνακα και στη συνέχεια διαιρεί τα στοιχεία της γραμμής με το στοιχείο αυτό.

3. μια συνάρτηση η οποία ελέγχει αν ο πίνακας είναι ορθογώνιος και επιστρέφει TRUE ή FALSE ανάλογα με το αποτέλεσμα του ελέγχου.

Κεφάλαιο 6

Δομές

Η πρώτη ομαδοποίηση δεδομένων που παρουσιάσαμε ήταν αυτή του πίνακα στοιχείων. Το μειονέκτημα όμως στους πίνακες είναι ότι ομαδοποιούνται ομοειδή δεδομένα, δηλαδή δεδομένα του ιδίου τύπου. Η *C* διαθέτει τη **δομή ή εγγραφή (*structure*)** για την συνάθροιση μεταβλητών διαφορετικών ή ίδιων τύπων. Για παράδειγμα μπορούμε να μοντελοποιήσουμε ένα σημείο του επιπέδου ως εξής:

```
struct point {
        float x;
        float y;
};
```

όπου η λέξη **struct** είναι λέξη κλειδί, η λέξη *point* είναι το **tag** της δομής που την χαρακτηρίζει και τα *x* και *y* είναι τα **μέλη** της δομής. Από την παραπάνω δήλωση προκύπτει ο τύπος *struct point* χωρίς να γίνεται δέσμευση χώρου μνήμης και λειτουργεί ως *πρότυπο (template)*. Ο τύπος αυτός μπορεί να χρησιμοποιηθεί για να δηλωθούν μεταβλητές όπως:

struct point p1, p2;

όπου οι μεταβλητές $p1$ και $p2$ είναι μεταβλητές τύπου *struct point* και στο σημείο αυτό απαιτείται από τον μεταγλωττιστή χώρος μνήμης για τα δύο μέλη x και y των μεταβλητών $p1$ και $p2$.

Τα ονόματα των μελών σε μια δομή πρέπει να είναι διαφορετικά. Μεταξύ όμως διαφορετικών δομών, τα ονόματα των μελών μπορεί να είναι ίδια. Για την επεξεργασία των μελών μιας δομής μεμονωμένα, η *C* διαθέτει τον τελεστή . ο οποίος όπως φαίνεται και από τον Πίνακα 2.5 έχει την μεγαλύτερη προτεραιότητα μαζί με τον τελεστή ( ) που ορίζει μια συνάρτηση, τον [] που ορίζει έναν πίνακα και τον τελεστή → που θα αναλύσουμε στη συνέχεια. Έτσι σε μια έκφραση για να χρησιμοποιήσουμε ένα μέλος μιας δομής γράφουμε:

$< structure\_variable > . < member\_name >$

Για παράδειγμα για να ορίσουμε το σημείο (5.5, 6.0) γράφουμε:

$p1.x = 5.5;$
$p1.y = 6.0;$

Αν οι μεταβλητές $p1$ και $p2$ που αναπαριστούν διαφορετικά σημεία είναι επιθυμητό κάποια στιγμή να αναπαριστούν το ίδιο σημείο τότε γράφουμε:

$p1 = p2;$

όπου οι τιμές των μελών της $p2$ καταχωρούνται στα μέλη της $p1$.

Μέχρι τώρα είδαμε μέλη που ήταν απλού τύπου. Τα μέλη μιας δομής μπορεί να είναι πιο περίπλοκου τύπου, όπως μπορεί να είναι ένας πίνακας ή ακόμα και δομή. Για παράδειγμα, η ημερομηνία μπορεί να αναπαρασταθεί από τρεις ακεραίους -έναν για την ημέρα, έναν για τον μήνα και έναν για τον χρόνο- και δύο αλφαριθμητικά ένα για το όνομα της ημέρας και ένα για το όνομα του μήνα. Ανακεφαλαιώνοντας, γράφουμε:

```
struct date {
        int day, month, year;
        char d_name[10], m_name[10];
};
```

και στη συνέχεια χρησιμοποιώντας το παραπάνω πρότυπο ορίζουμε τρεις μεταβλητές για το χθες, σήμερα και αύριο ως εξής:

```
struct date yesterday, today, tomorrow;
```

Η παραπάνω δήλωση είναι ισοδύναμη με την:

```
struct{
        int day, month, year;
        char d_name[10], m_name[10];
}yesterday, today, tomorrow;
```

από την οποία απουσιάζει το *tag* και οι μεταβλητές ακολουθούν την δήλωση της δομής. Η απουσία του *tag* έχει ως συνέπεια να μην ορίζεται κάποιο πρότυπο για τη δομή με αποτέλεσμα αν χρειαστεί σε κάποιο σημείο του προγράμματος να δηλώσουμε και άλλες μεταβλητές θα πρέπει να επαναλάβουμε την πλήρη δήλωση των μελών της δομής.

Ένα άλλο παράδειγμα χρήσης της δομής είναι αυτό της αναπαράστασης των μιγαδικών αριθμών όπου για την αναπαράσταση τους χρειάζονται δύο μέλη ένα για το πραγματικό και ένα για το φανταστικό μέρος. Η δημιουργία του προτύπου για τους μιγαδικους γίνεται με την παρακάτω δήλωση.

```
struct complex {
        float re, im;
};
```

Στη συνέχεια για να δηλώσουμε μιγαδικές μεταβλητές χρησιμοποιούμε μόνο τη λέξη κλειδί *struct* και το *tag complex* όπως φαίνεται παρακάτω:

struct complex x, y;

Μια καλή προγραμματιστική πρακτική είναι να αντικαταστήσουμε το *tag* μιας δομής με όνομα τύπου με τη βοήθεια της λέξης κλειδί *typedef*. Έτσι για παράδειγμα για την περίπτωση των μιγαδικών αριθμών δηλώνουμε:

```
typedef struct complex {
        float re, im;
} complex;
```

και έτσι προκύπτει ένας νέος τύπος ο *complex*. Τώρα για να δηλώσουμε μιγαδικές μεταβλητές απλά γράφουμε:

complex x, y;

Στην περίπτωση αυτή το *tag* είναι περιττό οπότε μπορούμε πλέον να γράψουμε:

```
typedef struct{
        float re, im;
} complex;
complex x, y;
```

Με ποιό τρόπο μπορούμε να αρχικοποιήσουμε όμως μια μεταβλητή τύπου δομής? Η τιμή την οποία θέλουμε να αναθέσουμε σε μια μεταβλητή τύπου δομής περιλαμβάνει τις τιμές των επιμέρους μελών κλεισμένες εντός αγκίστρων. Για παράδειγμα, έστω ότι θέλουμε να ορίσουμε ένα αρχικό σημείο με συντεταγμένες $(5.0, 6.0)$ τότε γράφουμε:

struct point a = {5.0, 6.0};

ενώ για να δηλώσουμε μια αρχική ημερομηνία γράφουμε:

struct date yesterday = {6, 5, 2008, “Monday”, “May”};

Στο εδάφιο 5.2 είδαμε τον ορισμό πινάκων στοιχείων απλού τύπου όπως *int*, *float*, *double* και *char*. Τα στοιχεία ενός πίνακα μπορεί να είναι και δομές. Μια καμπύλη N σημείων για παράδειγμα μπορεί να αναπαρασταθεί από έναν πίνακα μεγέθους N όπου τα στοιχεία του είναι τύπου *struct point*, δηλαδή:

```
struct point curve[N];
```

όπου το μέγεθος N μπορεί να καθορισθεί είτε ορίζοντας μια σταθερά N στην αρχή του προγράμματος μας είτε δίνοντας την τιμή ο προγραμματιστής κατά την είσοδο δεδομένων. Για την αρχικοποίηση ενός πίνακα που τα στοιχεία του είναι δομές ακολουθούνται οι ίδιοι κανόνες που αναφέρθηκαν παραπάνω. Ένα παράδειγμα αρχικοποίησης ενός τέτοιου πίνακα είναι αυτό του πίνακα διαστάσεων 2×2 με στοιχεία μιγαδικούς:

```
complex x[2][2]={
                {{3.0, 4.0}, {-1.0, -4.0}},
                {{2.0, 5.0}, {-5.0, -4.0}}
                };
```

όπου για κάθε γραμμή του παραπάνω πίνακα έχουμε εντός άγκιστρων τιμές για τα στοιχεία κάθε στήλης.

6.1. Οι δομές ως τυπική παράμετρος συνάρτησης

Μια συνάρτηση μπορεί να έχει ως τυπική παράμετρο μια δομή. Κατά την κλήση μιας συνάρτησης που έχει ως τυπική παράμετρο μια δομή, η καλούσα συνάρτηση αντιγράφει τα μέλη της δομής όρισμα στα μέλη της τυπικής παραμέτρου της συνάρτησης πράγμα που σημαίνει ότι η καλούμενη συνάρτηση δεν μπορεί να τροποποιήσει τις τιμές των μελών της δομής της καλούσας συνάρτησης. Μπορεί όμως να τροποποιήσει το αντίγραφο της και στη συνέχεια μέσω της εντολής *return* να επιστρέψει το τροποποιημένο αντίγραφο στην καλούσα συνάρτηση όπως φαίνεται και στον παρακάτω κώδικα.

```
point e;
e = updatePoint(e);
point updatePoint(point d) {
        εντολές τροποποίησης τυπικής παραμέτρου
        return d;
}
```

Η συνάρτηση *updatePoint* καλείται με το όρισμα *e* που είναι δομή το οποίο αντιγράφεται στην τυπική παράμετρο *d*. Στη συνέχεια τροποποιείται η τυπική παράμετρος και με την εντολή *return* επιστρέφεται στην καλούσα συνάρτηση. Η δομή που επιστρέφεται -η οποία είναι η τροποποιημένη τυπική παράμετρος - καταχωρείται στη δομή *e* όπως φαίνεται από την εντολή ανάθεσης με την οποία καλείται η συνάρτηση *updatePoint*.

```
 0:  #include <stdio.h>
 1:
 2:  typedef struct {
 3:    float x;
 4:    float y;
 5:  } point;
 6:
 7:  point AddPoint(point a, point b)
 8:  {
 9:    a.x = a.x + b.x;
10:    a.y = a.y + b.y;
11:
12:    return a;
13:  }
14:
15:  main(void)
16:  {
17:    point p1, p2, target;
18:
19:    p1.x = 3.0;
20:    p2.x = 4.0;
21:    p1.y = p2.y = 5.2;
22:    target = AddPoint(p1, p2);
23:    printf("The new coordinates are:(%f, %f)\n", target.x, target.y);
24:  }
```

Σχήμα 6.1: Πρόγραμμα πρόσθεσης δύο σημείων επιπέδου.

Στο Σχήμα 6.1 δίνεται ένα παράδειγμα που εφαρμόζει το παραπάνω μοντέλο. Στις γραμμές 2 – 5 ορίζεται ο τύπος *point*. Στην κύρια συνάρτηση ορίζονται δύο σημεία $p1$ και $p2$ με συντεταγμένες $(3.0, 5.2)$ και $(4.0, 5.2)$ αντίστοιχα. Στη συνέχεια καλείται η συνάρτηση *AddPoint* με τυπικές παραμέτρους δύο δομές τύπου *point* η οποία όταν εκτελείται προσθέτει τις συντεταγμένες των δύο σημείων καταχωρόντας το αποτέλεσμα στην πρώτη τυπική παράμετρο *a* και επιστρέφει στην κύρια συνάρτηση την τροποποιημένη τυπική παράμετρο. Η επιστρεφείσα δομή αποθηκεύεται στη μεταβλητή *target* τύπου *point* τα μέλη της οποίας (συντεταγμένες του νέου σημείου που προέκυψε από την άθροιση) τυπώνονται με την εκτέλεση της συνάρτησης *printf* στη γραμμή 25.

