ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΕΠΙΚΟΙΝΩΝΙΩΝ

Προγραμματισμός Ι

Δρ. Πάρις Αστ. Μαστοροκώστας
Επίκουρος Καθηγητής

Τομέας Υπολογιστικών Τεχνικών & Συστημάτων

Σέρρες 2004

Περιεχόμενα

Κατάλογος Σχημάτων vii

Κατάλογος Πινάκων ix

1. Βασικά στοιχεία προγράμματος 1-1

1.1 Εισαγωγή 1-1

1.2 Ιστορική αναδρομή 1-2

1.3 Εργαλεία ανάλυσης προβλημάτων 1-3

1.4 Στάδια υλοποίησης προγράμματος 1-5

1.5 Βασικά στοιχεία προγράμματος 1-6

1.6 Λεξιλόγιο της γλώσσας C 1-9

1.6.1 Δεσμευμένες λέξεις 1-10

1.6.2 Λέξεις κλειδιά 1-10

1.6.3 Αναγνωριστές 1-11

1.7 Κανόνες δημιουργίας ευανάγνωστων προγραμμάτων 1-11

2. Μεταβλητές - σταθερές - I/O κονσόλας 2-1

2.1 Η έννοια της μεταβλητής 2-1

2.1.1 Δήλωση μεταβλητής 2-1

2.1.2 Ονομασία μεταβλητής 2-2

2.2 Τύποι μεταβλητών 2-3

2.2.1 Ο τύπος του χαρακτήρα 2-3

2.2.2 Μη εκτυπούμενοι χαρακτήρες 2-5

Παράδειγμα 2.1 2-5

2.2.3 Ο τύπος του ακεραίου 2-6

Παράδειγμα 2.2 2-8

2.2.4 Τύποι πραγματικών αριθμών 2-9

Παράδειγμα 2.3 2-12

2.3 I/O κονσόλας 2-13

2.3.1 Οι συναρτήσεις getche, getch 2-13

2.3.2 Η συνάρτηση getchar 2-13
2.3.3 Η συνάρτηση putchar 2-14
2.3.4 Η συνάρτηση kbhit 2-14
Παράδειγμα 2.4 2-14
3. Τελεστές - εκφράσεις 3-1
3.1 Ορισμοί - σημειογραφίες 3-1
3.1.1 Σημειογραφία 3-2
3.2 Κατηγορίες εκφράσεων της C - προτεραιότητα και προσεταιριστικότητα 3-3
Παράδειγμα 3.1 3-5
3.3 Τελεστές αύξησης και μείωσης 3-5
Παράδειγμα 3.2 3-5
Παράδειγμα 3.3 3-6
3.4 Τελεστές ανάθεσης 3-7
3.5 Συσχετιστικοί τελεστές 3-7
3.6 Λογικοί τελεστές 3-8
Παράδειγμα 3.4 3-9
3.7 Μετατροπές τύπων 3-9
3.7.1 Υπονοούμενες μετατροπές 3-9
Παράδειγμα 3.5 3-10
3.7.2 Ρητές μετατροπές – τελεστής typecast 3-11
3.8 Τελεστής sizeof 3-11
4. Έλεγχος ροής - προτάσεις υπό συνθήκη διακλάδωσης 4-1
4.1 Έλεγχος ροής 4-1
4.2 Επιλεκτική εκτέλεση προτάσεων 4-2
Παράδειγμα 4.1 4-3
4.3 Υπό συνθήκη διακλάδωση if - else 4-4
Παράδειγμα 4.2 4-5

4.4 Ο υποθετικός τελεστής 4-6
Παράδειγμα 4.3 4-7
4.5 Υπό συνθήκη διακλάδωση switch 4-8
Παράδειγμα 4.4 4-9
Παράδειγμα 4.5 4-11
5. Προτάσεις επανάληψης - βρόχοι 5-1
5.1 Γενικά 5-1
5.1.1 Βρόχος while - do 5-1
5.1.2 Βρόχος do - while 5-2
5.2 Βρόχοι με συνθήκη εισόδου στη C 5-3
5.2.1 Βρόχος while 5-3
Παράδειγμα 5.1 5-4
Παράδειγμα 5.2 5-5
5.2.2 Βρόχος for 5-6
Παράδειγμα 5.3 5-7
Παράδειγμα 5.4 5-8
5.2.3 Ο τελεστής κόμμα (,). 5-9
5.2.4 Μετασχηματισμός βρόχων while – for 5-10
Παράδειγμα 5.5 5-10
5.3 Βρόχος με συνθήκη εξόδου στη C (do - while) 5-11
Παράδειγμα 5.6 5-12
Παράδειγμα 5.7 5-13
5.4 Ένθετοι βρόχοι 5-14
Παράδειγμα 5.8 5-14
Παράδειγμα 5.9 5-15
5.5 Διακοπτόμενοι βρόχοι στη C 5-16
5.5.1 Η κωδική λέξη break 5-16
Παράδειγμα 5.10 5-17
5.5.2 Η πρόταση continue 5-18
Παράδειγμα 5.11 5-18
5.5.3 Η πρόταση goto 5-19

