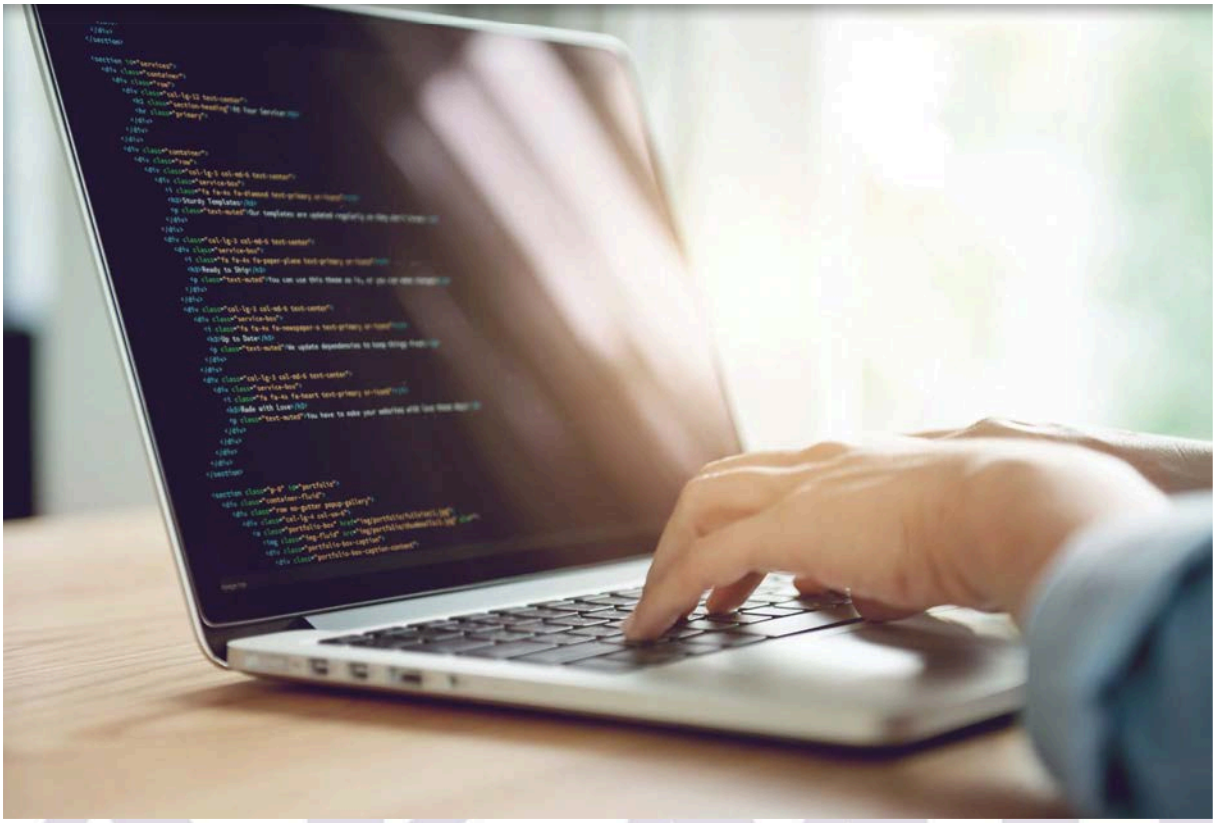
ΑΚΜΗ
Ο ΗΓΕΤΗΣ ΣΤΗΝ ΕΚΠΑΙΔΕΥΣΗ

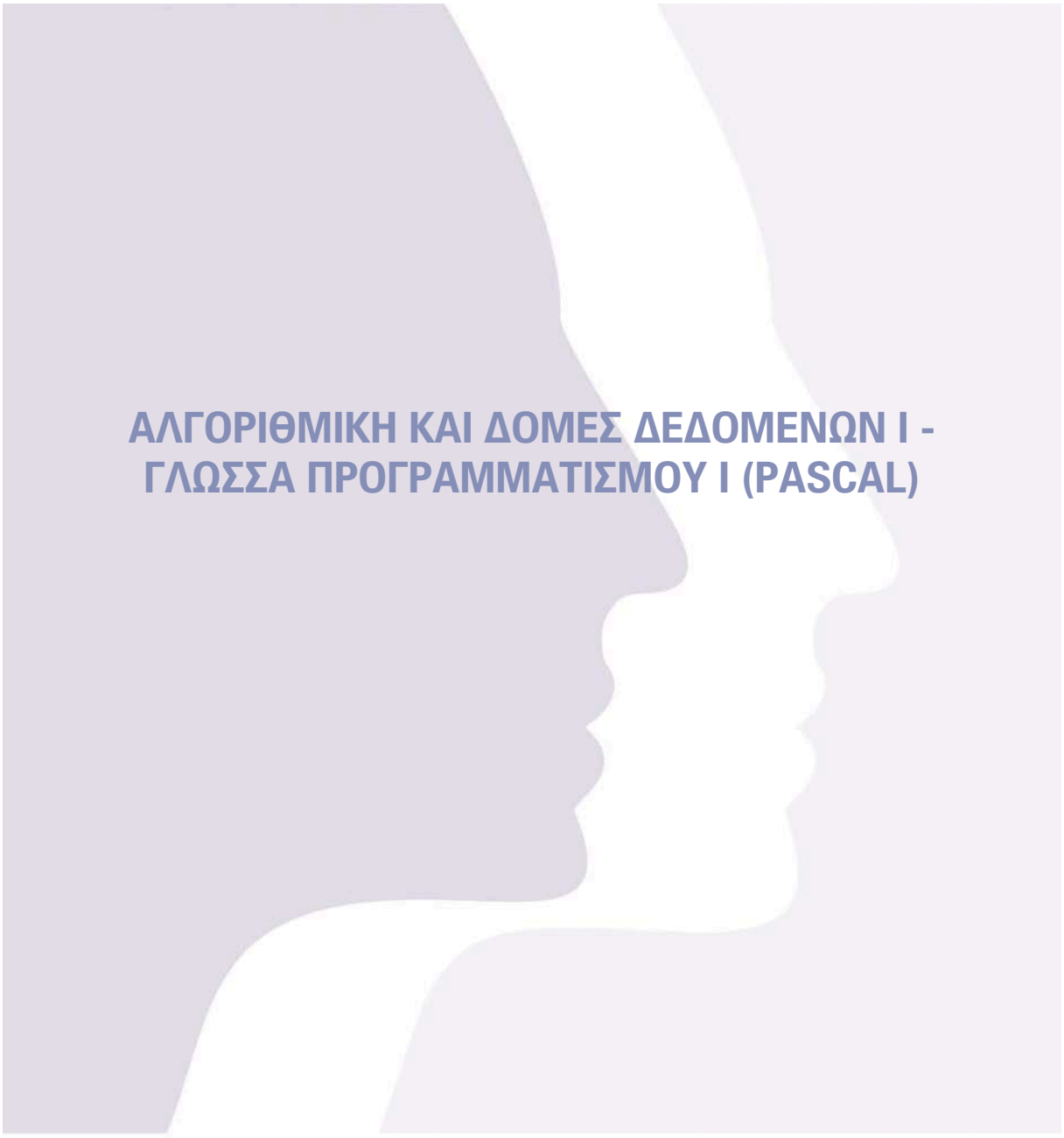
ΑΛΓΟΡΙΘΜΙΚΗ ΚΑΙ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ Ι - ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι (PASCAL)

ΑΘΑΝΑΣΙΟΣ ΓΚΟΓΚΙΔΗΣ - ΜΩΥΣΙΔΗΣ



ΠΛΗΡΟΦΟΡΙΚΗ

AKMH



ΑΛΓΟΡΙΘΜΙΚΗ ΚΑΙ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ Ι - ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι (PASCAL)

ΙΕΚ ΑΚΜΗ
ΤΟΜΕΑΣ: ΠΛΗΡΟΦΟΡΙΚΗ

ΑΚΜΗ ΑΝΩΝΥΜΗ ΕΚΠΑΙΔΕΥΤΙΚΗ ΕΤΑΙΡΕΙΑ
Κοδριγκτώνος 16, Αθήνα 112 57
Τηλ.: 210 8224074 • Fax: 210 8821801
e-mail: info@iek-akmi.edu.gr
http: //www.iek-akmi.edu.gr

ΑΘΗΝΑ | ΠΕΙΡΑΙΑΣ | ΘΕΣΣΑΛΟΝΙΚΗ | ΚΡΗΤΗ | ΛΑΡΙΣΑ | ΡΟΔΟΣ

Οργάνωση και εκτέλεση παραγωγής: Εκδόσεις Πεδίο

Πεδίο Α.Ε.
Συνταγματάρχου Δαβάκη 10 & Μυλοποτάμου, 115 26, Αθήνα
Τηλ.: 210 3390204 • Fax: 210 3390209
e-mail: info@pediobooks.gr
http: //www.pediobooks.gr

ΑΘΑΝΑΣΙΟΣ ΓΚΟΓΚΙΔΗΣ - ΜΩΥΣΙΔΗΣ

ΑΛΓΟΡΙΘΜΙΚΗ ΚΑΙ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ Ι - ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι (PASCAL)

ΙΕΚ ΑΚΜΗ
2019

AKMH

Περιεχόμενα

AKMH

1. Επικοινωνία Ανθρώπου – Η/Υ

1.1 Λογική Ανάλυση Προβλήματος

Είναι πλέον γνωστό ότι ένας Η/Υ δεν μπορεί να λύσει οποιοδήποτε πρόβλημα παρά μπορεί μόνο να κάνει στοιχειώδεις πράξεις και να παίρνει κάποιες μηχανικές αποφάσεις. Και σαν να μη φτάνει αυτό, περιμένει από τον χρήστη να του πει τι πράξεις πρέπει να κάνει και τι κριτήρια να χρησιμοποιήσει. Τη στιγμή που αρχίζει να δημιουργείται ένα πρόβλημα που μπορεί να λύσει ένας Η/Υ τίθεται αυτόματα το ερώτημα: τι πρέπει να κάνει κανείς για να λύσει το πρόβλημα; Η προεργασία αυτή αποτελεί μια λογική ανάλυση του προβλήματος, καταλήγει δε στη διατύπωση ορισμένων βημάτων που πρέπει να κάνουμε (διαδοχικά ή ταυτόχρονα) για να λύσουμε το πρόβλημα. Η λογική ανάλυση ενός προβλήματος είναι μερικές φορές πολύ δύσκολη και απαιτεί γνώσεις και ευφυΐα που μόνον ο άνθρωπος συνδυάζει. Το αποτέλεσμα της λογικής ανάλυσης ενός προβλήματος, δηλαδή τα απαιτούμενα βήματα για τη λύση, δεν είναι μονοσήμαντο για ένα πρόβλημα. Εκτός αυτού διαφορετικά προβλήματα μπορεί να απαιτούν τελείως διαφορετικά βήματα. Για τα προβλήματα εκείνα που απαιτούν την χρήση ενός Η/Υ η λογική υποδεικνύει τα ακόλουθα βήματα:

1. **Φραστική Διατύπωση του Προβλήματος.** Προσπαθούμε να διατυπώσουμε με λόγια το πρόβλημα όσο πιο καλά γίνεται. Καθορίζουμε τα δεδομένα, τον αντικειμενικό σκοπό (δηλ. τι θα θεωρήσουμε απάντηση στο πρόβλημα) και τους γενικούς κανόνες που θα διέπουν την επίλυση του προβλήματος.

2. **Μαθηματική Διατύπωση του Προβλήματος.** Διατυπώνουμε το πρόβλημα με μαθηματική αυστηρότητα και σαφήνεια. Είναι φανερό ότι η επιτυχία αυτού του βήματος εξαρτάται από την ικανότητα μας να κάνουμε συγκεκριμένο κάτι μάλλον αφηρημένο. Εξαρτάται όμως και από το ίδιο το πρόβλημα. Άλλα προβλήματα επιδέχονται εύκολη μαθηματική διατύπωση ή είναι σχεδόν έτοιμα από την αρχή και άλλα είναι από την φύση τους λιγότερο ή περισσότερο ασυμβίβαστα με τη μαθηματική γλώσσα.
3. **Αρχική Επίλυση του Προβλήματος.** Εκθέτουμε το πώς θα λύσουμε το πρόβλημα και εκτελούμε εκείνες τις πράξεις που δεν παρουσιάζουν δυσκολία στην εκτέλεσή τους από έναν άνθρωπο.
4. **Αριθμητική Ανάλυση του Προβλήματος.** Χρησιμοποιώντας μεθόδους αριθμητικής ανάλυσης διατυπώνουμε το μέρος εκείνο του προβλήματος, που θα λυθεί από τον Η/Υ έτσι ώστε να μη μένει παρά μόνο η εκτέλεση των πράξεων.
5. **Προγραμματισμός και Χρήση του Η/Υ.** Στο στάδιο αυτό επιλύουμε το μερικό πρόβλημα (όπως αυτό διατυπώθηκε στο προηγούμενο στάδιο) χρησιμοποιώντας τον Η/Υ. Επειδή το βήμα αυτό ενδιαφέρει περισσότερο, είναι σκόπιμο να αναλυθεί στα εξής χαρακτηριστικά υπο-προβλήματα:
   - 5.1 ***Λογικό Διάγραμμα*** **(ή διάγραμμα ροής ή flowchart ή block diagram).** Καθορίζουμε τη σειρά των πράξεων που θα κάνει ο Η/Υ. Η σειρά αυτή περιγράφεται με το λογικό διάγραμμα που δείχνει περιγραφικά την πορεία που θα ακολουθήσει ο Η/Υ.
   - 5.2 ***Εκλογή Γλώσσας.*** Ανάλογα με τη φύση του προβλήματος και τις γνώσεις μας, επιλέγουμε μια γλώσσα (απ' αυτές φυσικά που καταλαβαίνει ο Η/Υ) για να του μεταβιβάσουμε το πρόβλημα.
   - 5.3 ***Προγραμματισμός.*** Από το λογικό διάγραμμα γράφουμε στη γλώσσα που διαλέξαμε το τι πρέπει να κάνει ο Η/Υ. Έτσι παίρνουμε το πρόγραμμα, δηλ. μια σειρά από δεδομένα, οδηγίες παρατηρήσεις για τον Η/Υ. Αυτό το πρόγραμμα ονομάζεται πηγαίο πρόγραμμα (source program).

5.4 ***Εκτέλεση του Προγράμματος και Έλεγχος.*** Τροφοδοτούμε τον Η/Υ με το πρόγραμμα και ελέγχουμε τα αποτελέσματα που μας δίνει. Αν είναι δυνατόν ελέγχουμε όλες τις δυνατές περιπτώσεις και στη συνέχεια διορθώνουμε ή βελτιώνουμε το πρόγραμμα.

1.2 Αλγόριθμοι

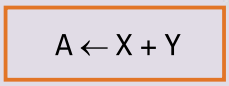
Το πρόβλημα ή το μέρος του προβλήματος που θα δοθεί στον Η/Υ πρέπει να είναι σαφές και να εξελίσσεται σύμφωνα με προκαθορισμένους νόμους (για την εκτέλεση των πράξεων) και κριτήρια (για τη λήψη αποφάσεων από τον Η/Υ). Το σύνολο των οδηγιών που θα δώσουμε στον Η/Υ αποτελεί μια συνταγή, έναν αλγόριθμο όπως λέμε. Ο αλγόριθμος αυτός σε συνδυασμό με τους κανόνες λειτουργίας του Η/Υ θα δώσει ένα αιτιοκρατικά προκαθορισμένο αποτέλεσμα. Αλγόριθμους συναντάμε πολλούς και καθημερινά στην ζωή μας, όπως π.χ. στη συνταγή ενός γιατρού, στο δέσιμο μιας γραβάτας, κλπ. Οι αλγόριθμοι αυτοί όμως είναι συχνά αφελείς αλλά και ατελείς. Χωρίς να δώσουμε μαθηματικό ορισμό του αλγόριθμου, ας αναφερθούμε σε μερικές ιδιότητες που πρέπει να έχει ένας καλός αλγόριθμος:

1. Δεν πρέπει να περιέχει αμφιβολίες, δηλ. σημεία όπου μια απόφαση πρέπει να ληφθεί χωρίς να έχουμε πει το πώς να ληφθεί.
2. Πρέπει να είναι αποτελεσματικός, δηλ. να δίνει το επιδιωκόμενο αποτέλεσμα σε πεπερασμένο χρόνο.
3. Πρέπει να είναι γενικός, δηλ. να εφαρμόζεται σε όσο το δυνατόν περισσότερες παρόμοιες περιπτώσεις.
4. Πρέπει να είναι αποδοτικός, δηλ. να φθάνει στο αποτέλεσμα με τον καλύτερο δυνατόν τρόπο.
5. Τέλος πρέπει να μπορεί να εκτελεσθεί από μια μηχανή.

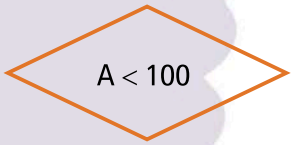
Προφανώς, εμείς θα συναντήσουμε αλγόριθμους που μπορούν να εκτελεσθούν από μια συγκεκριμένη μηχανή, έναν Η/Υ.

Για να περιγράψουμε έναν αλγόριθμο βολικά και παραστατικά για την ανθρώπινη λογική χρησιμοποιούμε το λογικό διάγραμμα. Το λογικό διάγραμμα αποτελεί μια εικόνα του τι θα γίνει μέσα στον Η/Υ και βοηθάει πολύ στο γράψιμο του προγράμματος και αποτελείται από διάφορα απλά γεωμετρικά σχήματα, από τα οποία τα πιο συνηθισμένα είναι τα εξής:

***Ένα ορθογώνιο παραλληλόγραμμο*:** Δηλώνει μια ενδιάμεση πράξη (όχι απόφαση), που περιγράφεται κατάλληλα μέσα στο ορθογώνιο, π.χ.



Ένας ρόμβος: Δηλώνει μια απόφαση και ανάλογα με τις δυνατές περιπτώσεις που παρουσιάζονται μπορεί να γίνει κάποιο άλλο πολύγωνο, π.χ.



Μια έλλειψη: Δηλώνει την αρχή ή το τέλος του λογικού διαγράμματος και μελλοντικά και του προγράμματος, π.χ.

Ένα πλάγιο παραλληλόγραμμο: Δηλώνει είσοδο δεδομένων ή εξαγωγή αποτελεσμάτων, ανάλογα με τη φορά του βέλους, π.χ.

Φυσικά, υπάρχουν και άλλα τέτοια σχήματα αλλά μάλλον τα παραπάνω είναι από τα πιο βασικά. Η γραμμή που συνδέει τα διάφορα αυτά σχήματα φέρει ένα βέλος κάθε φορά που διακόπτεται από ένα σχήμα. Το βέλος αυτό δείχνει τη σειρά διαγραφής του λογικού διαγράμματος που στην ουσία είναι η σειρά εκτέλεσης των οδηγιών από τον Η/Υ.

Εκτός από το λογικά διαγράμματα, ένας άλλος τρόπος περιγραφής ενός αλγόριθμου είναι με τη χρήση φυσικής γλώσσας και παράλληλα με συγκεκριμένα βήματα να δίνεται η λεπτομερής περιγραφή του. Αυτή είναι και η μέθοδος που θα ακολουθήσουμε στα παραδείγματα που ακολουθούν.

Παράδειγμα

Αν αγοράσουμε ΚΑ κιλά ροδάκινα Βέροιας με Α ευρώ και ΚΒ κιλά ροδάκινα Νάουσας με Β ευρώ, ποια είναι τα ακριβότερα; Να γραφεί αλγόριθμος που θα λύνει αυτό το πρόβλημα.

Λύση με χρήση φυσικής γλώσσας:

1. Εισαγωγή των δεδομένων ΚΑ, Α, ΚΒ, Β
2. Χ ← Α / ΚΑ
3. Υ ← Β / ΚΒ
4. Εάν Χ > Υ τότε ακριβότερα είναι της Βέροιας και ο αλγόριθμος τερματίζεται.
5. Εάν Χ < Υ τότε ακριβότερα είναι της Νάουσας και ο αλγόριθμος πάλι τερματίζεται.
6. Εάν Χ = Υ τότε κοστίζουν το ίδιο.

Παράδειγμα

Κάθε σπουδαστής εξετάζεται σε 3 μαθήματα. Για να περάσει το εξάμηνο πρέπει να περάσει τουλάχιστον 2 με βαθμό 5 ή μεγαλύτερο στο καθένα και να συγκε-

ντρώσει τουλάχιστον μέσο όρο 6. Να γίνει αλγόριθμος που θα λύνει αυτό το πρόβλημα.

1. Εισαγωγή των πέντε βαθμών, A, B, C
2. Έστω P ← 0 (έστω ότι δεν πέρασε κανένα μάθημα – Αρχική τιμή)
3. Έστω ΜΟ ← 0 (έστω ότι έχει μέσο όρο μηδέν – Αρχική τιμή)
4. Εάν A > 5 τότε P ← P +1
5. Εάν B > 5 τότε P ← P +1
6. Εάν C > 5 τότε P ← P +1
7. ΜΟ ← (A+B+C) / 5
8. Εάν (P > 3 και ΜΟ > 5)
   a. τότε Πέρασε
   b. αλλιώς Έμεινε
9. Τέλος Αλγόριθμου

1.3 Διαγράμματα Ροής

Αλγόριθμος (algorithm) λέγεται μία πεπερασμένη διαδικασία καλά ορισμένων βημάτων που ακολουθείται για τη λύση ενός προβλήματος. Το **διάγραμμα ροής προγράμματος (flowchart)**, σε συντομογραφία **ΔΡΠ**, μπορεί να το συναντήσετε σπανιότερα και ως **λογικό διάγραμμα**. Πρόκειται για μία σχηματική παράσταση του αλγορίθμου. Η έννοια του αλγορίθμου είναι πολύ βασική στην Πληροφορική, ενώ το διάγραμμα ροής ένα πολύ χρήσιμο περιγραφικό εργαλείο. Ένα διάγραμμα ροής αποτελείται από σύμβολα και λέξεις οι οποίες είναι σε θέση να περιγράψουν με λεπτομέρεια κάθε αλγόριθμο, κάθε διαδικασία δηλαδή επίλυσης οποιουδήποτε προβλήματος.

Η κατασκευή ενός διαγράμματος ροής προηγείται της φάσης της κωδικοποίησης. Με τον όρο **κωδικοποίηση (coding)** εννοούμε την ανάπτυξη του κώδικα. Το διάγραμμα ροής:

1. δίνει τη δυνατότητα να μελετήσουμε καλά ένα πρόβλημα και να εμβαθύνουμε σε αυτό,
2. επιτρέπει να σχεδιάσουμε μέρος του προγράμματος ή το πρόγραμμα στην ολότητά του,
3. δίνει μία χρήσιμη οπτική αναπαράσταση του αλγορίθμου,
4. επιτρέπει να διαπιστώσουμε εγκαίρως, αν ο τρόπος που έχουμε σκεφτεί για να λύσουμε το πρόβλημα είναι δόκιμος και να βεβαιωθούμε για την ορθότητα της λύσης,
5. δίνει τη δυνατότητα να αποτυπώσουμε τις σκέψεις μας για να μπορέσουμε να εργαστούμε με άλλους συνεργάτες επί της λύσης, πριν καν ακόμα ξεκινήσουμε την επίπονη διαδικασία της κωδικοποίησης και
6. επιτρέπει να συγκρίνουμε διαφορετικές μεταξύ τους λύσεις και να επιλέξουμε την καλύτερη, αρκετά νωρίς, αφού οι αλλαγές κατά την κωδικοποίηση είναι πολύ πιο ακριβές και επίπονες από ό,τι οι αλλαγές κατά τη σχεδίαση ενός λογισμικού.

Τα βασικότερα σύμβολα που χρησιμοποιούνται σε ένα διάγραμμα ροής είναι η **έλλειψη**, το **ορθογώνιο**, το **πλάγιο παραλληλόγραμμο**, ο **ρόμβος** και τα βέλη. Τα **ορθογώνια παραλληλόγραμμα** συμβολίζουν επεξεργασία. Τα **πλάγια παραλληλόγραμμα** είσοδο και έξοδο δεδομένων, ενώ οι **ρόμβοι** αποτελούν σημεία στα οποία πρέπει να ληφθεί κάποια απόφαση. Με τα **βέλη** απεικονίζουμε τη ροή του προγράμματος. Πέρα από αυτά τα σύμβολα, υπάρχουν μερικά ακόμα, δευτερεύουσας σημασίας, αλλά επίσης χρήσιμα. Ας μείνουμε όμως τώρα σε αυτά και ας τα δούμε αναλυτικά ένα ένα.

Τα **βέλη** τα χρησιμοποιούμε για να δείξουμε ότι η ροή εκτέλεσης ενός προγράμματος μεταφέρεται από το ένα σχήμα σε ένα άλλο. Όταν, δηλαδή, εκτελεστεί μία εντολή, το βελάκι που ξεκινάει από αυτό το σχήμα κατευθύνεται προς το σχήμα το οποίο πρέπει να εκτελεστεί στη συνέχεια. Από κάποιο σχήμα μπορεί να μην ξεκινά κανένα βέλος (συμβαίνει όταν τερματίζεται το πρόγραμμα), μπορεί να ξεκινά ένα βέλος, το οποίο καταλήγει στο επόμενο σχήμα στο οποίο θα περάσει η εκτέλεση, ή περισσότερα βέλη όταν πρέπει να λάβουμε κάποια απόφαση.

Η **έλλειψη** χρησιμοποιείται για να σημειωθεί η αρχή και το τέλος του αλγορίθμου. Ανάλογα με το αν συμβολίζουμε αρχή ή τέλος σημειώνουμε μέσα τη λέξη **ΑΡΧΗ** ή **ΤΕΛΟΣ** αντίστοιχα μέσα την έλλειψη. Φυσικά, κάθε πρόγραμμα έχει μία μόνο αρχή, αλλά δεν πειράζει αν τερματίζεται σε δύο διαφορετικά σημεία, ανάλογα με τη διαδρομή που θα ακολουθήσει η ροή εκτέλεσης. Από κάθε έλλειψη που συμβολίζει αρχή προγράμματος ξεκινάει ακριβώς ένα βέλος ροής, ενώ σε κάθε έλλειψη που συμβολίζει τέλος μπορούν να καταλήγουν ένα ή περισσότερα τέτοια βέλη. Φυσικά, από μία έλλειψη που συμβολίζει τέλος δεν μπορεί να ξεκινάει κανένα βέλος ροής.

Η διαδικασία εισόδου και εξόδου στοιχείων συμβολίζεται με ένα **πλάγιο παραλληλόγραμμο**. Μέσα στο πλάγιο παραλληλόγραμμο διευκρινίζουμε αν πρόκειται για είσοδο, αν πρόκειται για έξοδο και ποια μεταβλητή είναι αυτή που συμμετέχει. Αν, δηλαδή, θέλουμε να διαβάσουμε τη μεταβλητή **Α**, θα πρέπει μέσα στο πλάγιο παραλληλόγραμμο να γράψουμε κάτι σαν:

ΔΙΑΒΑΣΕ Α

Αν θέλουμε να τυπώσουμε στην οθόνη ένα μήνυμα το οποίο μας πληροφορεί για το αποτέλεσμα μιας πράξης, μπορούμε να γράψουμε:

ΤΥΠΩΣΕ “ΑΠΟΤΕΛΕΣΜΑ=”,Χ

όπου η λέξη **ΑΠΟΤΕΛΕΣΜΑ** θα τυπωθεί ως έχει (διότι είναι μέσα σε εισαγωγικά), ενώ το **X** θα αποτιμηθεί και θα εμφανιστεί η τιμή που έχει. Στο παράδειγμα της πρόσθεσης, είσοδο δεδομένων έχουμε στην αρχή του προγράμματος, όπου και πρέπει να δοθούν από τον χρήστη στον υπολογιστή οι δύο αριθμοί που θα προστεθούν. Σε εκείνο το σημείο έχουμε και έξοδο δεδομένων, αφού για κάθε έναν από τους δύο αριθμούς που ζητούνται προηγείται ένα μήνυμα διευκρινιστικό προς τον χρήστη:

Δώσε μου τον πρώτο αριθμό

και

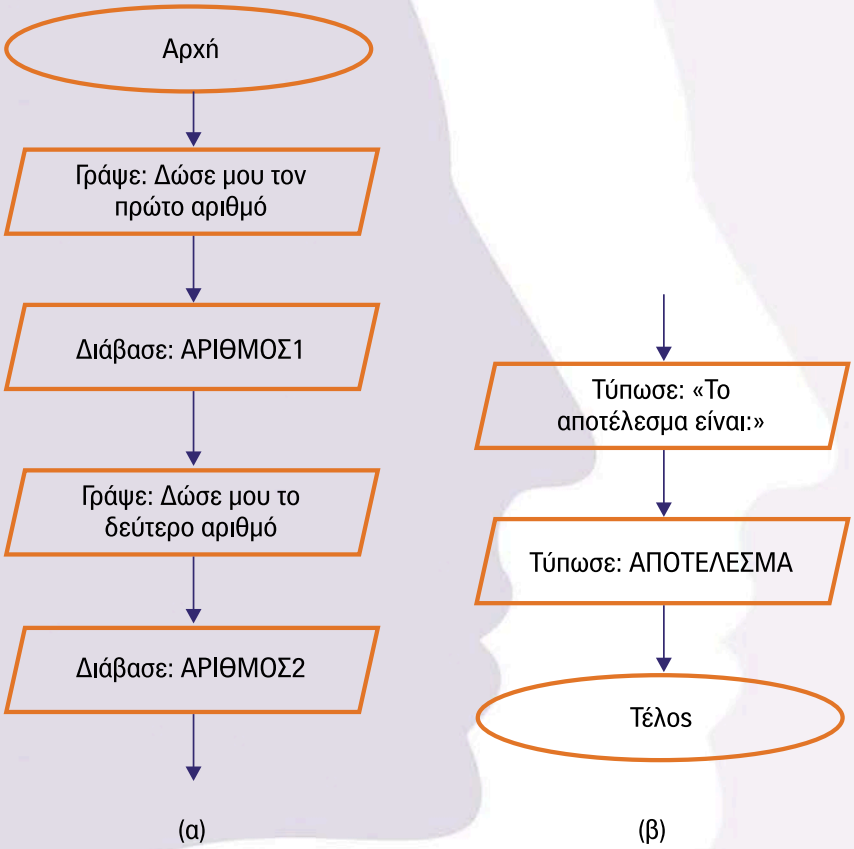
Δώσε μου τον δεύτερο αριθμό.

Έξοδο δεδομένων έχουμε στο τέλος του προγράμματος, όπου υπάρχει η εμφάνιση του αποτελέσματος στην οθόνη συνοδευόμενο πάλι από το κατάλληλο βοηθητικό μήνυμα:

Το αποτέλεσμα είναι:

Τα μέρη του διαγράμματος ροής προγράμματος για το πρόβλημα της πρόσθεσης που αναλογούν στην αρχή και στο τέλος του προγράμματος και στην είσοδο και έξοδο δεδομένων φαίνονται στο Σχήμα 1.1. Στο Σχήμα 1.1α. φαίνεται η έλλειψη στην οποία είναι σημειωμένη η αρχή του προγράμματος καθώς και τέσσερα παραλληλόγραμμα, δύο που τυπώνουν μηνύματα στην οθόνη, έτσι ώστε να γίνει πιο κατανοητό στον χρήστη τι του ζητείται, και δύο ώστε να γίνει η είσοδος από το πληκτρολόγιο για τις μεταβλητές ΑΡΙΘΜΟΣ1 και ΑΡΙΘΜΟΣ 2.