6.2. Δείκτες σε Δομές

Όπως αναφέραμε στο προηγούμενο εδάφιο, όταν καλείται μια συνάρτηση με τυπική παράμετρο δομή, η καλούσα συνάρτηση αντιγράφει όλα τα μέλη του ορίσματος της στα μέλη της τυπικής παραμέτρου της συνάρτησης για τα οποία έχει κατανεμηθεί χώρος μνήμης από το πλαίσιο στοίβας της συνάρτησης. Μια δομή μπορεί να έχει ως μέλη σύνθετους και μεγάλους τύπους όπως ένας πίνακας ή μια δομή όπως φαίνεται στο παρακάτω παράδειγμα:

```
typedef struct{
            char last_name[20];
            char first_name[20], grade;
            int student_id;
} student;
```

όπου η δομή *student* που μοντελοποιεί την καρτέλα φοιτητή αποτελείται από δύο πίνακες των 20 στοιχείων για το όνομα και το επώνυμο του φοιτητή, έναν χαρακτήρα για τον βαθμό του και ένα ακέραιο για τον κωδικό αναγνώρισης φοιτητή. Αυτό σημαίνει ότι θα αντιγραφούν 41 χαρακτήρες και ένας ακέραιος όταν κληθεί μια συνάρτηση με τυπική

παράμετρο τύπου *student*. Γίνεται φανερό ότι το πέρασμα μιας δομής ως παραμέτρου μιας συνάρτησης καθίσταται χρονοβόρα διαδικασία ειδικά όταν έχουμε πολλαπλές κλήσεις τέτοιων συναρτήσεων. Όπως και με τους άλλους τύπους δεδομένων στη *C*, έτσι και με τις δομές που είναι ένας σύνθετος τύπος μπορούμε να ορίσουμε δείκτες προς δομές. Έτσι για το παραπάνω παράδειγμα γράφουμε:

```
student *st;
```

όπου η μεταβλητή *st* είναι δείκτης σε δομή *student*.

Οι παραπάνω δηλώσεις μπορούν να γραφούν και με την ακόλουθη μορφή:

```
typedef struct{
            char last_name[20];
            char first_name[20], grade;
            int student_id;
} *student;
student st;
```

όπου πλέον ο τύπος *student* είναι τύπος δείκτη σε δομή με τα πεδία που περιγράφονται παραπάνω και όχι απλά δομή. Το αποτέλεσμα βέβαια είναι το ίδιο αλλά η διαφορά είναι ότι στη δεύτερη περίπτωση δεν έχουμε τη δυνατότητα να ορίσουμε μεταβλητές τύπου δομής *student*, μπορούμε να ορίσουμε μόνο δείκτες.

Η παραπάνω δήλωση σηματοδοτεί για τον μεταγλωττιστή ότι πρέπει να δεσμεύσει χώρο μνήμης (4 bytes) μόνο για τον δείκτη όχι όμως και για τα μέλη της δομής. Έτσι για την μεταβλητή *st* που είναι δείκτης στη δομή πρέπει να δεσμευθεί χώρος για την δομή με την χρήση της *malloc* (δυναμική κατανομή μνήμης):

```
st = (student) malloc(sizeof(*st));
if(st == NULL) {
  printf("there is no available memory for pointer st");
  exit(1);
}
```

όπου ο τελεστής *sizeof*(∗*st*) επιστρέφει τον αριθμό των *byte* που απαιτούνται για όλα τα μέλη της δομής καθώς η μεταβλητή *st* είναι δείκτης στη δομή και η δομή είναι προσβάσιμη μέσω του τελεστή *. Ο δείκτης γενικού τύπου που επιστρέφεται από την *malloc* μετατρέπεται σε τύπου *student*. Στη συνέχεια γίνεται έλεγχος αν ο δείκτης που επιστρέφηκε είναι *NULL* και σε περίπτωση που ο έλεγχος είναι αληθής τυπώνεται το παραπάνω μήνυμα λάθους και τερματίζει η εκτέλεση του προγράμματος.

Όμως με ποιό τρόπο μπορούμε να επεξεργαστούμε τα μέλη μιας δομής μέσω ενός δείκτη σε αυτή? Αν γράψουμε

```
*st.grade;
```

τότε θα λάβουμε μήνυμα λάθους από τον μεταγλωτιστή καθώς αυτός το αντιμετωπίζει σύμφωνα με τις προτεραιότητες των τελεστών ως εξής:

```
*(st.grade);
```

που είναι λάθος αφού η μεταβλητή είναι δείκτης σε δομή και όχι δομή. Η πρόσβαση στο μέλος *grade* είναι δυνατή μέσω της παρακάτω έκφρασης:

```
(*st).grade;
```

Τα πράγματα όμως είναι πιο εύκολα διότι η *C* διαθέτει τον τελεστή → ο οποίος είναι τρίτος τη τάξη σε προτεραιότητα έναντι όλων των τελεστών μετά τον τελεστή ( ) και τον τελεστή [ ] η χρήση του οποίου είναι ισοδύναμη με την παραπάνω έκφραση, δηλαδή:

$st \rightarrow grade \equiv (*st).grade$

6.3. Αφηρημένοι Τύποι Δεδομένων

Στην Επιστήμη των Υπολογιστών, ο όρος **Αφηρημένος Τύπος Δεδομένων** χρησιμοποιείται για την περιγραφή μιας δομής δεδομένων μαζί με τις λειτουργίες της χωρίς να δοθούν

λεπτομέρειες σχετικά με την υλοποίηση τους. Έστω παράδειγμα ότι θέλουμε να ορίσουμε πράξεις μεταξύ ακεραίων με περισσότερα από 64 *bit* που ένας υπολογιστής μπορεί να προσφέρει. Τότε ο νέος τύπος ακεραίων μαζί με την αριθμητική του είναι ένας αφηρημένος τύπος δεδομένων.

Το εργαλείο της δομής που η *C* παρέχει στον προγραμματιστή δίνει τη δυνατότητα για την υλοποίηση των δικών του αφηρημένων τύπων δεδομένων. Στο εδάφιο αυτό θα παρουσιάσουμε υλοποιήσεις με δομές δύο ευρέως διαδεδομένων δομών δεδομένων: της στοίβας και της γραμμικής διασυνδεδεμένης λίστας.

6.3.1. Στοίβα

Τη δομή αυτή μπορούμε να τη φανταστούμε ως μια στοίβα από δεδομένα από την οποία μπορούμε να τραβήξουμε πάντα το δεδομένο που βρίσκεται στην κορυφή της στοίβας δηλαδή αυτό που προστέθηκε τελευταίο στη στοίβα. Για να μπορέσουμε να πάρουμε το κατώτερο δεδομένο της στοίβας -το οποίο εισήλθε στη στοίβα πρώτο- πρέπει πρώτα να εξέλθουν αυτά που βρίσκονται από πάνω του. Ο τρόπος λειτουργίας της στοίβας την εντάσσει στην κατηγορία των δομών δεδομένων τύπου (**LIFO (Last In First Out)**). Βασικές λειτουργίες της στοίβας είναι αυτή της εξαγωγής του δεδομένου που βρίσκεται στην κορυφή *pop* και αυτή της εισαγωγής δεδομένου στην κορυφή της στοίβας *push*.