Παράδειγμα 5.12 5-19
5.6 Κανόνες για τη χρήση των προτάσεων ροής του ελέγχου . 5-20
6. Πίνακες - αλφαριθμητικά 6-1
6.1 Μονοδιάστατοι πίνακες 6-1
6.2 Πολυδιάστατοι πίνακες 6-3
6.3 Αρχικοποίηση πολυδιάστατου πίνακα 6-4
Παράδειγμα 6.1 6-6
6.4 Αποθήκευση των πινάκων στη μνήμη 6-8
6.5 Το αλφαριθμητικό 6-9
6.6 Αρχικοποίηση αλφαριθμητικού 6-9
6.7 Είσοδος - έξοδος αλφαριθμητικών 6-10
6.7.1 Εισαγωγή αλφαριθμητικού 6-10
6.7.2 Εκτύπωση αλφαριθμητικού 6-11
Παράδειγμα 6.2 6-11
6.8 Συναρτήσεις αλφαριθμητικών 6-12
6.8.1 Η συνάρτηση μήκους αλφαριθμητικού 6-13
Παράδειγμα 6.3 6-13
6.8.2 Η συνάρτηση αντιγραφής αλφαριθμητικού 6-13
6.8.3 Η συνάρτηση συνένωσης αλφαριθμητικών 6-14
Παράδειγμα 6.4 6-14
6.8.4 Η συνάρτηση σύγκρισης αλφαριθμητικών 6-15
Παράδειγμα 6.5 6-16
7. Δημιουργία προγραμμάτων - παραδείγματα . 7-1
7.1 Γενικά 7-1
Πρόβλημα 7.1 7-1
Πρόβλημα 7.2 7-5
Πρόβλημα 7.3 7-7
Πρόβλημα 7.4 7-9
Πρόβλημα 7.5 7-10
Πρόβλημα 7.6 7-11

Πρόβλημα 7.7 7-13

Βιβλιογραφία Β-1

Κατάλογος Σχημάτων

Σχήμα 2-1. *Αποθήκευση και ανάκληση ASCII χαρακτήρα* 2-4
Σχήμα 4-1. *Ακολουθιακή εκτέλεση προτάσεων* 4-1
Σχήμα 4-2. *Επιλεκτική εκτέλεση προτάσεων* 4-3
Σχήμα 5-1. *Βρόχος while-do* 5-2
Σχήμα 5-2. *Βρόχος do-while* 5-3
Σχήμα 5-3. *Διάγραμμα ροής της πρότασης επανάληψης* 5-5
Σχήμα 5-4. *Διάγραμμα ροής της πρότασης επανάληψης* 5-8
Σχήμα 5-5. *Διάγραμμα ροής της πρότασης επανάληψης* 5-13
Σχήμα 5-6. *Μπλοκ διάγραμμα της break* 5-17
Σχήμα 6-1. *Αποθήκευση πολυδιάστατου πίνακα* 6-4
Σχήμα 6-2. *Αποθήκευση πίνακα στη μνήμη* 6-8

Κατάλογος Πινάκων

Πίνακας 1-1. *Λέξεις κλειδιά της C* 1-11
Πίνακας 2-1. *Μη εκτυπούμενοι χαρακτήρες και αντίστοιχες ακολουθίες διαφυγής* 2-5
Πίνακας 3-1. *Κατηγορίες τελεστών*. 3-1
Πίνακας 3-2. *Σύμβολα δυαδικών τελεστών* 3-2
Πίνακας 3-3. *Προτεραιότητα και προσεταιριστικότητα τελεστών* 3-4
Πίνακας 3-4. *Συσχετιστικοί τελεστές* 3-7
Πίνακας 3-5. *Λογικοί τελεστές* 3-8
Πίνακας 3-6. *Πίνακας αληθείας* 3-8