Στο Σχήμα 1.1β. φαίνονται τα αντίστοιχα δύο πλάγια παραλληλόγραμμα για την εμφάνιση του αποτελέσματος στην οθόνη και του αντίστοιχου μηνύματος πριν από αυτό. Η έλλειψη στο τέλος υποδηλώνει το τέλος του προγράμματος.



Σχήμα 1.1 Η είσοδος (α) και η έξοδος (β) του προγράμματος της πρόσθεσης.

2. Εισαγωγή στη γλώσσα προγραμματισμού C

2.1 Σύντομη Ιστορία της C

Η γλώσσα προγραμματισμού C δημιουργήθηκε από τον Dennis Ritchie στα Bell Labs το 1972, όταν αυτός και ο Ken Thompson ασχολούνταν με τον σχεδιασμό του λειτουργικού συστήματος Unix. Η C ήταν μια εξέλιξη της γλώσσας B του Ken Thompson κι εκείνη με τη σειρά της ήταν μια εξέλιξη της γλώσσας BCPL και δημιουργήθηκε για να μπορέσει να καλύψει κάποιες αυξημένες ανάγκες στον προγραμματισμό. Αργότερα αναπτύχθηκαν πολλές παραλλαγές της C που ήταν επόμενο να έχουν ασυμφωνίες μεταξύ τους. Έτσι, δημιουργήθηκε μια επιτροπή στις αρχές του καλοκαιριού του 1983 που άρχισε να δουλεύει πάνω στη δημιουργία ενός προτύπου Ansi το οποίο και θα όριζε μια για πάντα τη γλώσσα C.

2.2 Μια Σύγκριση των Γλωσσών Προγραμματισμού

Η C θεωρείται γενικά γλώσσα μέσου επιπέδου κι αυτό γιατί συνδυάζει στοιχεία των γλωσ- σών υψηλού επιπέδου (high level languages), όπως είναι η Cobol και η Pascal και στοιχεία των γλωσσών χαμηλού επιπέδου (low level languages), όπως είναι η Assembly. Για να διευκρινίσουμε, πρέπει να πούμε ότι ως γλώσσα υψηλού επιπέδου θεωρείται η γλώσσα εκείνη που είναι αρκετά περιγραφική και που είναι έτσι πιο κοντά στην ανθρώπινη γραπτή γλώσσα και ως γλώσσα χαμηλού επιπέδου θεωρείται εκείνη που είναι πιο κοντά στη μηχανή. Αυτός ο χαρακτηρισμός δεν έχει καμία απολύτως σχέση με τις δυνατότητες της γλώσ-

σας. Σίγουρα, η κάθε γλώσσα προγραμματισμού έχει κάποιες προτεραιότητες να εκπληρώσει. Η Pascal, για παράδειγμα, χρησιμοποιείται κυρίως για τη σωστή διδασκαλία των αρχών του προγραμματισμού, ενώ η Basic δημιουργήθηκε έτσι ώστε να δώσει τη δυνατότητα σε αρχάριους στον προγραμματισμό να κάνουν με άνεση κι ευκολία τα πρώτα τους βήματα στον ιδιόμορφο αυτό χώρο. Ο Clipper και η Cobol είναι καθαρά επαγγελματικές γλώσσες προγραμματισμού.

Η C, όμως, μπόρεσε να φέρει τον προγραμματιστή πιο κοντά στο hardware, που με τις άλλες γνωστές γλώσσες προγραμματισμού κάτι τέτοιο θα ήταν πολύ δύσκολο να γίνει. Βέβαια, η κάθε γλώσσα προγραμματισμού κάνει και διαφορετική δουλειά και δεν θα ήταν σωστό να κάνουμε συγκρίσεις, για τον ίδιο λόγο που δεν μπορούμε να συγκρίνουμε ένα αεροπλάνο μ' ένα ποδήλατο καθώς το καθένα είναι προορισμένο να κάνει διαφορετική δουλειά.

2.3 Σύνταξη Προγράμματος σε C

Η C επιβάλλει τον καταμερισμό του προγράμματος σε ενότητες, που ονομάζονται συναρτήσεις (functions). Εάν είναι απαραίτητο, οι συναρτήσεις μπορούν να χωριστούν και σε μικρότερες συναρτήσεις. Επίσης, στη C το κύριο πρόγραμμα είναι κι αυτό μια συνάρτηση, που ονομάζεται main().

Μια μέθοδος για το γράψιμο ενός προγράμματος στη C είναι να ξεκινήσουμε γράφοντας τη συνάρτηση main(), την ενότητα του πιο πάνω επιπέδου και μετά ν' ασχοληθούμε με τις συναρτήσεις των πιο κάτω επιπέδων. Η διαδικασία αυτή ονομάζεται πάνω-προς-τα-κάτω προγραμματισμός (top-down programming).

Η αντίστροφη διαδικασία, δηλ. το ν' ασχοληθούμε πρώτα με τις συναρτήσεις των κατώτερων επιπέδων και μετά ν' ανεβαίνουμε προς τα πάνω, ονομάζεται κάτω-προς-τα-πάνω προγραμματισμός (bottom-up programming). Ένα πλεονέκτημα του πάνω-προς-τα-κάτω προγραμματισμού είναι ότι μπορούμε να χαράξουμε καλύτερα τη ροή του προγράμματος, μια και δεν ασχολούμαστε από την αρχή με τις λεπτομέρειες των επί μέρους συναρτήσεων.

2.4 Μεταγλώττιση και Σύνδεση Προγράμματος

Ο μεταγλωττιστής (compiler) της C μετατρέπει τον πηγαίο κώδικα (source program), δηλαδή το πρόγραμμα που γράφουμε σε C, σ' έναν αντικειμενικό κώδικα (object program) και το πρόγραμμα σύνδεσης (linker) συνδυάζει αυτό τον κώδικα με άλλους κώδικες και δημιουργείται έτσι το εκτελέσιμο αρχείο (executable file). Τα προγράμματα της C έχουν την επέκταση .c.

Ο ρόλος του προγράμματος σύνδεσης είναι να ενώσει τον τελικό κώδικα, τον κώδικα εκκίνησης (start-up code) του συστήματός μας και τον κώδικα βιβλιοθήκης (library code) στο εκτελέσιμο αρχείο. Ο κώδικας εκκίνησης έχει σχέση με την επικοινωνία μεταξύ του προγράμματος και του λειτουργικού συστήματος και ο κώδικας βιβλιοθήκης περιέχει τον ολοκληρωμένο κώδικα για πολλές συναρτήσεις. Σε μερικά συστήματα πρέπει να τρέξουμε τα προγράμματα μεταγλώττισης και σύνδεσης ξεχωριστά, ενώ σ' άλλα ο μεταγλωττιστής ενεργοποιεί το πρόγραμμα σύνδεσης αυτόματα μόνος του.

2.5 Το Πρώτο Πρόγραμμα

Το παρακάτω πρόγραμμα (το πρώτο πρόγραμμα σε C) εκτυπώνει στην οθόνη του Η/Υ ένα μήνυμα, το κλασσικό πρώτο πρόγραμμα της C, «Hello Wolrd!». Ανάλογα με το περιβάλλον που θα τρέξει το πρόγραμμα είναι δυνατόν τα μηνύματα να είναι και στα Ελληνικά, αλλά αυτό πρέπει να δοκιμασθεί πρώτα (στη συνέχεια, τα μηνύματα θα είναι πάντα με λατινικούς χαρακτήρες). Η αρίθμηση υπάρχει για την ευκολία αναφοράς σε συγκεκριμένα σημεία του προγράμματος και μόνο.

```
1. #include <stdio.h>
2. int main()
3. {
4.         printf(«Hello Wolrd \n»);
5.         return 0;
6. }
```

Εάν δεν παρουσιασθούν λάθη κατά τη διάρκεια της μεταγλώττισης, τότε ο μεταφραστής θα δημιουργήσει το εκτελέσιμο πρόγραμμα και το αποτέλεσμα θα είναι να εμφανισθεί στην οθόνη του Η/Υ το μήνυμα:

Hello World!

Επεξήγηση των προτάσεων

#include <stdio.h>

Το πρώτο πράγμα που δηλώνεται σε ένα πρόγραμμα είναι οι βιβλιοθήκες που θα χρησιμοποιηθούν. Στο παραπάνω παράδειγμα ενσωματώνεται μόνο η βιβλιοθήκη stdio.h (STandard Input Output), που προσφέρει τη χρήση βοηθητικών συναρτήσεων σχετιζόμενες με τις ροές εισόδου/εξόδου. Η ροή εισόδου σχετίζεται με τα δεδομένα που θα εισάγει ο χρήστης του προγράμματος μέσω π.χ. του πληκτρολογίου και η ροή εξόδου με το τι θα εμφανίσει το πρόγραμμα στον χρήστη. Η κατάληξη .h υποδηλώνει ότι πρόκειται για αρχείο βιβλιοθήκης (header file). Σε περίπτωση που παραληφθεί η ενσωμάτωση της συγκεκριμένης βιβλιοθήκης, τότε η εντολή printf δεν θα μπορεί να αναγνωριστεί κατά τη μεταγλώττιση και κατ' επέκταση δεν θα μπορεί να χρησιμοποιηθεί.

int main( )

Κάθε πρόγραμμα αποτελείται από κάποιες συναρτήσεις και η main( ) είναι η πιο βασική συνάρτηση από αυτές. Δεν είναι αναγκαίο να υπάρχουν περισσότερες από μια συναρτήσεις, ωστόσο η ύπαρξη της main( ) είναι απαραίτητη καθώς με αυτή ξεκινάει η εκτέλεση του προγράμματος. Οι παρενθέσεις που ακολουθούν τη main υποδηλώνουν ότι αυτή η συνάρτηση δεν δέχεται παραμέτρους (σε άλλες περιπτώσεις μπορεί να δέχεται παραμέτρους). Η λέξη int στην αρχή δηλώνει ότι η συνάρτηση main( ) θα επιστρέψει έναν ακέραιο αριθμό.

Το αριστερό και δεξί άγκιστρο ({ και }) υποδηλώνουν πού αρχίζει και πού τελειώνει η συνάρτηση, εσωκλείοντας το σώμα των εντολών που θα χρησιμοποιηθούν. Επίσης μπορούν να χρησιμοποιηθούν επιμέρους άγκιστρα που θα δείχνουν την

αρχή και το τέλος μιας δομής ή ενός συνόλου εντολών όπως θα φανεί σε επόμενα κεφάλαια.

Η **printf( )** είναι μία συνάρτηση εμφάνισης πληροφοριών στην οθόνη (ροή εξόδου), παρέχοντας περαιτέρω δυνατότητες μορφοποίησης και προσφέρεται μέσω της βιβλιοθήκης stdio.h. Η συνάρτηση πρόκειται να εκτυπώσει ό,τι υπάρχει μέσα στα εισαγωγικά εκτός από κάποιους ειδικούς χαρακτήρες που ακολουθούν μία ανάποδη κάθετο (π.χ. \n). Αυτοί οι χαρακτήρες αλλάζουν τη λειτουργία εκτύπωσης της συνάρτησης. για παράδειγμα ο n αναγκάζει την printf( ) να αλλάξει γραμμή.

Η εντολή **return** δηλώνει τι θα πρέπει να επιστραφεί από τη συνάρτηση. Η συγκεκριμένη συνάρτηση δεν επιστρέφει τίποτα και γι'αυτό τοποθετείται το μηδέν.

Προσοχή, τα κενά διαστήματα και οι αλλαγές γραμμής δεν λαμβάνονται υπόψη από τον μεταγλωττιστή. Αυτό σημαίνει ότι το παραπάνω πρόγραμμα θα μπορούσε να έχει γραφεί ως εξής:

1. #include <stdio.h> int main(){
2. printf(«Hello world!\n»); return 0;}

Παράδειγμα

Να γραφεί ένα πρόγραμμα που να εκτυπώνει το μήνυμα Hello world this is an awesome day σε τρεις διαφορετικές γραμμές.

```
#include <stdio.h>
int main()
{
        printf(«Hello world\n this is\n an awesome day \n»);
        return 0;
}
```

Το πρόγραμμα θα μπορούσε να είχε γραφεί και ως εξής (με ακριβώς ίδιο αποτέλεσμα):

```
#include <stdio.h>
int main()
{
        printf(«Hello world»);
        printf(«\n this is»):
        printf(\n an awesome day \n»);
        return 0 ;
}
```

2.5.1 Η main ως συνάρτηση εισαγωγής δεδομένων από τη γραμμή εντολών

Μία δεύτερη λειτουργία της main() αφορά στο γεγονός πως, μέσω της συνάρτησης αυτής, μπορούμε να εισάγουμε δεδομένα τα οποία θέλουμε το πρόγραμμά μας να χρησιμοποιεί κάθε φορά που εκτελείται, κατά περίπτωση. Η εισαγωγή των δεδομένων αυτών, γίνεται στη γραμμή εντολών ενός shell (unix-command line, dos-command line κ.λπ. αναλόγως του λειτουργικού συστήματος). Μία πολύ συνηθισμένη ανάγκη για εισαγωγή τέτοιων δεδομένων, αφορά στις περιπτώσεις όπου ένα πρόγραμμα χρησιμοποιεί αρχεία εισόδου και εξόδου. Μπορούμε, για παράδειγμα, να φανταστούμε μία εφαρμογή η οποία ανοίγει ένα αρχικό αρχείο κειμένου (αρχείο εισόδου), επεξεργάζεται τα δεδομένα του και παράγει ένα τελικό αρχείο (αρχείο εξόδου). Μία εφαρμογή αυτού του είδους, δεν έχει φυσικά γραφτεί για κάποια συγκεκριμένα αρχεία. Δηλώνοντας έτσι διαφορετικά αρχεία εισόδου και εξόδου κάθε φορά που τρέχουμε την εφαρμογή, ο κώδικάς μας μπορεί να επαναχρησιμοποιείται για διαφορετικά αρχεία.

Σε τέτοιες περιπτώσεις η main() συντάσσεται με διαφορετικό τρόπο:

```
int main(int argc, char *argv[]){
        ...
}
```

όπου, η μεταβλητή argc αφορά στο πλήθος ορισμάτων που το πρόγραμμα περιμένει να διαβάσει από τη γραμμή εντολών και argv[] ένας πίνακας στον οποίο αποθηκεύονται τα ορίσματα αυτά.

3. Μεταβλητές, Σταθερές, Τελεστές και Παραστάσεις

Οι τελεστές χειρίζονται μεταβλητές και σταθερές και σχηματίζουν παραστάσεις.

Αυτά τα τέσσερα στοιχεία –μεταβλητές, σταθερές, τελεστές και παραστάσεις– αποτελούν τον κορμό της γλώσσας C.

3.1 Ονοματοδοσία

Η γλώσσα C ονοματίζει τις μεταβλητές ή τις συναρτήσεις, με αυτό τον τρόπο ορίζονται τα αναγνωριστικά (identifiers) ώστε να γίνει χρήση εκάστοτε μεταβλητών ή συναρτήσεων. Στη C η ονοματοδοσία (δηλαδή το αναγνωριστικό) μπορεί να είναι ένας ή περισσότεροι χαρακτήρες και κάθε χαρακτήρας μπορεί να είναι κάποιο πεζό ή κεφαλαίο λατινικό γράμμα, κάποιος αριθμός ή η κάτω παύλα (underscore). Ωστόσο, ο πρώτος χαρακτήρας πρέπει να είναι γράμμα ή η κάτω παύλα. Επίσης, δεν μπορεί να χρησιμοποιηθεί το κενό στην ονοματοδοσία, αυτό σημαίνει ότι το όνομα μιας μεταβλητής δεν μπορεί να αποτελείται από δύο λέξεις διαχωριζόμενες από ένα κενό, σε αυτή την περίπτωση οι δύο λέξεις θα έπρεπε να διαχωρίζονται αποκλειστικά από την κάτω παύλα. Ακόμα, η C κάνει διάκριση μεταξύ κεφαλαίων και μικρών γραμμάτων. Για παράδειγμα, τα **vathmos** και **VATHMOS** είναι δύο διαφορετικά ονόματα. Ένα όνομα δεν μπορεί να είναι ίδιο με κάποια δεσμευμένη λέξη της C, με κάποια συνάρτηση από αυτές που έχουμε γράψει εμείς οι ίδιοι ή με αυτές που προσφέρονται από κάποια βιβλιοθήκη που έχει ενσωματωθεί.

Παρακάτω αναφέρονται ορισμένα παραδείγματα σωστών και λανθασμένων αναγνωριστικών.

Σωστό	**Λάθος**
sum	10sum
mesos_oros	mesos oros
counter	counter!

3.2 Τύποι Δεδομένων

Κάθε μεταβλητή της C θα πρέπει να δηλωθεί πριν χρησιμοποιηθεί προκειμένου να δεσμευτεί μια ή περισσότερες θέσεις μνήμης, ανάλογα με τη μεταβλητή, και να οριστεί ο τύπος της συγκεκριμένης μεταβλητής, δηλαδή τι είδος τιμών θα καταχωρούνται σε αυτή. Σε διαφορετική περίπτωση, αν δηλαδή δεν δηλωθεί η μεταβλητή, θα υπάρχει πρόβλημα κατά την μεταγλώττιση και δεν θα αναγνωρίζεται η μεταβλητή. Επίσης, το ίδιο πρόβλημα δημιουργείται σε περίπτωση που η δήλωση μιας μεταβλητής έπεται μιας πρότασης που χρησιμοποιεί την αντίστοιχη μεταβλητή. Γι'αυτό τον λόγο είναι καλή πρακτική όλες οι μεταβλητές να δηλώνονται στην αρχή του προγράμματος.

Στη C, υπάρχουν τρεις βασικοί τύποι δεδομένων: *χαρακτήρας*, *ακέραιος*, *αριθμός κινητής υποδιαστολής (πραγματικός αριθμός)*. Η δήλωση μεταβλητών τέτοιου τύπου γίνεται χρησιμοποιώντας τις λέξεις κλειδιά char, int και float αντίστοιχα.

Στις μεταβλητές τύπου char καταχωρούνται χαρακτήρες τύπου a, b, c, σύμβολα όπως !,@ ή ακόμα και αριθμητικοί χαρακτήρες '0', '1', '2',..., '9'. Σε κάθε περίπτωση αυτοί οι χαρακτήρες πρέπει να περιέχονται στον πίνακα κωδικοποίησης ASCII.

Οι μεταβλητές τύπου int μπορούν να περιέχουν ακέραιες τιμές και οι μεταβλητές τύπου float χρησιμοποιούνται όταν μια τιμή ενδεχομένως να διαθέτει κλασματικό μέρος, για παράδειγμα να καταχωρείται ο μέσος όρος κάποιων άλλων τιμών. Μαζί με τις μεταβλητές float χρησιμοποιούνται και οι μεταβλητές double, όπου η

διαφορά τους έγκειται στο μέγεθος του μεγαλύτερου (και του μικρότερου) αριθμού που μπορεί να εκχωρηθεί. Στον παρακάτω πίνακα δίνεται το ελάχιστο μέγεθος και το εύρος τιμών των βασικών τύπων δεδομένων της C.

Τύπος	Ελάχιστο πλάτος σε bit	Εύρος
char	8	0 έως 255
int	16	-32769 έως 32767
float	32	3.4E -38 έως 3.4E +38
double	64	1.7E – 308 έως 1.7E +308

3.3 Τροποποιητές Τύπων

Οι βασικοί τύποι δεδομένων μπορούν να χρησιμοποιηθούν μαζί με κάποιους τροποποιητές (modifiers), οι οποίοι βολεύουν σε κάποιες ιδιαίτερες περιπτώσεις.

Τροποποιητές	
signed	με χρήση προσήμου
unsigned	χωρίς χρήση προσήμου
long	μεγάλος
short	μικρός

Στον παρακάτω πίνακα εμφανίζονται όλοι οι πιθανοί τροποποιητές τύπων.

Τύπος	Ελάχιστο πλάτος σε bit	Εύρος
char	8	-128 έως 127
unsigned char	8	0 έως 255
signed char	8	-128 έως 127
int	16	-32768 έως 32767
unsigned int	16	0 έως 65535

signed int	16	-32768 έως 32767
short int	16	-32768 έως 32767
unsigned short int	16	0 έως 65535
signed short int	16	-32768 έως 32767
long int	32	-2147483648 έως 2147483649
signed long int	32	-2147483648 έως 2147483649
unsigned long int	32	0 έως 4294967296
float	32	3.4E -38 έως 3.4E +38
double	64	1.7E – 308 έως 1.7E +308
unsigned long double	64	3.4E -4932 έως 1.1E +4932

3.4 Δήλωση Μεταβλητών

Η πρόταση βάσει της οποία δηλώνεται μια μεταβλητή είναι:

τύπος όνομα_μεταβλητής;

Ο *τύπος* πρέπει να είναι ένας από τους προαναφερθέντες τύπους δεδομένων, ενώ το όνομα_μεταβλητής πρέπει να ακολουθεί τους κανόνες ονοματοδοσίας. Επίσης, στην ίδια πρόταση μπορούν να συμπεριληφθούν περισσότερα ονόμα_μεταβλητής διαχωριζόμενα με κόμμα, υποδηλώνοντας τη δήλωση περισσότερων μεταβλητών του ίδιου όμως τύπου.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int x, y, z;
        float average;
        return 0;
}
```

Κατά τη δήλωση των μεταβλητών υπάρχει η δυνατότητα αρχικοποίησή τους με κάποια τιμή. Δηλαδή, κατά την πρόταση που ορίζει τη μεταβλητή και δεσμεύονται κάποιες θέσεις μνήμης για τη συγκεκριμένη μεταβλητή μπορεί να καταχωρηθεί μια αρχική τιμή. Με αυτό τον τρόπο μειώνεται το πλήθος των γραμμών του κώδικα. Η πρόταση αρχικοποίησης μιας μεταβλητής είναι:

τύπος όνομα_μεταβλητής = τιμή;

Παράδειγμα:

```
#include <stdio.h>
int main()
{
        int sum = 0;
        return 0;
}
```

3.5 Οι Δεσμευμένες Λέξεις (Λέξεις – Κλειδιά) της C

Η C, όπως και κάθε γλώσσα προγραμματισμού, χρησιμοποιεί κάποιες δεσμευμένες λέξεις (λέξεις κλειδιά - keywords) και πρόκειται για ενέργειες που θα εκτελεστούν καθώς και για τον τρόπο με τον οποίο θα εκτελεστούν. Οι δεσμευμένες λέξεις του Προτύπου ANSI παρατίθενται στον επόμενο πίνακα.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Όλες οι δεσμευμένες λέξεις συντάσσονται με πεζά γράμματα, καθώς η C είναι case-sensitive, δηλαδή διακρίνει τα κεφαλαία με τα πεζά γράμματα. Επομένως, η λέξη break είναι μια δεσμευμένη λέξη, ενώ η BREAK δεν είναι.

3.6 Σχόλια

Τα σχόλια στις γλώσσες προγραμματισμού χρησιμοποιούνται προς επεξήγηση ορισμένων γραμμών κώδικα, ώστε να είναι καλύτερη η κατανόηση της λειτουργίας του προγράμματος σε κάποιον τρίτο ή σε περίπτωση που ακόμα και εμείς οι ίδιοι ανατρέξουμε μελλοντικά στο πρόγραμμα. Τα σχόλια μπορούν να τοποθετηθούν σε οποιοδήποτε σημείο του προγράμματος, και αν πρόκειται για σχόλια μόνο μιας γραμμής, τότε χρησιμοποιούνται δύο κάθετες γραμμές (//), ενώ αν πρόκειται για περισσότερες γραμμές, τότε χρησιμοποιείται ο σημειωτής έναρξης /* και ο σημειωτής λήξης */. Οποιεσδήποτε γραμμές εσωκλείονται μέσα στους σημειωτές θεωρούνται σχόλια και θα αγνοηθούν κατά τη μεταγλώττιση.

3.7 Σταθερές

Στις γλώσσες προγραμματισμού εκτός από τις μεταβλητές χρησιμοποιούνται και κάποιες σταθερές τιμές που μένουν αμετάβλητες καθ' όλη την έκταση του προγράμματος. Οι σταθερές μπορούν να ανήκουν σε οποιονδήποτε βασικό τύπο δεδομένων και η παράσταση της κάθε σταθεράς εξαρτάται από τον τύπο της. Όπως όλοι οι χαρακτήρες, έτσι και οι σταθερές τύπου χαρακτήρα τοποθετούνται μέσα σε μονά εισαγωγικά ('x', 'y', 'z'). Παρακάτω αναφέρονται ορισμένα παραδείγματα σταθερών.

Τύπος δεδομένου	Σταθερές
char	'a' '\t' '7'
int	2 234 2456 -453
long int	32000 -43

short int	7 -10 80
unsigned int	9000 879 30000
float	132.34 3.44e -3
double	132.34 132456 -0.96758

3.8 Αλφαριθμητικές Σταθερές

Η C υποστηρίζει και τις αλφαριθμητικές σταθερές. Πρόκειται για έναν ακόμα τύπο σταθερών πλην των προαναφερθέντων τύπων σταθερών. Τα αλφαριθμητικά είναι ένα σύνολο χαρακτήρων που εσωκλείονται μέσα σε διπλά εισαγωγικά. Για παράδειγμα το "Hello World" είναι ένα αλφαριθμητικό. Θα πρέπει να δοθεί προσοχή στην διαφορά των αλφαριθμητικών και των απλών χαρακτήρων, η διαφορά τους έγκειται αφενός στο πλήθος των χαρακτήρων, δηλαδή τα αλφαριθμητικά αποτελούνται από έναν ή περισσότερους χαρακτήρες. Αφετέρου ένας απλός χαρακτήρας τοποθετείται σε μονά εισαγωγικά σε αντίθεση με τα αλφαριθμητικά που τοποθετούνται σε διπλά. Για παράδειγμα το 'A' είναι χαρακτήρας, ενώ το "A" είναι αλφαριθμητικό.

3.9 Χαρακτήρες Διαφυγής

Η C διαθέτει κάποιους ειδικούς χαρακτήρες που χρησιμοποιούνται για κάποιες συγκεκριμένες ενέργειες, δεν εμφανίζονται στην οθόνη αλλά ορίζουν κάποιες από τις λειτουργίες της, όπως παραδείγματος χάριν, την αλλαγή γραμμής. Οι χαρακτήρες ή τελεστές διαφυγής εμφανίζονται στον παρακάτω πίνακα.

Χαρακτήρες Διαφυγής	Ενέργεια
\b	μετακίνηση μιας θέσης πίσω
\f	αλλαγή σελίδας
\n	αλλαγή γραμμής

\r	μετακίνηση στην αρχή της γραμμής
\t	οριζόντιος στηλοθέτης
\"	Εμφάνιση διπλού εισαγωγικού
\'	Εμφάνιση μονού εισαγωγικού
\?	Εμφάνιση λατινικού ερωτηματικού
\\	Εμφάνιση ανάποδης καθέτου
\v	κατακόρυφος στηλοθέτης

Παράδειγμα

```
#include <stdio.h>
int main()
{
        printf("Rory says\n\"Hi\" ");
        return 0;
}
```

Εμφάνιση

```
Rory says
"Hi"
```

Τα διπλά εισαγωγικά έχουν ειδικό νόημα στη C, καθώς σε αυτά εσωκλείονται τα αλφαριθμητικά. Γι' αυτό δεν μπορούν να χρησιμοποιηθούν σαν χαρακτήρας, οπότε σε περίπτωση που θέλουμε να τα εμφανίσουμε θα πρέπει να χρησιμοποιηθεί ο τελεστής διαφυγής \" .

3.10 Η συνάρτηση printf( )

Η συνάρτηση printf( ) προσφέρεται από την βιβλιοθήκη stdio.h και χρησιμοποιείται για να εμφανιστούν δεδομένα στην οθόνη του υπολογιστή. Η σύνταξη της συνάρτησης είναι:

printf("αλφαριθμητικό ελέγχου", ορίσματα);

Το αλφαριθμητικό ελέγχου αποτελείται από το κείμενο που θέλουμε να εμφανιστεί στην οθόνη, καθώς και από εντολές που καθορίζουν τον τρόπο εμφάνισης των ενδεχόμενων ορισμάτων. Οι εντολές που θα μπορούσαν να χρησιμοποιηθούν εμφανίζονται στον επόμενο πίνακα.

Κωδικός	Έξοδος
%c	απλός χαρακτήρας
%d	ακέραιος αριθμός με πρόσημο
%u	δεκαδικός ακέραιος χωρίς πρόσημο
%e	αριθμός κινητής υποδιαστολής, συμβολισμός e
%f	αριθμός κινητής υποδιαστολής, δεκαδικός συμβολισμός
%g	χρησιμοποιεί το συντομότερο από %e ή %f
%s	αλφαριθμητικό
%x	δεκαεξαδικός ακέραιος χωρίς πρόσημο (a – f)
%%	εμφανίζει το σύμβολο %
%p	εμφανίζει ένα δείκτη

Για να χρησιμοποιηθεί μια εντολή στην printf( ), πρέπει να συνταχθεί το σύμβολο % μαζί με τον εκάστοτε χαρακτήρα που αντιστοιχεί σε μια διαφορετική ενέργεια. Είναι απαραίτητο ο αριθμός των ορισμάτων να ισούται με τον αριθμό των εντολών που θα χρησιμοποιηθούν εντός του αλφαριθμητικού. Αυτό συμβαίνει, γιατί προκειμένου να

εμφανιστεί το όρισμα (μια μεταβλητή ή κάποια μαθηματική πράξη μεταξύ αριθμών ή μεταβλητών) είναι απαραίτητη η χρήση κάποιας εντολής εντός του αλφαριθμητικού που θα υποδηλώσει τον τρόπο εμφάνισης του ορίσματος, δηλαδή αν θα εμφανιστεί με ακέραια ή δεκαδική μορφή κλπ. Επίσης, η εντολή καθορίζει και σε ποιο ακριβώς σημείο του αλφαριθμητικού θα εμφανιστεί το όρισμα. Να σημειωθεί, πως η printf( ) μπορεί να εμφανίσει περισσότερα από ένα ορίσματα, όπου το πρώτο όρισμα αντιστοιχίζεται στην πρώτη εντολή, το δεύτερο με την δεύτερη εντολή κοκ.

Παράδειγμα