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2:
 3: #define MAXVAL 100
 4: #define EMPTY 0
 5: #define FULL MAXVAL
 6:
 7: typedef struct{
 8:  int sp;
 9:  double val[MAXVAL];
10: } stack;
11:
12: typedef enum {FALSE, TRUE} bool;
13:
14: void push(double f, stack *st)
15: {
16:  if (st -> sp < MAXVAL)st->val[st->sp++] = f;
17:  else printf("error: stack full, can't push %g\n", f);
18: }
19:
20: double pop(stack *st)
21: {
22:  if (st->sp > 0) return st->val[--st->sp];
23:  else {
24:        printf("error: stack empty\n");
25:        return 0.0;
26:   }
27: }
28:
29: void reset(stack *st)
30: {
31:  st->sp = EMPTY;
32: }
33:
34: bool isEmpty(stack *st)
35: {
36:  if (st->sp == EMPTY) return TRUE;
37:  return FALSE;
38: }
39:
40: bool isFULL(stack *st)
41: {
42:  if (st->sp == FULL) return TRUE;
43:  return FALSE;
44: }
```

Σχήμα 6.2: Module που υλοποιεί τη στοίβα με τη χρήση δομής

Με τη χρήση του μηχανισμού των δομών μπορούμε να μοντελοποιήσουμε τη στοίβα με την ακόλουθη δήλωση του τύπου *stack*:

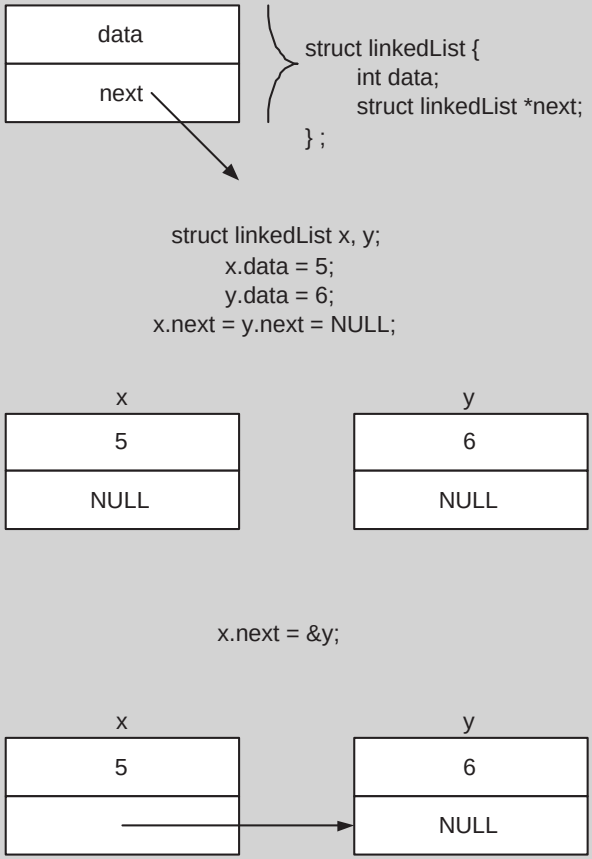
```
typedef struct{
                double val[MAXVAL];
                int sp;
} stack;
```

Η δομή αυτή περιλαμβάνει ένα μέλος που είναι ο πίνακας *val* με πεπερασμένο αριθμό θέσεων, και ένα δεύτερο μέλος που είναι η μεταβλητή *sp* που δείχνει στην πρώτη κενή θέση του πίνακα που θεωρείται η κορυφή της στοίβας. Βασικές λειτουργίες της *stack* είναι η *push* που εισάγει δεδομένα στη κορυφή της στοίβας και η *pop* που εξάγει δεδομένα από την κορυφή της στοίβας. Θα εμπλουτίσουμε τις λειτουργίες της στοίβας με μια συνάρτηση αρχικοποίησης της στοίβας *reset* και με δύο κατηγορηματικές συναρτήσεις την *isEmpty* η οποία επιστρέφει *TRUE* όταν η στοίβα δεν έχει κανένα δεδομένο και *isFull* η οποία επιστρέφει *TRUE* όταν η στοίβα είναι πλήρης.

Στο Σχήμα 6.2 δίνεται το module που υλοποιεί την στοίβα. Βασική τυπική παράμετρος όλων των συναρτήσεων του module είναι η *st* που είναι δείκτης στη δομή *stack*. Η πρόσβαση στα μέλη της δομής μέσω του δείκτη *st* γίνεται με τη βοήθεια του τελεστή →. Επίσης, στη γραμμή 12 ορίζεται ο τύπος *bool* ο οποίος είναι ένας τύπος απαρίθμησης.

6.3.2. Γραμμική διασυνδεμένη λίστα

Η γραμμική διασυνδεδεμένη λίστα είναι μια θεμελιώδης δομή δεδομένων καθώς με αυτή μπορούν να υλοποιηθούν πολλές άλλες δομές δεδομένων όπως είναι η **στοίβα (stack)**, η **ουρά (FIFO, First In First Out)**, οι **γράφοι** και τα **δένδρα**. Είναι μια δυναμική δομή δεδομένων καθώς αποτελείται απο μια συλλογή στοιχείων ακολουθιακά διασυνδεδεμένων μεταξύ τους η οποία μεταβάλλεται σε μέγεθος δυναμικά κατά την εκτέλεση του προγράμματος δεσμεύοντας μνήμη ανάλογα με τις απαιτήσεις της εφαρμογής καθώς και την διαθεσιμότητα των πόρων του συστήματος.



Σχήμα 6.3: Στοιχείο Γραμμικής Διασυνδεδεμένης λίστας

Σε μια διασυνδεδεμένη λίστα ισχύουν γενικά τα εξής:

1. Ένας δείκτης επικεφαλίδα (*head*) δείχνει στο πρώτο στοιχείο της λίστας,
2. κάθε στοιχείο της λίστας υλοποιείται με μια δομή η οποία περιέχει ένα μέλος που είναι δείκτης σε ένα άλλο στοιχείο της λίστας,
3. προαιρετικά ένας δείκτης ουρά (*tail*) δείχνει στο τελευταίο στοιχείο της λίστας.

Στο Σχήμα 6.3 ορίζεται ένα πρότυπο δομής με *tag linkedList* και δύο πεδία: έναν ακέραιο και ένα δείκτη σε μια δομή τύπου *linkedList*. Στη συνέχεια ορίζονται δύο μεταβλητές *x* και *y* τύπου *linkedList* τα πεδία *data* των οποίων δέχονται τις τιμές 5 και 6 αντίστοιχα ενώ τα πεδία *next* τίθενται *NULL* δηλαδή αρχικά δεν συνδέονται μεταξύ τους. Η διασύνδεση του στοιχείου *x* με το στοιχείο *y* γίνεται με την παρακάτω ανάθεση:

x.next = &*y*;

όπου ο δείκτης *next* του στοιχείου *x* δέχεται ως τιμή την διεύθυνση του στοιχείου *y*.

Οι βασικές λειτουργίες σε μια διασυνδεμένη λίστα είναι:

1. Δημιουργία της λίστας.
2. Μέτρηση των στοιχείων της λίστας.
3. Εκτύπωση των στοιχείων της λίστας.
4. Αναζήτηση ενός στοιχείου στη λίστα.
5. Συγχώνευση δύο λιστών.
6. Εισαγωγή στοιχείου σε λίστα.
7. Διαγραφή στοιχείου από τη λίστα.

Η υλοποίηση των παραπάνω λειτουργιών μπορεί να γίνει είτε με αναδρομικές συναρτήσεις είτε με συναρτήσεις που υλοποιούν την λειτουργία κατά τρόπο επαναληπτικό όπως αναφέραμε και στο Κεφάλαιο 4.

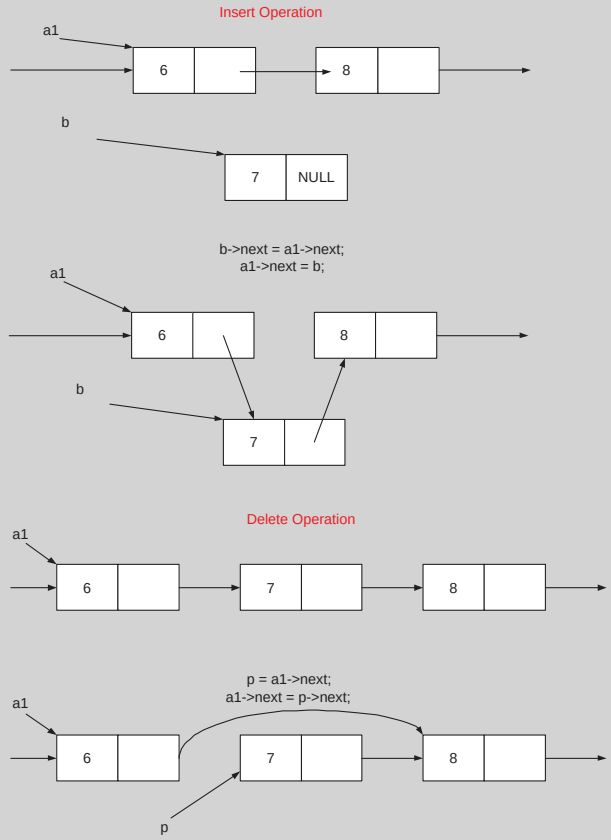
```
0:  #include <stdio.h>
1:  #include <stdlib.h>
2:
3:  typedef struct linkedList {
4:    int data;
5:    struct linkedList *next;
6:  } element;
7:
8:  typedef element *link;
9:
10: int countList(link head)
11: {
12:    int count=0;
13:
14:    for(; head!=NULL; head=head->next) ++count;
15:
16:    return count;
17: }
18:
19: void printList(link head)
20: {
21:    if (head == NULL) printf("NULL\n");
22:    else{
23:       printf("%d ->", head-> data);
24:       printList(head->next);
25:    }
26: }
27:
28: void insert(link a1, link b)
29: {
30:   b ->next = a1->next;
31:   a1->next = b;
32: }
33:
34: void delete(link a1)
35: {
36:    link p;
37:
38:    p = a1->next;
39:    a1->next = p->next;
40:
41:    free(p);
42: }
```

Σχήμα 6.4: Module που υλοποιεί μια διασυνδεμένη λίστα με τη χρήση δομής

Στο Σχήμα 6.4 παρουσιάζεται το *module* που υλοποιεί τέσσερεις από τις παραπάνω βασικές λειτουργίες. Αρχικά ορίζονται δύο τύποι: ο τύπος *element* που είναι μια δομή με δύο πεδία είναι το πρότυπο των στοιχείων της λίστας και ο τύπος *link* που είναι ένας δείκτης σε δομή τύπου *element*. Βέβαια ανάλογα με την εφαρμογή μπορεί να υπάρχουν περισσότερα από ένα πεδία που θα αφορούν δεδομένα.

Η συνάρτηση *countList* δέχεται ως τυπική παράμετρο την κεφαλή της λίστας που είναι ένας δείκτης σε *element* και απαριθμεί τα στοιχεία της λίστας αξιοποιώντας τον δείκτη παράμετρο. Στην συνάρτηση αυτή η τυπική παράμετρος *head* υφίσταται αλλοίωση, τι συμβαίνει όμως με την κεφαλή της λίστας, υφίσταται αλλοίωση (αφήνεται στον αναγνώστη η απάντηση)?

Η δεύτερη συνάρτηση η printList δέχεται ως τυπική παράμετρο την κεφαλή της λίστας. Στο σώμα της όμως έχουμε μια κλασική αναδρομική υλοποίηση της λειτουργίας εκτύπωσης των στοιχείων της λίστας.



Σχήμα 6.5: Διαδικασία εισαγωγής στοιχείου σε Γραμμική Διασυνδεδεμένη λίστα

Η εισαγωγή στοιχείου στη λίστα υλοποιείται με τη συνάρτηση *insert* η οποία δέχεται ως τυπικές παραμέτρους τον δείκτη $a1$ που δείχνει στο στοιχείο μετά από το οποίο θα γίνει η προσθήκη και τον δείκτη b που δείχνει στο στοιχείο το οποίο θα προστεθεί στη λίστα. Στο πάνω μέρος του Σχήματος 6.5 φαίνονται οι ενέργειες που λαμβάνουν χώρα κατά την εκτέλεση των εντολών ανάθεσης του σώματος της συνάρτησης.

Τέλος, η διαγραφή στοιχείου από τη λίστα υλοποιείται με την συνάρτηση *delete* η οποία δέχεται ως τυπική παράμετρο τον δείκτη $a1$ που δείχνει στο στοιχείο που προηγείται αυτού που θέλουμε να διαγράψουμε. Στο κάτω μέρος του Σχήματος 6.5 φαίνονται οι ενέργειες που λαμβάνουν χώρα κατά την εκτέλεση των εντολών ανάθεσης του σώματος της συνάρτησης. Οι πόροι που καταναλώνονται από το στοιχείο που διαγράφηκε επιστρέφονται πίσω στο σύστημα με την *free*.

6.4. Ασκήσεις

Άσκηση 6.4.1 *Δίνονται οι παρακάτω δηλώσεις και αναθέσεις τιμών. Να συμπληρωθεί ο πίνακας.*

```
struct student{
        int id;
        char firstName[20];
        char familyName[20];
        char grade;
} tmp, *w = &tmp;
tmp.grade = 'C';
tmp.id = 1040
tmp.familyName = "Petrou";
```

Έκφραση	Τιμή
$tmp.grade$	
$tmp.familyName$	
$(*w).id$	
$*w \rightarrow familyName + 4$	
$*(w \rightarrow familyName + 3)$	

Άσκηση 6.4.2 *Να γραφεί πρόγραμμα που να υπολογίζει το μήκος της ευθείας που διέρχεται από δύο σημεία P και Q με συνετεταγμένες (x_1, y_1) και (x_2, y_2) αντίστοιχα.*

Άσκηση 6.4.3 *Χρησιμοποιώντας δομές για την αναπαράσταση μιγαδικών αριθμών να γραφούν οι εξής συναρτήσεις:*

1. *Πρόσθεσης δύο μιγαδικών αριθμών.*
2. *Άθροισης δύο διανυσμάτων μιγαδικών αριθμών: η συνάρτηση θα έχει ως τυπικές παραμέτρους το διάνυσμα που θα αποθηκευθεί το άθροισμα των δύο διανυσμάτων, τα δύο διανύσματα που προστίθενται και το μέγεθος των διανυσμάτων.*

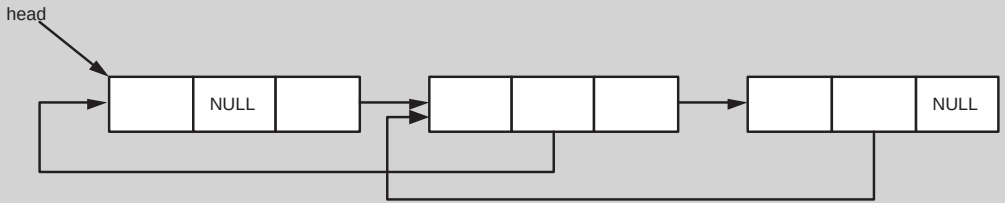
Στη συνέχεια να γραφεί πρόγραμμα που:

- *θα διαβάζει το μέγεθος N δύο διανυσμάτων μιγαδικών αριθμών και θα δεσμεύει μνήμη και για τα δύο διανύσματα,*
- *θα διαβάζει N μιγαδικούς αριθμούς για το ένα διάνυσμα και N για το δεύτερο διάνυσμα,*
- *θα δεσμεύει μνήμη για το διάνυσμα όπου θα αποθηκευθεί το αποτέλεσμα της άθροισης των δύο διανυσμάτων και θα καλεί την συνάρτηση άθροισης δύο διανυσμάτων μιγαδικών αριθμών,*

- *θα τυπώνει το αποτέλεσμα της άθροισης των διανυσμάτων.*

Άσκηση 6.4.4 *Το στοιχείο μιας διπλής διασυνδεδεμένης λίστας μοντελοποιείται με την παρακάτω δομή:*