1. Βασικά στοιχεία προγράμματος

1.1 Εισαγωγή

Αντικείμενο του παρόντος συγγράματος είναι η εισαγωγή του αναγνώστη στη λογική του προγραμματισμού Η/Υ. Το ενδιαφέρον εστιάζεται στον καλούμενο **διαδικαστικό προγραμματισμό** (procedural programming), βασικά στοιχεία του οποίου είναι η δόμηση του προγράμματος και η επαναλαμβανόμενη χρήση υποπρογραμμάτων, τα οποία είτε επιτελούν εργασίες γενικής φύσης είτε απευθύνονται σε ένα τμήμα του συνολικού προβλήματος. Στόχος είναι η κατανόηση των αρχών του προγραμματισμού και η εμπέδωση της φιλοσοφίας του, έτσι ώστε ο αναγνώστης να μπορεί χωρίς δυσχέρειες να προχωρήσει σε άλλες μορφές προγραμματισμού, όπως ο αντικειμενοστραφής προγραμματισμός (object–oriented programming).

Στην προσπάθεια αυτή θα χρησιμοποιηθεί ως πλατφόρμα μία γλώσσα προγραμματισμού **υψηλού επιπέδου**, η γλώσσα C. Ο όρος γλώσσα υψηλού επιπέδου υποδηλώνει ότι δεν είναι κατασκευασμένη για να λειτουργεί σε συγκεκριμένη αρχιτεκτονική υπολογιστή αλλά δύναται να λειτουργήσει σε πληθώρα αρχιτεκτονικών, γεγονός που σημαίνει ότι:

- Οι διάφορες αρχιτεκτονικές υπολογιστών για να λειτουργήσουν χρησιμοποιούν ένα σύνολο εντολών, το οποίο χρησιμοποιεί τη γλώσσα μηχανής της εκάστοτε αρχιτεκτονικής. Επομένως, έαν κάποιος προγραμματίσει κάνοντας χρήση της γλώσσας μηχανής μίας αρχιτεκτονικής, τα προγράμματά του δε θα είναι συμβατά με άλλες αρχιτεκτονικές. Το αντιστάθμισμα αυτού του μειονεκτήματος είναι ότι στο παρελθόν, που οι υπολογιστές είχαν λίγη μνήμη και χαμηλή ταχύτητα, ο προγραμματισμός σε γλώσσα μηχανής χειριζόταν αποδοτικότερα τους πόρους του μηχανήματος.
- Για να είναι μία γλώσσα συμβατή με τις γλώσσες μηχανής διαφόρων αρχιτεκτονικών, θα πρέπει να λαμβάνει χώρα μεταγλώττιση από τη γλώσσα προγραμματισμού στην εκάστοτε γλώσσα μηχανής. Όντως αυτό συμβαίνει και το λογισμικό που επιτελεί τη συγκεκριμένη εργασία ονομάζεται **μεταγλωττιστής** (compiler).

Με βάση τα παραπάνω, οι γλώσσες προγραμματισμού υψηλού επιπέδου είναι *μεταγλωττισμένες* γλώσσες, αποσκοπώντας στο να είναι ευανάγνωστες και κατανοητές.

Σε ό,τι αφορά τη C, παρουσιάζει μία σειρά από ενδιαφέροντα και χρήσιμα χαρακτηριστικά:

- Μπορεί να χρησιμοποιηθεί και ως γλώσσα προγραμματισμού χαμηλού επιπέδου, επιτρέποντας άμεση πρόσβαση στους πόρους του υπολογιστή.
- Είναι σχετικά μικρή και εύκολη στην εκμάθηση.
- Υποστηρίζει δομημένο προγραμματισμό.
- Είναι αποτελεσματική, παράγοντας συμπαγή και γρήγορα στην εκτέλεση προγράμματα.
- Αποτελεί μαζί με τη C++ τις ευρύτερα χρησιμοποιούμενες γλώσσες σε ερευνητικά και αναπτυξιακά προγράμματα, γεγονός που έχει δημιουργήσει μία πολλή μεγάλη εγκατεστημένη βάση εφαρμογών που αναπτύχθηκαν με αυτές τις γλώσσες και πρέπει να συντηρούνται και να εξελίσσονται.