```
int main( ){
        int math_grade=14;
        char mark='b';
        printf("Your grades are:\n math=%d\n chem=%f \n your mark is %c
        \n",math_grade,15.5,mark);
        return 0;
}
```

Εμφάνιση

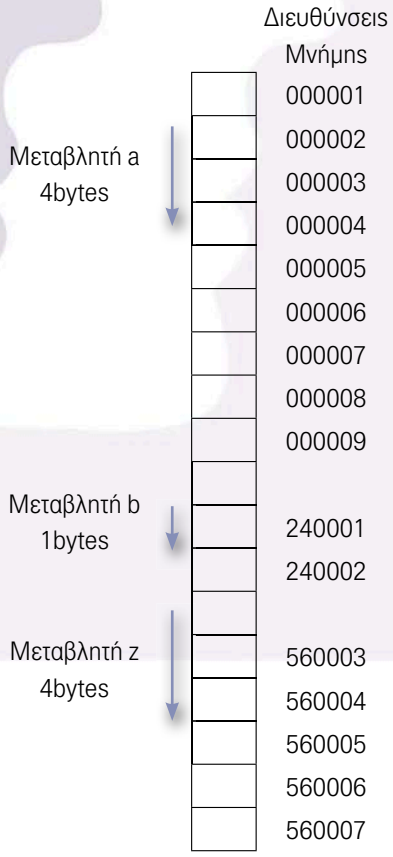
```
Your grades are:
math=14
chem=15.500000
your mark is b
```

Οι εντολές της printf( ) μπορούν να χρησιμοποιηθούν μαζί με κάποιους μετατροπείς, αλλάζοντας έτσι τον τρόπο εμφάνισης των ορισμάτων. Για παράδειγμα, για να καθοριστεί ο αριθμός των δεκαδικών ψηφίων που εμφανίζονται σε έναν πραγματικό αριθμό τροποποιούμε την αντίστοιχη εντολή ως εξής %5.2f. Με αυτή την τροποποίηση θα εμφανιστεί ένας αριθμός πλάτους τουλάχιστον 5 χαρακτήρων με δυο δεκαδικά ψηφία. Όταν εφαρμόζουμε μια εντολή σαν κι αυτή σε αλφαριθμητικά ή ακεραίους, ο αριθμός που ακολουθεί την τελεία καθορίζει

το μέγιστο μήκος του πεδίου. Για παράδειγμα το %5.7s θα εμφανίσει ένα αλφαριθμητικό μήκους τουλάχιστον πέντε χαρακτήρων και όχι μεγαλύτερο από 7 χαρακτήρες. Αν το αλφαριθμητικό είναι μεγαλύτερο από το μέγιστο μήκος πεδίου, ο υπολογιστής θα αποκόψει τους χαρακτήρες που βρίσκονται στο τέλος.

3.11 Η C και η Μνήμη

Η C χρησιμοποιεί τρεις βασικούς τύπους μεταβλητών, τις ακέραιες, τις πραγματικές και τις μεταβλητές χαρακτήρων. Μπορεί να θεωρηθεί ότι κάθε μεταβλητή είναι ένα κελί ή κουτάκι, που χαρακτηρίζεται από κάποιο όνομα και διαθέτει κάποιο περιεχόμενο. Κάθε μεταβλητή καταλαμβάνει έναν συγκεκριμένο χώρο στη μνήμη RAM του υπολογιστή. Κάθε θέση μνήμης της RAM έχει μέγεθος 1 byte, ωστόσο οι μεταβλητές της C καταλαμβάνουν 4 byte, αν πρόκειται για ακέραιες ή πραγματικές μεταβλητές, ενώ 1 byte αν πρόκειται για μεταβλητές χαρακτήρα. Επίσης, υπάρχουν και άλλοι τύποι μεταβλητών που στηρίζονται στους τρεις βασικούς τύπους μεταβλητών, που καταλαμβάνουν διαφορετικό μέγεθος σε bytes και που θα αναφερθούν αργότερα.



Στο διπλανό σχήμα φαίνεται η μνήμη ενός υπολογιστή ως σύνολο θέσεων μνήμης. Η κάθε θέση μνήμης καταλαμβάνει ένα κουτάκι μεγέθους 1 byte και προσδιορίζεται από έναν μοναδικό αριθμό που αποτελεί τη διεύθυνση της συγκεκριμένης θέσης μνήμης. Όταν δηλώνεται μια μεταβλητή δεσμεύονται κάποιες θέσεις μνήμης ανάλογα με τον τύπο της εκάστοτε μεταβλητής

Έστω ότι έχουν δεσμευτεί τρεις μεταβλητές με τις παρακάτω προτάσεις.

int a;

char b;

float z;

Η μεταβλητή a είναι μια ακέραια μεταβλητή και δεσμεύει 4 θέσεις μνήμης (3, 4, 5 και 6), η b είναι μεταβλητή χαρακτήρα και δεσμεύει μόνο μια θέση μνήμης (240001). Ενώ η z είναι πραγματική μεταβλητή και δεσμεύει 4 bytes (560003, 560004, 560005 και 560006)

Η διεύθυνση μιας μεταβλητής είναι η διεύθυνση του πρώτου byte των θέσεων (ή της θέσης) μνήμης που καταλαμβάνει μια μεταβλητή. Επομένως οι διευθύνσεις των προηγούμενων μεταβλητών είναι:

a → 000003

b → 240001

z → 560003

3.12 Ο Τελεστής &

Ο τελεστής & επιστρέφει τη διεύθυνση μνήμης μιας μεταβλητής. Βάσει του προηγούμενου σχήματος η πρόταση &a επιστρέφει τη διεύθυνση της μεταβλητής a, δηλαδή την τιμή 3 και η πρόταση &z επιστρέφει τη διεύθυνση της μεταβλητής z, δηλαδή την τιμή 560003. Στα περισσότερα συστήματα υπολογιστών οι διευθύνσεις των θέσεων μνήμης αναπαρίστανται σε δεκαεξαδική μορφή

3.13 Η συνάρτηση scanf( )

Η βασική συνάρτηση εισαγωγής δεδομένων από τον χρήστη στο πρόγραμμα είναι η scanf( ), που προσφέρεται από τη βιβλιοθήκη stdio.h. Η scanf( ) διαβάζει δεδομένα από το πληκτρολόγιο και τα καταχωρεί σε μεταβλητές που χρησιμο-

ποιούνται από το πρόγραμμα. Κατά τη χρήση της συνάρτησης είναι βασικό να καθοριστούν ο τύπος των δεδομένων που πρόκειται να εισαχθούν καθώς και οι διευθύνσεις των θέσεων μνήμης στις οποίες θα γίνει η καταχώρηση. Η σύνταξη της scanf( ) θυμίζει αυτή της print( ):

scanf("αλφαριθμητικό μορφοποίησης", διεύθυνση_μνήμης);

Το αλφαριθμητικό μορφοποίησης (format string) καθορίζει τον τύπο των δεδομένων που θα εισαχθούν. Υπάρχουν τριών ειδών αλφαριθμητικά μορφοποίησης:

1. Ειδικά σύμβολα, τα οποία χρησιμοποιούνται για να καθορίσουν τον τύπο των εισαγόμενων δεδομένων που θα καταχωριστούν στην διεύθυνση_μνήμης
   - %c για έναν χαρακτήρα
   - %d για ακέραιο αριθμό
   - %f για πραγματικό αριθμό
2. Χαρακτήρες διαστήματος, η scanf( ) έχει την δυνατότητα να διαβάσει δύο ή περισσότερες μεταβλητές, ακολουθώντας την εξής σύνταξη:

   scanf("%d %d", 1η_διεύθυνση_μνήμης, 2η_διεύθυνση_μνήμης);

   Η scanf( ), πρόκειται να διαβάσει δύο ακέραιους αριθμούς και να τους καταχωρήσει σε δύο θέσης μνήμης που έχουν δεσμευτεί. Ωστόσο, μεταξύ των ειδικών συμβόλων (δηλαδή μεταξύ των %d) μπορεί να τοποθετηθούν ένα ή περισσότερα κενά διαστήματα τα οποία θα αγνοηθούν
3. Άλλοι χαρακτήρες. Αν κατά τη σύνταξη της scanf( ) μεταξύ των ειδικών συμβόλων τοποθετηθεί κάποιος χαρακτήρας, τότε αυτός ο χαρακτήρας υποχρεωτικά θα πρέπει να διαβαστεί και να αγνοηθεί, ειδάλλως η scanf( ) θα τερματίσει. Για παράδειγμα, υπάρχει η ακόλουθη σύνταξη της scanf( ):

   scanf("%d,%d", 1η_διεύθυνση_μνήμης, 2η_διεύθυνση_μνήμης);

   Σε αυτή την περίπτωση η καταχώρηση των αριθμών θα πρέπει να γίνει διαχωριζόμενη από ένα κόμμα π.χ. 10,20. Ο χαρακτήρας που τοποθετήθηκε μεταξύ των %d είναι υποχρεωτικό να υπάρχει κατά την καταχώρηση των τιμών.

Στο παρακάτω παράδειγμα καταχωρείται έναs πραγματικός αριθμός.

Παράδειγμα

```
int main( ){
        float x;
        scanf("%f", &x);
}
```

Στο παρακάτω παράδειγμα καταχωρούνται δύο ακέραιες τιμές διαχωριζόμενες από έναν οποιοδήποτε χαρακτήρα

Παράδειγμα

```
int main( ){
        int x,y;
        scanf("%d%*c%d", &x,&y);
}
```

Στο πιο πάνω παράδειγμα θα μπορούσε να δοθεί σαν είσοδος 10/20, το 10 θα είχε καταχωρηθεί στη μεταβλητή x, το 20 θα είχε καταχωρηθεί στη μεταβλητή y και ο χαρακτήρα / θα είχε αγνοηθεί λόγω του %*c.

3.14 Συναρτήσεις Χειρισμού Χαρακτήρων

Για την είσοδο και την έξοδο κάποιου χαρακτήρα μπορούν εναλλακτικά να χρησιμοποιηθούν οι συναρτήσεις getchar( ) και putchar( ) αντίστοιχα.

Η συνάρτηση getchar( ) περιμένει μέχρι να πληκτρολογηθεί έναs χαρακτήραs, ο οποίος καταχωρείται στη μεταβλητή που έχει δοθεί ως όρισμα στην getchar( ). Η καταχώρηση του χαρακτήρα δεν γίνεται με το πάτημα του πλήκτρου Enter. Επίσης, δεν είναι απαραίτητο να δοθεί κάποια μεταβλητή ως όρισμα στη συνάρτηση,

απλά σε αυτή την περίπτωση ο χαρακτήρας που θα πατηθεί δεν θα εκχωρηθεί πουθενά.

Η συνάρτηση putchar( ) εμφανίζει έναν χαρακτήρα στην οθόνη. Θα πρέπει να δοθεί ως όρισμα κάποια μεταβλητή χαρακτήρα ή ένας απλός χαρακτήρας. Υπάρχει η δυνατότητα να δοθεί ως όρισμα και κάποια ακέραια παράμετρος (μεταβλητή ή απλά ακέραια τιμή), σε αυτή την περίπτωση θα εμφανιστεί ο χαρακτήρας που βρίσκεται στην αντίστοιχη θέση στον πίνακα ASCII.

Παράδειγμα

```
main( ){
        char x;
        printf("Hello press any key to continue\n");
        getchar( );
        printf("Please insert a character\n");
        x=getchar( );
        printf("The character is ");
        putchar(x);
}
```

3.15 Τελεστές

Μια από τις πιο βασικές λειτουργίες ενός προγράμματος είναι η εκτέλεση μαθηματικών και λογικών πράξεων, οι οποίες εκτελούνται με τη χρήση κάποιου τελεστή, αριθμητικού ή λογικού αντίστοιχα.

Αριθμητικοί Τελεστές

Οι αριθμητικοί τελεστές φαίνονται στον πίνακα που ακολουθεί:

Τελεστής	Πράξη
-	αφαίρεση ή το αρνητικό πρόσημο
+	πρόσθεση
*	πολλαπλασιασμός
%	ακέραιο υπόλοιπο διαίρεσης (mod)
/	διαίρεση
++	αύξηση κατά ένα
--	μείωση κατά ένα

Η χρήση των αριθμητικών τελεστών μπορεί να γίνει σε όλους τους υποστηριζόμενους τύπους δεδομένων. Για παράδειγμα, αν χρησιμοποιηθεί ο τελεστής άθροισης σε δύο ακέραιες τιμές θα υπολογιστεί το αποτέλεσμα, που θα είναι και αυτό προφανώς ακέραια τιμή. Αν διαιρεθούν δύο πραγματικές τιμές το αποτέλεσμα που θα προκύψει θα είναι και αυτό πραγματικό. Ωστόσο, αξίζει να σημειωθεί πως αν διαιρεθούν δύο ακέραιες τιμές τότε το αποτέλεσμα θα είναι αυστηρά ακέραιο και αυτό, ακόμα και αν οι τιμές στις οποίες έγινε η πράξη είναι 3 και 2. Αυτό συμβαίνει γιατί ο τύπος του παραγόμενου αποτελέσματος εξαρτάται από τον τύπο των ορισμάτων που συντέλεσαν στην πράξη. Ωστόσο, αν διαιρούνταν ένας πραγματικός αριθμός με έναν ακέραιο το αποτέλεσμα θα είχε πραγματικό τύπο, γιατί η μια από τις δύο τιμές θα ήταν πραγματική και ο πραγματικός τύπος έχει μεγαλύτερη προτεραιότητα από τον ακέραιο. Ο τελεστής υπολοίπου % παράγει το υπόλοιπο μιας διαίρεσης ακεραίων. Δεν μπορεί να χρησιμοποιηθεί με τους τύπους float ή double.

Παράδειγμα

```
main( ){
        int grade1 = 15, grade2 = 16,sum;
        sum= grade1+ grade2;
        printf("%d\n", sum); // θα εμφανιστεί 31
        printf("%f\n", sum/2); // θα εμφανιστεί 15.500000
```

```
        grade1 = 1;
        grade2= 2;
        printf("%d %d\n", x /y, x % y); // θα δώσει 0 και 1
}
```

Η τελευταία printf( ) θα δώσει 0 και 1 γιατί το ½ στη διαίρεση ακεραίων είναι 0 με υπόλοιπο 1.

Τελεστής Εκχώρησης Τιμής

Η εκχώρηση τιμής σε μια μεταβλητή γίνεται με το σύμβολο =, τοποθετώντας την τιμή που βρίσκεται δεξιά του συμβόλου στην αριστερή πλευρά όπου πρέπει να βρίσκεται το όνομα μιας μεταβλητής

Παράδειγμα

```
main( ){
        int salary1, salary2, sum;
        printf("Δώσε τον μισθό για δύο μήνες : ");
        scanf("%d %d",& salary1,& salary2);
        sum = salary1+ salary2;
        printf("Το άθροισμα των μισθών ειναι %d \n",sum);
}
```

Τελεστές Αύξησης και Μείωσης κατά 1 Μονάδα

Όπως πολλές γλώσσες προγραμματισμού, έτσι και η C διαθέτει δύο αρκετά εύχρηστους τελεστές, αυτόν της αύξησης κατά μια μονάδα (++) και αυτόν της μείωσης κατά μια μονάδα (--). Η χρήση κάποιου από τους δύο τελεστές γίνεται πάνω σε μια μεταβλητή, αυξάνοντας/μειώνοντας την τιμή που διαθέτει η μεταβλητή κατά μια μονάδα. Αυτό σημαίνει ότι η πρόταση x++; είναι ακριβώς ίδια με την πρόταση x = x + 1; . Το αντίστοιχο ισχύει και για τον τελεστή μείωσης.

Και οι δύο τελεστές μπορούν να χρησιμοποιηθούν τόσο μπροστά από κάποια μεταβλητή όσο και μετά από αυτήν αλλάζοντας όμως την συμπεριφορά της εκάστοτε περίπτωσης. Για παράδειγμα:

a = 5;

b = ++a;

Σε αυτή την περίπτωση το περιεχόμενο της μεταβλητής a αυξάνεται κατά μια μονάδα και το αποτέλεσμα εκχωρείται στη μεταβλητή b. Δηλαδή πλέον η μεταβλητή b περιέχει το 6. Όμως, αν ο τελεστής αύξησης βρισκόταν μετά τη μεταβλητή (a++), τότε πρώτα θα είχε εκχωρηθεί στη μεταβλητή b το περιεχόμενο της μεταβλητής a και μετά θα είχε γίνει η αύξηση της μεταβλητής a. Δηλαδή, στη μεταβλητή b θα είχε εκχωρηθεί η τιμή 5. Και στις δύο περιπτώσεις η τιμή της μεταβλητής a θα είχε γίνει 6, η διαφορά έγκειται στο πότε θα γινόταν αυτή η αύξηση.

Παρακάτω εμφανίζεται η σειρά προτεραιότητας των αριθμητικών τελεστών.

Υψηλή προτεραιότητα	++ --
	* / %
Χαμηλή προτεραιότητα	+ -

Σε περίπτωση που σε μια πρόταση εμφανίζονται τελεστές ίδιας προτεραιότητας προηγείται ο τελεστής που βρίσκεται στα αριστερά. Επίσης, εάν πρέπει να αλλάξει η σειρά προτεραιότητας κάποιων τελεστών θα πρέπει να γίνει χρήση των παρενθέσεων. Κλασικό παράδειγμα αποτελεί η εύρεση μέσης τιμής τριών μεταβλητών (a + b + c) / 3.

Για λόγους ταχύτητας κάποιες παραστάσεις όπως η x = x + 10, όπου η αρχική μεταβλητή του αριστερού μέλους επαναλαμβάνεται αμέσως στην αρχή του δεξιού μέλους, μπορούν να γραφούν ως x += 10. Το ίδιο ισχύει και για τους υπόλοιπους τελεστές (-=, *=, /=, %=).

Παράδειγμα

```
main( ){
        int x, y;
        int temp;
        printf("Δώσε την πρώτη μετβλητή : ");
```

```
        scanf("%d",&x); //έστω ότι καταχωρείται η τιμή 5
        temp = x;
        printf("Δώσε την δύτερη μετβλητή : ");
        scanf("%d",&y); //έστω ότι καταχωρείται η τιμή 10
        x += y;
        printf("%d\n",x); //εμφανίζεται η τιμή 15
        x = temp;
        x -= y;
        printf("%d\n",x); //εμφανίζεται η τιμή -5
        x = temp;
        x *= y;
        printf("%d\n", x); //εμφανίζεται η τιμή -50
        x = temp;
        x /= y;
        printf("%d\n", x); //εμφανίζεται η τιμή -5
}
```

3.16 Μετατροπή Τύπων σε Παραστάσεις

Όταν έχουμε σταθερές και μεταβλητές διαφορετικών τύπων σε μια παράσταση, η C τις μετατρέπει όλες στον ίδιο τύπο. Ο μεταγλωττιστής της C θα μετατρέψει πράξη προς πράξη όλους τους τελεστές προς τα «επάνω» στον τύπο του μεγαλύτερου τελεστή, όπως περιγράφεται στους παρακάτω κανόνες μετατροπής τύπων:

1. Όλοι οι τύποι char και short int μετατρέπονται σε int. Όλοι οι τύποι float μετατρέπονται σε double.
2. Για όλα τα ζευγάρια τελεστών, η σειρά μετατροπής είναι η παρακάτω. Αν ο ένας από τους τελεστές είναι long double, και ο άλλος τελεστής μετατρέπεται σε long double. Αν ο ένας τελεστής είναι double, και ο άλλος τελεστής μετατρέπεται σε double. Αν ο ένας από τους τελεστές είναι long, και ο άλλος τελεστής μετατρέπεται σε long. Αν ο ένας από τους τελεστές είναι unsigned και ο άλλος τελεστής μετατρέπεται σε unsigned.

3.17 Η Τεχνική Type Casting

Υπάρχει η δυνατότητα μια παράσταση ή μια μεμονωμένη μεταβλητή να αλλάξει προσωρινά τύπο, δηλαδή να αλλάξει τύπο μόνο μέσα σε μια πρόταση, χρησιμοποιώντας την τεχνική type casting. Για παράδειγμα, αν πρέπει να βρεθεί ο μέσος όρος τριών ακέραιων μεταβλητών η πρόταση (x1 + x2 + x3) / 3 θα παράξει ακέραιο αποτέλεσμα, ακόμα και αν τιμές των μεταβλητών ήταν τέτοιες που θα έπρεπε το αποτέλεσμα να είναι πραγματικός αριθμός. Για να παραχθεί σωστά αποτέλεσμα θα μπορούσε απλά η πρόταση να επαναδιατυπωθεί ως (x1 + x2 + x3) / 3.0, δηλαδή ο παρανομαστής πλέον είναι πραγματική τιμή, επομένως το αποτέλεσμα όλης της πράξης θα είναι πραγματικό.

Θα κάνουμε μια τροποποίηση στο προηγούμενο παράδειγμα, έστω η προηγούμενη πρόταση έχει την μορφή (x1 + x2 + x3) / count_x. Σε αυτή την περίπτωση δεν μπορεί να χρησιμοποιηθεί η προηγούμενη λύση, καθώς δεν εμπλέκεται κάποιος σταθερός αριθμός. Προκειμένου να παραχθεί σωστό αποτέλεσμα θα πρέπει να δοθεί οδηγία στον μεταγλωττιστή να συμπεριφερθεί σε έναν από τους δύο όρους της παράστασης ως πραγματική τιμή, επαναδιατυπώνοντας την πρόταση ως εξής: (x1 + x2 + x3) / (float) count_x. Με αυτό τον τρόπο μόνο για τη συγκεκριμένη πρόταση η μεταβλητή count_x παύει να είναι ακέραια και θεωρείται ως πραγματική, επομένως το αποτέλεσμα όλης της πρότασης θα είναι πραγματικό.

Παράδειγμα

```
#include <stdio.h>
main(){
        float average;
        int x,y;
        printf("Δώσε δύο ακέραιες τιμές : \n ");
        scanf("%d",&x,&y);
        average = (float) ( x + y ) / 2;
        printf("%.2f\n", average);
}
```

3.18 Η οδηγία #define

Η οδηγία #define ορίζει ένα αναγνωριστικό και μια τιμή για αυτό το αναγνωριστικό, που θα χρησιμοποιείται σε όλη την έκταση του προγράμματος αντικαθιστώντας το αναγνωριστικό. Η σύνταξη της οδηγίας είναι:

#define αναγνωριστικό τιμή

Τις περισσότερες φορές η οδηγία #define χρησιμοποιείται για να οριστεί κάποια σταθερά στο πρόγραμμα. Στο παρακάτω παράδειγμα ορίζεται η σταθερά p (που είναι ίση με 3.141593) και υπολογίζεται το εμβαδόν ενός κύκλου. Ο μεταγλωττιστής της C, όπου συναντάει τη μεταβλητή p θα την αντικαθιστά με την τιμή 3.141593.

Παράδειγμα

```
#include <stdio.h>
#define p 3.141593
int main( ){
        float aktina = 20, embado;
        printf("Το εμβαδό του κύκλου είναι %f\n", aktina * p);
}
```

3.19 Τελεστές Bitwise

Οι τελεστές bitwise επιτρέπουν στη C να χειρίζεται τους ακέραιους αριθμούς σε επίπεδο bit. Σχεδόν όλοι οι τελεστές bitwise, έχουν δύο μέλη, ένα αριστερό και ένα δεξιό. Τα μέλη αυτά (μέλος – A, μέλος –B), πρέπει να είναι ακέραιες παραστάσεις, δηλαδή να αποδίδουν ακέραιες τιμές. Οι τελεστές bitwise είναι οι εξής:

& (bitwise AND), εκτελεί μια πράξη AND στα αντίστοιχα bit των δύο μελών.

& 1 → 1

1 & 0 → 0

0 & 1 → 0

0 & 0 → 0

| (bitwise OR), εκτελεί μια πράξη OR στα αντίστοιχα bit των δύο μελών.

1 | 1 → 1

1 | 0 → 1

0 | 1 → 1

0 | 0 → 0

| (bitwise XOR), εκτελεί μια πράξη XOR στα αντίστοιχα bit των δύο μελών.

1 ^ 1 → 0

1 ^ 0 → 1

0 ^ 1 → 1

0 ^ 0 → 0

~ (συμπλήρωμα), αντιστρέφει τα bit του μέλους – Α.

~1 → 0

~ 0 → 1

<< (ολίσθηση αριστερά), μετακινεί προς τα αριστερά όλα τα bit του μέλους – Α τόσες θέσεις όση η τιμή του μέλους – Β. Για παράδειγμα, x << 10, μετακινούνται τα bit της μεταβλητής x 10 θέσεις αριστερά.

>> (ολίσθηση δεξιά), μετακινεί προς τα δεξιά όλα τα bit του μέλους – Α τόσες θέσεις όση η τιμή του μέλους – Β. Για παράδειγμα, x >> 10, μετακινούνται τα bit της μεταβλητής x 10 θέσεις δεξιά.

4. Στοιχεία Λογικής & Δομή Επιλογής

4.1 Δομή Επιλογής

Στο συγκεκριμένο κεφάλαιο θα αναλυθεί η δομή επιλογής, που είναι γνωστή και ως δομή διακλάδωσης. Τις περισσότερες φορές ανάλογα με τα δεδομένα που εισάγονται στο πρόγραμμα πρέπει να εκτελείται μια διαφορετική διαδικασία, το ίδιο ενδεχομένως θα πρέπει να συμβεί ανάλογα και από το αποτέλεσμα μιας αριθμητικής πράξης. Η δομή επιλογής είναι η δομή που χρησιμοποιείται για την επιλογή μιας επεξεργασίας μεταξύ δύο ή περισσότερων διαφορετικών διαδικασιών ανάλογα με την τιμή μιας λογικής συνθήκης. Η C διαθέτει δύο εντολές που παρέχουν τη δυνατότητα εκτέλεσης προτάσεων υπό συνθήκη, δηλαδή μόνο αν πληρούνται κάποιοι όροι ή προϋποθέσεις. Είναι οι εντολές if και switch-case. Η δομή επιλογής χρησιμοποιείται με δύο διαφορετικές εκδοχές: την απλή εντολή επιλογής και τη σύνθετη εντολή επιλογής.

4.2 Απλή Εντολή Επιλογής

Η σύνταξη της εντολής είναι:

if *(συνθήκη ελέγχου)*
{
εντολή 1

εντολή 2
...
εντολή ν
}

Τα άγκιστρα έναρξης (**{**) και λήξης (**}**) του σώματος εντολών είναι απαραίτητα όταν υπάρχουν παραπάνω από μια εντολές που περιλαμβάνονται στη δομή επιλογής. Σε περίπτωση που εκτελείται μόνο μια εντολή τα άγκιστρα μπορούν να παραλειφθούν. Αν υπάρχουν παραπάνω από μια εντολές που περιλαμβάνονται στη δομή επιλογής, αλλά τα άγκιστρα έχουν παραλειφθεί, τότε θα θεωρηθεί ότι μόνο η πρώτη εντολή ανήκει στη δομή επιλογής.

Λειτουργία της εντολής

Αν η συνθήκη ελέγχου έχει τιμή ΑΛΗΘΗΣ, εκτελείται η ομάδα εντολών που βρίσκεται μεταξύ του άγκιστρου έναρξης και του άγκιστρου τερματισμού. Διαφορετικά, η συγκεκριμένη ομάδα εντολών αγνοείται και η ροή του προγράμματος μεταφέρεται στην εντολή που ακολουθεί μετά το άγκιστρο τερματισμού.

4.2.1 Λογικές Προτάσεις

Σε αυτό το σημείο θα πρέπει να επεξηγηθεί τι ακριβώς είναι η συνθήκη ελέγχου. Πρόκειται για μια λογική πρόταση που εκτελείται και οι πιθανές εκβάσεις του αποτελέσματος είναι μια από τις δύο λογικές τιμές ΑΛΗΘΗΣ ή ΨΕΥΔΗΣ. Η C δεν υποστηρίζει λογικές μεταβλητές (Boolean), για τον λόγο αυτό αν μια λογική πράξη είναι ΑΛΗΘΗΣ, τότε θα έχει τιμή διάφορη του μηδέν, ενώ αν το αποτέλεσμα είναι ΨΕΥΔΗΣ τότε θα έχει τιμή ίση με μηδέν.

Μια λογική πρόταση μπορεί να αποτελείται από σταθερές, μεταβλητές και αριθμητικές εκφράσεις που συνδέονται μεταξύ τους με τους συγκριτικούς τελεστές. Οι συγκριτικοί τελεστές μπορούν να χρησιμοποιηθούν σε όλους τους τύπους δεδομένων της C. Οι συγκριτικοί τελεστές που υποστηρίζονται από τη C δίνονται στον παρακάτω πίνακα:

Τελεστής	Σύγκριση
<	μικρότερο
>	μεγαλύτερο
<=	μικρότερο ή ίσο
>=	μεγαλύτερο ή ίσο
!=	διάφορο
= =	ίσο με

Παράδειγμα

```
#include <stdio.h>
main( ){
        int x = 10;
        if (x > 0)
                printf("Η τιμή της μεταβλητής x είναι θετικός αριθμός");
}
```

Στο προηγούμενο παράδειγμα ο λογικός έλεγχος είναι x > 0 και σε περίπτωση που το αποτέλεσμα είναι αληθές, δηλαδή η τιμή της μεταβλητής είναι μεγαλύτερη του μηδενός, θα εκτελεστεί η printf. Επειδή πρόκειται για μια εντολή που θα εκτελεστεί αν ο έλεγχος είναι αληθής, τα άγκιστρα δεν είναι απαραίτητα. Επίσης, η εντολή printf βρίσκεται ένα tab πιο μέσα, υποδεικνύοντας ότι ανήκει στο σώμα εντολών της if. Δεν θα υπήρχε κανένα απολύτως πρόβλημα αν δεν είχε χρησιμοποιηθεί το tab, ο μόνος λόγος είναι για να διευκολύνεται ο αναγνώστης του κώδικα.

Παράδειγμα

```
#include <stdio.h>
main( ){
```

```
        int x = 10;
        int y = 100;
        if (x > y)
                printf("Η μεταβλητή y είναι μεγαλύτερη της μεταβλητής x");
}
```

Στο προηγούμενο παράδειγμα ο λογικός έλεγχος θα είναι ΨΕΥΔΗΣ, καθώς η τιμή της μεταβλητής x δεν είναι μεγαλύτερη από την τιμή της μεταβλητής y. Επομένως δεν θα εκτελεστεί η printf που αποτελεί το σώμα εντολών της if.

Παράδειγμα

```
#include <stdio.h>
main( ){
        char letter1 = 'A';
        char letter2 = 'B';
        if (letter1 >  letter2)
                printf("Ο χαρακτήρας Α είναι μεγαλύτερος από τον χαρακτήρα Β");
}
```

Στο προηγούμενο παράδειγμα ο λογικός έλεγχος θα είναι ΑΛΗΘΗΣ. Υπό κανονικές συνθήκες δεν θα μπορούσε να πραγματοποιηθεί ο λογικός έλεγχος, για την ακρίβεια δεν υφίσταται καν η σύγκριση μεταξύ χαρακτήρων. Ωστόσο, ο μηχανισμός που επιτρέπει να πραγματοποιηθεί η σύγκριση επιτυχώς είναι ο πίνακας κωδικοποίησης ASCII. Πιο συγκεκριμένα ο χαρακτήρας Α βρίσκεται στη θέση 65 του πίνακα, ενώ ο Β στη θέση 66, επομένως στην πραγματικότητα γίνεται σύγκριση μεταξύ των θέσεων στις οποίες βρίσκονται οι χαρακτήρες και το 65 είναι μικρότερος αριθμός από το 66, άρα και ο χαρακτήρας αυτής της θέσης είναι μικρότερος.

Παράδειγμα

```
# include <stdio.h>
main( ){
        int a = 25, b = 25;
        if (a == b)
        printf("Οι δύο μεταβλητές είναι ίσες");
}
```

4.2.2 Λογικές Πράξεις

Υπάρχουν τρεις λογικές πράξεις οι οποίες μπορούν να πραγματοποιηθούν μεταξύ λογικών προτάσεων, κάθε λογική πράξη εκτελείται με τους λογικούς τελεστές ! (Άρνηση - Not), && (Σύζευξη - And), ll (Διάζευξη - Or).

Άρνηση

Η άρνηση μιάς λογικής πρότασης είναι ΑΛΗΘΗΣ, όταν η πρόταση είναι ΨΕΥΔΗΣ, αντίστοιχα η άρνηση μιας λογικής πρότασης είναι ΨΕΥΔΗΣ, όταν η πρόταση είναι ΑΛΗΘΗΣ.

Παράδειγμα

Η πρόταση ! (10 > 200) δίνει αποτέλεσμα 1 (ΑΛΗΘΗΣ) αφού η πρόταση (10 > 200) δίνει αποτέλεσμα 0 (ΨΕΥΔΗΣ). Δηλαδή η άρνηση αντιστρέφει το αποτέλεσμα

Σύζευξη

Η σύζευξη δύο λογικών προτάσεων έχει τιμή ΑΛΗΘΗΣ, όταν και οι δύο προτάσεις έχουν τιμή ΑΛΗΘΗΣ. Αν μία από τις προτάσεις έχει τιμή ΨΕΥΔΗΣ, τότε το αποτέλεσμα της σύζευξης έχει τιμή ΨΕΥΔΗΣ.

Παράδειγμα

Η πρόταση (5 < 15) && (60 > 40) δίνει αποτέλεσμα 1 (ΑΛΗΘΗΣ) αφού και η πρόταση (5 < 15) δίνει αποτέλεσμα 1 (ΑΛΗΘΗΣ) και η παράσταση (60 > 40).

Διάζευξη

Η διάζευξη δύο λογικών προτάσεων έχει τιμή ΑΛΗΘΗΣ όταν μία τουλάχιστον από τις δύο προτάσεις έχει τιμή ΑΛΗΘΗΣ. Προκειμένου το αποτέλεσμα της διάζευξης να είναι ΨΕΥΔΕΣ θα πρέπει όλες οι προτάσεις να έχουν τιμής ΨΕΥΔΗΣ.

Παράδειγμα

Η πρόταση (10 > 20) || (100 < 200) δίνει αποτέλεσμα 1 (ΑΛΗΘΗΣ) αφού η παράσταση (10 > 20) δίνει αποτέλεσμα 0 (ΨΕΥΔΗΣ) αλλά η παράσταση (100 < 200) δίνει αποτέλεσμα 1 (ΑΛΗΘΗΣ).

Τα αποτελέσματα των λογικών πράξεων δίνονται συγκεντρωτικά στον παρακάτω πίνακα που είναι γνωστός και ως πίνακας αληθείας λογικών πράξεων.

ΛΟΓΙΚΗ ΜΕΤΑΒΛΗΤΗ	**X**	**Y**	**! X**	**X && Y**	**X || Y**
ΤΙΜΗ	ΑΛΗΘΗΣ	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ	ΑΛΗΘΗΣ
	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ
	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ
	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ

4.2.3 Ιεραρχία Τελεστών στις Λογικές Πράξεις

1. Ο τελεστής άρνησης ! έχει την υψηλότερη προτεραιότητα.
2. Οι αριθμητικοί τελεστές (*, /, %, +, -) έχουν μεγαλύτερη προτεραιότητα από τους τελεστές σύγκρισης.

3. Οι τελεστές σύγκρισης (>, >=, <, <=) έχουν όλοι την ίδια προτεραιότητα.
4. Την ακριβώς επόμενη προτεραιότητα έχουν οι τελεστές ισότητας (==, !=).
5. Οι λογικοί τελεστές (&&, ||) έχουν τη χαμηλότερη προτεραιότητα. Οι προτάσεις που περιλαμβάνουν τους συγκεκριμένους τελεστές υπολογίζονται από τα αριστερά προς τα δεξιά.

Άσκηση

Ένας αθλητής σε έναν αγώνα είχε τρεις προσπάθειες στο άλμα επί κοντώ. Να γραφεί πρόγραμμα το οποίο:

α. θα διαβάζει τις τιμές των προσπαθειών
β. θα υπολογίζει και θα εμφανίζει τη μέση τιμή των προσπαθειών
γ. θα εμφανίζει το μήνυμα «Προκρίθηκε», αν η παραπάνω μέση τιμή είναι μεγαλύτερη των 8 μέτρων.