```
typedef struct doubleLList{
        int data;
        struct doubleLList *prev;
        struct doubleLList *next;
        } DLLElement;
```

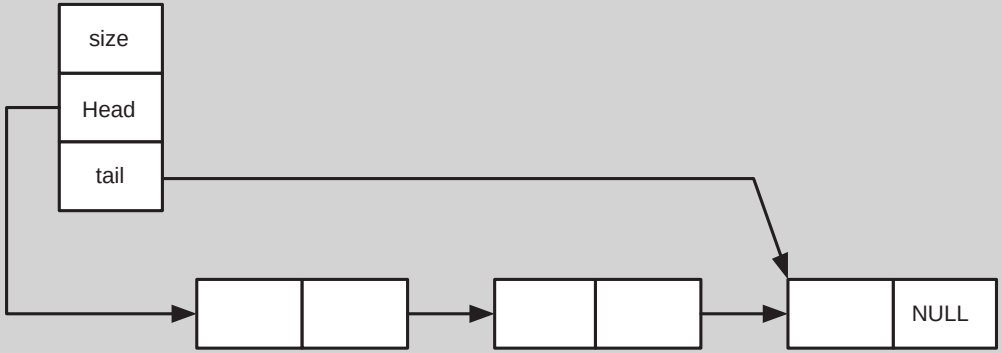


Σχήμα 6.6: Διπλή διασυνδεδεμένη λίστα

Στο Σχήμα 6.6 δίνεται ένα διάγραμμα που περιγράφει την δομή μιας διπλής διασυνδεδεμένης λίστας. Να γραφούν οι συναρτήσεις insert και delete που εισάγουν και διαγράφουν ένα στοιχείο από τη λίστα αντίστοιχα.

Άσκηση 6.4.5 *Η ουρά (queue) είναι μια FIFO (First In First Out) δομή δεδομένων της οποίας η δομή δίνεται στο διάγραμμα του Σχήματος 6.7 όπου size είναι το πλήθος των*

στοιχείων της ουράς, front είναι δείκτης στο πρώτο στοιχείο της ουράς και tail είναι δείκτης στο τελευταίο στοιχείο της ουράς.



Σχήμα 6.7: Δομή ουράς

Να μοντελοποιηθεί η δομή δεδομένων της ουράς με τη χρήση δομής και να γραφούν οι συναρτήσεις:

1. *εισαγωγής στοιχείου στην ουρά,*
2. *διαγραφής στοιχείου από την ουρά,*
3. *που επιστρέφει το πρώτο στοιχείο της ουράς*
4. *που επιστρέφει TRUE αν η ουρά είναι κενή διαφορετικά FALSE.*
5. *που επιστρέφει TRUE αν η ουρά είναι πλήρης διαφορετικά FALSE.*

Κεφάλαιο 7

Αρχεία Δεδομένων

Όπως είδαμε στα προηγούμενα κεφάλαια σε κάθε πρόγραμμα εισρέουν δεδομένα και εξάγονται τα αποτελέσματα των υπολογισμών που έχουν εκτελεσθεί κατά την διάρκεια εκτέλεσης του. Οι πηγές και οι προορισμοί δεδομένων συχνά αναφέρονται ως **συσκευές (devices)**. Αυτές μπορεί να είναι μόνο εισόδου, μόνο εξόδου ή και τα δύο. Ενδεικτικά αναφέρονται οι εξής:

- Πληκτρολόγιο: Συσκευή εισόδου δεδομένων.
- Οθόνη: Συσκευή εξόδου δεδομένων.
- Αρχεία Δίσκων: Συσκευές εισόδου και εξόδου.

Στη *C* οι λειτουργίες εισόδου και εξόδου δεδομένων από και προς συσκευές γίνεται μέσω των **ροών δεδομένων (streams)**.

Μια ροή δεν είναι τίποτα άλλο παρά μια ακολουθία χαρακτήρων (ή εναλλακτικά μια ακολουθία από byte δεδομένων). Μια ακολουθία χαρακτήρων η οποία εισέρχεται σε ένα

πρόγραμμα ονομάζεται **ροή εισόδου (input stream)**. Μια ακολουθία χαρακτήρων η οποία εξέρχεται από ένα πρόγραμμα ονομάζεται **ροή εξόδου (output stream)**. Μια ροή δεδομένων συσχετίζεται με μια συσκευή. Μέσω των ροών, οι λειτουργίες εισόδου και εξόδου δεδομένων σε ένα *C* πρόγραμμα καθίστανται ανεξάρτητες από το είδος των συσκευών από τις οποίες προέρχονται ή στις οποίες πηγαίνουν τα δεδομένα και αυτό γιατί ο προγραμματιστής μέσω των συναρτήσεων που θα μας απασχολήσουν στη συνέχεια χειρίζεται μόνο ακολουθίες από *byte* αγνοώντας τυχόν λεπτομέρειες που αφορούν τον τρόπο αποθήκευσης τους στη συσκευή. Θα λέγαμε ότι μια ροή δεδομένων παρεμβάλεται μεταξύ προγραμματιστή και συσκευής κρύβοντας λεπτομέρειες που χαρακτηρίζουν τη συσκευή.

Στη *C* μια ροή μπορεί να είναι:

1. **κειμένου (text)**, δηλαδή αποτελείται μόνο από χαρακτήρες. Αυτού του τύπου οι ροές είναι οργανωμένες σε γραμμές των 255 χαρακτήρων.

2. **δυαδική (binary)**, δηλαδή πραγματεύεται οποιασδήποτε μορφής δεδομένα (συμπεριλαμβανομένου και αυτής του κειμένου).

Είναι πολύ σημαντικό να γίνει κατανοητή η διαφορά μεταξύ των δύο παραπάνω αρχείων. Σε μια ροή κειμένου, η τιμή κάθε *byte* αντιμετωπίζεται ως ASCII κωδικός ενώ στη περίπτωση μια δυαδικής ροής αντιμετωπίζεται ως δυαδικός αριθμός. Έτσι για παράδειγμα, η τιμή 65 στη μια περίπτωση εκλαμβάνεται ως χαρακτήρας A ενώ στη δεύτερη είναι ο αριθμός 65. Στο παράδειγμα αυτό έτυχε η επιλογή να είναι εκτυπώσιμος χαρακτήρας. Υπάρχουν όμως και περιπτώσεις όπου δεν είναι εκτυπώσιμοι με αποτέλεσμα όταν ένα δυαδικό αρχείο επιχειρηθεί να τυπωθεί εμφανίζονται ακατανόητα σχήματα και χαρακτήρες.

Η *ANSI C* διαθέτει τρεις προκαθορισμένες ροές τύπου κειμένου που ονομάζονται **standard input / output files**:

- **stdout**: προκαθορισμένη (**default**) ροή εξόδου του προγράμματος, η οποία συσχετίζεται με τη συσκευή της οθόνης,

- **stdin**: προκαθορισμένη (**default**) ροή εισόδου του προγράμματος, η οποία συσχετίζεται με τη συσκευή του πληκτρολογίου,
- **stderr**: προκαθορισμένη (**default**) ροή εξόδου σφαλμάτων του προγράμματος, η οποία συσχετίζεται επίσης με τη συσκευή της οθόνης.

Με την έναρξη του προγράμματος, οι παραπάνω ροές αρχικοποιούνται και είναι διαθέσιμες στο πρόγραμμα μέσω των συναρτήσεων των οποίων οι δηλώσεις των προτύπων βρίσκονται στο αρχείο επικεφαλίδα $< stdlib.h >$:

- Η συνάρτηση *scanf*() διαβάζει δεδομένα από το πληκτρολόγιο μέσω της προκαθορισμένης ροής stdin.
- Η συνάρτηση *printf*() απεικονίζει δεδομένα στην οθόνη μέσω της προκαθορισμένης ροής stdout.

Στις δηλώσεις των προτύπων των παραπάνω συναρτήσεων δεν εμφανίζονται οι παραπάνω ροές γιατί είναι προκαθορισμένες.

7.1. Είσοδος/Έξοδος από/σε αρχεία

Όπως προαναφέραμε, σε ένα πρόγραμμα η εισαγωγή δεδομένων από το πληκτρολόγιο γίνεται μέσω της προκαθορισμένης ροής stdin η οποία συσχετίζεται με το πληκτρολόγιο με την έναρξη εκτέλεσης τους προγράμματος, ενώ η έξοδος δεδομένων στην οθόνη γίνεται μέσω της προκαθορισμένης ροής stdout. Τι γίνεται όμως όταν το πρόγραμμα πρέπει να διαβάσει ή αποθηκεύσει δεδομένα από ή σε αρχεία του δίσκου?

Το πρόγραμμα πρέπει καταρχήν να δημιουργήσει μια ροή η οποία θα συσχετισθεί με το αρχείο του δίσκου. Η ροή που θα δημιουργηθεί πρέπει να είναι είτε ροή κειμένου (text stream) είτε δυαδική ροή (binary stream): ροή κειμένου χρησιμοποιείται με αρχεία κειμένου τα οποία είναι οργανωμένα σε γραμμές από 0 έως 255 χαρακτήρες το πολύ και

χωρισμένες μεταξύ τους με χαρακτήρες τέλους γραμμής ενώ δυαδική ροή χρησιμοποιούμε με δυαδικά αρχεία όπου τα δεδομένα που διαβάζονται και γράφονται δεν υφίστανται κανενός είδους μορφοποίηση.

r	Ανοίγει το αρχείο για ανάγνωση. Αν το αρχείο δεν υπάρχει επιστρέφει NULL.
w	Ανοίγει το αρχείο για εγγραφή. Αν το αρχείο δεν υπάρχει το δημιουργεί. Αν το αρχείο υπάρχει, τα δεδομένα του διαγράφονται και ένα νέο αρχείο δημιουργείται.
a	Ανοίγει το αρχείο για ενημέρωση. Αν το αρχείο δεν υπάρχει το δημιουργεί. Αν το αρχείο υπάρχει, τα νέα δεδομένα προστίθενται στο τέλος του αρχείου.
r+	Ανοίγει το αρχείο για ανάγνωση και εγγραφή. Αν το αρχείο δεν υπάρχει το δημιουργεί. Αν το αρχείο υπάρχει, τα νέα δεδομένα του προστίθενται στην αρχή του αρχείου επανεγγράφοντας τα υπάρχοντα δεδομένα.
w+	Ανοίγει το αρχείο για ανάγνωση και εγγραφή. Αν το αρχείο δεν υπάρχει το δημιουργεί. Αν το αρχείο υπάρχει, το αρχείο επανεγγράφεται.
a+	Ανοίγει το αρχείο για ανάγνωση και ενημέρωση. Αν το αρχείο δεν υπάρχει το δημιουργεί. Αν το αρχείο υπάρχει, τα νέα δεδομένα προστίθενται στο τέλος του αρχείου.

Πίνακας 7.1: Τρόποι ανοίγματος αρχείου

Κάθε αρχείο δίσκου έχει ένα όνομα το οποίο είναι απαραίτητο για τη δημιουργία της ροής. Το όνομα ενός αρχείου στη *C* αντιμετωπίζεται ως αλφαριθμητικό (string). Για παράδειγμα σε command line στα Windows XP το όνομα του αρχείου καθορίζεται με τον

ακόλουθο τρόπο C:\ tmp\ myfile.txt. Στη *C* χρησιμοποιούμε μια αλφαριθμητική μεταβλητή για το όνομα του αρχείου char *filename="C:\\ tmp\\ myfile.txt". Ο λόγος που βάζουμε διπλό \ είναι διότι όπως αναφέραμε και στο Κεφάλαιο 2 ο χαρακτήρας \ είναι ο ειδικός χαρακτήρας διαφυγής και για να συμμετέχει σε μια αλφαριθμητική μεταβλητή ή σταθερά ως κανονικός χαρακτήρας πρέπει να γραφεί \\. Στο Unix τα directories διαχωρίζονται με τον χαρακτήρα /.

Η διαδικασία δημιουργίας μιας ροής κα της συσχέτισης της με το αρχείο του δίσκου ονομάζεται **άνοιγμα του αρχείου**. Η συνάρτηση βιβλιοθήκης fopen() της οποίας η δήλωση πρωτοτύπου βρίσκεται στο αρχείο επικεφαλίδα < *stdio.h* > χρησιμοποιείται για το άνοιγμα του αρχείου. Η σύνταξη της συνάρτησης αυτής είναι:

```
FILE *fopen(const char *filename, const char *mode)
```

όπου η πρώτη τυπική παράμετρος είναι το όνομα του αρχείου και η δεύτερη είναι μια αλφαριθμητική σταθερά της οποίας οι πιθανές τιμές φαίνονται στον Πίνακα 7.1 και η οποία καθορίζει τον τρόπο λειτουργίας του αρχείου. Ο προκαθορισμένος τύπος του αρχείου όταν ανοίγει είναι ο τύπος κειμένου. Αν είναι επιθυμητό να ανοιχθεί ως δυαδικό τότε πρέπει να γραφεί το *b* ως κατάληξη του τρόπου λειτουργίας του αρχείου. Για παράδειγμα αν η δεύτερη παράμετρος της συνάρτησης είναι η "w+b" το αρχείο ανοίγει ως δυαδικό έτοιμο για ανάγνωση και εγγραφή όπως ορίζεται στον Πίνακα 7.1. Η συνάρτηση *fopen* επιστρέφει δείκτη σε μια δομή τύπου FILE η οποία είναι δηλωμένη στο αρχείο < *stdio.h* > και έχει την παρακάτω μορφή:

```
typedef  struct _iobuf {
            char *_ptr;
            int _cnt;
            char *_base;
            int _flag;
            int _file;
            int _charbuf;
```

```
        int _bufsiz;
        char *_tmpfname;
}FILE;
```

τα πεδία της οποίας κρατούν πληροφορία σχετικά με την κατάσταση του αρχείου. Δεν θα επεκταθούμε σε λεπτομέρειες για τον ρόλο των πεδίων στον μηχανισμό εισόδου / εξόδου καθώς δεν χρησιμοποιούνται σε προγραμματιστικό επίπεδο. Από τη στιγμή που θα ανοιχθεί ένα αρχείο και μετά μπορούν να εκτελεσθούν λειτουργίες εισόδου / εξόδου μέσω των συναρτήσεων που θα αναλύσουμε παρακάτω. Οι συναρτήσεις που θα χρησιμοποιηθούν για ανάγνωση και εγγραφή χρησιμοποιούν τον δείκτη που επιστρέφει η *fopen*() και για το λόγο αυτό ορίζουμε πάντα μια μεταβλητή τύπου δείκτη στη δομή *FILE*.

Όταν ολοκληρωθεί η επεξεργασία στην οποία εμπλέκονται τα αρχεία τότε αυτά μπορούν να κλείσουν με τη χρήση της συνάρτησης *fclose* της οποίας η δήλωση πρωτοτύπου είναι:

```
int fclose(FILE *fp);
```

όπου *fp* είναι ο δείκτης που σχετίζεται με τη ροή του αρχείου. Επιστρέφει 0 όταν είναι επιτυχές το κλείσιμο και −1 όταν υπάρχει λάθος. Όταν ένα πρόγραμμα τερματίζει (με την ολοκλήρωση της *main*() ή μετά την εκτέλεση της *exit*()) αυτόματα κλείνουν όλα τα αρχεία.

Όταν δημιουργείται μια ροή για ένα αρχείο δίσκου αυτόματα δεσμεύεται μια προσωρινή μνήμη την οποία διαχειρίζεται το λειτουργικό σύστημα και η οποία συσχετίζεται με την ροή του εν λόγω αρχείου. Για λόγους απόδοσης, αλλά και του γεγονότος ότι ένας δίσκος είναι μια συσκευή στην οποία εγγράφονται τμήματα από *byte* και όχι χαρακτήρες σε αντίθεση με τη ροή, οποτεδήποτε το πρόγραμμα γράφει δεδομένα στη ροή τα δεδομένα μεταφέρονται στην προσωρινή αυτή μνήμη. Όταν η προσωρινή μνήμη γεμίσει τότε τα δεδομένα μεταφέρονται από το λειτουργικό αυτόματα στο δίσκο. Επειδή είναι πιθανό το πρόγραμμα να καταρρεύσει (π.χ. αστοχία ισχύος, ατέρμονοι βρόχοι) τα δεδομένα στην προσωρινή μνήμη χάνονται, το αρχείο δεν ενημερώνεται και τα δεδομένα χάνονται. Η

ενημέρωση του αρχείου μπορεί να επιτευχθεί σε κάθε σημείο του προγράμματος με τη χρήση της συνάρτησης *fflush* της οποίας η δήλωση πρωτοτύπου είναι:

int fflush(FILE *fp)

Στην περίπτωση μεγάλων αρχείων είναι επιθυμητό το πρόγραμμα να γνωρίζει αν έφτασε στο τέλος του αρχείου. Για τα αρχεία κειμένου υπάρχει μια συμβολική σταθερά η *EOF* η οποία είναι δηλωμένη στο *stdio.h* και έχει την τιμή -1. Επειδή δεν υπάρχει χαρακτήρας με αυτή την τιμή όταν μια συνάρτηση ανάγνωσης διαβάσει *EOF* είναι εύκολο να διαπιστωθεί το τέλος του αρχείου. Για την περίπτωση των δυαδικών αρχείων, όπου η τιμή -1 είναι μια δυνατή τιμή μπορεί να χρησιμοποιηθεί η συνάρτηση *feof*() η δήλωση πρωτοτύπου της οποίας είναι η εξής:

int feof(FILE *fp);

Η *feof* επιστρέφει 0 όταν δεν συναντηθεί το τέλος του αρχείου και μη-μηδενική τιμή διαφορετικά.

7.2. Συναρτήσεις Εισόδου και Εξόδου

Αντικείμενο του παρόντος εδάφιου είναι οι συναρτήσεις εισόδου και εξόδου δεδομένων από/σε αρχεία. Οι συναρτήσεις εισόδου δεδομένων διακρίνονται σε δύο κατηγορίες:

- εισόδου χαρακτήρων (*getc*(), κλπ)
- φορμαρισμένης εισόδου (*scanf*(), κλπ)

Οι συναρτήσεις της πρώτης κατηγορίας κάθε φορά που καλούνται διαβάζουν ένα χαρακτήρα από τη ροή εισόδου τη φορά τον οποίο και επιστρέφουν στην καλούσα συνάρτηση. Διακρίνονται σε δύο κατηγορίες:

- Σε αυτές με απομονωτή (buffered) όπου οι χαρακτήρες που πληκτρολογούνται αποθηκεύονται από το λειτουργικό σύστημα σε μια προσωρινή μνήμη αποθήκευσης μέχρι να πατηθεί ο χαρακτήρας τέλους γραμμής, οπότε και μεταφέρονται στη ροή *stdin*. Συνεπώς, μόνο όταν πατηθεί ο χαρακτήρας τέλους γραμμής οι χαρακτήρες θα είναι διαθέσιμοι για επεξεργασία στο πρόγραμμα.

- Σε αυτές χωρίς απομονωτή (unbuffered) όπου οι χαρακτήρες που πληκτρολογούνται μεταφέρονται απευθείας στη ροή *stdin* χωρίς προσωρινή αποθήκευση.

Οι συναρτήσεις φορμαρισμένης εισόδου, τους χαρακτήρες που διαβάζουν από την ροή εισόδου τους μετατρέπουν στον τύπο που ορίζεται στην παράμετρο μορφοποίησης των προς ανάγνωση δεδομένων των συναρτήσεων αυτών. Στην κατηγορία αυτή ανήκει η συνάρτηση *scanf*.

Τέλος, οι συναρτήσεις εξόδου δεδομένων διακρίνονται και αυτές σε δύο κατηγορίες:

- εξόδου χαρακτήρων (*putc*, κλπ.)

- φορμαρισμένης εξόδου (*printf*, κλπ.)

Οι συναρτήσεις εξόδου χαρακτήρων διακρίνονται και αυτές σε δύο κατηγορίες: με απομονωτή και χωρίς απομονωτή. Ενώ οι συναρτήσεις φορμαρισμένης εξόδου όταν εγγράφουν δεδομένα στη ροή του αρχείου υφίστανται μορφοποίηση όπως αυτή ορίζεται στην παράμετρο μορφορποίησης της συνάρτησης. Η έξοδος δεδομένων μέσω της ροής *stderr* γίνεται χωρίς απομονωτή. Στο πλαίσιο αυτού του βιβλίου δεν θα αναφερθούμε με ποιό τρόπο μπορούμε να ορίσουμε απομονωτή για μια ροή.

7.2.1. Είσοδος/Έξοδος χαρακτήρων

Μπορούν να αποθηκευτούν απλοί χαρακτήρες ή γραμμές κειμένου σε αρχείο καθώς και να διαβαστούν χαρακτήρες ή γραμμές κειμένου από αρχείο. Αυτό μπορεί να γίνει κατά βάση σε αρχεία που έχουν ανοιχθεί ως αρχεία κειμένου. Πρωταρχικός στόχος αυτού του

τρόπου εγγραφής είναι να δημιουργηθούν αρχεία τα οποία θα μπορούν να διαβαστούν από πρόγραμμα σε *C*.

Για την ανάγνωση/εγγραφή απλών χαρακτήρων χρησιμοποιούνται οι συναρτήσεις της πρότυπης βιβλιοθήκης:

- int fgetc(FILE *fp): η οποία έχει ως τυπική παράμετρο τη ροή *fp* και επιστρέφει τον χαρακτήρα τον οποίο διαβάζει από τη ροή *fp* ή *EOF*. Η συνάρτηση *getchar*() είναι ισοδύναμη με την *fgetc*(*stdin*).

- int fputc(int ch, FILE *fp): η οποία έχει ως τυπική παράμετρο τον χαρακτήρα *ch* που πρόκειται να εγγραφεί και τη ροή *fp*, και επιστρέφει τον χαρακτήρα ο οποίος εγγράφηκε ή *EOF* σε περίπτωση λάθους. Η συνάρτηση *putchar*(*ch*) είναι ισοδύναμη με την *fputc*(*ch, stdout*).

Για την εγγραφή/ανάγνωση αλφαριθμητικών μπορούν να χρησιμοποιηθούν οι συναρτήσεις της πρότυπης βιβλιοθήκης:

- char *fgets(char *buf, int N, FILE *fp): η οποία μεταφέρει χαρακτήρες από το αρχείο με ροή *fp* στη μεταβλητή αλφαριθμητικό *buf* μέχρι να συναντηθεί ο χαρακτήρας τέλους γραμμής '\n' ή μεταφερθούν $N-1$ χαρακτήρες. Θέτοντας το N ίσο με το μέγεθος του αλφαριθμητικού διασφαλίζουμε να μην γίνει παραβίαση των ορίων της μνήμης του αλφαριθμητικού. Η συνάρτηση επιστρέφει τον δείκτη *buf* ή *NULL* όταν διαβασθεί το *EOF*.

- int fputs(char *buf, FILE *fp): η οποία μεταφέρει τους χαρακτήρες του αλφαριθμητικού *buf* στο αρχείο με ροή *fp*. Πρέπει το αλφαριθμητικό να έχει τερματιστεί με τον χαρακτήρα '\0' ο οποίος δεν εγγράφεται στη ροή. Για να αποθηκευτεί ο χαρακτήρας τέλους γραμμής '\n' πρέπει να προστεθεί στο τέλος του αλφαριθμητικού πριν το χαρακτήρα '\0'. Επιστρέφει μη αρνητικό αριθμό ή *EOF* σε περίπτωση λάθους.