1.2 Ιστορική αναδρομή

Η C επινοήθηκε το 1972 από τον Dennis Ritchie, στα εργαστήρια Bell. Δημιουργήθηκε για να εξυπηρετήσει το λειτουργικό σύστημα Unix, το οποίο έως τότε ήταν γραμμένο σε assembly. Ο δημιουργός του Unix, Ken Thompson, φίλος και συνεργάτης του Ritchie, είχε δημιουργήσει την πρόγονο της C, τη γλώσσα B. Και οι δύο γλώσσες έχουν κοινή καταγωγή από τη γλώσσα BCPL, η οποία είχε αναπτυχθεί από τον Martin Richards κατά το πέρασμά του απο το Τεχνολογικό Ινστιτούτο της Μασσαχουσέττης (MIT) το 1967, στηριζόμενη στη γλώσσα CPL (Cambridge Programming Language) του πανεπιστημίου του Cambridge. Και οι τρεις γλώσσες κατασκευάστηκαν στα πλαίσια του προγράμματος MAC και του απόγονού του Multics, τα οποία στόχευαν στην κατανομή των πόρων των υπολογιστών σε πολλούς χρήστες. Τα δύο αυτά προγράμματα, στα οποία συνέπραξαν το MIT, η General Electric και τα εργαστήρια Bell, απετέλεσαν τη θερμοκοιτίδα πολλών προγραμμάτων λογισμικού, που κυριαρχούν από τη δεκαετία του 1960 έως σήμερα. Για ενδελεχή μελέτη της ιστορίας του λογισμικού, ο αναγνώστης μπορεί να ανατρέξει στην αναφορά [13].

Η C, ούσα ευέλικτη και αποδοτική χρησιμοποιήθηκε αρχικά για τον προγραμματισμό συστημάτων στο Unix. Το 1974 εμφανίσθηκε από τον Brian Kernighan το πρώτο γραπτό κείμενο για τη γλώσσα, υπό τον τίτλο "Programming in C: A Tutorial". Το 1977 έγινε η πρώτη επίσημη τεκμηρίωση της γλώσσας με το βιβλίο "The C Programming Language" από τους Kernighan και Ritchie. Το βιβλίο αυτό απετέλεσε το «ευαγγέλιο» των προγραμματιστών της C, αποκαλούμενο «Λευκή Βίβλος» ή «πρότυπο K&R».

Με την πάροδο του χρόνου η γλώσσα C άρχισε να χρησιμοποιείται και σε άλλα πεδία εφαρμογών, πέραν του προγραμματισμού συστημάτων. Η εμφάνιση των μεταγλωττιστών της γλώσσας στο MS–DOS και ο μεγάλος αριθμός προγραμμάτων βιβλιοθήκης που κατασκευάστηκαν, οδήγησαν τη γλώσσα στο απόγειό της στα τέλη της δεκαετίας του 1980. Βέβαια η γλώσσα γνώρισε πολλές αλλαγές, οδηγούμενη τελικά στην επονομαζόμενη **ANSI έκδοση**. Η τελευταία ενημέρωση της γλώσσας έγινε το 1999.

Τα τελευταία χρόνια τα ηνία έχει λάβει ο αντικειμενοστρεφής προγραμματισμός και στην κατεύθυνση αυτή συνέβαλλε η γλώσσα C++, που επινοήθηκε το 1983 από το Δανό Bjarne Stroustrup στα εργαστήρια της AT&T (το τμήμα των εργαστήρια Bell που πήγαν στην AT&T όταν η εταιρεία διασπάσθηκε το 1995). Η C++ – ή *C με τάξεις* – μπορεί να θεωρηθεί απόγονος της C, αν και έχει αρκετές διαφορές. Γεγονός είναι ότι αντικατέστησε τη C σε πολύ μεγάλο ποσοστό και αποτελεί σήμερα μία από τις κυρίαρχες γλώσσες προγραμματισμού.

1.3 Εργαλεία ανάλυσης προβλημάτων

Για την ανάλυση ενός προβλήματος και την κατάρτιση του μοντέλου που θα υλοποιηθεί σε κώδικα υπάρχουν τρία εργαλεία:

1. Η **φυσική γλώσσα** (natural language), σύμφωνα με την οποία το πρόβλημα αναλύεται σε απλές προτάσεις της καθομιλουμένης γλώσσας, χρησιμοποιώντας το συντακτικό αυτής.

2. **Το διάγραμμα ροής** (flow chart), σύμφωνα με το οποίο απεικονίζεται γραφικά η εξέλιξη του προβλήματος, με χρήση ειδικών συμβόλων.

3. **Ο ψευδοκώδικας** (pseudocode), ο οποίος μετασχηματίζει τη φυσική γλώσσα σε μία σειρά προτάσεων που χρησιμοποιούν το συντακτικό γλώσσας προγραμματισμού,

χωρίς να ακολουθούν επακριβώς το φορμαλισμό συγκεκριμένης γλώσσας.