```
#include <stdio.h>
int main()
{
        float a, b, c, mo;
        printf("Δώσε τις τιμές των τριών προσπαθειών : ");
        scanf("%f %f %f",&a,&b,&c);
        mo = (a + b + c)/3;
        printf("Η μέση τιμή των τριών επιδόσεων είναι %.2f\n",mo);
        if (mo > 8)
        printf("Προκρίθηκε.\n");
        return 0;
}
```

4.3 Σύνθετη Εντολή Επιλογής

Η σύνθετη εντολή επιλογής προσθέτει κάποιες παραπάνω δυνατότητες στην εντολή if. Η σύνθετη εντολή επιλογής έχει τη μορφή **if – else** και χρησιμοποιείται όταν υπάρχουν δύο ενδεχόμενες καταστάσεις εκ των οποίων μόνο μια πρέπει να εκτελεστεί. Η κατάσταση που θα εκτελεστεί καθορίζεται από τη συνθήκη ελέγχου που βρίσκεται στην if όπως είχαμε δει και στην απλή εντολή επιλογή. Η σύνταξη της εντολής είναι:

if *(λογική συνθήκη)*

ομάδα εντολών 1

else

ομάδα εντολών 2

Λειτουργία της εντολής

Σε αυτή τη μορφή η εντολή if ελέγχει τη λογική συνθήκη και αν είναι ΑΛΗΘΗΣ, τότε εκτελείται η *ομάδα εντολών 1*, σε διαφορετική περίπτωση εκτελείται η *ομάδα εντολών 2*. Σε κάθε περίπτωση θα εκτελεστεί μόνο μια ομάδα εντολών και έπειτα το πρόγραμμα συνεχίζει με την εκτέλεση της επόμενης εντολής που βρίσκεται αμέσως μετά τη δομή επιλογής.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int x;
        scanf("%d",&x);
        if (x > 0)
                printf("Η τιμή της μεταβλητής x είναι θετικός αριθμός");
```

```
        else
                printf("Η τιμή της μεταβλητής x δεν είναι θετικός αριθμός");
        return 0;
}
```

Άσκηση

Να γραφεί πρόγραμμα που θα εμφανίζει αν σε μια δοσοληψία υπήρχε κέρδος ή όχι.

```
main (void){

        float cost, sell;
        printf("Δώσε τιμή κόστους και τιμή πώλησης\n");
        scanf(«%f%f», &cost, &sell);
        if (sell > cost)
                printf("Η συμφωνία ήταν κερδοφόρα");
        else
                printf("Η συμφωνία δεν ήταν κερδοφόρα ");
}
```

4.4 Εμφωλευμένες Δομές Επιλογής

Το σώμα των εντολών που εκτελείται στην if ή στην else μπορεί να περιέχει οποιαδήποτε υποστηριζόμενη εντολή της C, αυτό σημαίνει ότι θα μπορούσε να περιλαμβάνεται μια ακόμη δομή επιλογής. Σε αυτή την περίπτωση λέμε πως η δομή επιλογής που βρίσκεται στο σώμα εντολών μιας άλλη δομής επιλογής είναι εμφωλευμένη. Οι εμφωλευμένες δομές επιλογής χρησιμοποιούνται όταν η διάκριση δύο καταστάσεων και η εκτέλεση μιας εξ' αυτών μπορεί να οδηγήσει σε επιμέρους διάκριση.

Παράδειγμα

Να γραφεί πρόγραμμα το οποίο ζητάει την ηλικία ενός ατόμου και το φύλο του. Σε περίπτωση που η ηλικία είναι άνω των 18 και το φύλο 'Α' (Άνδρας), να εμφανίζεται μήνυμα «Υποχρεωτική Στράτευση».

```
int main( ){
        int age;
        char gender;
        printf ("Δώσε την ηλικία και φύλο του ατόμου:\n ");
        scanf("%d%c",&age, &gender);
        if (age >= 18) //διάκριση 1ης κατάστασης
        {
                if (gender == 'A') // επιμέρους διάκριση της 1ης κατάστασης
                        printf("Υποχρεωτική Στράτευση");
        }
}
```

Το προηγούμενο παράδειγμα χρησιμοποιεί μια εμφωλευμένη δομή επιλογής. Εναλλακτικά θα μπορούσε να έχει αποφευχθεί η εμφωλευμένη δομή επιλογής και να χρησιμοποιηθεί μια σύνθετη πρόταση και να διατυπωθεί ως εξής:

```
#include <stdio.h>
int main( )
{
        int age;
        char gender;
        printf ("Δώσε την ηλικία του ατόμου:\n ");
        scanf("%d",&age);
        printf ("Δώσε το φύλο του ατόμου:\n ");
        scanf("%c",&gender);
```

```
        if (age >= 18  && gender == 'A')
                printf("Υποχρεωτική Στράτευση");
        return 0;
}
```

Σε περιπτώσεις που μια κατάσταση μπορεί να διακριθεί σε περισσότερες από δύο επιμέρους καταστάσεις χρησιμοποιείται μια άλλη σύνταξη της if η οποία είναι η if - else if – else. Η σύνταξη της εντολής είναι:

if (συνθήκη ελέγχου 1)

ομάδα εντολών 1

else if (συνθήκη ελέγχου 2)

ομάδα εντολών 2

else if (συνθήκη ελέγχου 3)

ομάδα εντολών 3

…

else

ομάδα εντολών ν

Λειτουργία της εντολής

Στην εντολή if - else if – else υπάρχουν παραπάνω από μια συνθήκη ελέγχου, το πρόγραμμα ξεκινάει ελέγχοντας την πρώτη συνθήκη και αν αυτή είναι ΑΛΗΘΗΣ τότε εκτελείται η *ομάδα εντολών 1*, και δεν ελέγχεται καμία από τις υπόλοιπες. Αν όμως ο έλεγχος της πρώτης συνθήκης είναι ΨΕΥΔΗΣ, τότε ελέγχεται η δεύτερη συνθήκη κ.ο.κ. Αν δεν βρεθεί κανένας έλεγχος συνθήκης ΑΛΗΘΗΣ, τότε εκτελείται η *ομάδα εντολών n* που ανήκει στην else.

Παράδειγμα

```
int main( ){
        int x;
        printf("Δώσε έναν ακέραιο αριθμό: ");
        scanf("%d",&x);
        if (x > 0)
                printf("Ο αριθμός είναι θετικός.\n");
        else if (x < 0)
                printf("Ο αριθμός είναι αρνητικός.\n");
        else
                printf("Ο αριθμός είναι το μηδέν.\n");
}
```

Άσκηση

Δίνονται δύο αριθμοί από το πληκτρολόγιο. Να βρεθεί και να εμφανιστεί στην οθόνη ο μεγαλύτερος από αυτούς.

```
main( ){
        int a, b, max;
        printf("Δώσε δύο αριθμούς : ");
        scanf("%d %d",&a,&b);
        if (a == b)
                printf("Οι αριθμοί είναι ίσοι");
        else
        {
                if ( a < b)
                max = b;
                else
                max = a;
                printf("Μεγαλύτερος είναι ο %d\n",max);
        }
}
```

Άσκηση

Να φτιάξετε πρόγραμμα που να διαβάζει τον μέσο όρο βαθμολογίας (ΜΟ) ενός σπουδαστή και αν είναι σωστός (0 <= ΜΟ <= 10) να εμφανίζει:

- 'FAIL' αν 0 <= ΜΟ <5
- 'GOOD' αν 5 <= ΜΟ < 8
- 'EXCELLENT' αν 8 <= ΜΟ <10

```
int main (void){
        float x;
        printf(«Dwse ton meso oro tou spoudasti\n»);
        scanf(«%f», &x);
        if (x >= 0 && x < 5)
                printf(«Fail»);
        else if (x >= 5 && x < 8)
                printf(«Good»);
        else if (x >= 8 && x <= 10)
                printf(«Excellent»);
}
```

4.5 Η Εντολή Swtich - Case

Μια εναλλακτική προσέγγιση επίλυσης προβλημάτων που διαθέτουν διακριτές περιπτώσεις αποτελεί η εντολή switch – case, η οποία ελέγχει αν η τιμή μιας παράστασης ισούται με συγκεκριμένες σταθερές τιμές. Η σύνταξη της εντολής είναι:

```
switch (παράσταση)
{
        case  τιμή 1 : ομάδα εντολών 1;
                break;
        case  τιμή 2 : ομάδα εντολών 2;
```

```
            break;
        .....
        case τιμή v : ομάδα_εντολών v;
            break;
        default : ομάδα εντολών default;
            break;
}
```

Λειτουργία της εντολής

Αρχικά γίνεται σύγκριση της παράστασης με τις τιμές που έχουν τα διάφορα cases. Στην πρώτη case που θα βρεθεί να υπάρχει ταύτιση παράστασης και τιμής της case, θα εκτελεστούν οι προτάσεις της συγκεκριμένης case μέχρι και την πρώτη εντολή break που θα συναντηθεί.

Η ύπαρξη της εντολής break είναι απαραίτητη σε κάθε case, καθώς η εκτέλεση της εντολής διακόπτει τη συγκεκριμένη δομή επιλογής μεταφέροντας την εκτέλεση του προγράμματος στην πρώτη εντολή μετά τη δομή switch – case και παραλείποντας όλες τις υπόλοιπες εντολές που διαθέτουν τα υπόλοιπα cases.

Οι εντολές της default εκτελούνται σε περίπτωση που δεν γίνει ταύτιση της παράστασης με καμία από τις σταθερές των cases, δηλαδή έχει αντίστοιχη συμπεριφορά με αυτή της else.

Θα πρέπει να δοθεί προσοχή στην άνω και κάτω τελεία (:) που χρησιμοποιείται μετά τη σταθερά του εκάστοτε case, καθώς και στο γεγονός ότι οι σταθερές μπορεί να είναι ακέραιοι ή χαρακτήρες, αλλά όχι συμβολοσειρές ή πραγματικοί αριθμοί.

Στο επόμενο παράδειγμα ζητείται ένας αριθμός και εμφανίζεται μήνυμα σχετικά με το αν είναι άρτιος ή περιττός (χρησιμοποιείται το mod).

Παράδειγμα

```
main( )
{
        int num;
        printf("Δώσε έναν αριθμό : ");
        scanf("%d",&num);
        switch (num % 2)
        {
                case 0 :printf("Ο αριθμός είναι άρτιος.\n");
                        break;
                case 1 :printf("Ο αριθμός είναι περιττός.\n");
                        break;
        }
}
```

Άσκηση

Να γραφεί πρόγραμμα το οποίο θα υπολογίζει το επίδομα παιδιών ενός μισθωτού σαν ποσοστό του βασικού του μισθού ως εξής: για ένα παιδί →5%, για δύο παιδιά →10%, από το τρίτο παιδί και πάνω → +6% για κάθε παιδί.

```
#include <stdio.h>
int main (void){
        float salary,epidoma;
        int kids;
        printf(«Please insert your salary\n»);
        scanf(«%f»,&salary);
        printf(«How many kids do you have\n»);
        scanf(«%d»,&kids);
        switch(kids){
                case 1: epidoma=salary*0.05;
                                break;
```

```
                case 2: epidoma=salary*0.1;
                                break;
                case 3: epidoma=salary*(kids*0.6);
                                break;
                default: break;
        }
        printf(«to epidoma einai %f»,epidoma);
}
```

Άσκηση

Να γραφεί πρόγραμμα το οποίο θα διαβάζει από τον χρήστη έναν ακέραιο αριθμό και θα εμφανίζει με μήνυμα ποιο είναι το υπόλοιπο της ακέραιης διαίρεσης με το 3.

```
main (void){
        int num;
        printf(«Please insert the number\n»);
        scanf(«%d»,&num);
        switch( num%3 ){
                case 0: printf(«The remaining is 0»);
                                break;
                case 1: printf(«The remaining is 1»);
                                break;
                case 2: printf(«The remaining is 2»);
                                break;
                case 3: printf(«The remaining is 3»);;
                                break;
                default: break;}
}
```

Άσκηση

Να γραφεί πρόγραμμα το οποίο θα διαβάζει έναν ακέραιο αριθμό από 1 μέχρι 5 και θα εμφανίζει την αντίστοιχη εργάσιμη ημέρα της εβδομάδας (1 για Δευτέρα, 2 για Τρίτη κ.ο.κ.).

```
main( ){
        int choise;
        printf("Δώσε έναν ακέραιο αριθμό (1 – 5) : ");
        scanf("%d",&choise);
        switch (choise){
                case 1 : printf("Δευτέρα\n");
                        break;
                case 2 : printf("Τρίτη\n");
                        break;
                case3 : printf("Τετάρτη\n");
                        break;
                case4 : printf("Πέμπτη\n");
                        break;
                case5 : printf("Παρασκευή\n");
                        break;}
}
```

4.6 Ο τελεστής ?

Ο τελεστής ? αποτελεί έναν εναλλακτικό τρόπο ελέγχου λογικών παραστάσεων, αντικαθιστώντας ορισμένες φορές την πρόταση if – else. Η σύνταξη της εντολής είναι:

λογική_παράσταση ? παράσταση_1 : παράσταση_2

Αρχικά ελέγχεται η λογική παράσταση και σε περίπτωση που είναι αληθής εκτελείται η παράσταση_1 και το αποτέλεσμα αποτελεί το αποτέλεσμα όλης της πα-

ράστασης, ειδάλλως εκτελείται η παράσταση_2 και το αποτέλεσμα αποτελεί το αποτέλεσμα όλης της παράστασης.

Στο επόμενο παράδειγμα, ζητείται ένας ακέραιος αριθμός και υπολογίζεται η απόλυτη τιμή του.

Παράδειγμα

```
#include <stdio.h>
main( )
{
        int absolute_value, number ;
        printf("Δώσε έναν ακέραιο αριθμό : ");
        scanf("%d",&number);
        absolute_value = (number > 0) ? number : -number;
        printf("Η απόλυτη τιμή είναι %d\n", absolute_value);
}
```

Στο παραπάνω παράδειγμα θα μπορούσε να έχει χρησιμοποιηθεί η εντολή if – else.

```
#include <stdio.h>
main( )
{
        int absolute_value, number ;
        printf("Δώσε έναν ακέραιο αριθμό : ");
        scanf("%d",&number);
        if ( number > 0 )
                absolute_value = number;
        else
                absolute_value = -number;
        printf("Η απόλυτη τιμή είναι %d\n", absolute_value);
}
```

Άσκηση

Να γραφεί πρόγραμμα που θα ζητάει δύο ακέραιους αριθμούς και με τη χρήση του τελεστή ?, θα υπολογίζεται ο μεγαλύτερος αριθμός από αυτούς που δόθηκαν.

```
#include <stdio.h>
int main()
{
        int x, y, max;
        printf("Δώσε δύο ακέραιους αριθμούς : ");
        scanf("%d %d", &x, &y);
        max = (x > y) ? x : y;
        printf("Ο μεγαλύτερος αριθμός είναι ο %d\n", max);
        return 0;
}
```

4.7 Ασκήσεις

1. Να γραφεί πρόγραμμα το οποίο θα διαβάζει 5 αριθμούς από τον χρήστη και θα εμφανίζει αν οι άρτιοι αριθμοί που έδωσε ήταν περισσότεροι ή λιγότεροι από τους περιττούς.
2. Να γίνει πρόγραμμα το οποίο θα διαβάζει μια θερμοκρασία (πραγματικός αριθμός) και έναν χαρακτήρα. Στη συνέχεια το πρόγραμμα θα ελέγχει αν ο χαρακτήρας είναι ο 'C', ο 'F' ή οτιδήποτε άλλο. Στη πρώτη περίπτωση θα μετατρέπει τη θερμοκρασία σε βαθμούς Φαρενάιτ και θα την τυπώνει, στη δεύτερη θα τη μετατρέπει σε βαθμούς Κελσίου και θα την τυπώνει, ενώ στην τελευταία θα εμφανίζει μήνυμα λάθους.F=1.8*C+32
3. Να γίνει πρόγραμμα το οποίο θα ζητά τον αριθμό των ωρών εργασίας ενός υπαλλήλου με ωριαία αποζημίωση 7.5€ και θα εμφανίζει τις συνολικές απο-

δοχές του. Αν οι ώρες εργασίας είναι περισσότερες από 40, τότε η αποζημίωση κάθε επιπλέον ώρας θα προσαυξάνεται κατά 25%.

4. Στο τμήμα φυσικής ενός πανεπιστημίου μιας χώρας κάποιος εισάγεται αν ο βαθμός του στη φυσική είναι μεγαλύτερος από 17 ή αν ο μέσος όρος στη φυσική, μαθηματικά, πληροφορική είναι μεγαλύτερος από 16 (με άριστα το 20) . Να γραφεί πρόγραμμα που να διαβάζει τους βαθμούς ενός μαθητή στη φυσική, μαθηματικά και πληροφορική, να εμφανίζει κατάλληλο μήνυμα στην περίπτωση που πληρούνται ή δεν πληρούνται οι όροι εισαγωγής.

5. Να γραφεί πρόγραμμα το οποίο να ζητά το εισόδημα ενός φορολογούμενου και να υπολογίζει τον φόρο σύμφωνα με τα εξής:

α. Αν το εισόδημα είναι κάτω από 10500 τότε ο φόρος είναι 0.

β. Για εισοδήματα από 10500 ως 13500 τότε ο φόρος είναι 15% .

γ. Για τα ποσά πάνω από 13500 τότε ο φόρος είναι 29% .

6. Να γραφεί πρόγραμμα το οποίο να δέχεται ένα έτος και να αποφασίζει αν είναι δίσεκτο ή όχι. Δίσεκτο είναι ένα έτος αν διαιρείται ακριβώς με το 4. Τα έτη όμως που διαιρούνται με το 100 δεν είναι δίσεκτα εκτός και αν διαιρούνται και με το 400.

7. Σε κάποια εξεταστική ένα γραπτό αξιολογείται από δύο βαθμολογητές στην κλίμακα [0, 100]. Αν η διαφορά μεταξύ των βαθμολογιών είναι μικρότερη ή ίση των 20 μονάδων, ο τελικός βαθμός είναι ο μέσος όρος των δύο βαθμολογιών. Αλλιώς το γραπτό δίνεται για αναβαθμολόγηση σε τρίτο βαθμολογητή και ο τελικός βαθμός προκύπτει από τον μέσο όρο των τριών βαθμολογιών. Να γίνει πρόγραμμα το οποίο θα διαβάζει τους βαθμούς των 2 (ή 3) βαθμολογητών και θα τυπώνει τον τελικό βαθμό του γραπτού στην κλίμακα [0, 20]

5. Δομές Επανάληψης

Οι δομές επανάληψης δίνουν τη δυνατότητα σε ένα σύνολο εντολών να επαναλαμβάνεται μέχρι να ικανοποιηθεί μια συνθήκη. Η C υποστηρίζει τρεις δομές επανάληψης, που είναι οι while, do - while και for.

5.1 Η Δομή Επανάληψης While

Η συγκεκριμένη δομή επανάληψης εκτελεί το σύνολο των προτάσεων, απλών ή σύνθετων, που περιλαμβάνει για όσο μια λογική συνθήκη είναι αληθής. Όταν η συνθήκη γίνει ψευδής, η εκτέλεση του προγράμματος μεταβαίνει στην πρώτη εντολή μετά τη δομή επανάληψης. Η σύνταξη της εντολής είναι η εξής:

while *(συνθήκη)*

{

Εντολή 1;
Εντολή 2;
} Σώμα Εντολών του βρόχου, δηλαδή της δομή επανάληψης

.....

Εντολή N;

}

Παράδειγμα

```
#include <stdio.h>
int main( ){
        int number  = 1, count = 0;
        while (number != 0)
        {
                printf("Δώσε έναν ακέραιο αριθμό (0 για τερματισμό) : ");
                scanf("%d",&number);
                count++;
        }
        printf("Το πλήθος των αριθμών που δόθηκαν είναι %d.\n",count);
        return 0;
}
```

Το πρόγραμμα του παραδείγματος ζητάει συνεχόμενα τιμές από τον χρήστη μέχρι να δοθεί η τιμή μηδέν. Όταν δοθεί κάποια στιγμή το μηδέν και πάψει να ισχύει η συνθήκη του βρόχου επανάληψης, τότε τερματίζει ο βρόχος και εμφανίζεται το πλήθος των τιμών που δόθηκαν. Το πλήθος των επαναλήψεων που θα εκτελέσει ο βρόχος είναι άγνωστο και εξαρτάται αποκλειστικά από τον χρήστη. Θεωρητικά η επανάληψη θα μπορούσε να εκτελείται επ' άπειρον. Η μεταβλητή count, όπως ειπώθηκε, μετράει τους αριθμούς που δόθηκαν και ουσιαστικά έχει ευθεία σύνδεση με το πλήθος των επαναλήψεων που εκτελέστηκαν.

Η συνθήκη της while βρίσκεται στην αρχή του βρόχου, αυτό σημαίνει ότι ενδέχεται ο βρόχος να μην κάνει καμία επανάληψη αν ο πρώτος έλεγχος της συνθήκης είναι ψευδής. Στο παράδειγμα που δόθηκε αν δεν είχε αρχικοποιηθεί η μεταβλητή number με την τιμή 1, ο βρόχος το πιο πιθανό είναι να μην είχε εκτελεστεί ούτε μια φορά. Αυτό συμβαίνει γιατί μια μεταβλητή που έχει δηλωθεί αλλά δεν έχει αρχικοποιηθεί με καμία τιμή είναι πολύ πιθανό να περιέχει το 0, αυτό όμως δεν συμβαίνει πάντα, μπορεί να υπάρχει και κάποια άλλη τυχαία τιμή. Λόγω αυτού λέμε πως μια μη αρχικοποιημένη μεταβλητή έχει απροσδιόριστο περιεχόμενο.

Παράδειγμα

```
#include <stdio.h>
main()
{
        int sum=0, count=0, number;
        while (sum <= 100)
        {
                printf("Δώσε το πλήθος των αριθμών : ");
                scanf(«%d»,&number);
                sum += number;
                count++;
        }
        printf("Ο μέσος όρος των αριθμών που έδωσες είναι %f \n", sum /
        (float) count);
}
```

Το παραπάνω πρόγραμμα υπολογίζει τον μέσο όρο των αριθμών που δόθηκαν. Προκειμένου να εμφανιστεί σωστά ο μέσο όρος, πρέπει να χρησιμοποιηθεί η τεχνική type casting.

Άσκηση

Να γραφεί πρόγραμμα το οποίο να κάνει τα εξής:

1. Να διαβάζει συνέχεια χαρακτήρες από το πληκτρολόγιο
2. Να σταματάει μόλις πληκτρολογηθούν δύο αστεράκια
3. Να εμφανίζει πόσοι χαρακτήρες μεσολάβησαν μεταξύ του πρώτου και δεύτερου αστεριού.

```
#include <stdio.h>
main(void){
        char c;
```

```
        int counter = 0, plithos = 0;
        while (counter < 2)
        {
                printf(«Δώσε χαρακτήρα\n»);
                scanf(«%c»,&c);
                if (c=='*')
                        counter++;
                if (counter==1 && c!='*')
                        plithos++;
        }
        printf("Το πλήθος των χαρακτήρων που μεσολάβησαν ήταν %d",plithos);
}
```

Άσκηση

Σε έναν λογαριασμό καταθέσεων τοποθετήσατε αρχικό κεφάλαιο 1000 Ευρώ. Με δεδομένο ότι κάθε χρόνο το ποσό αυξάνεται κατά 3%, να υπολογιστεί σε πόσα χρόνια θα έχετε διπλασιάσει το αρχικό σας κεφάλαιο. Στο τέλος του αλγορίθμου να εμφανιστεί επεξηγηματικό μήνυμα.

```
#include <stdio.h>
main (void){
        int years=0;
        float capital=1000;

        while(capital<2000)
        {
                capital=capital*1.03;
                years++;
        }
        printf("Το κεφάλαιο θα διπλασιαστεί μετά από %d χρόνια",years);
}
```

5.2 Η Δομή Επανάληψης Do While

Η δομή επανάληψης do while είναι παρόμοια με τη while, η διαφορά των δύο δομών έγκειται στο ότι η συνθήκη της do while βρίσκεται στο τέλους του βρόχου σε αντίθεση με τη while που η συνθήκη βρίσκεται στην αρχή. Λόγω αυτής της διαφοροποίησης είναι δεδομένο πως η do while θα εκτελεστεί τουλάχιστον μια φορά, κάτι που για τη while δεν είναι δεδομένο. Η σύνταξη της do while είναι:

Σε αντίθεση με τον βρόχο while που ελέγχει τη συνθήκη στην αρχή του, ο βρόχος do – while ελέγχει τη συνθήκη στο τέλος του. Αυτό το χαρακτηριστικό προκαλεί πάντοτε την εκτέλεση του βρόχου do – while τουλάχιστον μία φορά. Η γενική μορφή της επαναληπτικής δομής do – while είναι:

```
do
{
        Εντολή 1;   } Σώμα Εντολών του βρόχου,
        Εντολή 2;   } δηλαδή της δομή επανάληψης
        .....
        Εντολή N;
} while (συνθήκη) ;
```

Μια από τις πιο κοινές περιπτώσεις χρήσης της do while είναι η ανάγνωση μιας επιλογής του χρήστη από ένα μενού επιλογών. Σε αυτή την περίπτωση η δομή επανάληψης πρέπει να εκτελεστεί τουλάχιστον μια φορά και λειτουργεί ως ένας μηχανισμός ελέγχου. Οι επιλογές του μενού είναι συγκεκριμένες, αυτό σημαίνει πως αν ο χρήστης δώσει μια τιμή που δεν παρέχεται από το μενού, ο βρόχος της do while θα ξαναεκτελεστεί ζητώντας από τον χρήστη να ξαναδώσει τιμή. Αυτό θα επαναλαμβάνεται μέχρι ο χρήστης να δώσει κάποια τιμή που αντιστοιχίζεται σε αυτές που προσφέρει το μενού.

Παράδειγμα

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

```
int main()
{
int choice;
        do
        {
                printf("1. Εισαγωγή\n");
                printf("2. Διόρθωση\n");
                printf("3. Διαγραφή\n");
                printf("4. Έξοδος\n");
                printf("Δώσε την επιλογή σου (1 - 4) : ");
                scanf(«%d»,&choice);
        } while (choice < 1 || choice > 4);
        printf("Επέλεξες την περίπτωση %d\n",choice);
        return 0;
}
```

Η προηγούμενη λειτουργία της do while, μπορεί να γενικευθεί και να θεωρηθεί ότι η εντολή do while μπορεί να χρησιμοποιηθεί για έλεγχο εγκυρότητας, δηλαδή να εξαναγκάζεται ο χρήστης να εισάγει αριθμό ή χαρακτήρα συγκεκριμένου πεδίου τιμών ή εύρους. Στο παρακάτω πρόγραμμα ζητείται να καταχωρηθεί ο βαθμός (σε εικοσαβάθμια κλίμακα) ενός μαθητή στα μαθηματικά.