```
 0:  #include <stdio.h>
 1:
 2:  main(int argc, char *argv[])
 3:  {
 4:     FILE *fp=NULL;
 5:     char c;
 6:
 7:     fp = fopen(argv[1], "r");
 8:
 9:     while((c=fgetc(fp)) != EOF) printf("%c", c);
10:     printf("\\text");
11:  }
```

Σχήμα 7.1: Παράδειγμα εισόδου/εξόδου χαρακτήρων από/σε αρχείο.

Στο Σχήμα 7.1 δίνεται ένα πολύ απλό πρόγραμμα χρήσης των παραπάνω συναρτήσεων. Υποθέτοντας ότι το πρόγραμμα είναι αποθηκευμένο στο αρχείο *readFile* η εκτέλεση του μπορεί να γίνει σε Command περιβάλλον (σε Windows επιλέξτε **έναρξη** και μετά **εκτέλεση**, τρέξτε την εντολή **cmd** και μετά μετακινηθείτε στο φάκελο που βρίσκεται το εκτελέσιμο που προέκυψε από τη μεταγλώττιση) γράφοντας:

readFile <filename>

όπου < *filename* > είναι το όνομα του αρχείου από όπου θέλουμε να διαβάσουμε. Κατά την εκτέλεση του προγράμματος, η τυπική παράμετρος *argc* της *main* έχει την τιμή 2 και *argv*[0] =“readFile” *argv*[1] =< *filename* >. Έτσι στη γραμμή 7 ανοίγει το αρχείο με όνομα αυτό που περιέχεται στη παράμετρο *argv*[1] ως αρχείο κειμένου και ο δείκτης που επιστρέφεται καταχωρείται στη μεταβλητή *fp* που είναι δείκτης στη δομή *FILE*. Στην εντολή επανάληψης της γραμμής 9 διαβάζονται χαρακτήρες από το αρχείο που μόλις ανοίχθηκε μέσω της ροής *fp* μέχρι να διαβασθεί ο χαρακτήρας τέλους αρχείου *EOF*. Κάθε χαρακτήρας που διαβάζεται τυπώνεται στην οθόνη.

7.2.2. Μορφοποιημένη Είσοδος/Έξοδος

Μπορεί να χρησιμοποιηθεί η φορμαρισμένη έξοδος για την αποθήκευση δεδομένων σε αρχείο δίσκου και η φορμαρισμένη είσοδος για την ανάγνωση δεδομένων από αρχείο δίσκου. Χρησιμοποιείται για αρχεία που έχουν ανοιχθεί ως αρχεία κειμένου. Σκοπός από τη χρήση αυτού του τρόπου εγγραφής είναι να δημιουργηθεί ένα αρχείο με κείμενο ή αριθμητικές τιμές τα οποία θα διαβαστούν από άλλο πρόγραμμα όπως spreadsheets και βάσεις δεδομένων και σπάνια από άλλο πρόγραμμα *C*.

Για την φορμαρισμένη έξοδο χρησιμοποιείται η συνάρτηση της πρότυπης βιβλιοθήκης *fprintf*() της οποίας η σύνταξη είναι ίδια με αυτή της *printf*() με τη διαφορά ότι επιλέγεται η ροή του αρχείου στο οποίο θέλουμε να γράψουμε:

```
void fprintf(FILE *fp, const char *fmt, . . .);
```

Για την φορμαρισμένη είσοδο χρησιμοποιείται η συνάρτηση της πρότυπης βιβλιοθήκης *fscanf*() της οποίας η σύνταξη είναι ίδια με αυτή της *scanf*() με τη διαφορά ότι επιλέγεται η ροή του αρχείου από το οποίο θέλουμε να διαβάσουμε:

```
void fscanf(FILE *fp, const char *fmt, . . .);
```

```
 0:  #include <stdlib.h>
 1:  #include <stdio.h>
 2:
 3:  main(int argc, char *argv[])
 4:  {
 5:      FILE *fp1, *fp2;
 6:      char a[20], b[20];
 7:
 8:      if ((fp1 = fopen(argv[1], "r")) == NULL) {
 9:          printf("Error in opening Input File\n");
10:          exit(1);
11:      }
12:
13:      fgets(a, 20, fp1);
14:      fgets(b, 20, fp1);
15:
16:      if ((fp2 = fopen(argv[2], "w")) == NULL) {
17:          printf("Error in opening Output File\n");
18:          exit(1);
19:      }
20:      fprintf(fp2, "%f = %f + %f\n", atof(a)+atof(b), atof(a), atof(b));
21:
22:      fclose(fp1);
23:      fclose(fp2);
24:  }
25:
```

Σχήμα 7.2: Παράδειγμα μορφοποιημένης εισόδου και εξόδου.

Στο Σχήμα 7.2 δίνεται ένα απλό παράδειγμα μορφοποιημένης εξόδου. Υποθέτοντας ότι το πρόγραμμα είναι αποθηκευμένο στο αρχείο *addNumbers* η εκτέλεση του μπορεί να γίνει σε Command περιβάλλον (σε Windows επιλέξτε **έναρξη** και μετά **εκτέλεση**, τρέξτε την εντολή **cmd** και μετά μετακινηθείτε στο φάκελο που βρίσκεται το εκτελέσιμο που

προέκυψε από τη μεταγλώττιση) γράφοντας:

addNumbers <InputFile> <OutputFile>

όπου < *InputFile* > είναι το όνομα του αρχείου από όπου θα διαβάσουμε και < *OutputFile* > το όνομα του αρχείου στο οποίο θα γράψουμε. Κατά την εκτέλεση του προγράμματος η τυπική παράμετρος *argc* της *main* έχει την τιμή 3 και *argv*[0] =“addNumbers” *argv*[1] =< *InputFile* > *argv*[2] =< *OutputFile* >. Στο αρχείο εισόδου το οποίο είναι αρχείο κειμένου είναι αποθηκευμένοι οι αριθμοί χωρισμένοι με λευκούς χαρακτήρες.

Στις γραμμές 8 – 11 ανοίγει το αρχείο με όνομα *argv*[1] ως αρχείο κειμένου για ανάγνωση. Η ροή που επιστρέφεται αποθηκεύεται στη μεταβλητή *fp*1 και ελέγχεται για την τιμή *NULL*. Σε περίπτωση που παρουσιαστεί πρόβλημα στο άνοιγμα του αρχείου η εκτέλεση του προγράμματος τερματίζει. Με διαδοχική κλήση της συνάρτησης *fgets* στις γραμμές 13 – 14 διαβάζονται δύο αλφαριθμητικά μήκους το πολύ 20 χαρακτήρων και οι οποίοι αποθηκεύονται στου πίνακες *a* και *b*.

Στις γραμμές 16 – 19 ανοίγει το αρχείο με όνομα *argv*[2] ως αρχείο κειμένου για εγγραφή. Η ροή που επιστρέφεται αποθηκεύεται στη μεταβλητή *fp*2 και ελέγχεται για την τιμή *NULL*. Σε περίπτωση που παρουσιαστεί πρόβλημα στο άνοιγμα του αρχείου η εκτέλεση του προγράμματος τερματίζει. Τέλος, με την *fprintf*() της γραμμής 20 εγγράφονται στο αρχείο με ροή *fp*2, το αποτέλεσμα της άθροισης των αριθμών που πρόεκυψαν μετά τις διαδοχικές κλήσεις της συνάρτησης *atof*() με ορίσματα *a* και *b* καθώς και τους αριθμούς αυτούς. Για να δείτε το αποτέλεσμα της εκτέλεσης γράψτε:

cat <OutputFile>

7.2.3. Απευθείας Είσοδος/Έξοδος

Το περιεχόμενο ολόκληρου τμήματος μνήμης αποθηκεύεται σε αρχείο με την απευθείας έξοδο. Είναι ο καλύτερος τρόπος να αποθηκευτούν δεδομένα τα οποία θα διαβαστούν

μελλοντικά από άλλο πρόγραμμα *C*. Η ανάγνωση γίνεται με τον αντίστροφο τρόπο, δηλαδή ένα μέρος του αρχείου διαβάζεται και αποθηκεύεται στη μνήμη του προγράμματος. Μπορούμε να χρησιμοποιήσουμε την απευθείας έξοδο όταν το αρχείο έχει ανοιχθεί ως δυαδικό. Ένα παράδειγμα είναι η αποθήκευση ενός πίνακα από αριθμούς κινητής υποδιαστολής διπλής ακρίβειας σε ένα αρχείο με την κλήση μιας συνάρτησης απευθείας αποθήκευσης και στη συνέχεια η ανάγνωση τους από το ίδιο ή ένα άλλο πρόγραμμα *C*.

Απευθείας εγγραφή δεδομένων σε αρχείο επιτυγχάνεται με την κλήση της συνάρτησης της πρότυπης βιβλιοθήκης *fwrite*() η οποία συντάσσεται ως εξής:

```
int fwrite(void *buf, int size, int count, FILE *fp)
```

όπου *fp* είναι η ροή του αρχείου στο οποίο θα γράψουμε, *buf* είναι δείκτης σε οτιδήποτε, *size* είναι το μέγεθος σε *byte* ενός από τα στοιχεία που θα γραφούν και *count* είναι το πλήθος των στοιχείων που θα αποθηκεύτουν. Επιστρέφει το πλήθος των στοιχείων που έχουν εγγραφεί. Για παράδειγμα για την αποθήκευση ενός πίνακα (*buf*) αριθμών κινητής υποδιαστολής απλής ακρίβειας με 20 στοιχεία γράφουμε:

```
fwrite(buf, sizeof(float), 20, fp);
```

Απευθείας ανάγνωση δεδομένων από αρχείο επιτυγχάνεται με την κλήση της συνάρτησης βιβλιοθήκης *fread*() η σύνταξη της οποίας έχει ως εξής:

```
int fread(void *buf, int size, int count, FILE *fp)
```

όπου *fp* είναι η ροή του αρχείου εισόδου, *buf* είναι δείκτης σε μια περιοχή μνήμης, *size* είναι το μέγεθος σε *byte* ενός από τα στοιχεία που θα διαβαστούν και *count* είναι το πλήθος των στοιχείων που θα διαβαστούν. Επιστρέφει το πλήθος των στοιχείων που έχουν διαβασθεί.