Δεν υπάρχουν γενικοί κανόνες για την υιοθέτηση κάποιου από τα τρία εργαλεία. Η χρήση καθενός εκ των τριών εξαρτάται από τον εκάστοτε προγραμματιστή και το συγκεκριμένο πρόβλημα. Μία ενδεικτική περιγραφή της λειτουργίας των ανωτέρω εργαλείων θα γίνει με τη βοήθεια του ακόλουθου παραδείγματος:

Να μετατραπούν οι βαθμοί Farenheit σε βαθμούς Κελσίου, χρησιμοποιώντας την εξίσωση μετασχηματισμού C=(F-32))5/9.

1. Με χρήση φυσικής γλώσσας

Ζήτησε από το χρήστη τη θερμοκρασία σε βαθμούς F

Διάβασε την τιμή που δίνει ο χρήστης

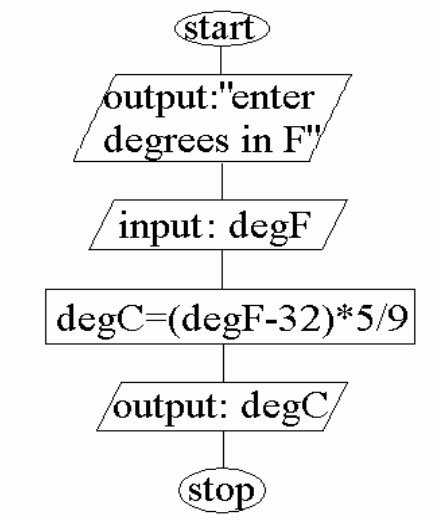
Αποθήκευσε την τιμή σε θέση αποθήκευσης που ονομάζεται degF

Υπολόγισε τους βαθμούς C με χρήση μαθηματικής σχέσης

Αποθήκευσε το αποτέλεσμα σε θέση αποθήκευσης που ονομάζεται degC

Τύπωσε το περιεχόμενο της degC

2. Με χρήση διαγράμματος ροής



2. Με χρήση ψευδοκώδικα

print "enter degrees in Farenheit"

read degF

degC = (degF - 32) * 5 / 9

print degC

1.4 Στάδια υλοποίησης προγράμματος

Το υπολογιστικό πρόγραμμα είναι μία ακολουθία εντολών, με τις οποίες ο υπολογιστής εκτελεί μία συγκεκριμένη εργασία και επιλύει ένα δοθέν πρόβλημα. Η υλοποίηση ενός προγράμματος περιλαμβάνει τέσσερα στάδια:

1. Η συγγραφή του πηγαίου κώδικα (source code). Στο βήμα αυτό χρησιμοποιείται ένας **συντάκτης προγράμματος** (text editor) για τη συγγραφή του κώδικα. Συνήθως χρησιμοποιείται ο ενσωματωμένος συντάκτης της C. Το αποτέλεσμα είναι ένα αρχείο κειμένου, αναγνώσιμο από οποιονδήποτε συντάκτη (DOS editor, notepad, wordpad, πρόγραμμα επεξεργασίας κειμένου), το οποίο έχει κατάληξη **.c** (ή **.cpp** εάν χρησιμοποιείται η C++).

2. Η μεταγλώττιση του πηγαίου κώδικα (compilation). Η διαδικασία της μεταγλώττισης εκτελείται αυτόματα από το **μεταγλωττιστή** (compiler) και παράγεται ο **τελικός ή αντικείμενος κώδικας** (object code), που είναι ο κώδικας σε γλώσσα μηχανής. Στο στάδιο αυτό ανιχνεύονται τα **συντακτικά σφάλματα** (syntax errors), τα οποία είναι σφάλματα που οφείλονται σε παραβίαση των συντακτικών κανόνων, και αναφέρονται υπό μορφή λίστας (οπότε μπορούν να διορθωθούν προτού εκτελεσθεί το πρόγραμμα). Εάν δεν υπάρχουν συντακτικά σφάλματα, το αρχείο που προκύπτει ονομάζεται έχει το όνομα του αρχείου του πηγαίου κώδικα και κατάληξη **.obj**.

3. Η σύνδεση του αντικείμενου κώδικα. Στο τρίτο στάδιο επιτελείται η διεργασία της σύνδεσης (linking), η οποία είναι πλήρως αυτοματοποιημένη και γίνεται με χρήση του **συνδέτη** (linker). Το αποτέλεσμα της σύνδεσης είναι η δημιουργία του εκτελέσιμου κώδικα ή η αναφορά τυχόν προβλημάτων, όπως για παράδειγμα η αδυναμία εντοπισμού μίας συνάρτησης ή μίας εξωτερικής μεταβλητής. Ο εκτελέσιμος κώδικας αποθηκεύεται σε αρχείο που έχει το όνομα του πηγαίου κώδικα και επέκταση **.exe**. Πλέον το πρόγραμμα έχει ξεφύγει από το περιβάλλον της γλώσσας προγραμματισμού και μπορεί να εκτελεσθεί όπως ένα οποιοδήποτε εκτελέσιμο αρχείο του υπολογιστή.