Παράδειγμα

```
#include  <stdio.h>
main(){
        int grade;
        do {
                printf("Εισάγεται τον βαθμό στα μαθηματικά ( 0 – 20)\n");
                scanf(«%d»,&grade);
        } while (grade < 0 || grade > 20);
```

```
        printf("Ο βαθμός που καταχωρήθηκε είναι %d\n", grade);
}
```

Στο προηγούμενο παράδειγμα όσο πληρείται η λογική παράσταση, δηλαδή όσο ο χρήστης καταχωρεί τιμή που δεν είναι έγκυρη (μεγαλύτερη από 20 ή μικρότερη από 0), ο βρόχος θα συνεχίσει να εκτελείται μέχρι να δοθεί τιμή εντός των αποδεκτών ορίων. Ο έλεγχος εγκυρότητας θα μπορούσε να πραγματοποιηθεί και με την εντολή while.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int grade;
        printf("Εισάγετε τον βαθμό στα μαθηματικά ( 0 – 20)\n");
        scanf(«%d»,&grade);

        while (grade < 0 ll grade > 20)
        {
                printf("Εισάγετε τον βαθμό στα μαθηματικά ( 0 – 20)\n");
                scanf("%d",&grade);
        }
        printf("Ο βαθμός που καταχωρήθηκε είναι %d\n", grade);
        return 0;
}
```

Προφανώς σε περιπτώσεις που απαιτείται έλεγχος εγκυρότητας η εντολή do while είναι πιο εύχρηστη.

Άσκηση

Να γραφεί πρόγραμμα το οποίο θα ζητάει ένα ζεύγος ακέραιων αριθμών από τον χρήστη και θα εμφανίζει το μέσο όρο τους. Το πρόγραμμα θα σταματάει όταν και οι δύο αριθμοί που θα δοθούν θα είναι 0.

```
#include <stdio.h>
int main(void)
{
        int x,y;
        float mo;
        printf("Δώσε δύο ακέραιες τιμές\n");
        do
        {
                scanf(«%d %d»,&x,&y);
                mo=(x+y)/2.0;
                printf("ο μέσος όρος είναι:%f\n",mo);
        }while (x!=0 && y!=0);
        return 0;
}
```

5.3 Η Δομή Επανάληψης For

Η συγκεκριμένη δομή χρησιμοποιείται σε περιπτώσεις που το πλήθος των επαναλήψεων που θα πραγματοποιηθούν είναι ένας πεπερασμένος αριθμός, δηλαδή το πλήθος των επαναλήψεων έχει μια συγκεκριμένη τιμή. Σε αντίθεση με τη while ή do while που χρησιμοποιούνται όταν το πλήθος των επαναλήψεων δεν είναι περασμένος αριθμός. Η σύνταξη της εντολής είναι η εξής:

for *(πρόταση αρχικοποίησης - αρχική τιμή; λογικός έλεγχος – τελική τιμή; ρυθμός αύξησης)*

Η εντολή for περιλαμβάνει τρεις παραστάσεις προκειμένου να οριστεί το πλήθος των επαναλήψεων που θα εκτελεστούν. Γενικά η φιλοσοφία της δομής for είναι η χρήση μιας βοηθητικής μεταβλητής που θα ξεκινήσει από μια συγκεκριμένη τιμή (1η παράσταση) και θα καταλήξει σε μια άλλη (2η παράσταση). Σε κάθε επανάληψη η βοηθητική μεταβλητή θα έχει έναν συγκεκριμένο ρυθμός αύξησης (3η παράσταση). Με αυτό τον τρόπο καθορίζεται αυστηρά το πλήθος των επαναλήψεων που θα πραγματοποιηθούν.

Λειτουργία της εντολής

Η πρόταση αρχικοποίησης εκτελείται μόνο μια φορά πριν την πρώτη επανάληψη της δομής, συνήθως χρησιμοποιείται μια βοηθητική μεταβλητή i που αρχικοποιείται με την τιμή 0. Η δεύτερη παράσταση, που είναι ο λογικός έλεγχος, εκτελείται πριν από κάθε επανάληψη της for και όσο είναι ΑΛΗΘΗΣ επιτρέπει στη δομή να εκτελέσει μια ακόμα επανάληψη. Αφού εκτελεστεί το σώμα εντολών της δομής εκτελείται η 3η παράσταση που ορίζει τον ρυθμό αύξησης της βοηθητικής μεταβλητής.

Παράδειγμα

```
main(){
        int i, max=10;
        for (i = 0 ; i < max ; i++)
                printf("Αριθμός επανάληψης %d\n», i + 1);
}
```

Οι παρενθέσεις που ακολουθούν τη for περιέχουν τρείς εκφράσεις χωρισμένες με ελληνικά ερωτηματικά. Η πρώτη έκφραση αρχικοποιεί τη μεταβλητή που χειρίζεται η for και εκτελείται μια φορά όταν ο βρόχος ξεκινάει. Η δεύτερη έκφραση είναι μια συνθήκη ελέγχου, και υπολογίζεται πριν από κάθε εκτέλεση του βρόχου. Όταν η συνθήκη γίνει ψευδής (όταν η η μεταβλητή i γίνει ίση με τη μεταβλητή max), ο βρόχος τερματίζεται. Η τρίτη έκφραση υπολογίζεται στο τέλος κάθε βρόχου.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int i;
        for (i = 1 ; i <= 100 ; i++)
        {
                if ( i % 2 == 0 )
                printf("Ο αριθμός %d είναι άρτιος\n", i);
        }
        return 0;
}
```

Στο προηγούμενο παράδειγμα θέλουμε να εμφανιστούν ποιοι αριθμοί από το 1 έως το 100 είναι άρτιοι. Επομένως εσκεμμένα τέθηκε η τιμή έναρξης και λήξης της βοηθητικής μεταβλητής ώστε να μας βολεύει καθώς οι τιμές που θα πάρει «τυχαίνει» να είναι οι τιμές που θα ελεγχθούν ώστε να εμφανιστεί το κατάλληλο μήνυμα. Εναλλακτικά θα μπορούσε να έχει χρησιμοποιηθεί η επόμενη προσέγγιση:

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int i;
        for (i = 2 ; i <= 100 ; i = i + 2)
                printf("Ο αριθμός %d είναι άρτιος\n", i);
        return 0;
}
```

Κατά τη δεύτερη εκδοχή του προηγούμενου παραδείγματος τροποποιήθηκε η τιμή έναρξης της βοηθητικής μεταβλητής καθώς και ο ρυθμός αύξησης. Ως τιμή έναρξης τέθηκε η πρώτη άρτια τιμή στο πεδίο τιμών που μας ενδιαφέρει και στη συνέχεια ο ρυθμός αύξησης ήταν τέτοιος που οδηγούσε στην αμέσως επόμενη άρτια τιμή.

Άσκηση

Ένας τουρίστας ενοικίασε ένα αυτοκίνητο με τον όρο να το επιστρέψει είτε μετά την πάροδο 5 ημερών είτε όταν διανύσει περισσότερα από 5000 χλμ. Να γραφεί πρόγραμμα που θα διαβάζει πόσα χιλιόμετρα διήνυσε ο τουρίστας με το αυτοκίνητο ανά ημέρα. Το πρόγραμμα θα τερματίζει είτε όταν περάσουν 5 ημέρες είτε όταν ξεπεραστούν τα 5000 χλμ. Στο τέλος θα τυπώνονται τα χιλιόμετρα και οι συνολικές ημέρες που χρησιμοποίησε το αυτοκίνητο.

```
#include <stdio.h>
int main (void)
{
        int i,sum=0,xiliometra,flag=0;

        for (i = 1 ; i <= 5 ; i++)
        {
                printf("Δώσε τα χιλιόμετρα της %dης ημέρας \n",i);
                scanf(«%d»,&xiliometra);
                sum+=xiliometra;
                if  ( sum > 5000)
                {
                        flag = 1; //η μεταβλητή flag χρησιμοποιείται για να υπο-
                        δειχθεί για ποίο  // λόγο ολοκληρώθηκε η δομή επανά-
                        ληψης
                        break;
                }
```

```
        }
        if (i==5 && flag==1)
                printf("Το αμάξι επιστράφηκε λόγω ημερών αλλά και %d χιλιο-
                μέτρων",sum);
        else if (flag==1)
                printf("Το αμάξι επιστράφηκε λόγω %d χιλιομέτρων ",sum);
        else if (i==6)
                printf("Το αμάξι επιστράφηκε λόγω ημερών");
        return 0;
}
```

5.4 Εμφωλευμένες Δομές Επανάληψης

Η δομή επανάληψης for στο σώμα εντολών μπορεί να περιέχει οποιαδήποτε πρόταση, αυτό σημαίνει ότι μπορεί να υπάρχει και κάποιος εμφωλευμένος βρόχος επανάληψης. Το ίδιο ισχύει και για τις άλλες δύο δομές επανάληψης.

Στο επόμενο παράδειγμα πρέπει να γίνει ένα πρόγραμμα το οποίο θα εμφανίζει την προπαίδεια. Ο εξωτερικός βρόχος θα εκτελεστεί δέκα φορές και για κάθε επανάληψη της εξωτερικής for ο εσωτερικός βρόχος θα εκτελείται δέκα φορές και αυτός. Αυτό σημαίνει ότι στο σύνολο θα γίνουν 100 επαναλήψεις (10 X 10)

Παράδειγμα

```
main(void){
        for ( int i = 1  ; i < 10 ; i++){
                for (int j = 1 ; j <= 10 ; j++)
                        printf(«%d  X  %d  = % d \n», i , j, i*j );
        }
}
```

Άσκηση

Να γίνει πρόγραμμα που θα διαβάζει τις θερμοκρασίες 3 πόλεων (Θεσσαλονίκη, Αθήνα, Πάτρα) για 3 ημέρες και θα εμφανίζει τη μέση θερμοκρασία της κάθε πόλης για τις 3 μέρες .

```
#include <stdio.h>
int main(void){
        int thermokrasia, sunolo_thermokrasias = 0;
        float mesi_thermokrasia_polis = 0;
        for (int  i= 1 ; i <= 3 ; i++){
                for (int  j = 1 ; j <= 3 ; j++){
                        printf("Δώσε για την πόλη %d την θερμοκρασία της μέ-
                        ρας: %d\n",i,j);
                        scanf(«%d»,&thermokrasia);
                        sunolo_thermokrasias+=thermokrasia;
                }
                mesi_thermokrasia_polis=sunolo_thermokrasias/3;
                printf("Η μέση θερμοκρασία της πόλη είναι:%f\n",mesi_
                thermokrasia_polis);
                sunolo_thermokrasias=0;
        }
        return 0;
}
```

Άσκηση

Σύμφωνα με τον κώδικα οδικής κυκλοφορίας, όταν ένας οδηγός κάνει μια παράβαση η τροχαία του επιβάλλει ποινή από 5 μέχρι 40 βαθμούς, ανάλογα με το είδος της παράβασης. Για κάθε νέα παράβαση οι βαθμοί ποινής αθροίζονται με τους προηγούμενους. Αν το άθροισμα ποινών υπερβεί τους 40 βαθμούς, αφαιρείται το δίπλωμα για ένα τρίμηνο, ενώ αν υπερβεί τους 60 αφαιρείται για ένα έτος. Να γραφεί πρόγραμμα το οποίο:

α. θα διαβάζει για 4 οδηγούς το πλήθος των παραβάσεών τους.

β. θα διαβάζει για κάθε οδηγό τους βαθμούς ποινής κάθε παράβασης, ελέγχοντας την εγκυρότητά τους,

γ. θα βρίσκει το άθροισμα των βαθμών ποινής για κάθε οδηγό και θα εμφανίζει 'Σύνολο βαθμών ποινής κάτω από 40' ή 'Αφαίρεση για 3 μήνες' ή 'Αφαίρεση για 1 χρόνο'.

```
main(void){
        int paravaseis,poini,sum=0;
        char c;
        for (int i=0; i<4; i++){
                printf("Για τον %do οδηγό δώσε πλήθος παραβάσεων\n",i+1);
                scanf(«%d», &paravaseis);
                for (int j = 0 ; j < paravaseis ; j++){
                        do{
                        printf("Δώσε τους βαθμούς ποινής της %do παράβασεις\n",j+1);
                        scanf(«%d»,&poini);
                        }while(poini<5 || poini>40);
                }
                sum+=poini;
                if (sum<40)
                        printf("Το σύνολο των βαθμών είναι μικρότερο από το όριο\n");
                else if (sum>40 && sum<60)
                        printf("Αφαίρεση διπλώματος για 3 μήνες\n");
                else
                        printf("Αφαίρεση διπλώματος για 1 χρόνο\n");
                sum=0;
        }
}
```

5.5 Η εντολή goto

Η εντολή goto μεταφέρει τον έλεγχο του προγράμματος σε ένα διαφορετικό σημείο του προγράμματος. Η μεταφορά του ελέγχου γίνεται χρησιμοποιώντας ετικέτες που καταδεικνύουν τα σημεία του προγράμματος στα οποία θέλουμε να γίνεται η μεταφορά του ελέγχου. Μια ετικέτα είναι ένα αναγνωριστικό που ακολουθείται από άνω κάτω τελεία. Η σύνταξη της εντολή είναι:

goto ετικέτα;

Στο επόμενο παράδειγμα εμφανίζεται συνεχώς ένας χαρακτήρας που δίνεται από τον χρήστη.

Παράδειγμα

```
main(void)
{
        char x;
        scanf(" %c ", &x);
        again:
                printf(" ο χαρακτήρας είναι %c \n");
        goto pali;
}
```

Η ετικέτα δεν ακολουθείται από ερωτηματικό, καθώς με το ερωτηματικό τερματίζει μια πρόταση. Ωστόσο η ετικέτα όταν βρίσκεται μετά την goto, τότε ακολουθείται από το ερωτηματικό και όχι από την άνω κάτω τελεία.

Η goto ως μέθοδος μεταφοράς του ελέγχου σε ένα πρόγραμμα είναι ο πιο εύκολος τρόπος για να καταστραφεί ο δομημένος προγραμματισμός, γι'αυτό δεν χρησιμοποιείται και αναφέρεται μόνο για λόγους πληρότητας. Σε κάθε περίπτωση, για να είναι δομημένο ένα πρόγραμμα και ευανάγνωστο χρησιμοποιούνται οι υπόλοιπες επαναληπτικές διαδικασίες της C.

5.6 Η εντολή continue

Αυτή η εντολή χρησιμοποιείται με τις τρείς επαναληπτικές δομές. Η continue όπως και η break διακόπτει τη ροή ενός προγράμματος. Όμως αντί να τερματίσει ολόκληρο τον βρόχο, η continue έχει ως αποτέλεσμα τη μη εκτέλεση του υπόλοιπου τμήματος του βρόχου και την επανάληψή του από την αρχή.

Στο παρακάτω παράδειγμα εμφανίζονται μόνο οι άρτιοι αριθμοί. Κάθε φορά που παράγεται από το πρόγραμμα ένας περιττός αριθμός, εκτελείται η εντολή if, γιατί το υπόλοιπο της διαίρεσης ενός περιττού αριθμού με το 2 είναι ίσο πάντοτε με 1, δηλαδή ΑΛΗΘΗΣ τιμή. Έτσι, ένας περιττός αριθμός προκαλεί την εκτέλεση της εντολής continue, οπότε εκτελείται η επόμενη επανάληψη του βρόχου, ενώ η εντολή printf( ) παρακάμπτεται.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int i;
        for (i = 0; i < 100; i++)
        {
                if (i % 2)
                        continue;
                printf("%5d",i);
        }
        return 0;
}
```

5.7 Ασκήσεις

1. Να γραφεί πρόγραμμα το οποίο να υπολογίζει τη μέγιστη και την ελάχιστη ημερήσια θερμοκρασία που εμφανίστηκε την προηγούμενη βδομάδα (η θερμοκρασία δίνεται από τον χρήστη).

2. Να γραφεί πρόγραμμα το οποίο:

α. θα διαβάζει από το πληκτρολόγιο μια σειρά θετικών ακέραιων τιμών. Η διαδικασία εισαγωγής τιμών θα τερματίζεται όταν το άθροισμά τους γίνει μεγαλύτερο από 1000.

β. θα εμφανίζει το άθροισμα των τιμών εισόδου.

3. Να γραφεί πρόγραμμα το οποίο:

α. θα διαβάζει από το πληκτρολόγιο μια σειρά θετικών ακέραιων τιμών. Η διαδικασία εισαγωγής τιμών θα τελειώνει δίνοντας την τιμή 0 ή αρνητικό αριθμό

β. θα εμφανίζει το πλήθος, το άθροισμα και το μέσο όρο των τιμών εισόδου.

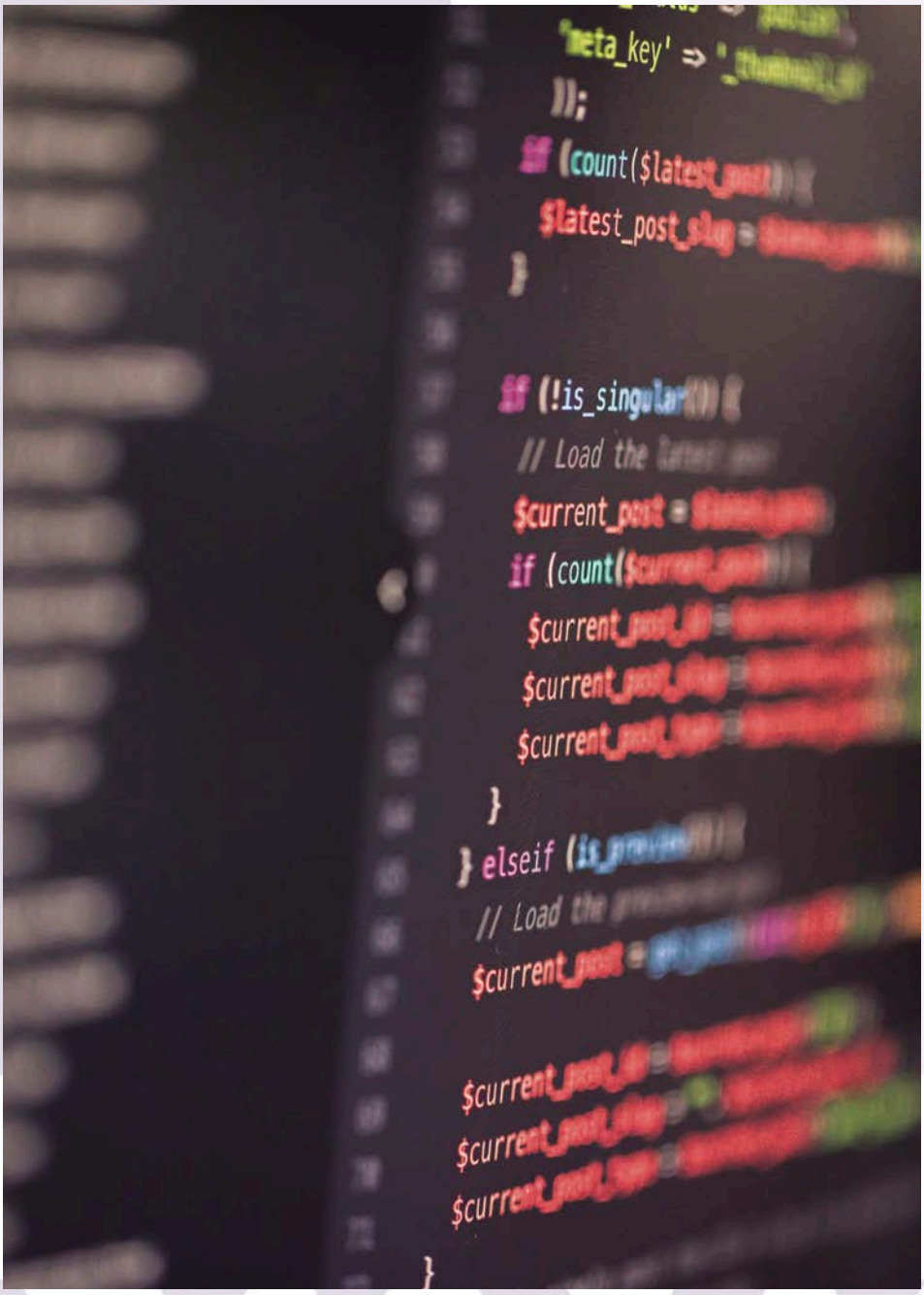
4. Δύο παίκτες ρίχνουν διαδοχικά ένα ζάρι. Αν ο αριθμός της ζαριάς είναι μονός, τότε ο πρώτος παίκτης κερδίζει έναν πόντο. Αν συμβεί να είναι ζυγός, κερδίζει ένα πόντο ο δεύτερος παίκτης. Να γραφεί πρόγραμμα το οποίο θα διαβάζει τον αριθμό της ζαριάς κάθε παίκτη μετά από 50 προσπάθειες του καθενός και θα βρίσκει και θα εμφανίζει ποιος από τους δύο παίκτες είναι ο νικητής.

5. Σε ένα υποκατάστημα μιας τράπεζας λειτουργούν τρία ταμεία (1, 2 και 3). Για την καλύτερη οργάνωση της λειτουργίας της, η τράπεζα χρειάζεται στατιστικά στοιχεία για τα άτομα που εξυπηρετεί κάθε ταμείο την ημέρα. Να γραφεί πρόγραμμα το οποίο:

α. θα διαβάζει τον αριθμό του ταμείου που εξυπηρετεί κάθε πελάτη. Το τέλος της ημέρας θα δηλώνεται όταν δοθεί ως αριθμός ταμείου το 0,

β. θα εμφανίζει τον αριθμό των ατόμων που εξυπηρετήθηκαν από κάθε ταμείο και από το υποκατάστημα συνολικά,

γ. θα εμφανίζει το ποσοστό ατόμων που εξυπηρετήθηκαν από κάθε ταμείο ξεχωριστά.

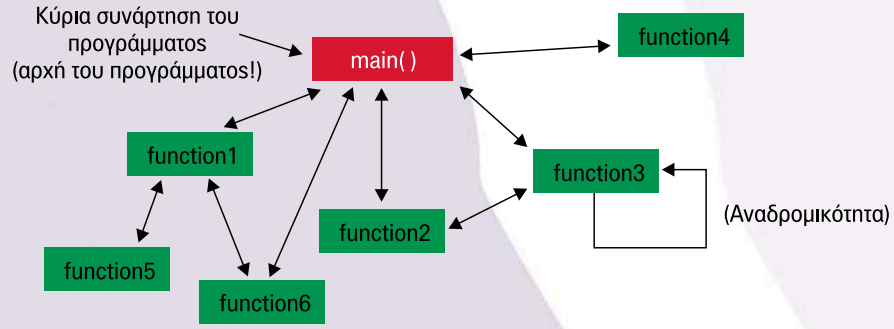

'meta_key' =>
));
if (count($latest_post)) {
$latest_post_slug =
}
if (!is_singular()) {
// Load the latest post
$current_post =
if (count($current_post)) {
$current_post_id =
$current_post_slug =
$current_post_type =
}
} elseif (is_preview()) {
// Load the preview post
$current_post =
$current_post_id =
$current_post_slug =
$current_post_type =
}

6. Συναρτήσεις (Functions)

Ένα εκτενές πρόγραμμα στη C ενδεχομένως να αποτελείται από μικρότερα τμήματα κώδικα που ονομάζονται συναρτήσεις. Η πιο βασική συνάρτηση που είναι πάντα παρούσα σε κάθε πρόγραμμα είναι η main ( ), όμως μέχρι στιγμής έχουν χρησιμοποιηθεί και κάποιες εντολές όπως η printf ( ), scanf ( ) ή η getchar ( ), που στην πραγματικότητα είναι συναρτήσεις οι οποίες προσφέρονται από κάποια βιβλιοθήκη. Οι συναρτήσεις που χρησιμοποιούνται σε ένα πρόγραμμα μπορεί να είναι ήδη έτοιμες και να προσφέρονται από κάποια βιβλιοθήκη ή μπορεί να έχουν δημιουργηθεί από εμάς τους ίδιους.

Ο σκοπός της χρήσης συναρτήσεων είναι η κατάτμηση του προγράμματος σε μικρότερα υποπρογράμματα, όπου το καθένα εκτελεί μια συγκεκριμένη λειτουργία. Αφενός, η χρήση των συναρτήσεων θα μειώσει τις γραμμές του κώδικα καθώς αντί να επαναλαμβάνεται ένα σύνολο εντολών που εκτελεί ακριβώς την ίδια λειτουργία θα καλείται απλώς η συνάρτηση που θα είναι υπεύθυνη για την εκάστοτε λειτουργία. Αφετέρου, καθίσταται πιο εύκολη η διαδικασία εύρεσης πιθανών σφαλμάτων (συντακτικών ή λογικών), καθώς η προσοχή μας θα πρέπει να εστιαστεί στις εντολές της συνάρτησης και όχι σε μπλοκ εντολών της main ( ), που μάλιστα μπορεί να επαναλαμβάνεται σε διάσπαρτα σημεία της main ( ).

Κάθε συνάρτηση μπορεί να καλείται τόσο από κάποια άλλη όσο και από τον ίδιο της τον εαυτό. Όμως είναι βασικό για έναν σωστό σχεδιασμό προγράμματος οι συναρτήσεις να έχουν την κατάλληλη δόμηση και ιεράρχηση.



Εικόνα 6.1 Κλήση Συναρτήσεων

Ορισμός μιας συνάρτησης

Μια συνάρτηση ορίζεται με τον εξής τρόπο:

```
τύπος_συνάρτησης  όνομα_συνάρτησης (όρισμα1, όρισμα2, ..., όρισμα N)
{
        δήλωση μεταβλητών συνάρτησης;
        εκτελέσιμες προτάσεις συνάρτησης;
        return τιμή;
}
```

Κατά τη σύνταξη μιας συνάρτησης αρχικά θα πρέπει να προσδιοριστεί ο τύπος της. Ο τύπος κάθε συνάρτησης καθορίζεται αυστηρά από τον τύπο της τιμής που επιστρέφεται στη main () ή σε όποια άλλη συνάρτηση την καλέσει. Ας υποθέσουμε ότι δημιουργείται μια συνάρτηση που υπολογίζει και επιστρέφει το άθροισμα τριών ακέραιων τιμών. Σε αυτή την περίπτωση ο τύπος της συνάρτησης θα είναι int καθώς η τιμή που θα επιστρέψει η συνάρτηση είναι και αυτή int. Κατ' επέκταση, οι βασικοί τύποι των συναρτήσεων είναι αντίστοιχοι των τύπων των βασικών δεδομένων που διαθέτει η C, δηλαδή int, float και char.

Το επόμενο δομικό στοιχείο της συνάρτησης που πρέπει να προσδιοριστεί είναι το όνομα της και εξαρτάται από εμάς τους ίδιους, συνήθως ως όνομα δίνεται κάτι

αντιπροσωπευτικό της λειτουργίας που επιτελεί. Τέλος πρέπει να τοποθετηθούν τα ορίσματα, δηλαδή τα δεδομένα που θα δέχεται η συνάρτηση και που είναι βασικά, προκειμένου να εκτελεστεί σωστά η λειτουργία της.

- Στις συναρτήσεις υπάρχει και ο τύπος void. Συναρτήσεις τύπου void δεν επιστρέφουν καμία τιμή.

6.1 Συναρτήσεις που επιστρέφουν τιμή

Όπως ήδη έχει αναφερθεί κάθε συνάρτηση επιστρέφει μια τιμή. Αν δεν οριστεί ο τύπος της συνάρτησης τότε θεωρείται ότι επιστρέφεται ακέραια τιμή. Σε περίπτωση που επιστρέφεται τιμή άλλου τύπου τότε είναι απαραίτητο να δηλωθεί ο αντίστοιχος τύπος στη συνάρτηση. Η τιμή επιστρέφεται μέσω της εντολής return και σε εκείνο το σημείο τερματίζεται η συνάρτηση.

Στο παρακάτω παράδειγμα δημιουργείται ένα πρόγραμμα το οποίο διαβάζει τρεις ακέραιους βαθμούς ενός μαθητή από το πληκτρολόγιο και μετά καλείται μια συνάρτηση που υπολογίζει και επιστρέφει τον μέγιστο βαθμό.

Παράδειγμα – 1ος τρόπος κλήσης συνάρτησης, εκχώρηση σε μεταβλητή

```
#include <stdio.h>
int max (int a, int b, int c){
        int max;
        if (a > b && a > c)
                max = a;
        else if (b > a && b > c)
                max = b;
        else
                max = c;
        return max;
}
```

```
main(void){
        int grade1, grade2, grade3, result;
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3);
        result = max(grade1, grade2, grade3);  // κλήση συνάρτησης max
        printf("ο μεγαλύτερος βαθμός είναι %d\n", result);
}
```

Κατά την κλήση της συνάρτησης περνιούνται ως ορίσματα οι τρεις βαθμοί, δηλαδή οι μεταβλητές grade1, grade2 και grade3, καθώς αποτελούν βασικά δεδομένα ώστε να εκτελεστεί σωστά η λειτουργία της συνάρτησης. Τα ορίσματα αυτά ονομάζονται πραγματικά ορίσματα, γιατί σε αυτές τις μεταβλητές θα καταχωρηθούν οι βαθμοί που θα δώσει ο χρήστης του προγράμματος. Όταν κληθεί η συνάρτηση οι τιμές των πραγματικών ορισμάτων θα αντιγραφούν γραμμικά στα ορίσματα που βρίσκονται στη δήλωση της συνάρτησης, που ονομάζονται τυπικά ορίσματα. Δηλαδή η τιμή του πρώτου πραγματικού ορίσματος (grade1) θα αντιγραφεί στο πρώτο τυπικό όρισμα (a), η τιμή του δεύτερου πραγματικού ορίσματος (grade2) θα αντιγραφεί στο δεύτερο τυπικό όρισμα (b) κ.ο.κ. Αυτός ο τύπος κλήσης (όχι τρόπος κλήσης) της συνάρτησης, ονομάζεται κλήση με τιμή (call by value). Σε αυτό τον τύπο κλήσης τα πραγματικά και τα τυπικά ορίσματα δεν έχουν καμία είδους σύνδεση μεταξύ τους, παρά μόνο τις ίδιες τιμές όπως αναφέρθηκε προηγουμένως. Επίσης, ο τύπος των πραγματικών ορισμάτων πρέπει να είναι ίδιος με τον τύπο των τυπικών ορισμάτων. Κατά τ' άλλα οι έξι μεταβλητές στο σύνολο είναι ανεξάρτητες μεταξύ τους και θα μπορούσαν ακόμα τα πραγματικά ορίσματα να έχουν ίδιο όνομα με τα τυπικά, κάτι που δεν αποτελεί καλή πρακτική λόγω διάκρισης.

Στον πρώτο τρόπο κλήσης καλείται η συνάρτηση και η τιμή που επιστρέφεται εκχωρείται στη μεταβλητή result. Επομένως ο μέγιστος βαθμός που υπολογίστηκε και επιστράφηκε μέσω της συνάρτησης βρίσκεται πλέον καταχωρημένος στη result. Οπότε χρησιμοποιείται η printf για να εμφανίσει τον μεγαλύτερο βαθμό μέσω της result. Εναλλακτικά θα μπορούσε να έχει ακολουθηθεί η εξής προσέγγιση:

Παράδειγμα – 2ος τρόπος κλήσης συνάρτησης, μέσω printf

```
main(void){
        int grade1, grade2, grade3;
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3);
        printf("ο μεγαλύτερος βαθμός είναι %d\n", max(grade1, grade2,
        grade3));
}
```

Στον δεύτερο τρόπο κλήσης της συνάρτησης παραλείπεται η χρήση της μεταβλητής result στην οποία εκχωρείται η επιστρεφόμενη τιμή της συνάρτησης. Σε αυτή την περίπτωση η συνάρτηση καλείται μέσω της printf και κατευθείαν εμφανίζεται η τιμή που επιστρέφεται από τη συνάρτηση. Παρατηρήστε ότι η printf έχει ένα %d τοποθετημένο στο αλφαριθμητικό της, που αντιστοιχίζεται στην τιμή που θα επιστρέψει η κλήση της συνάρτησης max.