```
 0: #include <stdio.h>
 1: #include <stdlib.h>
 2:
 3: int main()
 4: {
 5:    int i, x, outBuffer[30], inBuffer[30];
 6:    FILE *fp;
 7:
 8:    for (i = 0; i < 30; i++)
 9:       outBuffer[i] = 2 * i;
10:
11:    if ((fp = fopen("direct", "w+b")) == NULL)
12:    {
13:       fprintf(stderr, "Error opening file.");
14:       exit(1);
15:    }
16:
17:    if (fwrite(outBuffer, sizeof(int), 30, fp) != 30)
18:    {
19:       fprintf(stderr, "Error writing to file.");
20:       exit(1);
21:    }
22:
23:    rewind(fp);
24:
25:    if (fread(inBuffer, sizeof(int), 30, fp) != 30)
26:    {
27:       fprintf(stderr, "Error reading file.");
28:       exit(1);
29:    }
30:
31:    for (i = 0; i < 30; i++)
32:       printf("%d\t%d\n", inBuffer[i], outBuffer[i]);
33:
34:    fclose(fp);
35:    return(0);
36: }
```

Σχήμα 7.3: Παράδειγμα απευθείας εισόδου/εξόδου δεδομένων από/σε αρχείο.

Έστω για παράδειγμα ότι θέλουμε να αποθηκεύσουμε τους 30 πρώτους όρους της αριθμητικής συνάρτησης $a_i = 2 * i, \forall i \in N$. Ένας τρόπος είναι να δημιουργήσουμε τους όρους και να τους αποθηκεύσουμε προσωρινά σε ένα πίνακα 30 στοιχείων και στη συνέχεια να αποθηκεύσουμε τον πίνακα σε ένα αρχείο κάνοντας χρήση της συνάρτησης απευθείας εξόδου δεδομένων *fwrite*. Στο Σχήμα 7.3 δίνεται μια υλοποίηση του παραδείγματος αυτού. Με την επαναληπτική δομή των γραμμών 8 – 9 δημιουργούνται οι όροι της αριθμητικής προόδου και αποθηκεύονται στον πίνακα *outBuffer*.

Με τις εντολές των γραμμών 11 – 15 ανοίγουμε το αρχείο με όνομα *direct* ως δυαδικό αρχείο για ανάγνωση και εγγραφή ("w+b") και ο δείκτης που επιστρέφεται από την *fopen* αποθηκεύεται στην μεταβλητή *fp*. Πριν συνεχίσουμε οποιαδήποτε λειτουργία ελέγχεται η τιμή του δείκτη *fp*. Σε περίπτωση που είναι *NULL* ένα μήνυμα λάθους τυπώνεται και τερματίζει η εκτέλεση του προγράμματος.

Με την κλήση της συνάρτησης *fwrite*() της γραμμής 17 τα δεδομένα του πίνακα *outBuffer* εγγράφονται στο αρχείο με ροή *fp*. Σε περίπτωση που η επιστραφείσα τιμή είναι μικρότερη του 30 τότε η εγγραφή για κάποιο λόγο δεν εκτελέστηκε σωστά, τυπώνεται μήνυμα λάθους και τερματίζει η εκτέλεση του προγράμματος.

Μια σημαντική παράμετρος της δομής *FILE* όπως θα δούμε και στο επόμενο εδάφιο είναι ο **ενδείκτης θέσης** του αρχείου ο οποίος καθορίζει την τρέχουσα θέση του αρχείου από όπου θα εκτελεσθεί η τρέχουσα λειτουργία εγγραφής και ανάγνωσης. Με το άνοιγμα του αρχείου ο ενδείκτης θέσης τοποθετείται στην αρχή του αρχείου που είναι η θέση 0. Με την κλήση της συνάρτησης *fwrite*() συνολικά $30 \times sizeof(int) = 30 \times 4 = 120$ *byte* (θεωρούμε ότι ένας *int* καταλαμβάνει 4 *byte*) μεταφέρονται στο αρχείο. Αυτό έχει σαν αποτέλεσμα ο ενδείκτης θέσης του αρχείου να έχει την τιμή 120 μετά την επιστροφή της συνάρτησης *fwrite*().

Σε περίπτωση που θέλουμε να διαβάσουμε τους όρους της αριθμητικής συνάρτησης που εγγράφησαν με την εκτέλεση της εντολής της γραμμής 17 πρέπει να επιστρέψουμε τον ενδείκτη θέσης του αρχείου στην αρχή του αρχείου δηλαδή στη θέση 0. Αυτό πολύ εύκολα μπορεί να γίνει με την κλήση της συνάρτησης *rewind*() στη γραμμή 23 η λειτουργία της

οποίας είναι να επαναφέρει τον ενδείκτη θέσης στην αρχή του αρχείου όπως θα γίνονταν με την εκτέλεση της λειτουργίας *rewind*() σε ένα κασετόφωνο.

Με την κλήση της συνάρτησης *fread*() της γραμμής 25 διαβάζονται οι 30 όροι της αριθμητικής συνάρτησης τους οποίους γράψαμε στο αρχείο προηγουμένως και αποθηκεύονται στον πίνακα *inBuffer*. Σε περίπτωση που η συνάρτηση επιστρέψει τιμή μικρότερη του 30 (που σημαίνει ότι για κάποιο λόγο διαβάστηκαν λιγότερα στοιχεία από το αρχείο) τυπώνεται μήνυμα λάθους και τερματίζει η εκτέλεση του προγράμματος. Αν η ανάγνωση ήταν επιτυχής, ο ενδείκτης θέσης του αρχείου τοποθετείται και πάλι στη θέση 120 από όπου θα εκτελεσθούν οι επόμενες λειτουργίες.

Τέλος, με την εκτέλεση των εντολών των γραμμών 31 – 32 τυπώνονται οι πίνακες *outBuffer* και *inBuffer* οπότε μπορούμε να συγκρίνουμε τι εγγράφηκε στο αρχείο και τι διαβάσθηκε.

7.3. Ακολουθιακή και Τυχαία Προσπέλαση

Όπως αναφέρθηκε στο προηγούμενο εδάφιο στην ανάλυση του παραδείγματος του Σχήματος 7.3 μια σημαντική παράμετρος της δομής *FILE* είναι ο **ενδείκτης θέσης** του αρχείου ο οποίος καθορίζει την τρέχουσα θέση του αρχείου από όπου θα εκτελεσθεί η τρέχουσα λειτουργία εγγραφής και ανάγνωσης. Με το άνοιγμα ενός αρχείου, ένας ενδείκτης θέσης μέσα στο αρχείο δημιουργείται ο οποίος ισούται με τον αριθμό των *byte* από την αρχή του αρχείου. Όταν ένα νέο αρχείο δημιουργείται ο ενδείκτης έχει την τιμή 0 ενώ μόνο όταν ένα αρχείο ανοίγει για ενημέρωση τότε ο ενδείκτης είναι στο τέλος του αρχείου και η τιμή του ισούται με το μέγεθος του αρχείου. Όλες οι συναρτήσεις ανάγνωσης και εγγραφής που αναφέρθηκαν προηγουμένως εγγραφούν / διαβάζουν δεδομένα στην/από την τρέχουσα θέση του ενδείκτη και με την ολοκλήρωση της λειτουργίας, αυξάνουν τον ενδείκτη τόσα *byte* όσα διαβάσθηκαν ή εγγράφηκαν.

Από τα παραπάνω γίνεται φανερό ότι οι λειτουργίες ανάγνωσης και εγγραφής γίνονται κατά τρόπο ακολουθιακό δηλαδή τα δεδομένα προσπελαύνονται ακολουθιακά. Η *C*

διαθέτει ένα σύνολο από συναρτήσεις με τις οποίες ο προγραμματιστής μπορεί να ελέγξει την τιμή του ενδείκτη θέσης δίνοντας την δυνατότητα κατα κάποιο τρόπο στην τυχαία προσπέλαση των στοιχείων του αρχείου. Για να καταλάβουμε την έννοια της τυχαίας προσπέλασης ας δούμε το εξής παράδειγμα. Έστω ότι ένα αρχείο έχει ανοιχθεί ως δυαδικό με τον ενδείκτη θέσης στην αρχή του αρχείου. Για να διαβάσουμε το δεδομένο που βρίσκεται στη θέση 150 με βάση αυτά που έχουμε πει μέχρι τώρα πρέπει να διαβάσουμε όλα τα προηγούμενα δεδομένα (ακολουθιακός τρόπος προσπέλασης). Μπορούμε όμως, όπως θα δούμε, να μετακινήσουμε τον ενδείκτη θεσης στη θέση 150 και να διαβάσουμε το δεδομένο που βρίσκεται εκεί χωρίς να χρειαστεί να διαβάσουμε όλα τα προηγούμενα. Με την τυχαία πρόσβαση εννοούμε ότι μπορούμε να μετακινηθούμε σε οποιαδήποτε θέση μέσα στο αρχείο και να εκτελέσουμε λειτουργίες ανάγνωσης και εγγραφής.

Οι συναρτήσεις της πρότυπης βιβλιοθήκης με τα πρωτότυπα της δηλωμένα στο αρχείο *stdio.h* είναι οι εξής:

1. int fseek(FILE *fp, long offset, int position): Θέτει τον ενδείκτη θέσης του αρχείου με ροή *fp* για την επόμενη λειτουργία ανάγνωσης ή εγγραφής. Η παράμετρος *position* μπορεί να πάρει μια από τις παρακάτω συμβολικές σταθερές *SEEK_SET*, *SEEK_CUR* και *SEEK_END*, οι οποίες αντιστοιχούν στην αρχή του αρχείου, την τρέχουσα θέση και στο τέλος του αρχείου. Η συνάρτηση μετατοπίζει τον ενδείκτη θέσης του αρχείου *offset* θέσεις από την *position*. Έαν η εκτέλεση της συνάρτησης είναι επιτυχής τότε ο ενδείκτης τέλους αρχείου γίνεται *reset* και επιστρέφει 0.

2. void rewind(FILE *fp): Θέτει τον ενδείκτη θέσης του αρχείου με ροή *fp* στην αρχή του αρχείου, κάνει *reset* τον ενδείκτη τέλους αρχείου. Είναι ισοδύναμη με την fseek(fp, 0L, SEEK_SET) δηλαδή θέτει τον ενδείκτη θέσης 0 *byte* από την αρχή του αρχείου *SEEK_SET*.

3. long ftell(FILE *fp): Επιστρέφει την τρέχουσα την τιμή του ενδείκτη θέσης του αρχείου. Αποθηκεύοντας την τιμή αυτή, ο χρήστης μπορεί να κάνει χρήση της *fseek*

για να μετατοπίσει τον ενδείκτη θέσης σε μια νέα θέση αλλά στη συνέχεια να μπορεί να γυρίσει στην αρχική.

Οι παραπάνω συναρτήσεις δουλεύουν καλά όταν το αρχείο ανοιχθεί ως δυαδικό αρχείο σε περιβάλλον *MS – DOS* ενώ στο *UNIX* με κάθε *mode* λειτουργίας.