Ο συνδέτης έχει τη δυνατότητα να συνδέσει περισσότερα του ενός αρχεία αντικείμενου κώδικα. Επιπλέον, αναζητά σε βιβλιοθήκες τα σώματα των

συναρτήσεων που ο προγραμματιστής χρησιμοποίησε στο πρόγραμμά του (λεπτομέρειες στην §1.5 και σε επόμενα κεφάλαια).

4. Η εκτέλεση του προγράμματος. Στο τελευταίο στάδιο εκτελείται ή «τρέχει» το αρχείο **.exe** και ελέγχεται η ορθή λειτουργία του προγράμματος. Τα σφάλματα που ανακύπτουν στο στάδιο αυτό ονομάζονται **σημασιολογικά σφάλματα** (semantic errors). Οφείλονται σε κακή σχεδίαση της λύσης του προβλήματος και, δυστυχώς, αναγνωρίζονται στο χρόνο εκτέλεσης. Χαρακτηριστικό σφάλμα εκτέλεσης είναι η διαίρεση αριθμού με το μηδέν, ενέργεια που δεν ανιχνεύεται ως λανθασμένη κατά τη μεταγλώττιση.

Σε περίπτωση σφάλματος, ο προγραμματιστής επιστρέφει στο πρώτο στάδιο, όπου κάνει τις διορθώσεις, και επαναλαμβάνει τα υπόλοιπα στάδια έως ότου το πρόγραμμα λειτουργήσει επιτυχώς.

1.5 Βασικά στοιχεία προγράμματος

Έστω το ακόλουθο απλό πρόγραμμα:

```
/*****************************************
  This program prints out the sentence "This is a test."
***************************************** */
#include<stdio.h>
void main ()
{
     printf( "This is a test.\n" );
}  // end of main
```

- Η καρδιά ενός προγράμματος της γλώσσας C είναι η λέξη ***main***. Αντιστοιχεί στο κύριο τμήμα του κώδικα και χρησιμοποιείται για να γνωστοποιήσει το σημείο έναρξης εκτέλεσης του προγράμματος. Μετά τη ***main*** ακολουθεί το εισαγωγικό αριστερό άγκιστρο ({) και κατόπιν οι γραμμές του προγράμματος. Η ***main***, και μαζί το πρόγραμμα, τερματίζει με το καταληκτικό δεξί άγκιστρο (}). Στο παρόν πρόγραμμα η ***main*** αποτελείται από μία μόνο πρόταση κλήσης της συνάρτησης

printf, η οποία ανήκει στη βασική βιβλιοθήκη συναρτήσεων της C. Η βιβλιοθήκη συναρτήσεων περιλαμβάνει τον κώδικα πρότυπων συναρτήσεων και αποτελείται από μία σειρά αρχείων, τα οποία έχουν την κατάληξη **h**. Τα αρχεία αυτά ονομάζονται **αρχεία κεφαλίδας** (header files) και περιλαμβάνουν συναρτήσεις συναφούς λειτουργίας. Δηλώνονται πριν από τη ***main***, με χρήση της οδηγίας προς τον προεπεξεργαστή ***include*** ως εξής:

#include <όνομα_αρχείου_κεφαλίδας.h>

Στην παρούσα περίπτωση το αρχείο κεφαλίδας είναι το ***stdio.h*** (**st**an**d**ard **i**nput–**o**utput), το οποίο περιλαμβάνει συναρτήσεις σχετιζόμενες με τις συσκευές εισόδου (π.χ. πληκτρολόγιο) και εξόδου (π.χ. οθόνη).

- Η συνάρτηση ***main*** εκτός από προτάσεις και κλήσεις σε συναρτήσεις βιβλιοθήκης μπορεί να καλεί και άλλες συναρτήσεις, οι οποίες δημιουργούνται από το προγραμματιστή. Μία συνάρτηση είναι ένα σύνολο προτάσεων με ένα δεδομένο όνομα, όπως είναι η ***main*** ή η ***printf***. Οι προτάσεις αποτελούν το **σώμα** (body) της συνάρτησης και περικλείονται σε άγκιστρα. Οι συναρτήσεις πριν από τη ***main*** και αναπτύσσουν τον κώδικά τους είτε πάνω είτε κάτω από τη ***main***[1].