Μέχρι στιγμής έχουμε δει δύο τρόπους κλήσης μιας συνάρτησης, κλήση και εκχώρηση τιμής σε μια μεταβλητή και κλήση συνάρτησης μέσω printf. Υπάρχει και ένας τρίτος τρόπος κλήσης και λαμβάνει χώρα σε έναν λογικό έλεγχο. Θα τροποποιήσουμε λίγο την εκφώνηση του προηγούμενου παραδείγματος κάνοντας τη συνάρτηση να υπολογίζει τον μέσο όρο των τριών βαθμών και σε περίπτωση που ο μέσος όρος (επιστρεφόμενη τιμή) είναι κάτω του 10 να εμφανίζεται μήνυμα «Κόπηκες».

Παράδειγμα – 3ος τρόπος κλήσης συνάρτησης, χρήση σε λογικό έλεγχο

```
float mo (int a, int b, int c)
{
        int sum = 0;
        sum = a + b + c;
        return sum/3.0;
}
```

```
main(void){
        int grade1, grade2, grade3;
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3);
        if ( mo(grade1, grade1, grade1) < 10 )
                printf("Κόπηκες");
}
```

Στον 3ο τρόπο κλήσης της συνάρτησης η επιστρεφόμενη τιμή συμμετέχει σε έναν λογικό έλεγχο, όπου ανάλογα με την τιμή ο έλεγχος είναι ΑΛΗΘΗΣ ή ΨΕΥΔΗΣ. Κατά τους δύο τελευταίους τρόπους κλήσης της συνάρτησης η επιστρεφόμενη τιμή δεν θα είναι διαθέσιμη στο κυρίως πρόγραμμα μετά το πέρας της γραμμής στην οποία κλήθηκε η συνάρτηση, καθώς η επιστρεφόμενη τιμή δεν έχει εκχωρηθεί σε κάποια μεταβλητή. Μόνο στον 1ο τρόπο κλήσης υπάρχει η δυνατότητα περαιτέρω χειρισμού της επιστρεφόμενης τιμή μέσω της μεταβλητής result. Δεν υπάρχει κάποια δέσμευση μεταξύ των τρόπων κλήσεως και γενικά θα μπορούσε πάντα να χρησιμοποιείται ο πρώτος τρόπος και έπειτα είτε πρόκειται για εμφάνιση της επιστρεφόμενης τιμής είτε για χρήση σε κάποιο λογικό έλεγχο να χρησιμοποιείται η μεταβλητή στην οποία έχει εκχωρηθεί η επιστρεφόμενη τιμή (η result στην προκειμένη περίπτωση). Ο καθένας από τους τρεις τρόπους κλήσης επιλέγεται προς χρήση ανάλογα με την περίπτωση και ποιος είναι πιο βολικός κάθε φορά.

Άσκηση

Να δημιουργηθεί μια συνάρτηση που θα δέχεται ως παράμετρο έναν χαρακτήρα και θα επιστρέφει 1 αν είναι πεζός, 2 αν είναι κεφαλαίος, 3 αν είναι αριθμητικό ψηφίο (0-9) και 0 σε άλλη περίπτωση. Τέλος, θα καλείται στο κυρίως πρόγραμμα από όπου θα εμφανίζεται το κατάλληλο μήνυμα.

```
#include <stdio.h>
int isletter(char c){
        int result;
        if (c >= 'a' && c <= 'z')
                result = 1;
        else if (c >= 'A'  &&  c <= 'Z')
                result = 2;
        else if (c >= '0' && c <= '9')
                result = 3;
        else
                result = 0;
        return result;
}
int main(void)
{
        char ch;
        printf("Δώσε ένα χαρακτήρα\n");
        scanf(«%c», &ch);
        if ( isletter( c ) == 1)
                printf(" Πρόκειται  για πεζό λατινικό χαρακτήρα");
else if ( isletter( c ) == 2)
                printf(" Πρόκειται για κεφαλαίο λατινικό χαρακτήρα");
else if ( isletter( c ) == 3)
                printf(" Πρόκειται για αιρθμητικό χαρακτήρα");
        else
                printf(" Άλλη περίπτωση");
}
```

6.2 Συναρτήσεις που δεν επιστρέφουν τιμή

Εκτός από τους τύπους των συναρτήσεων που αναφέρθηκαν, υπάρχει και ο τύπος void. Σε αυτή την περίπτωση η συνάρτηση δεν επιστρέφει κάποια τιμή απλά εκτελεί μια συγκεκριμένη λειτουργία. Στο επόμενο παράδειγμα δημιουργείται ένα πρόγραμμα στο οποίο θα εμφανίζεται ένα μενού επιλογών και ο χρήστης θα πρέπει να επιλέξει μια από τις διαθέσιμες επιλογές ώστε να εκτελεστεί η κατάλληλη ενέργεια. Η εμφάνιση του μενού θα γίνει μέσω της συνάρτησης display ( );

Η ενέργεια προς εκτέλεση θα παραλειφθεί καθώς δεν αποτελεί το θέμα συζήτησης αυτή τη στιγμή.

Παράδειγμα

```
void display ( )
{
        printf("1. Εισαγωγή\n");
        printf("2. Διόρθωση\n");
        printf("3. Διαγραφή\n");
        printf(«4. Έξοδος\n»);
}
int main(void){
        int choice;
        display( );  // κλήση συνάρτησης
        scanf(«%d», &choice);
}
```

Οι συναρτήσεις που είναι τύπου void έχουν μόνο τον παραπάνω τρόπο κλήσης, σε αντίθεση με αυτές που επιστρέφουν κάποια τιμή, καθώς δεν επιστρέφουν κάποια τιμή παρά μόνο εκτελούν κάποια λειτουργία ή εμφανίζουν κάποιες πληροφορίες, επομένως επί της αρχής δεν τίθεται θέμα εκχώρησης κάποιας τιμής ή χρήσης σε λογικό έλεγχο.

Στα προηγούμενα παραδείγματα η συνάρτηση max επιστρέφε κάποια τιμή. Θα μπορούσαμε να έχουμε τροποποιήσει τη συνάρτηση ορίζοντάς της τύπο void και προσαρμόζοντάς την ώστε να μην επιστρέφει τον μεγαλύτερο βαθμό αλλά απλά να τον υπολογίζει και να τον εμφανίζει εντός του σώματος εντολών της συνάρτησης.

Παράδειγμα

```
void max (int a, int b, int c){
        int result;
        if (a > b && a > c)
                result = a;
        else if (b > a && b > c)
                result = b;
        else
                result = c;
printf("ο μεγαλύτερος βαθμός είναι %d\n", result);
}

main(void){
        int grade1, grade2, grade3;
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3);
}
```

Άρα ο τύπος της συνάρτησης που θα χρησιμοποιηθεί είναι κάπως σχετικός και εξαρτάται από την προσέγγιση που θέλουμε να ακολουθήσουμε για την επίλυση του προβλήματος. Αυτό θα γίνει πιο οφθαλμοφανές και στα επόμενα κεφάλαια.

Στο παρακάτω παράδειγμα δημιουργείται ένα πρόγραμμα το οποίο δέχεται δύο ακέραιες τιμές και καλεί τη συνάρτηση swap που θα ανταλλάσσει τις τιμές των δυο μεταβλητών.

Παράδειγμα

```
#include <stdio.h>
void swap(int a, int b){
        int temp = 0;
        temp = a;
        a = b;
        b = temp;
        printf("Οι τιμές είναι :x = %d y = %d \n",a, b); //εμφανίζονται οι τιμές
        αλλαγμένες
}
int main(void){
        int x, y;
        printf("Δώσε δύο ακέραιες τιμές:\n");
        scanf(«%d %d»,&x, &y);
        swap(x, y);
        printf("Οι τιμές είναι x = %d y = %d",x ,y);  //οι τιμές δεν εμφανίζονται
        αλλαγμένες
}
```

Στο παραπάνω παράδειγμα, ενώ καλείται η συνάρτηση swap για να ανταλλάξει τις τιμές των δύο μεταβλητών, το αποτέλεσμα που θα εμφανιστεί δεν θα είναι απολύτως σωστό. Ενώ όταν καλείται η swap και η λειτουργία εκτελείται σωστά όταν χρησιμοποιείται η printf εντός της συνάρτησης, οι τιμές είναι αλλαγμένες, όμως όταν χρησιμοποιείται η printf εντός main, οι τιμές δεν είναι αλλαγμένες και οι μεταβλητές διατηρούν τις αρχικές τους τιμές. Το πρόβλημα έγκειται στον τύπο κλήσης της συνάρτησης, δηλαδή στο call by value. Δηλαδή η τιμή της μεταβλητής x αντιγράφεται στη μεταβλητή a, το ίδιο συμβαίνει και στη μεταβλητή y και b. Όμως στη συνέχεια η διαδικασία της ανταλλαγής τιμών γίνεται στις μεταβλητές a και b που είναι ανεξάρτητες από τις μεταβλητές x και y. Γι' αυτό όταν χρησιμο-

ποιείται η printf εντός της συνάρτησης οι τιμές είναι αλλαγμένες, ενώ κατά την χρήση της printf εντός main οι τιμές δεν είναι αλλαγμένες.

Άρα η διαδικασία της ανταλλαγής δεν είναι μόνιμη καθώς δεν επιτελείται στις μεταβλητές που μας ενδιαφέρουν (πραγματικά ορίσματα) αλλά στις μεταβλητές της συνάρτησης (τυπικά ορίσματα). Το πρόβλημα δεν θα μπορούσε να επιλυθεί αν προσπαθούσαμε να χρησιμοποιήσουμε εντός της main τις μεταβλητές a και b που έχουν αλλαγμένες τιμές γιατί τίθεται θέμα εμβέλειας. Δηλαδή οι μεταβλητές της εκάστοτε συνάρτησης δημιουργούνται κατά την κλήση της συνάρτησης και καταστρέφονται με τον τερματισμό της και ταυτόχρονα δεν είναι γνωστές στη main, όπως οι μεταβλητές τις main δεν είναι γνωστές στην εκάστοτε συνάρτηση. Οπότε αν χρησιμοποιούσαμε κάποιες μεταβλητές μια συνάρτησης στην main τότε αυτές δεν θα μπορούσαν να αναγνωριστούν και θα εμφανιζόταν σφάλμα.

Η μόνιμη ανταλλαγή θα μπορούσε να επιτευχθεί με τον δεύτερο τύπο κλήσης των συναρτήσεων, δηλαδή κάνοντας κλήση με αναφορά (call by reference), στο συγκεκριμένο τύπο θα αναφερθούμε σε επόμενο κεφάλαιο.

Άσκηση

1 Να δημιουργηθεί ένα πρόγραμμα που θα δέχεται έναν ακέραιο αριθμό από το πληκτρολόγιο, έστω n, θα καλεί μια συνάρτηση η οποία θα εμφανίζει τους n πρώτους αριθμούς της σειράς Fibonacci. Η σειρά Fibonacci ξεκινάει με τους αριθμούς 0 και 1, οι υπόλοιποι αριθμοί προκύπτουν από το άθροισμα των δύο προηγούμενων αριθμών. Π.χ. 0, 1, 1, 2, 3, 5, 8, 13, 21

```
#include <stdio.h>
void fibonacci(int x)
{
        int next,first=0,second=1;
        for (int i=0;i<x;i++)
```

```
        {
                if (x<=1)
                        next=x;
                else
                {
                        next=first+second;
                        first=second;
                        second=next;
                }
                printf(«%d\n»,next);
        }
}
int main(void)
{
        int plithos;
        printf("Δώσε το πλήθος των αριθμών\n");
        scanf(«%d», &plithos);
        fibonacci(plithos);
        return 0;
}
```

6.3 Εμβέλεια Μεταβλητών

Στη C δίνεται η δυνατότητα δήλωσης μεταβλητών σε διάφορα σημεία του προγράμματος. Το σημείο αλλά και το είδος της δήλωσης καθορίζει την εμβέλεια της εκάστοτε μεταβλητής, δηλαδή το εύρος του κώδικα στο οποίο θα είναι έγκυρη και αναγνωρίσιμη. Οι μεταβλητές διακρίνονται σε τοπικές (local), καθολικές (global) και στατικές (static).

6.3.1 Τοπικές Μεταβλητές (Local Variables)

Όπως ήδη έχουμε δει σε κάθε συνάρτηση υπάρχει η δυνατότητα δήλωσης μεταβλητών. Σε περίπτωση που μια συνάρτηση διαθέτει και τυπικά ορίσματα, που είναι και αυτά μεταβλητές, πρέπει να δηλωθούν στις παρενθέσεις μετά το όνομα της συνάρτησης. Τόσο οι μεταβλητές μιας συνάρτησης όσο και τα τυπικά ορίσματα είναι αναγνωρίσιμα μόνο μέσα στην συνάρτηση και παραμένουν άγνωστες στις υπόλοιπες συναρτήσεις συμπεριλαμβάνοντας και τη main.

Στο παρακάτω παράδειγμα δημιουργείται ένα πρόγραμμα που δέχεται τρεις ακέραιους βαθμούς ενός μαθητή και καλεί μια συνάρτηση που υπολογίζει και επιστρέφει τον μέσο όρο των βαθμών. Σε περίπτωση που ο μέσος όρος είναι κάτω από 10 εμφανίζεται μήνυμα «Κόπηκες»

Παράδειγμα

```
#include <stdio.h>
float mo (int a, int b, int c){  // ορίσματα που ταυτόχρονα είναι τοπικές μεταβλητές
        float result; // δήλωση τοπικής μεταβλητής με εμβέλεια μόνο εντός συνάρτησης
        result = (a + b + c)/3.0;
        return result;
}
int main(void)
{
        int grade1, grade2, grade3;  // ← τοπικές μεταβλητές της main
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3);
        if ( mo(grade1, grade1, grade1) < 10 )
                printf("Κόπηκες");
        return 0;
}
```

Στη συνάρτηση mo, η μεταβλητή result έχει ισχύ μόνο μέσα στη mo. Επίσης, οι τυπικές παράμετροι a, b και c αποτελούν και αυτές τοπικές μεταβλητές και είναι αναγνωρίσιμες μόνο μέσα στη συνάρτηση, η διαφορά τους με τη μεταβλητή result είναι ότι σε αυτές αντιγράφονται οι τιμές των πραγματικών ορισμάτων κατά την κλήση της συνάρτησης.

6.3.2 Καθολικές Μεταβλητές (Global Variables)

Στη C υπάρχει η δυνατότητα να δηλωθεί μια μεταβλητή έξω από οποιαδήποτε συνάρτηση και κατ' επέκταση να είναι αναγνωρίσιμη από όλο το πρόγραμμα και από κάθε συνάρτηση. Τέτοιου είδους μεταβλητές ονομάζονται καθολικές μεταβλητές και λέμε ότι έχουν καθολική εμβέλεια. Παρακάτω χρησιμοποιείται το προηγούμενο παράδειγμα έχοντας αλλάξει κάποιες από τις μεταβλητές σε καθολικές .

Παράδειγμα

```
#include <stdio.h>
int grade1, grade2, grade3; // δήλωση καθολικών μεταβλητών
float mo ( ){
        float result;
        result = (grade1 + grade2 + grade3)/3.0; // χρήση των μεταβλητών
        εντός συνάρτησης
        return result;
}
int main(void)
{
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3); // χρήση μεταβλη-
        τών εντός main
        if ( mo( ) < 10 )
                printf("Κόπηκες");
}
```

Στο προηγούμενο παράδειγμα η χρήση των καθολικών μεταβλητών παρείχε έναν εναλλακτικό τρόπο μεταβίβασης τιμών στη συνάρτηση. Ωστόσο, η μεταβίβαση τιμών με τέτοιο τρόπο είναι μια μέθοδος η οποία πρέπει να αποφεύγεται. Σε περίπτωση που μια τοπική και μια καθολική μεταβλητή έχουν ίδιο όνομα, τότε στο πεδίο εμβέλειας της τοπικής μεταβλητής αναφερόμαστε στην τοπική μεταβλητή και σε οποιοδήποτε άλλο σημείο στην καθολική.

6.3.3 Δήλωση τοπικών μεταβλητών εντός Δομής

Ακολουθώντας την ίδια φιλοσοφία δήλωσης τοπικών μεταβλητών εντός συναρτήσεων, υπάρχει η δυνατότητα δήλωσης μιας τοπικής μεταβλητής μέσα σε μια οποιαδήποτε δομή (επιλογής ή επανάληψης) αμέσως μετά το αριστερό άγκιστρο.

Στο παρακάτω παράδειγμα παραθέτεται μια for η οποία χρησιμοποιεί ως βοηθητική μεταβλητή την τοπική μεταβλητή i.

Παράδειγμα

```
for ( int i = 0 ; i < 10 ; i++)
        printf(" Hello \n");
printf ("%d", i);
```

Στην προηγούμενη περίπτωση κατά την έναρξη της δομής επανάληψης θα δηλωθεί η τοπική μεταβλητή i και στη συνέχεια θα εκτυπωθεί 10 φορές η λέξη "Hello" μέσω της δομής επανάληψης, όταν όμως ολοκληρωθεί η δομή και εκτελεστεί το τελευταίο printf η μεταβλητή i δεν θα είναι αναγνωρίσιμη και επομένως δεν θα είναι δυνατόν να εμφανιστεί κάτι.

Στο επόμενο παράδειγμα υπολογίζεται ο μέσος όρος τριών ακέραιων βαθμών ενός μαθητή μόνο εάν και οι τρείς βαθμοί είναι μεγαλύτεροι από 0.

Παράδειγμα

```
main(void)
{
        int grade1, grade2, grade3;
        printf("Δώσε τρεις ακέραιους βαθμούς\n");
        scanf(«%d %d %d», &grade1, &grade2, &grade3);
        if ( grade1 > 0 && grade2 > 0 && grade3 > 0 )
        {
                float mo;
                mo = (grade1 + grade2 + grade3)/3.0;
                printf( "  Ο μέσος όρος είναι %f", mo);
        }
}
```

6.3.4 Στατικές Τοπικές Μεταβλητές (Static Local Variables)

Οι τοπικές μεταβλητές μιας συνάρτησης δηλώνονται κατά την κλήση της συνάρτησης και καταστρέφονται όταν αυτή τερματιστεί και οποιαδήποτε τιμή διέθετε μια μεταβλητή χάνεται. Στη συνέχεια όταν η συνάρτηση ξανακληθεί οι τοπικές μεταβλητές δηλώνονται ξανά. Όταν όμως θέλουμε μια τοπική μεταβλητή να διατηρεί την τιμή της, πρέπει να δηλώνεται ως static.

```
#include <stdio.h>
void func ( )
{
        static int x = 0;
        x++;
        printf(" %d \n", x);
}
int main(void)
{
        func( );
```

```
        func( );
        return 0;
}
```

Στην προηγούμενη περίπτωση την πρώτη φορά που θα κληθεί η συνάρτηση func θα δημιουργηθεί η τοπική μεταβλητή x και θα αρχικοποιηθεί με το 0. Σε οποιαδήποτε άλλη φορά κλήσης της συνάρτησης η τοπική μεταβλητή x εξακολουθεί να υπάρχει χωρίς να γίνεται ξανά η αρχικοποίησή της και διατηρώντας την τιμή που διέθετε από πριν. Επομένως, κατά την πρώτη κλήση της func θα εμφανιστεί ο αριθμός 1 και κατά τη δεύτερη κλήση ο αριθμός 2. Ωστόσο, ο χώρος εμβέλειας της x παραμένει ο ίδιος.

6.4 Αναδρομή (Recursion)

Στην C μια συνάρτηση μπορεί να καλεί τον εαυτό της. Αν στο σώμα μιας συνάρτησης, υπάρχει κάποια πρόταση που καλεί την ίδια την συνάρτηση τότε η συνάρτηση λέγεται αναδρομική. Κάθε αναδρομική διαδικασία πρέπει να διαθέτει και μια μη αναδρομική περίπτωση, γιατί σε διαφορετική περίπτωση η διαδικασία συνεχίζεται επ' άπειρον.

Στο επόμενο παράδειγμα καλείται μια αναδρομική συνάρτηση, που υπολογίζει το παραγοντικό ενός αριθμού. Το παραγοντικό ενός αριθμού n (n!) είναι το άθροισμα όλων των ακεραίων από το 1 μέχρι το n. Επίσης, το παραγοντικό ενός αριθμού ορίζεται ως το γινόμενο του n και του παραγοντικού του προηγούμενου αριθμού, δηλαδή το παραγοντικό του 10 είναι 10 * 9!. Άρα, ο υπολογισμός του παραγοντικού μπορεί να γίνει με αναδρομική διαδικασία.

Παράδειγμα

```
#include <stdio.h>
int paragontiko (int x )
{
```

```
        int result;
        if ( x == 1)
                return 1;
        result = x * paragontiko(x-1);
        return result;
}
int main(void)
{
        int number;
        printf("δώσε έναν αριθμό για τον οποίο θα υπολογιστεί το παραγοντικό
        του\n");
        scanf("%d", &number);
        printf(" το παραγοντικό του αριθμού είναι %d ", paragontiko(number));
        return 0;
}
```

6.5 Ασκήσεις

1. Να δημιουργηθεί πρόγραμμα όπου θα καταχωρούνται τιμές σε μια ακέραια μεταβλητή και σε μια μεταβλητή χαρακτήρα και μετά να καλείται μια συνάρτηση τύπου void που θα εκτυπώνει τον χαρακτήρα τόσες φορές όσο είναι η τιμή της ακέραιας μεταβλητής.
2. Να γραφεί συνάρτηση η οποία να δέχεται σαν παράμετρο ένα έτος και να αποφασίζει αν είναι δίσεκτο ή όχι. Δίσεκτο είναι ένα έτος αν διαιρείται ακριβώς με το 4. Τα έτη όμως που διαιρούνται με το 100 δεν είναι δίσεκτα εκτός και αν διαιρούνται και με το 400. Το αποτέλεσμα να εμφανίζεται στην οθόνη
3. Να δημιουργηθεί πρόγραμμα το οποίο θα δέχεται έναν αριθμό από το πληκτρολόγιο, έστω n, θα καλεί μια συνάρτηση η οποία θα εμφανίζει τις n πρώτες

γραμμές του τρίγωνου του Floyd. Οι τέσσερεις πρώτες γραμμές του τριγώνου είναι:

1

23

456

78910

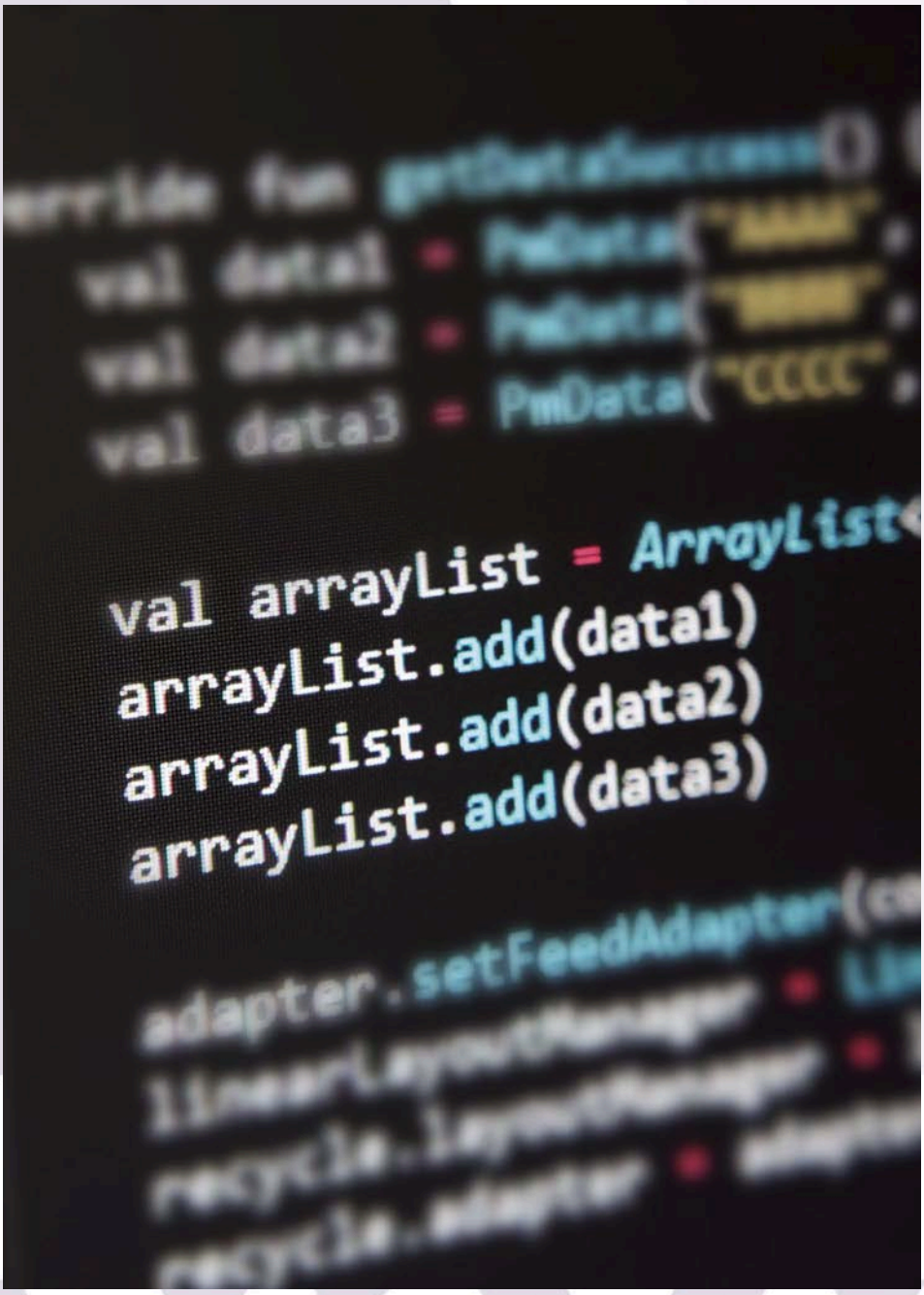
4 Να γίνει πρόγραμμα το οποίο θα διαβάζει έναν φυσικό αριθμό από το πληκτρολόγιο και θα καλεί μια συνάρτηση nextPrime η οποία θα ελέγχει αν είναι πρώτος αριθμός και θα εμφανίζει τους τρεις πρώτους αριθμούς μετά από αυτόν.

5 Να γίνει πρόγραμμα, όπου θα καλείται μια συνάρτηση που θα δέχεται δύο ακέραιους και θα υπολογίζει το ελάχιστο κοινό πολλαπλάσιο των δύο.

6 Να δημιουργηθεί πρόγραμμα που θα δέχεται έναν ακέραιο αριθμό από το πληκτρολόγιο και θα καλεί μια συνάρτηση η οποία θα επιστρέφει 1 αν αυτός ο αριθμός είναι παλινδρομικός αλλιώς 0 (π.χ. 12321).

7 Να δημιουργηθεί πρόγραμμα που θα δέχεται δύο ακέραιους αριθμούς από το πληκτρολόγιο, θα καλεί μια συνάρτηση η οποία θα εμφανίζει τους αριθμούς Armstrong σε αυτό το διάστημα. Επίσης να χρησιμοποιηθεί μια δεύτερη συνάρτηση η οποία θα υπολογίζει τη δύναμη. Οι αριθμοί Armstrong είναι π.χ. 7=7^1, 371=3^3+7^3+1^3

8 Να γίνει πρόγραμμα το οποίο θα διαβάζει έναν θετικό ακέραιο αριθμό από το πληκτρολόγιο και θα καλεί μια συνάρτηση η οποία θα τυπώνει: στην πρώτη γραμμή το άθροισμα 1+2+3+..,+N, όπου N ο αριθμός, στη δεύτερη γραμμή το άθροισμα 1+2+3+.+N-1 κοκ.


val data1 = PmData("AAAA",
val data2 = PmData("BBBB",
val data3 = PmData("CCCC",
val arrayList = ArrayList
arrayList.add(data1)
arrayList.add(data2)
arrayList.add(data3)
adapter.setFeedAdapter

7. Πίνακες

Πολλές φορές χρειάζεται να χειριστούμε τιμές που είναι σχετικές μεταξύ τους. Για παράδειγμα θέλουμε να καταχωρήσουμε τους βαθμούς δέκα μαθητών για ένα μάθημα. Προκειμένου να δοθεί η δυνατότητα να καταχωρηθούν οι βαθμοί κανονικά θα έπρεπε να έχουν δεσμευτεί δέκα διαφορετικές μεταβλητές, μια για κάθε βαθμό. Κάτι τέτοιο δεν είναι καθόλου ευέλικτο και θα ήταν εξαιρετικά δύσκολο αν μιλούσαμε για μεγαλύτερο πλήθος βαθμών. Σε αυτές τις περιπτώσεις χρησιμοποιούνται οι πίνακες καθώς αποτελούν ένα σύνολο μεταβλητών ίδιου τύπου. Η χρήση πινάκων παρέχει έναν διαφορετικό τρόπο χειρισμού της μνήμης στις οποίες αναφερόμαστε με ένα κοινό όνομα. Κάθε πίνακας αποτελείται από διαδοχικές θέσεις μνήμης, όπου η θέση μνήμης με τη χαμηλότερη διεύθυνση περιέχει το πρώτο στοιχείο του πίνακα. Η διάκριση των θέσεων μνήμης ή απλά των θέσεων του πίνακα γίνεται με τη χρήση κάποιου δείκτη. Οι πίνακες μπορεί να είναι μονοδιάστατοι, δισδιάστατοι ή και πολυδιάστατοι.

7.1 Μονοδιάστατοι Πίνακες

Ένας μονοδιάστατος πίνακας μπορεί να αναπαρασταθεί από ένα σύνολο θέσεων μνήμης όπου η μια βρίσκεται κάτω από την άλλη και όπου κάθε θέση διαθέτει έναν διαφορετικό αριθμό αναφοράς, ξεκινώντας από τον αριθμό 0. Αυτό σημαίνει πως αν δεσμευτεί ένας πίνακας δέκα θέσεων, τότε η πρώτη θέση είναι αυτή με δείκτη 0 και η τελευταία αυτή με δείκτη 9. Στη δήλωση ενός πίνακα εκτός

από το πλήθος των θέσεων που θα περιλαμβάνει πρέπει να προσδιορίσουμε και τον τύπο των δεδομένων που θα καταχωρούνται. Η δήλωση ενός πίνακα γίνεται ως εξής:

x[0]	
x[1]	
x[2]	
x[3]	
x[4]	
x[5]	
x[6]	
x[7]	
x[8]	
x[9]	

τύπος_δεδομένων όνομα_πίνακα[μέγεθος];

Για παράδειγμα:

int x[10];

Με την παραπάνω πρόταση δηλώθηκε ένας πίνακας δέκα θέσεων στις οποίες μπορούν να εκχωρηθούν αυστηρά μόνο ακέραιες τιμές.

Η πρώτη θέση του πίνακα είναι αυτή με δείκτη 0 και η τελευταία αυτή με δείκτη 9. Ο αριθμός που βρίσκεται μέσα στην εκάστοτε αγκύλη ονομάζεται δείκτης.

Η προσπέλαση και καταχώρηση του αριθμού σε μια θέση του πίνακα, για παράδειγμα της 5ης θέσης, γίνεται με την επόμενη πρόταση:

x[4] = 20; // η 5η θέση του πίνακα είναι αυτή με δείκτη 4

Η προσπέλαση και εμφάνιση της ίδιας θέσης του πίνακα, γίνεται με την επόμενη πρόταση:

printf(" Η θέση %d περιέχει την τιμή %d", 4 + 1, x[4]);

Κατά τη δήλωση ενός πίνακα μπορεί να γίνει και η αρχικοποίησή του, θέτοντας σε άγκιστρα τις τιμές που θα περιλαμβάνει ως εξής:

int x[5]={1,4,2,3,1}; // δηλώθηκε ένας πίνακας 5 θέσεων αρχικοποιημένος με στοιχεία

Η προσπέλαση όλων των θέσεων ενός πίνακα γίνεται χρησιμοποιώντας μια βοηθητική μεταβλητή και μια δομή επανάληψης.