```
 0:  #include <stdio.h>
 1:  #include <stdlib.h>
 2:
 3:  main(int argc, char *argv[])
 4:  {
 5:     FILE *fp=NULL;
 6:     char c;
 7:
 8:     fp = fopen(argv[1], "rb");
 9:     fseek(fp, 0, SEEK_END);
10:     fseek(fp, -1, SEEK_CUR);
11:
12:     while(ftell(fp) > 0){
13:        c=getc(fp);
14:        putchar(c);
15:        fseek(fp, -2, SEEK_CUR);
16:     }
17:     return 0;
18:  }
```

Σχήμα 7.4: Παράδειγμα Τυχαίας Προσπέλασης.

Ας δούμε ένα παράδειγμα χρήσης των παραπάνω συναρτήσεων. Έστω ότι θέλουμε να διαβάσουμε τους χαρακτήρες ενός αρχείου κειμένου από το τέλος προς την αρχή. Στο Σχήμα 7.4 δίνεται το πρόγραμμα που υλοποιεί το ζητούμενο και έστω ότι το όνομα του αρχείου στο οποίο είναι αποθηκευμένο είναι το *reverseRead.c*. Μετά τη μεταγλώττιση

του προγράμματος, η εκτέλεση του λαμβάνει χώρα δίνοντας σε Command περιβάλλον (σε Windows επιλέξτε **έναρξη** και μετά **εκτέλεση**, τρέξτε την εντολή **cmd** και μετά μετακινηθείτε στο φάκελο που βρίσκεται το εκτελέσιμο που προέκυψε από τη μεταγλώττιση) και το αρχείο που θέλουμε να διαβάσουμε:

reverseRead <textFile>

Κατά την εκτέλεση του προγράμματος, οι παράμετροι της *main* έχουν τις εξής τιμές: argv[0]="reverseRead" και argv[1]=<textFile>. Στη αρχή του προγράμματος (γραμμή 8) ανοίγουμε το αρχείο ως δυαδικό και για ανάγνωση. Στη συνέχεια με την κλήση της συνάρτησης *fseek* στη γραμμή 9, ο ενδείκτης θέσης τοποθετείται στο τέλος του αρχείου που είναι η πρώτη ελεύθερη θέση του αρχείου. Με την *fseek* της γραμμής 10 μετατοπίζουμε τον ενδείκτη θέσης ένα *byte* πριν το τέλος του αρχείου δηλαδή ουσιαστικά στον τελευταίο χαρακτήρα του αρχείου.

Στην εντολή while, η κλήση της ftell επιστρέφει την τρέχουσα τιμή του ενδείκτη θέσης. Στην περίπτωση κενού αρχείου ο έλεγχος θα δώσει την τιμή *FALSE* και το σώμα της while δεν θα εκτελεσθεί. Την πρώτη φορά που θα εκτελεσθεί το σώμα της while θα διαβασθεί ο τελευταίος χαρακτήρας του αρχείου μετατοπίζοντας τον ενδείκτη θέσης ένα *byte* προς το τέλος και στη συνέχεια θα τυπωθεί στην οθόνη. Με την *fseek* της γραμμής 15 μετατοπίζουμε τον ενδείκτη θέσης 2 *byte* πίσω από την τρέχουσα θέση για να τυπώσουμε τον προτελευταίο χαρακτήρα και αυτό γιατί το ένα από τα δύο *byte* είναι αυτό που τυπώθηκε προηγουμένως και το δεύτερο είναι αυτό που θα τυπωθεί. Το σώμα της while εκτελείται μέχρι η συνάρτηση *ftell*() να επιστρέψει 0 δηλαδή την αρχή του αρχείου.

7.4. Εφαρμογή

Στο παρόν εδάφιο θα εμπλουτίσουμε το λειτουργικό σύστημα με την εντολή createRand η οποία έχει την παρακάτω σύνταξη:

createRand – *o filename* – *s seed* – *n N*

η οποία θα δημιουργεί *N* τυχαίους αριθμούς στο διάστημα 0 . . . *RAND_MAX* τους οποίους θα αποθηκεύει στο αρχείο με όνομα *filename*. Η εντολή επίσης θα λαμβάνει και την τιμή *seed* για την αρχικοποίηση της γεννήτριας τυχαίων αριθμών.

Είναι φανερό από τις παραπάνω προδιαγραφές ότι το πλήθος των ορισμάτων της εντολής συμπεριλαβανομένου και του ονόματος της είναι 7 και τα οποία θα εισέρχονται στο πρόγραμμά μας μέσω της τυπικής παραμέτρου *argv* της *main*. Συγκεκριμένα, οι τιμές του θα έχουν ως εξής:

argv[0]=“createRand”
argv[1]=“-o”
argv[2]=“filename”
argv[3]=“-s”
argv[4]=“seed”
argv[5]=“-n”
argv[6]=“N”

Ο πρώτος έλεγχος που θα γίνεται πριν την εκτέλεση οποιασδήποτε ενέργειας από το πρόγραμμα είναι αν το πλήθος των ορισμάτων χωρίς το όνομα της εντολής αν είναι 6 καθώς απαιτούνται και τα 6 ορίσματα για την λειτουργία της εντολής. Αν το πλήθος των ορισμάτων δεν είναι 6 τότε τυπώνουμε μήνυμα με το οποίο διευκρινίζεται η σύνταξη της εντολής και το πρόγραμμα τερματίζει διαφορετικά η εκτέλεση του προγράμματος συνεχίζεται με το άνοιγμα του αρχείου με όνομα *argv*[2] ως δυαδικό και για εγγραφή και ανάγνωση. Στο Σχήμα 7.5 δίνεται μια υλοποίηση της εντολής που θέλουμε να κατασκευάσουμε.

```
 0: #include <stdlib.h>
 1: #include <stdio.h>
 2:
 3: void createRandNum(int, int *, int);
 4:
 5: int main(int argc, char *argv[])
 6: {
 7:    FILE *fp=NULL;
 8:    int *randNum, size, seed, i;
 9:
10:    if (--argc == 6) {
11:
12:        //Opens the file as binary in read/write mode
13:       fp = fopen(argv[2], "w+b");
14:
15:        //Converts strings argv[6] and argv[4] into size seed respectively.
16:       size = atoi(argv[6]);
17:       seed = atoi(argv[4]);
18:
19:        //Dynamic Array of size Elements
20:       randNum = (int *) malloc(size * sizeof(int));
21:
22:        //Random Number Generation
23:       createRandNum(seed, randNum, size);
24:
25:        //Writes Dynamic Array into file fp
26:       fwrite(randNum, sizeof(int), size, fp);
27:    }
28:    else {
29:        printf("The syntax of the command:");
30:        printf("createRand -o filename -s <seed> -n number_of_Rands\n");
31:        return -1;
32:    }
33:    fclose(fp);
34:
35:    return 0;
36: }
37:
38: void createRandNum(int seed, int *randArray, int N)
39: {
40:    int i;
41:
42:    srand(seed);
43:    for(i=0; i<N; i++) randArray[i] = rand();
44: }
```

Σχήμα 7.5: Κώδικας που υλοποιεί την εντολή *createRand – o filename – s seed – n number_of_Rands*

Στη συνέχεια τα αλφαριθμητικά *argv*[4] και *argv*[6] μετατρέπονται σε ακεραίους με την κλήση της συνάρτησης *atoi* προκειμένου να δημιουργήσουμε το πλήθος των τυχαίων αριθμών που θα δημιουργήσουμε και το *seed* αρχικοποίησης της γεννήτριας τυχαίων αριθμών της *C*. Στο σημείο αυτό αφήνουμε ως άσκηση στον αναγώστη να τροποποιήσει το πρόγραμμα έτσι να μην μετατρέπονται τα *argv*[4] και *argv*[6] απευθείας αλλά μετά από έλεγχο για το αν προηγούμενο όρισμα είναι το "-s" και "-n" αντίστοιχα. Με την κλήση της *malloc* δημιουργούμε τον πίνακα *randNum N* στοιχείων. Στο σημείο αυτό έχουν ολοκληρωθεί όλες οι λειτουργίες αρχικοποίησης και αρχίζει το τμήμα της επεξεργασίας.

Για την δημιουργία των τυχαίων αριθμών έχουμε δημιουργήσει μια συνάρτηση την createRandNum η οποία έχει τρεις τυπικές παραμέτρους: το *seed* αρχικοποίησης, τον πίνακα στον οποίο θα αποθηκεύσει του τυχαίους αριθμούς και το πλήθος των τυχαίων αριθμών που θα δημιουργήσει. Στην αρχή του σώματος της συνάρτησης αρχικοποιείται η ψευδογεννήτρια τυχαίων αριθμών της *C* με την κλήση της συνάρτησης *srand*(), η δήλωση της οποίας στο αρχείο επικεφαλίδα *stdlib.h* έχει ως εξής:

```
int srand(unsigned int seed);
```

Στη συνέχεια με την επαναληπτική κλήση της συνάρτησης *rand*() της *stdlib.h*, η οποία κάθε φορά επιστρέφει έναν τυχαίο αριθμό από 0 έως *RAND_MAX* (συμβολική σταθερά που αναπαριστά τον μέγιστο ακέραιο), δημιουργούνται και αποθηκεύονται στον πίνακα τυπική παράμετρο οι τυχαίοι αριθμοί. Όταν επαναφέρεται ο έλεγχος από την συνάρτηση *createRandNum*() στην *main*() ο πίνακας είναι έτοιμος να αποθηκευθεί στο αρχείο που ανοίξαμε στην αρχή του προγράμματος. Για το σκοπό αυτό χρησιμοποιούμε την συνάρτηση *fwrite*() της πρότυπης βιβλιοθήκης της *C*.

7.5. Ασκήσεις

Άσκηση 7.5.1 *Χρησιμοποιώντας την συνάρτηση fread() να γραφεί πρόγραμμα που θα διαβάζει το περιεχόμενο ενός αρχείου κειμένου και θα το τυπώνει στην οθόνη. Συγκεκριμένα,*

η εντολή show myFile θα διαβάζει το περιεχόμενο του αρχείου myFile και θα τυπώνει τα δεδομένα στην οθόνη.

Άσκηση 7.5.2 Χρησιμοποιώντας την συνάρτηση fgets() να γραφεί πρόγραμμα που θα αναζητεί συγκεκριμένο αλφαριθμητικό εντός ενός αρχείου κειμένου. Συγκεκριμένα, η εντολή find great testFile έχει ως αποτέλεσμα να αναζητηθεί το αλφαριθμητικό great στο αρχείο testFile και σε περίπτωση που βρεθεί να τυπώνονται οι γραμμές στις οποίες εντοπίζεται.

Άσκηση 7.5.3 Να τροποποιηθεί το παραπάνω πρόγραμμα έτσι ώστε όταν δοθεί η επιλογή $-n$ να τυπώνεται και ο αριθμός της γραμμής του κειμένου.

Άσκηση 7.5.4 Ένας μετεωρολογικός σταθμός αποθηκεύει τις μετρήσεις που λαμβάνει σε ένα δυαδικό αρχείο ως εξής:ο πρώτος αριθμός N είναι τύπου int και προσδιορίζει το πλήθος των αριθμών που ακολουθούν, και στη συνέχεια ακολουθούν N αριθμοί τύπου double. Να γραφεί πρόγραμμα που διαβάζει από το αρχείο του σταθμού τις μετρήσεις και στη συνέχεια για κάθε ένα από τους αριθμούς αυτούς υπολογίζει την τιμή του πολυωνύμου $4x^3 + 5x + 4$ όπου x είναι καθένας από τους αριθμούς αυτούς. Οι αριθμοί που προκύπτουν από τον παραπάνω υπολογισμό αποθηκεύονται σε ένα νέο δυαδικό αρχείο με τον ίδιο τρόπο που είναι αποθηκευμένοι και οι αριθμοί του αρχείου πηγής. **Υπόδειξη**

Άσκηση 7.5.5 Έστω ότι σε ένα δυαδικό αρχείο βρίσκονται αποθηκευμένοι $N > 10$ αριθμοί τύπου double με τον ακόλουθο τρόπο: ο πρώτος αριθμός N είναι τύπου int και προσδιορίζει το πλήθος των αριθμών που ακολουθούν, και στη συνέχεια ακολουθούν N αριθμοί τύπου double. Να γραφεί πρόγραμμα που θα διαβάζει κατά τυχαίο τρόπο 10 αριθμούς από το αρχείο και θα υπολογίζει το άθροισμα των τιμών της συνάρτησης $f(x) = 3x^2 + 6$. Στο τέλος θα τυπώνεται το υπολογισμένο άθροισμα καθως και οι αριθμοί που διαβάσθηκαν από το αρχείο. **Υπόδειξη**

Κεφάλαιο 8

Βιβλιογραφία

[KR] Η Γλώσσα Προγραμματισμού C Brian W. Kernighan, Dennis M. Ritchie, Δεύτερη Έκδοση, Εκδόσεις Κλειδάριθμος.

[ER] Η Τέχνη και Επιστήμη της C, Eric S. Roberts, Εκδόσεις Κλειδάριθμος.

[KP] A Book On C, Programming in C Fourth Edition, Al Kelley and Ira Pohl, Addison Wesley.

[DPV] Algorithms, Sanjoy Dasgupta, Christos Papadimitriou and Umesh Vazirani, Mc Graw Hill.

Υποδείξεις

Υπόδειξη: Αν υπάρχει ακέραιος $1 < k < n$ τέτοιος ώστε $n\%k == 0$ τότε ο n είναι σύνθετος.
□

Πίσω στην Άσκηση 4.8.2

Υπόδειξη: Να γίνει χρήση των *fread* και *fwrite*. □

Πίσω στην Άσκηση 7.5.4

Υπόδειξη: Μπορούν να διαβασθούν από το αρχείο 10 αριθμοί κατά τυχαίο τρόπο υπολογίζοντας τη θέση καθενός από αυτούς με την έκφραση $(rand()\%N) + 1$, αφού έχει αρχικοποιηθεί με την *srand* η ψευδογεννήτρια της *C*. Κατά αυτόν τον τρόπο επιστρέφεται ένας αριθμός από το διάστημα $1 \ldots N$ και χρησιμοποιώντας τον κατάλληλα στην *fseek* μπορεί να βρεθεί η θέση του στοιχείου μέσα στο αρχείο. □

Πίσω στην Άσκηση 7.5.5