- Όλες οι προτάσεις τελειώνουν με **ερωτηματικό** (**;**) (semicolon), το οποίο ονομάζεται **σύμβολο του τερματιστή προτάσεων** (statement terminator symbol). Εξαίρεση αποτελούν οι οδηγίες προς τον προεπεξεργαστή, όπως είναι η ***include***, οι οποίες αρχίζουν πάντοτε με **δίεση** (#) και δεν τελειώνουν με ερωτηματικό**.**

- Η έξοδος του προγράμματος είναι η έξοδος της ***printf***: **This is a test.**

- Οι τρεις πρώτες γραμμές του ανωτέρω προγράμματος είναι **σχόλια** (comments) και δεν αποτελούν τμήμα του κώδικα, καθώς δε λαμβάνονται υπόψη από το μεταγλωττιστή. Το σχόλιο είναι κείμενο ανάμεσα σε /* και */ και μπορεί να τοποθετηθεί οπουδήποτε μέσα στο πρόγραμμα. Μπορεί να επεκταθεί σε περισσότερες από μία γραμμές. Σε περίπτωση που πρέπει να τοποθετηθεί ένα σχόλιο σε κάποιο σημείο μίας γραμμής και να μην επεκταθεί σε άλλη γραμμή, χρησιμοποιείται το

[1] Λεπτομερής ανάλυση του συντακτικού και της λειτουργίας των συναρτήσεων στο αντίστοιχο κεφάλαιο.

σύμβολο // και ακολούθως τοποθετείται το σχόλιο, όπως συμβαίνει στο τέλος του ανωτέρω προγράμματος (το κείμενο **end of main** είναι σχόλιο).

Τα σχόλια πρέπει να χρησιμοποιούνται αφειδώς γιατί καταστούν τον κώδικα ευανάγνωστο και συνεισφέρουν στην επεξήγηση δυσνόητων σημείων. Είναι θεμιτό να ευθυγραμμίζονται τα σύμβολα των σχολίων και να μην τοποθετούνται ποτέ σχόλια μέσα σε σχόλια (ένθετα σχόλια) γιατί μπορεί να δημιουργηθεί πρόβλημα σε μερικούς μεταγλωττιστές.

Παρακάτω παρουσιάζονται μερικές περιπτώσεις σωστών και λανθασμένων σχολίων:

α) **/* Αυτό /* το σχόλιο */ είναι λανθασμένο */**

β) **/*
Αυτό το
σχόλιο
είναι σωστό
*/**

<u>Παρατηρήσεις:</u>

1. Η γλώσσα C διαχωρίζει τα κεφαλαία γράμματα από τα μικρά (case sensitive). Η εντολή ***Printf*** <u>δεν</u> είναι ίδια με την ***printf.*** Όλες οι εντολές στη C γράφονται με μικρά γράμματα!

2. Η σωστή στηλοθεσία είναι πολύ σημαντική καθώς καθιστά τον κώδικα ευανάγνωστο.

3. Η συνάρτηση ***printf*** ανήκει στις **μορφοποιούμενες** συναρτήσεις εισόδου–εξόδου. Ονομάζεται μορφοποιούμενη γιατί δίνει τη δυνατότητα στο χρήστη να μορφοποιήσει την έξοδό της, δυνάμενη να εκτυπώσει μεταβλητές διαφόρων τύπων και με διάφορους τρόπους, χρησιμοποιώντας κατάλληλα σύμβολα. Δυαδική της ***printf*** είναι η ***scanf***, η οποία λαμβάνει πληροφορία από την είσοδο (πληκτρολόγιο).

Η πρόταση **printf("This is a test.\n");** καλεί την ***printf*** για να τυπωθεί το καθορισμένο κείμενο. Τα **ορίσματα εισόδου** (input arguments) περικλείονται από παρενθέσεις και προσδιορίζουν το προς εκτύπωση κείμενο και τη μορφή με την οποία θα εκτυπωθεί. Τέλος, το σύμβολο **\n**, που ανήκει στις **ακολουθίες διαφυγής**,

σημαίνει «μετακινήσου στην επόμενη γραμμή». Λεπτομερής περιγραφή της λειτουργίας των συναρτήσεων εισόδου – εξόδου δίνεται στο επόμενο κεφάλαιο.