Παράδειγμα

```
#include <stdio.h>
int main( )
{
        int table[10], i;
        for (i = 0 ; i < 10; i++)
                table[i] = 20;
        for (i = 0 ; i < 10; i++)
                printf("Η θέση %d περιέχει την τιμή %d\n", i + 1, table[i]);
        return 0;
}
```

Στο προηγούμενο παράδειγμα έχει χρησιμοποιηθεί μια for που θα κάνει δέκα επαναλήψεις. Ταυτόχρονα έχουμε αξιοποιήσει τη βοηθητική μεταβλητή της for, δηλαδή τη μεταβλητή i, ώστε να πάρει τιμές τέτοιες ώστε σε κάθε επανάληψη να προσπελάζεται διαφορετική θέση του πίνακα table. Η μεταβλητή I θα πάρει τιμές 0, 1, ..., 9 που είναι και όλοι οι δείκτες του πίνακα. Όταν αναφερόμαστε στη μεταβλητή i, τότε αναφερόμαστε απλά σε ένα αριθμό, όμως όταν αναφερόμαστε στο table[i], τότε αναφερόμαστε στο στοιχείο της i-οστής θέσης του πίνακα table.

Θα πρέπει να δοθεί ιδιαίτερη προσοχή στις τιμές που επιτρέπουμε τη μεταβλητή i να πάρει καθώς μέσω αυτής προσπελάζεται ο πίνακας. Αυτό σημαίνει πως αν η μεταβλητή i πάρει τιμή 10, τότε προσπελάζεται η 11η θέση του πίνακα. Σε αυτή την περίπτωση πρόκειται για μια θέση μνήμης που δεν είναι δεσμευμένη για τον πίνακα και αντ' αυτού να είναι δεσμευμένη για κάποια άλλη μεταβλητή. Έτσι μπορεί να χαθούν χρήσιμα δεδομένα. Σε άλλες γλώσσες προγραμματισμού κάτι τέτοιο δεν θα επιτρεπόταν και θα εμφανιζόταν σχετικό μήνυμα.

Στο παρακάτω παράδειγμα υπολογίζεται το άθροισμα των στοιχείων ενός πίνακα 10 θέσεων.

Παράδειγμα

```
#include <stdio.h>
int main( )
{
        int table[10], i, sum = 0;
        for (i = 0; i < 10; i++)
        {
                printf("Δώσε το table[%d] στοιχείο : ",i + 1);
                scanf("%d",&table[i]);
                sum += table[i];
        }
        printf("Το άθροισμα των στοιχείων του πίνακα είναι %d.\n",sum);
        return 0;
}
```

Άσκηση

Να γράψετε πρόγραμμα το οποίο να διαβάζει 10 αριθμούς και να τους τοποθετεί σε έναν μονοδιάστατο πίνακα. Στη συνέχεια να εμφανίζει αυτούς τους αριθμούς με την αντίστροφη σειρά.

```
#include <stdio.h>
int main(void)
{
        int array[5];
        printf(«Δώσε αριθμό:\n»);
        for (int i=0;i<5;i++)
                scanf(«%d»,&array[i]);

        for (int i=4;i>=0;i--)
                printf(«%d\n»,array[i]);
        return 0;
}
```

Άσκηση

Να γραφεί πρόγραμμα που διαβάζει ακέραιους αριθμούς και τους αποθηκεύει σε έναν πίνακα 10 θέσεων μέχρι να δοθεί ως είσοδο ο αριθμός 0 ή να συμπληρωθούν οι 10 θέσεις και να εμφανίζει τον πίνακα με τους αριθμούς.

```
#include <stdio.h>
main(void){
        int array[10], N=0;
        for (int i=0;i<10;i++)
        {
                int arithmos;
                printf(«Δώσε έναν αριθμό:\n»);
                scanf(«%d»,&arithmos);
                if (arithmos==0)
                        break;
                array[i]=arithmos;
                N++;
        }
        for (int i=0;i<N;i++)
                printf(«%d\n»,array[i]);
}
```

7.2 Πίνακες και Συναρτήσεις

Υπάρχει η δυνατότητα να μεταβιβαστεί ένας πίνακας ως όρισμα σε μια συνάρτηση. Παρακάτω χρησιμοποιείται το προηγούμενο παράδειγμα με τη διαφορά ότι θα κληθεί η συνάρτηση sum που θα επιστρέψει το άθροισμα των στοιχείων.

Παράδειγμα

```
#include <stdio.h>
int sum ( int new_table[ ]){
        int result=0;
        for (i = 0; i < 10; i++)
                result += table[i];
        return result;
}
int main( ){
        int table[10], i;
        for (i = 0; i < 10; i++)
        {
                printf("Δώσε το table [%d] στοιχείο : ",i + 1);
                scanf("%d",&table[i]);
        }
        result = sum(table) // κλήση της sum και μεταβίβαση της πρώτης θέσης
        του πίνακα
        printf("Το άθροισμα των στοιχείων του πίνακα είναι %d.\n",result);
}
```

Στο παραπάνω παράδειγμα έχει χρησιμοποιηθεί διαφορετικός τύπος κλήσης της συνάρτησης sum από αυτόν που έχουμε δει μέχρι στιγμής (call by value). Στο συγκεκριμένο παράδειγμα κατά την κλήση της συνάρτησης έγινε αντιγραφή της διεύθυνσης της πρώτης θέσης του πίνακα table στο τυπικό όρισμα της sum, δηλαδή τον new_table που είναι και αυτός πίνακας. Λόγω αυτής της αντιγραφής στον πίνακα που είναι τυπικό όρισμα δεν έχει τεθεί πλήθος των θέσεων, στη συνέχεια όμως γνωρίζουμε ότι μέσω τις for θα προσπελαστούν μόνο οι εννέα επόμενες θέσεις. Σε αυτή την περίπτωση και οι δύο πίνακες, table και new_table, προσπελάζουν τις ίδιες θέσεις μνήμης καθώς χειρίζονται τις ίδιες διευθύνσεις μνήμης. Αυτός ο τρόπος κλήσης ονομάζεται κλήση με αναφορά (call by

reference). Στο παρακάτω παράδειγμα καλείται μια συνάρτηση store η οποία καταχωρεί στοιχεία στον πίνακα της main.

Παράδειγμα

```
#include <stdio.h>
void store ( int new_table[ ]){
        for (i = 0; i < 10; i++)
        {
                printf("Δώσε το table[%d] στοιχείο : ",i + 1);
                scanf("%d",&new_table[i]);
        }
}
main( ){
        int table[10], i;
        store(table) // θα ήταν ίδια η πρόταση store(table[0])
        for (i = 0; i < 10; i++)
                printf(" Το %d στοιχείο του πίνακα είναι % d\n : ",i + 1, table[i]);
}
```

Στο προηγούμενο παράδειγμα η καταχώρηση στοιχείων του πίνακα table γίνεται με χρήση της store, η οποία χειρίζεται τον πίνακα new_table. Έχει γίνει κλήση με αναφορά επομένως η scanf που χρησιμοποιείται στη συνάρτηση ουσιαστικά καταχωρεί στοιχεία στον πίνακα table.

7.3 Πίνακες και Αλφαριθμητικά

Στη C ένα αλφαριθμητικό (συμβολοσειρά - string) είναι ένα σύνολο χαρακτήρων, καθώς δεν υποστηρίζεται συγκεκριμένος τύπος μεταβλητής για αλφαριθμητικά. Προκειμένου να καταχωρηθεί ένα αλφαριθμητικό πρέπει να γίνει χρήση ενός μονοδιάστατου πίνακα χαρακτήρων, όπου σε κάθε θέση του πίνακα θα κατα-

χωρείται και ένας χαρακτήρας του αλφαριθμητικού. Πάντα στην τελευταία θέση κάθε πίνακα χαρακτήρων υπάρχει το '\0' υποδεικνύοντας ότι εκεί τελειώνει το αλφαριθμητικό. Κατά τη δήλωσή του, ένας πίνακας χαρακτήρων μπορεί και να αρχικοποιηθεί ως εξής:

char name[100] = "Robert Plant";

Στην προηγούμενη περίπτωση δηλώθηκε ο πίνακας χαρακτήρων name με 100 διαθέσιμες θέσεις. Το σκεπτικό είναι ότι δεν γνωρίζουμε εξαρχής ποιο θα είναι το πλήθος των χαρακτήρων που θα καταχωρηθούν, επομένως δεσμεύουμε αρκετές θέσεις. Σε κάθε περίπτωση εφόσον καταχωρηθούν οι χαρακτήρες, ο πίνακας name δεν θα προσπελάζεται μέχρι και την τελευταία θέση (name[99]), καθώς ενδεχομένως κάποιες θέσεις να μην έχουν αξιοποιηθεί, αλλά θα προσπελάζεται μέχρι να εντοπισθεί το '\0'.

Εναλλακτικά υπάρχει η δυνατότητα δέσμευσης λιγότερων θέσεων και σε περίπτωση που χρειαστούν περισσότερες από αυτές που ήδη έχουν δεσμευθεί να προχωρήσουμε σε δυναμική δέσμευση μνήμης. Αυτή η μέθοδος έχει το πλεονέκτημα ότι περιορίζονται αρκετά οι θέσεις που δεσμεύονται και εν τέλει δεν αξιοποιούνται. Η δήλωση του πίνακα name δεσμεύει 100 θέσεις από τις οποίες αξιοποιούνται μόνο οι 13, με αποτέλεσμα όλες οι υπόλοιπες να μένουν αναξιοποίητες, άρα να σπαταλώνται.

Η καταχώρηση ενός αλφαριθμητικού από το πληκτρολόγιο γίνεται ως εξής:

scanf("%s", &name) // θα ήταν ίδια η πρόταση scanf("%s", &name[0]);

Η C χειρίζεται τα αλφαριθμητικά γνωρίζοντας μόνο τη διεύθυνση του πρώτου χαρακτήρα, δηλαδή της πρώτης θέσης μνήμης. Το τέλος αναγνωρίζεται με το '\0'. Το %s (string) χρησιμοποιείται για να καταχωρηθούν χαρακτήρες. Η εμφάνιση ενός αλφαριθμητικού γίνεται ως εξής:

printf(" Το όνομα είναι %s", name);

Για την καταχώρηση και την εμφάνιση ενός αλφαριθμητικού μπορούν να χρησιμοποιηθούν οι συναρτήσεις gets( ) και puts( ) αντίστοιχα.

Παράδειγμα

```
#include <stdio.h>
#include <string.h>
main( ){
        char name[100];
        printf("Δώσε το όνομά σου : ");
        gets(name);
        printf("Το ονομά σου είναι ");
        puts(name);
}
```

Στο παραπάνω παράδειγμα χρησιμοποιούμε την **gets( )** για να διαβάσουμε ένα αλφαριθμητικό και την **puts( )** για να το εμφανίσουμε στην οθόνη. Η gets( ) περιμένει να πληκτρολογηθούν όλοι οι χαρακτήρες και μόλις πατηθεί ENTER τους αποθηκεύει έναν έναν σε διαφορετική θέση του πίνακα ξεκινώντας από την πρώτη. Η gets ( ) τοποθετεί αυτόματα το '\0' στην τελευταία θέση του πίνακα. Η puts( ) αντίστοιχα εμφανίζει έναν έναν τους χαρακτήρες ξεκινώντας από τη διεύθυνση που δόθηκε μέχρι να εντοπιστεί το '\0'.

7.4 Συναρτήσεις Χειρισμού Αλφαριθμητικών

Η βιβλιοθήκη string.h προσφέρει αρκετές συναρτήσεις χειρισμού αλφαριθμητικών.

Η συνάρτηση strlen( )

Η συνάρτηση **strlen( )** επιστρέφει το μήκος του αλφαριθμητικού που δίνεται ως όρισμα. Η συνάρτηση δεν μετράει το '\0'.

Παράδειγμα

```
#include <stdio.h>
#include <string.h>
main( )
{
        char name[100];
        int length;
        printf("Δώσε το όνομα σου : ");
        gets(name);
        length = strlen(name);
        printf("Το μήκος του ονόματος σου είναι %d\n",length);
}
```

Η συνάρτηση strcpy( )

Η συνάρτηση **strcpy( )** χρησιμοποιείται για την αντιγραφή των περιεχομένων ενός πίνακα χαρακτήρων σε ένα δεύτερο. Ο δεύτερος πίνακας πρέπει να διαθέτει αρκετές θέσεις ώστε να χωρέσει τους χαρακτήρες του πρώτου.

Παράδειγμα

```
#include <stdio.h>
#include <string.h>
int main()
{
        char name1[100] = "Robert Plant" ,name2[100];
        printf("Δώσε το νέο όνομα  : ");
        gets(name2);  // καταχωρείται το Jimmy Page
        strcpy(name1, name2);
        puts(name1); // εμφανίζεται το Jimmy Page και όχι το αρχικό (Robert Plant)
        return 0;
}
```

Η συνάρτηση strcat( )

Η συνάρτηση strcat( ) αντιγράφει τους χαρακτήρες ενός πίνακα στο τέλος ενός δεύτερου πίνακα, χωρίς να επηρεάζεται ο αρχικός πίνακας.

Παράδειγμα

```
#include <stdio.h>
#include <string.h>
main( ){
        char table1[100]= “Led  ”, table2[50] = “Zeppelin ”;
        strcat(table1, table2);
        puts(table1);  // Εμφανίζεται Led Zeppelin
}
```

Η συνάρτηση strcmp( )

Η συνάρτηση strcmp( ) συγκρίνει δύο πίνακες χαρακτήρων και επιστρέφει:

α. τιμή = 0, αν τα δύο αλφαριθμητικά είναι ίσα,

β. τιμή > 0, αν το πρώτο αλφαριθμητικό είναι μεγαλύτερο από το δεύτερο,

γ. τιμή < 0, αν το πρώτο αλφαριθμητικό είναι μικρότερο από το πρώτο.

Παράδειγμα

```
#include <stdio.h>
#include <string.h>
int main()
{
        char password[] = “1212”;
        char code[15];
        do
        {
                printf(“Δώσε τον κωδικό : ”);
```

```
                gets(code);
        } while (strcmp(code,password)); // n while τερματίζει όταν n strcmp
επιστρέψει 0
        return 0;
}
```

Παράδειγμα

```
#include <stdio.h>
#include <string.h>
int main()
{
        char string[80];
        while (1)  //ατέρμων βρόχος
        {
                printf("Δώσε μια συμβολοσειρά : ");
                gets(string);
                if (! strcmp(string,"έξοδος"))
                        break;
        }
        return 0;
}
```

Στο παραπάνω παράδειγμα δίνεται συνέχεια μια συμβολοσειρά μέχρι να δοθεί η συμβολοσειρά "έξοδος". Η διακοπή του βρόχου θα συμβεί όταν εκτελεστεί το if και κατ' επέκταση το break. Όμως η if εκτελείται όταν η πρόταση είναι αληθής. Από την άλλη, η strcmp επιστρέφει το 0 όταν πατηθεί η συμβολοσειρά "έξοδος", γι'αυτό χρησιμοποιείται η άρνηση στην πρόταση strcmp(string,"έξοδος").

Άσκηση

Να γίνει πρόγραμμα το οποίο θα διαβάζει μια πρόταση και θα καλεί μια συνάρτηση που θα μετατρέπει

- τα πεζά γράμματα σε κεφαλαία
- τα κεφαλαία γράμματα σε πεζά

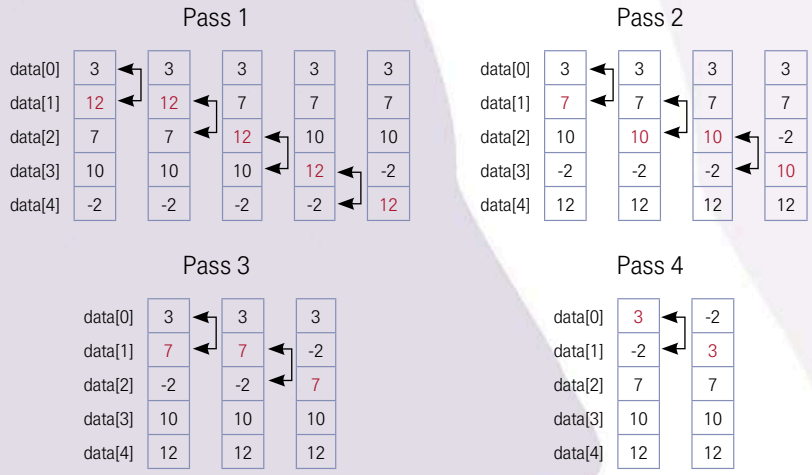
```
#include <stdio.h>
void upper(char x[ ]){
        int i=0;
        while (x[i]!='\0'){
                if (x[i]>='a' && x[i]<='z')
                        x[i]-=32;
                i++;
        }
}
void lower(char x[ ]){
        int i=0;
        while (x[i]!='\0'){
                if (x[i]>='A' && x[i]<='Z')
                        x[i]+=32;
                i++;
        }
}
main (void){
        char s[100];
        puts("Δώσε μια πρόταση");
        gets(s);
        upper(s);
        puts(s);
}
```

7.5 Ταξινόμηση

Η ταξινόμηση είναι μια από τις πιο βασικές λειτουργίες σε ένα πίνακα, δηλαδή η επανατοποθέτηση των στοιχείων ενός πίνακα σε αύξουσα ή φθίνουσα διάταξη. Υπάρχουν διάφοροι αλγόριθμοι που υλοποιούν τη διαδικασία της ταξινόμησης, όπου καθένας από αυτούς ακολουθεί διαφορετική προσέγγιση, έχοντας και διαφορετική πολυπλοκότητα. Παρακάτω θα αναλυθούν μερικοί από τους πιο βασικούς αλγορίθμους ταξινόμησης.

7.5.1 Ταξινόμηση Φυσαλίδας (Bubble Sort)

Ο αλγόριθμος φυσαλίδας ταξινομεί ένα πίνακα ελέγχοντας ανά ζεύγη τα στοιχεία, δηλαδή το πρώτο με το δεύτερο, το δεύτερο με το τρίτο κ.ο.κ. και αν εντοπιστεί κάποιο ζεύγος να έχει διαφορετική διάταξη από την επιθυμητή, αντιμεταθέτει τα στοιχεία. Δηλαδή, αν επιθυμούμε αύξουσα ταξινόμηση και βρεθεί κάποιο ζεύγος στοιχείων όπου το πρώτο είναι μεγαλύτερο από το δεύτερο τότε αντιμετατίθενται. Σε διαφορετική περίπτωση ο αλγόριθμος προχωράει στον έλεγχο του επόμενου ζεύγους. Ο αλγόριθμος εκτελεί κάποιες επαναλήψεις που ονομάζονται περάσματα, για την ακρίβεια χρειάζονται Ν-1 περάσματα (όπου Ν είναι το πλήθος των στοιχείων του πίνακα). Αυτό που επιτυγχάνεται σε κάθε πέρασμα είναι να τοποθετείται στο τέλος το μεγαλύτερο στοιχείο από αυτά που ελέγχθηκαν. Δηλαδή κατά το πρώτο πέρασμα ελέγχονται όλα τα στοιχεία και με τη διαδικασία της αντιμετάθεσης είναι δεδομένο πως το μέγιστο στοιχείο θα τοποθετηθεί στο τέλος του πίνακα. Κατά το δεύτερο πέρασμα θα επαναληφθεί η ίδια διαδικασία χωρίς όμως να ελεγχθεί το τελευταίο στοιχείο του πίνακα, εφόσον σε αυτό ήδη βρίσκεται το μέγιστο στοιχείο. Αυτό σημαίνει πως θα τοποθετηθεί στην προτελευταία θέση το δεύτερο μεγαλύτερο στοιχείο. Με τον ίδιο τρόπο θα λειτουργήσουν και τα υπόλοιπα περάσματα, κάνοντας κάθε φορά έναν λιγότερο έλεγχο. Στην παρακάτω εικόνα φαίνεται η λειτουργία του αλγορίθμου σε ένα πίνακα 5 στοιχείων.



Εικόνα 7.1 Εκτέλεση Αλγορίθμου BubbleSort

Παρακάτω παρατίθεται η συνάρτηση που υλοποιεί την ταξινόμηση φυσαλίδας.

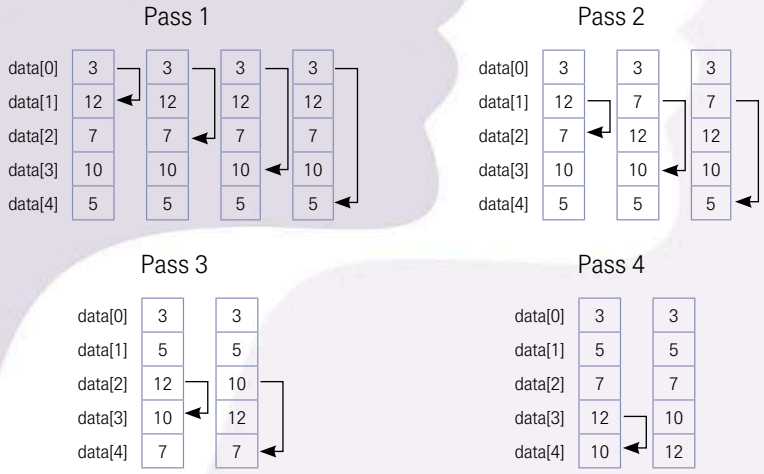
```
Void bubblesort ( int x[], int n)
{
        int temp;
        for (int i=0; i<n ; i++)
        {
                for (int j=0; j<n-i-1; j++) // n – i – 1, γίνονται κατά ένα λιγότεροι έλεγχοι
                {
                if (x[j] > x[j+1]) // έλεγχος λάθος διάταξης στοιχείων
                        {
                                temp = x[j];
                                x[j] =  x[j+1];
                                x[j] = temp;
                        }
                }
        }
}
```

} Σε αυτό το σημείο γίνεται η αντιμετάθεση των στοιχείων

7.5.2 Ταξινόμηση Επιλογής (Selection Sort)

Ο αλγόριθμος επιλογής ακολουθεί διαφορετική προσέγγιση από τον αλγόριθμο φυσαλίδας, ωστόσο η λογική των περασμάτων παραμένει ίδια. Αρχικά ελέγχεται όλος ο πίνακας ώστε να εντοπιστεί το μικρότερο στοιχείο, το οποίο τοποθετείται στην πρώτη θέση. Αντίστοιχα το στοιχείο που βρισκόταν στην πρώτη θέση τοποθετείται στη θέση όπου βρέθηκε να είναι το μικρότερο στοιχείο. Στο επόμενο πέρασμα επαναλαμβάνεται η ίδια διαδικασία αγνοώντας όμως την πρώτη θέση του πίνακα. Δηλαδή θα εντοπιστεί το αμέσως μικρότερο στοιχείο και θα αντικατασταθεί με το στοιχείο της δεύτερης θέσης του πίνακα. Ο έλεγχος εφαρμόζεται κάθε φορά σε ολοένα και μικρότερο τμήμα του πίνακα. Η διαδικασία θα ολοκληρωθεί όταν εκτελεστούν τα απαιτούμενα περάσματα.

Στην παρακάτω εικόνα φαίνεται η λειτουργία του αλγορίθμου σε ένα πίνακα 5 στοιχείων καθώς και η εκτέλεση των 4 περασμάτων που απαιτούνται.



Εικόνα 7.2 Εκτέλεση Αλγορίθμου SelectionSort

Παρακάτω παρατίθεται η συνάρτηση που υλοποιεί την ταξινόμηση επιλογής.

```
void selectionSort(int x[ ],int N){
        int min, minpos, temp;
        for (int i=0;i<N-1;i++){
                min =x[i];  //αρχικοποίηση min με κάποιο στοιχείο του πίνακα
                minpos =i; //αρχικοποίηση θέσης όπου βρέθηκε το min στοιχείο
                for (int j = i ; j <= N-1 ; j++){
                        if (x[j] < min){
                                min =x[j];
                                minpos =j;
                        }
                }
                temp=x[i];
                x[i]=min;
                x[minpos]=temp;
        }
}
```

Σε αυτό το σημείο γίνεται η αντιμετάθεση των στοιχείων

7.6 Αναζήτηση Στοιχείου

Η αναζήτηση κάποιου στοιχείου σε έναν πίνακα είναι εξίσου σημαντική λειτουργία με αυτή της ταξινόμησης. Στην αναζήτηση δίνεται μια τιμή (κλειδί – key) ώστε να εντοπιστεί σε ποια θέση του πίνακα βρίσκεται. Φυσικά, υπάρχει και το ενδεχόμενο το στοιχείο προς αναζήτηση να μην υπάρχει στον πίνακα. Υπάρχουν δύο μέθοδοι αναζήτησης στοιχείου, ο πρώτος τρόπος είναι η σειριακή αναζήτηση και ο δεύτερος η δυαδική αναζήτηση, όπου και οι δύο μέθοδοι αναλύονται παρακάτω.

7.6.1 Σειριακή Αναζήτηση

Η σειριακή αναζήτηση ακολουθεί μια απλή λογική, θα ελέγξει όλα τα στοιχεία του πίνακα ένα προς ένα προκειμένου να εντοπιστεί το κλειδί. Παρακάτω παρατίθεται η συνάρτηση που υλοποιεί τη δυαδική αναζήτηση:

```
int serial_search(int x[ ],int N, int key){
        int position = -1;
        for (int i = 0;i < N ; i++){
                if ( x[i] == i){
                        position = i;
break;
}
        }
        return position;
}
```

Στην προηγούμενη συνάρτηση N είναι το μέγεθος του πίνακα και key το στοιχείο προς αναζήτηση. Ο αλγόριθμος ελέγχει όλα τα στοιχεία του πίνακα και όταν το key εντοπιστεί τότε εκχωρείται η θέση στην οποία βρέθηκε (i) στη μεταβλητή position, η οποία και επιστρέφεται. Σε περίπτωση που δεν βρεθεί το key, τότε επιστρέφεται η τιμή -1 που υποδηλώνει ότι το key δεν βρίσκεται στον πίνακα.

7.6.2 Δυαδική Αναζήτηση

Ο αλγόριθμος δυαδικής αναζήτησης είναι πιο αποδοτικός από αυτόν της σειριακής, ωστόσο το μειονέκτημά του είναι ότι μπορεί να εφαρμοστεί μόνο σε ταξινομημένους πίνακες. Ο αλγόριθμος συγκαταλέγεται στην κατηγορία των αλγορίθμων διαίρει και βασίλευε (divide and conquer), καθώς λειτουργεί επαναληπτικά χωρίζοντας τον πίνακα σε δύο τμήματα επιλέγοντας μόνο το ένα από τα δύο για έλεγχο.

Αρχικά ο αλγόριθμος θέτει ένα δείκτη στην αρχή και ένα στο τέλος του πίνακα, έπειτα ελέγχει το μέσο στοιχείο του πίνακα αν είναι ίσο με το key. Σε περίπτωση που ο έλεγχος είναι αληθής, το στοιχείο βρέθηκε και τελειώνει η διαδικασία. Αν όμως ο έλεγχος είναι ψευδής και το key είναι μεγαλύτερο από το μέσο στοιχείο, τότε ο δείκτης της αρχής τοποθετείται στην επόμενη θέση από το μέσο. Η διαδικασία συνεχίζει βρίσκοντας το νέο μέσο, καθώς ο δείκτης της αρχής έχει μετακινηθεί, και ελέγχοντας το νέο μέσο με το κλειδί. Αν το κλειδί είναι μικρότερο από το νέο μέσο τότε ο δείκτης τέλους μετακινείται μια θέση πριν το μέσο και επαναϋπολογίζεται το νέο μέσο. Η διαδικασία συνεχίζεται όσο ο δείκτης της αρχής είναι μικρότερος από τον δείκτη τέλους ή εάν κάποια στιγμή βρεθεί το key.

Στο παρακάτω παράδειγμα το στοιχείο προς εντοπισμό είναι το 135. Στην αρχή ο δείκτης αρχής (A) είναι στη θέση 0 και ο δείκτης τέλους (E) είναι στη θέση 9, η διαδικασία ξεκινάει ελέγχοντας τον μέσο (A + E) / 2 = 4. Με αυτό τον τρόπο ο πίνακας χωρίζεται σε δύο τμήματα, το πρώτο είναι από την αρχή έως το μέσο και το δεύτερο από το μέσο έως το τέλος. Ο μέσος είναι το 69, επομένως το 135 σίγουρα δεν βρίσκεται στο πρώτο τμήμα αλλά στο δεύτερο, εφόσον ο πίνακας είναι ταξινομημένος. Στη συνέχεια ο δείκτης Α τοποθετείται στη θέση 5 (μια θέση μετά τον μέσο) ενώ ο δείκτης Ε παραμένει σταθερός και επαναϋπολογίζεται ο νέος μέσος που είναι το στοιχείο στη θέση (5 + 9) / 2 = 7. Ο νέος μέσος είναι το 180 που είναι μεγαλύτερο από το 135. Άρα η διαδικασία συνεχίζεται διατηρώντας τον δείκτη Α στη θέση του (A=5) και τοποθετώντας τον δείκτη Ε μια θέση πριν το μέσο (E=6), υπολογίζεται το νέο μέσο (5 + 6) / 2 = 5. Το στοιχείο του νέου μέσου, δηλαδή το στοιχείο της θέσης 5, το 100, είναι μικρότερο από το 135. Άρα ο δείκτης Ε παραμένει σταθερός (E=6) ενώ ο δείκτης της αρχής τοποθετείται στην επόμενη θέση από τον μέσο (A=6) και υπολογίζεται το νέο μέσο (6 + 6) / 2 = 6. Στον τελευταίο έλεγχο υπάρχει ταύτιση του key με τον μέσο, οπότε το στοιχείο προς αναζήτηση βρέθηκε στη θέση του μέσου.

	Πρώτος έλεγχος		Δεύτερος έλεγχος		Τρίτος έλεγχος		Τέταρτος έλεγχος	
a[0]	7	A	7		7		7	
a[1]	12		12		12		12	
a[2]	20		20		20		20	
a[3]	48		48		48		48	
a[4]	69	M	69		69		69	
a[5]	100		100	A	100	A/M	100	
a[6]	135		135		135	E	135	A/M/E
a[7]	180		180	M	180		180	
a[8]	196		196		196		196	
a[9]	205	E	205	E	205		205	

Εικόνα 7.3 Εκτέλεση Δυαδικής Αναζήτησης και Εύρεση Κλειδιού

Παρακάτω παρατίθεται η συνάρτηση που υλοποιεί τη δυαδική αναζήτηση.

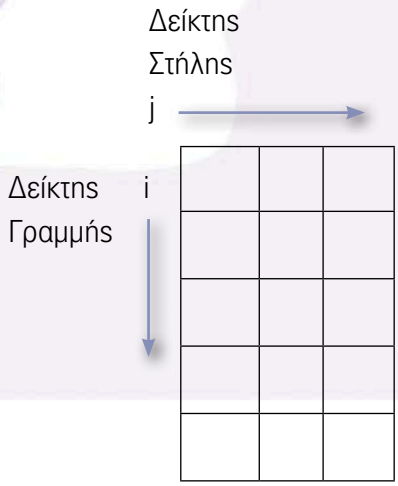
```
void  binary(int a[10])
{
        int apo=0,eos=9;
        int flag=0;         //το flag παίρνει τιμή 1 όταν βρεθεί το key
        int middle;        // ο δείκτης της μέσης
        int value=5;
        while (!flag && apo<=eos)
        {
                middle=(apo+eos)/2;
                if (a[middle]==value)
                        flag=1;
                if (a[middle]>value)
                        eos=middle-1;
                if (a[middle]<value)
                        apo=middle+1;
        }
```

```
        if (flag)
                printf(«Το key βρέθηκε στη θέση:%d»,middle);
        else
                printf(«Το key δεν βρέθηκε»);
}
```

7.7 Δισδιάστατοι Πίνακες

Οι δισδιάστατοι πίνακες μπορούν να θεωρηθούν ως ένα σύνολο θέσεων χωρισμένες σε γραμμές και στήλες. Σε αυτή την περίπτωση κάθε θέση του πίνακα προσδιορίζεται από έναν δείκτη γραμμής και από έναν δείκτη στήλης, όπου για την προσπέλαση της εκάστοτε θέσης απαιτούνται και οι δύο θέσεις. Προκειμένου να προσπελαστεί ολόκληρος ο δισδιάστατος πίνακας πρέπει να γίνει χρήση μιας εμφωλευμένης δομής επανάληψης τύπου for και ανάλογα με τις βοηθητικές μεταβλητές που θα χρησιμοποιηθούν στις δομές επανάληψης και στους δείκτες του πίνακα η προσπέλαση γίνεται ανά γραμμές ή ανά στήλες.

Η δήλωση ενός δισδιάστατου πίνακα απαιτεί δύο αγκύλες, όπου η πρώτη υποδεικνύει το πλήθος των γραμμών και η δεύτερη το πλήθος των στηλών. Παρακάτω δηλώνεται ένας πίνακας 5 γραμμών και 3 στηλών, δηλαδή στο σύνολο θα διαθέτει 15 θέσεις.



int table[5][3];

Στο επόμενο παράδειγμα γίνεται η καταχώρηση στοιχείων σε έναν πίνακα 5 γραμμών και 3 στηλών ανά γραμμές. Δηλαδή προσπελάζοντας όλες τις θέσεις μιας γραμμής και μετά πηγαίνοντας στην επόμενη.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int i, j, table[5][3];
        for(i = 0; i < 5; i++)
        {
                for (j = 0; j < 3; j++)
                        scanf("%d", &table[i][j]);
        }
        return 0;
}
```

Στο επόμενο παράδειγμα πάλι θα γίνει η καταχώρηση στοιχείων στον ίδιο πίνακα αλλά προσπελάζοντας τις θέσεις ανά στήλη. Συνήθως ο δείκτης των γραμμών είναι ο i και των στηλών ο j αλλά αυτό δεν είναι υποχρεωτικό. Σε κάθε περίπτωση ο δείκτης της πρώτης αγκύλης καθορίζει τις γραμμές και ο δείκτης της δεύτερης αγκύλης τις στήλες.

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int i, j, table[5][3];
        for(j = 0; j < 3; j ++)
        {
                for (i = 0; i < 5; i++)
                        scanf("%d", &table[i][j]);
        }
        return 0;
}
```

Παράδειγμα

Δίνεται πίνακας Α δύο διαστάσεων, που τα στοιχεία του είναι ακέραιοι αριθμοί με Ν γραμμές και Μ στήλες. Να γραφεί πρόγραμμα που να υπολογίζει το ελάχιστο στοιχείο του πίνακα. Θεωρείται ότι ο πίνακας έχει ήδη καταχωρημένα στοιχεία.

```
#include <stdio.h>
int main(void)
{
        int pinakas[100][100],x,y,min;
        min=pinakas[0][0];
        for (int i=0;i<x;i++)
        {
                for (int j=0;j<y;j++)
                {
                        if (pinakas[i][j]<min)
                                min=pinakas[i][j];
                }
        }
        printf("Το μικρότερο στοιχείο του πίνακα είναι %d",min);
        return 0;
}
```

Άσκηση

Μια αλυσίδα ξενοδοχείων έχει 5 ξενοδοχεία. Σε έναν μονοδιάστατο πίνακα Α[5] καταχωρούνται τα ονόματα των ξενοδοχείων. Σε έναν άλλο δισδιάστατο πίνακα C[5,3] καταχωρούνται οι εισπράξεις κάθε ξενοδοχείου για κάθε μήνα του πρώτου τριμήνου του έτους 2019, έτσι, ώστε στη i γραμμή καταχωρούνται οι εισπράξεις του i ξενοδοχείου. Να γραφεί πρόγραμμα, το οποίο:

α. διαβάζει τα στοιχεία των δύο πινάκων,

β. εκτυπώνει το όνομα κάθε ξενοδοχείου και τις τριμηνιαίες εισπράξεις του για το έτος 2019,

γ. εκτυπώνει το όνομα του ξενοδοχείου με τις μεγαλύτερες εισπράξεις για το πρώτο τρίμηνο του έτους 2019.

```
#include <stdio.h>
#include <windows.h>
main(void){
        char name[5];
        int revenue[5][3], max = 0,sum = 0, position;
        for (int i=0; i<5; i++){
                printf("Δώσε την ονομασία για το %do ξενοδοχείο\n",i+1);
                scanf(«%c»,&name[i]);
        }
        for (int i=0; i<5; i++){
                for (int j=0; j<3; j++){
                        printf("Δώσε τις εισπράξεις για τον %do μήνα\n",j+1);
                        scanf(«%d»,&revenue[i][j]);
                        sum+= revenue[i][j];
                }
                system(«cls»);
                printf("Οι εισπράξεις του ξενοδοχείου %c είναι %d\n",name[i],sum);
                system(«cls»);
                if (sum > max){
                        max = sum;
                        position = i;
                }
                sum = 0;
        }
        printf("Τις περισσότερες εισπράξεις είχε το %c ξενοδοχείο\n", name[position]);
}
```

7.8 Ασκήσεις

1. Να γίνει πρόγραμμα το οποίο θα δέχεται από τον χρήστη το μέγεθος ενός πίνακα καθώς και τους ακέραιους αριθμούς που περιλαμβάνει. Στη συνέχεια θα μπορεί ο χρήστης να καταχωρεί έναν νέο ακέραιο σε οποιαδήποτε θέση του πίνακα επιθυμεί.
2. Να γίνει πρόγραμμα το οποίο θα δέχεται από τον χρήστη το μέγεθος ενός πίνακα καθώς και τους ακέραιους αριθμούς που περιλαμβάνει. Στη συνέχεια ο χρήστης θα δίνει τη θέση του στοιχείου που θέλει να διαγραφεί. Θα γίνεται έλεγχος αν η διαγραφεί είναι εφικτή (π.χ. να διαγραφεί το έκτο στοιχείο ενός πενταθέσιου πίνακα)
3. Να γίνει πρόγραμμα το οποίο θα δέχεται έναν ακέραιο αριθμό από τον χρήστη και θα ψάχνει να βρει αυτό το στοιχείο, σε περίπτωση που υπάρχει θα εμφανίζεται η θέση αυτού του στοιχείου στον πίνακα. Να υπάρχει η δυνατότητα εμφάνισης των θέσεων σε περίπτωση που το στοιχείο υπάρχει πολλαπλές φορές και να εμφανίζεται το πλήθος των φορών που πιθανώς να εμφανίστηκε το στοιχείο.
4. Να γραφεί πρόγραμμα το οποίο:
   α. θα εισάγει τις τιμές των στοιχείων ενός μονοδιάστατου πίνακα 10 ακεραίων αριθμών,
   β. θα υπολογίζει και θα τυπώνει τη μέση τιμή των στοιχείων του πίνακα,
   γ. θα τυπώνει το πλήθος των στοιχείων του πίνακα που είναι μεγαλύτερα από τη μέση τιμή.
5. Σ' έναν μετεωρολογικό σταθμό πρόκειται να γίνει επεξεργασία των θερμοκρασιακών δεδομένων. Να γραφεί πρόγραμμα το οποίο:
   α. θα δημιουργεί έναν μονοδιάστατο πίνακα 31 θέσεων, όπου θα καταχωρείται η πρωινή ημερήσια θερμοκρασία (ώρα 8 π.μ.) του Ιανουαρίου 2004,

β. θα υπολογίζει και θα τυπώνει στην οθόνη τη μέση πρωινή θερμοκρασία του μήνα Ιανουαρίου,

γ. θα εμφανίζει στην οθόνη τις ημερομηνίες με θερμοκρασία μικρότερη των 0_0 C.

6. Τρία υποκαταστήματα διακινούν 6 εφημερίδες το καθένα. Να καταχωρηθούν οι ποσότητες των εφημερίδων που διακινούν τα υποκαταστήματα σε μία ημέρα και να υπολογιστεί:

α. η συνολική ποσότητα των εφημερίδων που διακινούν όλα τα υποκαταστήματα,

β. η ποσότητα των εφημερίδων που διακινεί το κάθε υποκατάστημα,

γ. η ποσότητα που διακινείται για κάθε εφημερίδα και από τα τρία υποκαταστήματα συνολικά και

δ. η μέγιστη ποσότητα εφημερίδων, ποιο υποκατάστημα τη διακινεί και για ποια εφημερίδα.

7. Στον διαγωνισμό πρόσληψης προσωπικού μιας Τράπεζας έλαβαν μέρος 5 υποψήφιοι οι οποίοι εξετάστηκαν σε τρία μαθήματα. Η βαθμολογία σε κάθε μάθημα ήταν ακέραια στο διάστημα 1 έως 50. Η απόφαση της Τράπεζας ήταν να προσληφθούν οι υποψήφιοι με μέσο όρο βαθμολογίας μεγαλύτερο από 40. Να γραφεί πρόγραμμα, το οποίο:

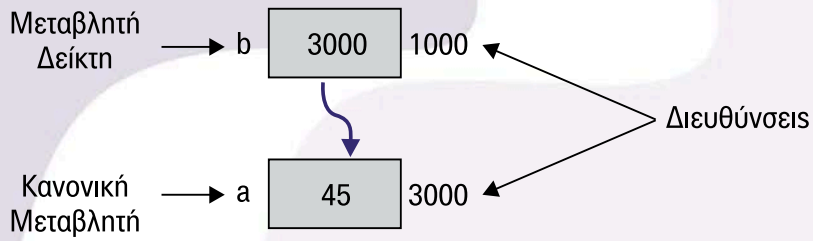
α. θα διαβάζει από το πληκτρολόγιο τους βαθμούς των διαγωνιζομένων σε κάθε μάθημα και θα τους καταχωρεί σε δισδιάστατο πίνακα,

β. θα εμφανίζει τον μέσο όρο βαθμολογίας κάθε διαγωνιζομένου,

γ. θα εμφανίζει το πλήθος των υποψηφίων που προσλαμβάνονται στην Τράπεζα.

8. Δείκτες (Pointers)

Οι δείκτες είναι ένας διαφορετικός τύπος μεταβλητών, που παρουσιάζουν την εξής ιδιομορφία: στη θέση μνήμης κάθε δείκτη βρίσκεται πάντα καταχωρημένη η διεύθυνση μνήμης μιας άλλης μεταβλητής. Γενικά μπορούμε να πούμε ότι ένας δείκτης χρησιμοποιείται ως ένας έμμεσος τρόπος αναφοράς σε μια άλλη θέση μνήμης, δηλαδή σε μια άλλη μεταβλητή. Στο παρακάτω σχήμα φαίνεται κάπως η φιλοσοφία των δεικτών. Η μεταβλητή δείκτη b συνδέεται με την κανονική μεταβλητή a, καθώς η θέση μνήμης της b περιέχει τη διεύθυνση της a. Λέμε πως ο δείκτης b «δείχνει» στη μεταβλητή a.



Εικόνα 8.1 Μεταβλητές Δείκτη και Κανονικές Μεταβλητές

8.1 Δήλωση και αρχικοποίηση μιας μεταβλητής δείκτη

Όπως ήδη έχει ειπωθεί ένας δείκτης είναι μια μεταβλητή, επομένως η δήλωση του γίνεται όπως και μιας οποιασδήποτε άλλης μεταβλητής, με τη διαφορά ότι

πριν από το όνομα του δείκτη πρέπει να υπάρχει το *. Παρακάτω δηλώνονται ορισμένες μεταβλητές δείκτη.

```
int *a;
char *b;
float *k;
```

Προηγουμένως δηλώθηκαν τρεις μεταβλητές δείκτη, όπου η μια είναι int, η άλλη char και η τρίτη float. Ποιος είναι ο λόγος δήλωσης τύπου σε μια μεταβλητή δείκτη εφόσον σε αυτή τη μεταβλητή καταχωρείται διεύθυνση κάποιας θέσεως μνήμης; Ο τύπος ενός δείκτη αναφέρεται στον τύπο των δεδομένων που είναι εκχωρημένα στη θέση που δείχνει. Επίσης, το μέγεθος μιας μεταβλητής δείκτη δεν εξαρτάται από τον τύπο του δείκτη, αλλά είναι κοινό για όλους τους δείκτες ανεξαρτήτως τύπου και καθορίζεται από την έκδοση της C που διαθέτουμε, καθώς και από το λειτουργικό μας σύστημα.

Προκειμένου να βάλουμε έναν δείκτη να «δείχνει» σε μια άλλη μεταβλητή πρέπει να τον αρχικοποιήσουμε. Δηλαδή πρέπει να εκχωρήσουμε στον δείκτη τη διεύθυνση της άλλης μεταβλητής. Προφανώς πρέπει να χρησιμοποιηθεί ο τελεστής &, που μέσω αυτού έχουμε στη διάθεση μας τη διεύθυνση της εκάστοτε μεταβλητής. Στο επόμενο παράδειγμα δηλώνεται και αρχικοποιείται ο δείκτης p ώστε να δείχνει στη μεταβλητή a.

Παράδειγμα

```
main( ){
        int a;
        int *p;
        p = &a; //αρχικοποίηση δείκτη
        printf(«Give a number: «);
        scanf("%d",p);  //εκχώρηση τιμής στη μεταβλητή a μέσω του δείκτη
        printf(«The number is %d\n»,a);
}
```

Η scanf για να λειτουργήσει σωστά χρειάζεται ως όρισμα μια διεύθυνση μνήμης. Αυτός είναι ο λόγος που μέχρι στιγμής σε κάθε χρήση της scanf υπήρχε ο τελεστής &. Στο παραπάνω παράδειγμα όμως από μόνος του ο δείκτης p παρέχει μια διεύθυνση μνήμης, αυτή της μεταβλητής a στην οποία δείχνει. Θα ήταν λάθος αν χρησιμοποιηθεί &p, γιατί έτσι θα παρεχόταν η διεύθυνση του ίδιου του δείκτη.

8.2 Ο τελεστής *

Κάνοντας χρήση του τελεστή * έχουμε πρόσβαση στο περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης. Ας δούμε τις παρακάτω προτάσεις βάσει του προηγούμενου σχήματος.

printf(" %d \n", b); //εμφανίζεται το περιεχόμενο του b, δηλαδή το 3000 (η διεύθυνση της a)

printf(" %d \n", *b); // εμφανίζεται το περιεχόμενο της θέσης που δείχνει ο b, δηλαδή το 45

Παράδειγμα

```
int main (void){
        int a = 50, *c;
        c=&a;
        printf(" το a είναι =%d\n",a); // εμφανίζεται το 50
        printf(" το *c είναι =%d\n",*c);
        *c=150; // εκχωρείται η τιμή 150 στη μεταβλητή a μέσω του δείκτη
        printf(" το a είναι =%d\n",a); // εμφανίζεται το 150
        printf(" το *c είναι =%d\n",*c); // εμφανίζεται το 150
}
```

Στο επόμενο παράδειγμα φαίνεται σχηματικά τι συμβαίνει με τις θέσεις μνήμης των μεταβλητών βάσει των εντολών που ακολουθούν.

Παράδειγμα

1. int a, *ptr, b, c, *m;
2. a=25;
3. ptr=&a;
4. b=a;
5. c=*ptr;
6. m=ptr;

	1	2	3	4	5	6
	a	a	a	a	a	a
100		25	25	25	25	25
	ptr	ptr	ptr	ptr	ptr	ptr
200			100	100	100	100
	b	b	b	b	b	b
300				25	25	25
	c	c	c	c	c	c
400					25	25
	m	m	m	m	m	m
500						100

8.3 Αριθμητική Δεικτών

Μια μεταβλητή δείκτη μπορούμε να την αυξήσουμε ή να τη μειώσουμε προσθέτοντας ή αφαιρώντας αντίστοιχα κάποια ακέραια τιμή. Αν θεωρητικά προσθέσουμε μια μονάδα σε έναν δείκτη τύπου char, τότε η διεύθυνση που μπορεί να περιέχεται στον δείκτη αυξάνεται κατά μια μονάδα και αυτή, στην ουσία όμως ο

δείκτης έχει τεθεί να δείχνει στην επόμενη θέση μνήμης. Σε μια άλλη περίπτωση που ένας δείκτης είναι τύπου int και προστεθεί μια μονάδα, τότε η διεύθυνση που περιέχεται αυξάνεται κατά τέσσερα. Αυτό μπορεί να φαίνεται κάπως περίεργο αλλά είναι απολύτως λογικό αν σκεφτούμε πως κάθε ακέραια μεταβλητή καταλαμβάνει 4 bytes. Προσθέτοντας μια μονάδα σε έναν δείκτη, τότε αυτός τοποθετείται στην επόμενη θέση μνήμης, που στην περίπτωση των ακεραίων η επόμενη θέση μνήμης είναι 4 bytes μετά. Στην προηγούμενη περίπτωση που ο δείκτης ήταν τύπου char, η αύξηση της διεύθυνσης ήταν μια μονάδα, καθώς μια μεταβλητή χαρακτήρα καταλαμβάνει 1 byte.

Στο παρακάτω παράδειγμα εκχωρείται αρχικά στον δείκτη η διεύθυνση 0023FF74 και στη συνέχεια εκτελούνται κάποιες πράξεις, στα δεξιά εμφανίζεται το αποτέλεσμα.

Παράδειγμα

```
int a, *p;
p = &a;
printf("p=%p\n",p);
p=p+1;
printf("p=%p\n",p);
++p;
printf("p=%p\n",p);
p+=3;
printf("p=%p\n",p);
```

```
p=0023FF74
p=0023FF78
p=0023FF7C
p=0023FF88
```

Οι τελεστές ++ και - - έχουν υψηλότερη προτεραιότητα από τον τελεστή *, αυτό σημαίνει ότι η εντολή *p++ είναι ταυτόσημη με την *(p++) ή με την p++ και όχι με την (*p)++. Το *(p++) σημαίνει μετακίνησε τον δείκτη p μία θέση μνήμης μετά και πάρε το περιεχόμενο της νέας θέσης μνήμης, ενώ το (*p)++ σημαίνει πάρε το περιεχόμενο της θέσης στην οποία δείχνει ο p και αύξησέ το κατά 1.

8.4 Δείκτες και Πίνακες

Στην πραγματικότητα οι δείκτες και οι πίνακες είναι έννοιες ταυτόσημες, η μόνη τους διαφορά είναι ότι ένας πίνακας επιτρέπεται να προσπελάσει συγκεκριμένο πλήθος θέσεων. Οι δείκτες είναι ιδιαίτερα χρήσιμοι σε περιπτώσεις χειρισμού ενός πίνακα χαρακτήρων. Στο παρακάτω παράδειγμα δημιουργείται πρόγραμμα που θα μετράει τους χαρακτήρες 'Α', που βρίσκονται σε μια συμβολοσειρά που δίνει ο χρήστης.

Παράδειγμα

```
#include <stdio.h>
int main(void)
{
        char string[50], *ptr;
        int counter=0;
        puts("Δώσε μια συμβολοσειρά");
        gets(string);
        ptr=&string[0];
        while (*ptr!='\0') // συνθήκη τερματισμού επανάληψης
        {
                if (*ptr=='A') // αν το περιεχόμενο της θέσης στην οποία δείχνει
                είναι Α
                        counter++;
                *ptr++; // μετακίνηση δείκτη σε επόμενη θέση
        }
        printf("Ο χαρακτήρα περιέχεται %d φορές",counter);
        return 0;
}
```

8.5 Δείκτες και Συναρτήσεις

Οι δείκτες μπορούν να χρησιμοποιηθούν στις συναρτήσεις κατά τον τύπο κλήσης call by reference. Δηλαδή κατά την κλήση μιας συνάρτησης το πραγματικό όρισμά της μπορεί να είναι μια διεύθυνση μνήμης που αντιγράφεται σε έναν δείκτη.

Στο επόμενο παράδειγμα καλείται η συνάρτηση swap που έχουμε ξαναδεί, μόνο που αυτή τη φορά η ανταλλαγή των τιμών στις δύο μεταβλητές είναι μόνιμη, καθώς η μεταβλητή x και a αναφέρονται στις ίδιες θέσεις μνήμης, το ίδιο ισχύει και με τις μεταβλητές y και b.

Παράδειγμα

```
#include <stdio.h>
void Swap(int *a, int *b)
{
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}
int main(void)
{
        int x,y;
        printf("Δώσε δύο αριθμούς\n");
        scanf(«%d %d»,&x,&y);
        Swap(&x,&y);  //call by reference
        printf("Πλέον οι τιμές είναι: x = %d και y = %d",x,y); //εμφάνιση αλλαγ-
        μένων τιμών
        return 0;
}
```

Στο επόμενο παράδειγμα δημιουργείται πρόγραμμα, το οποίο διαβάζει μια συμβολοσειρά και καλεί μια συνάρτηση με όνομα length, που επιστρέφει το μήκος της συμβολοσειράς. Σε αυτή την περίπτωση αντιγράφεται η διεύθυνση της πρώτης θέσης του πίνακα στον δείκτη *s.

Παράδειγμα

```
#include <stdio.h>
int length(char *s){
        int length=0,counter=0;
        while (*s!='\0')
        {
                length++;
                s++;

        }
        return length;
}
int main(void)
{
        char s[40];
        gets(s);
        printf(«%d»,length(s));
        return 0;
}
```

Άσκηση

Να γίνει συνάρτηση countCapital η οποία θα δέχεται στην είσοδο μία συμβολοσειρά (1 όρισμα) και θα επιστρέφει το πλήθος των κεφαλαίων λατινικών γραμμάτων.

```
#include <stdio.h>
int count_capital(char *ptr)
{
        int counter=0;
        while (*ptr!='\0')
        {
                if (*ptr>='A' && *ptr<='Z')
                        counter++;
                ptr++;
        }
        return counter;
}

int main (void)
{
        char s[40];
        puts("Δώσε συμβολοσειρά");
        gets(s);
        printf(«Το πλήθος των κεφαλαίων είναι:%d»,count_capital(s));
        return 0;
}
```

8.6 Συναρτήσεις που επιστρέφουν ως τιμή έναν δείκτη

Η C δίνει τη δυνατότητα σε μια συνάρτηση να επιστρέψει τιμή δείκτη. Μια τέτοια συνάρτηση επιστρέφει τη διεύθυνση μιας θέσης μνήμης. Στη δήλωση της συνάρτησης πρέπει να φαίνεται ότι η συνάρτηση επιστρέφει δείκτη, καθώς και ο τύπος του δείκτη που επιστρέφεται. Ο τελεστής * ως πρόθεμα στο όνομά της υποδηλώνει ότι η συνάρτηση επιστρέφει δείκτη.

Στο επόμενο παράδειγμα, η συνάρτηση getName επιστρέφει ως τιμή έναν δείκτη σε μια συμβολοσειρά (δηλαδή στο όνομα μιας μέρας) ανάλογα με την τιμή που έχει το τυπικό όρισμα που δέχεται η συνάρτηση.

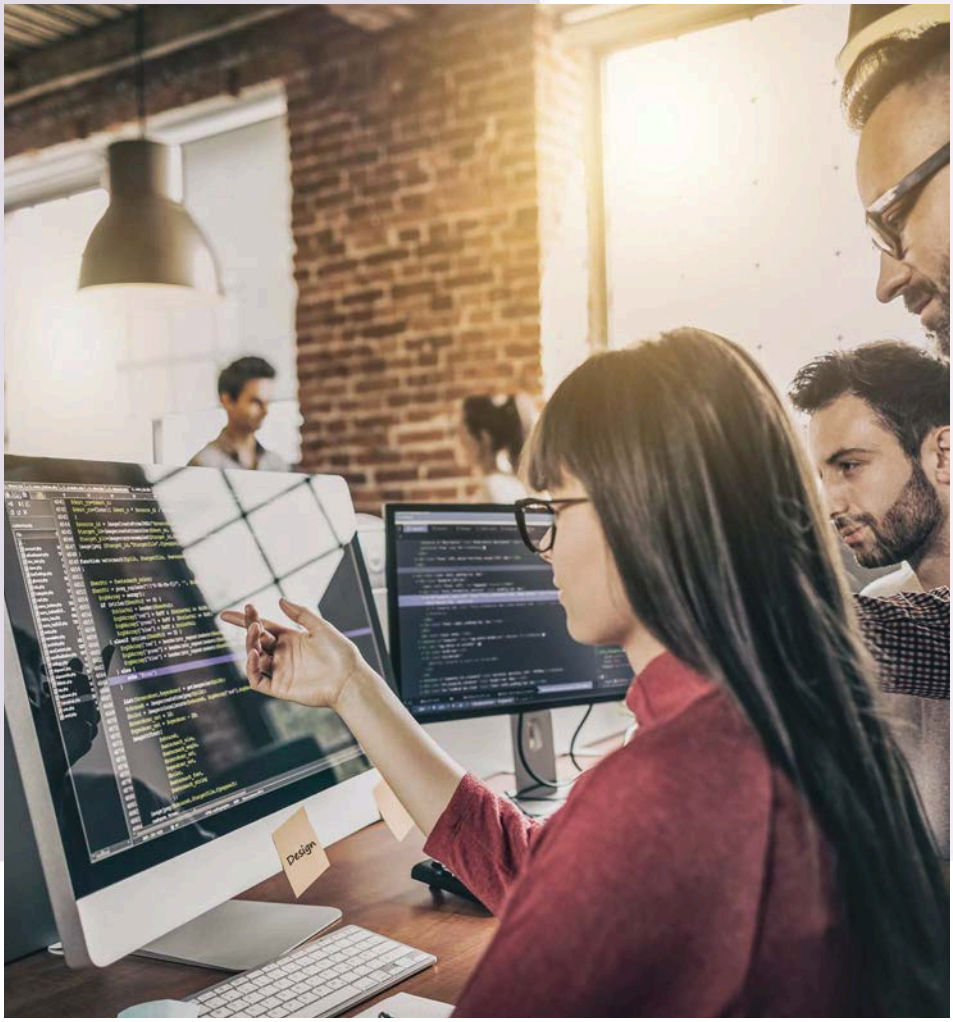
Παράδειγμα

```
#include <stdio.h>
char *getName(int day)
{
        switch(day)
        {
                case 0 :return(«Sunday»);
                case 1 :return(«Monday»);
                case 2 :return(«Tuesday»);
                case 3 :return(«Wednesday»);
                case 4 :return(«Thursday»);
                case 5 :return(«Friday»);
                case 6 :return(«Saturday»);
                default:return(«Error in getName() module.Invalid argument
                passed»);
        }
}
int main(void)
{
        int number;
        printf(“ δώσε έναν ακέραιο αριθμό από το 0 εώς το 6\n”);
        scanf(“%d”, &number);
        printf(“ η μέρα είναι %s”, getname( ));
        return 0;
}
```

8.7 Ασκήσεις

1. Να γίνει συνάρτηση με όνομα reversestring η οποία θα δέχεται στην είσοδο μία συμβολοσειρά (1 όρισμα) και θα την αντιστρέφει. Π.χ. αν δεχτεί σαν είσοδο τη συμβολοσειρά “Thessaloniki” θα επιστρέψει την “ikinolassehT”
2. Να γίνει συνάρτηση με όνομα last_char η οποία θα δέχεται στην είσοδο μία συμβολοσειρά (1 όρισμα) και θα αντικαθιστά κάθε χαρακτήρα της συμβολοσειράς με το τελευταίο της γράμμα. Π.χ. αν η last_char κληθεί με είσοδο το “abcefg” θα το αντικαταστήσει με το “gggggg”.
3. Να γίνει συνάρτηση deleteSpace η οποία θα δέχεται στην είσοδο μία συμβολοσειρά με πεζούς λατινικούς χαρακτήρες και κενούς χαρακτήρες (1 όρισμα) και την μετατρέπει σε συμβολοσειρά με τους αντίστοιχους χαρακτήρες έχοντας αφαιρέσει τα κενά. Π.χ. αν δεχτεί σαν είσοδο τη συμβολοσειρά “th ess alo niki” θα επιστρέψει την “ thessaloniki “.
4. Να γίνει συνάρτηση με όνομα countWords η οποία θα δέχεται στην είσοδο μία συμβολοσειρά (1 όρισμα), που θα περιλαμβάνει μία πρόταση και θα επιστρέφει το πλήθος των λέξεων που περιέχει. Θεωρήστε ότι οι λέξεις χωρίζονται με ένα ή περισσότερα κενά και η συμβολοσειρά δεν περιλαμβάνει σημεία στίξης ή χαρακτήρες αλλαγής γραμμής. Π.χ. η countWords για είσοδο τη συμβολοσειρά “ 1234 Hello world how are you “ θα επιστρέψει την τιμή 5.
5. Να γίνει συνάρτηση με όνομα concat η οποία θα δέχεται ως είσοδο 2 συμβολοσειρές και θα επικολλά στο τέλος της πρώτης τη δεύτερη συμβολοσειρά. Η συνάρτηση να επιστρέφει το πλήθος των χαρακτήρων της συμβολοσειράς που προκύπτει. (να μη χρησιμοποιηθούν έτοιμες συναρτήσεις όπως η strcopy, η strcat, η strlen κ.τ.λ.). Για παράδειγμα η concat για είσοδο “3450” και “ number” θα τροποποιεί την πρώτη συμβολοσειρά σε “3450 number”.
6. Να γίνει συνάρτηση η οποία θα δέχεται μια συμβολοσειρά και θα εξετάζει αν η συμβολοσειρά αποτελείται μόνο από αριθμητικά ψηφία (0-9). Στη συνέχεια θα επιστρέφει τον ακέραιο αριθμό που εμπεριέχεται στη συμβολοσειρά.

7. Να γίνει συνάρτηση compare η οποία θα δέχεται ως είσοδο δύο συμβολοσειρές (2 ορίσματα) και θα επιστρέφει το πλήθος των χαρακτήρων της πρώτης που εμπεριέχονται και στη δεύτερη. Π.χ. η compare για είσοδο “ball” και “lamp” θα επιστρέψει τον αριθμό 3.



Βιβλιογραφία

- Harbison, Samuel P. and Steele, Guy R., *C: A Reference Manual* (5th Edition), Pearson, 2002.
- Kernighan, Brian W., Ritchie, Dennis, *C Programming Language*, Prentice Hall, 1988.
- Perry, Greg and Miller, Dean, *C Programming Absolute Beginner's Guide*, Que Publishing, 2013.
- Shaw, Zed, *Learn C the Hard Way*, Addison-Wesley Professional, 2015.
- Χατζηγιαννάκης, Νίκος, *Η Γλώσσα C++ σε Βάθος*, Αθήνα, Εκδόσεις Κλειδάριθμος, 2017.

АКМИ