4. Πέραν της ***include***, μία σημαντική οδηγία προς τον προεπεξεργαστή είναι η ***define***, η οποία αντιστοιχίζει ένα όνομα με μία σταθερά ή με μία σειρά χαρακτήρων. Οποτεδήποτε εμφανίζεται το όνομα μέσα στον κώδικα, αντικαθίσταται αυτόματα με την τιμή της σταθεράς ή τη συμβολοσειρά. Για παράδειγμα, εάν χρησιμοποιηθεί η λέξη **TRUE** στη θέση της τιμής **1** και η λέξη **FALSE** στη θέση της τιμής **0**, θα δοθούν δύο ***define*** ως εξής:

```
#define TRUE 1
#define FALSE 0
```

Εάν αντικατασταθεί ολόκληρη φράση, μπορεί να εμφανισθεί στην οθόνη με χρήση της ***printf***:

```
#define TITLOS "TEI of Serres\nDpt of Informatics and Communications\n"
printf( TITLOS );
```

Το αποτέλεσμα είναι:

```
TEI of Serres
Dpt of Informatics and Communications
```

Μία συνηθισμένη χρήση της define είναι για τον καθορισμό του μεγέθους στοιχείων, όπως είναι η διάσταση ενός πίνακα, τα οποία μπορεί να αλλάξουν κατά την εκτέλεση του προγράμματος. Περισσότερα για αυτό το ζήτημα θα αναφερθούν στο κεφάλαιο 6.

1.6 Λεξιλόγιο της γλώσσας C

Από τα αναφερθέντα στην προηγούμενη παράγραφο γίνεται φανερό ότι η λειτουργία τής C στηρίζεται σε ένα λεξιλόγιο. Το λεξιλόγιο της C περιλαμβάνει τέσσερις μείζονες κατηγορίες λέξεων:

1. Δεσμευμένες λέξεις

2. Λέξεις κλειδιά

3. Τελεστές

4. Αναγνωριστές

1.6.1 Δεσμευμένες λέξεις

Οι δεσμευμένες λέξεις (reserved words) χρησιμοποιούνται από τη C κατά τρόπο αποκλειστικό και πρέπει να αποφεύγεται η χρήση τους ως ονόματα. Αποτελούνται από:

- Ονόματα συναρτήσεων πρότυπης βιβλιοθήκης (runtime function names), όπως ***printf***, ***abs*** κ.λ.π.
- Macro names. Είναι ονόματα που περιέχονται σε αρχεία κεφαλίδας για ορισμό μακροεντολών, π.χ. **EOF**, **INT_MAX**.
- Type names. Είναι ονόματα τύπων σε ορισμένα αρχεία κεφαλίδας, π.χ. **time_t**, **va_list**.
- Ονόματα εντολών προεπεξεργαστή (preprocessor). Είναι ονόματα που χρησιμοποιεί προεπεξεργαστής της C και έχουν προκαθορισμένη σημασία, π.χ. ***include***, ***define***.
- Ονόματα που αρχίζουν με το χαρακτήρα υπογράμμισης _ και έχουν δεύτερο χαρακτήρα τον ίδιο ή κεφαλαίο γράμμα, π.χ. **_DATE_**, **_FILE_**.

1.6.2 Λέξεις κλειδιά

Οι λέξεις κλειδιά (keywords) είναι λεκτικές μονάδες, οι οποίες είτε μόνες τους είτε με άλλες λεκτικές μονάδες χαρακτηρίζουν κάποια γλωσσική κατασκευή. Π.χ. η λέξη **int** αναπαριστά τον ακέραιο τύπο δεδομένων και το ζεύγος **if–else** χρησιμοποιείται στον έλεγχο ροής προγράμματος.

Οι λέξεις κλειδιά, αν και είναι ένας περιορισμός των γλωσσών, αυξάνουν την αναγνωσιμότητα και αξιοπιστία των προγραμμάτων ενώ ταυτόχρονα επιταχύνουν τη διαδικασία της μεταγλώττισης. Λέξεις κλειδιά όπως **if**, **else**, **for**, **case**, **while**, **do** έχουν γίνει κοινά αποδεκτές, διευκολύνοντας την εκμάθηση των γλωσσών προγραμματισμού.

Στον πίνακα που ακολουθεί παρατίθενται οι λέξεις κλειδιά της C:

auto	**else**	**register**	**union**
break	**enum**	**return**	**unsigned**
case	**extern**	**short**	**void**

char	**float**	**signed**	**volatile**
const	**for**	**sizeof**	**while**
continue	**goto**	**static**	
default	**if**	**struct**	
do	**int**	**switch**	
double	**long**	**typedef**	

Πίνακας 1-1. *Λέξεις κλειδιά της C*

1.6.3 Αναγνωριστές

Οι αναγνωριστές (identifiers) είναι λεκτικές μονάδες που κατασκευάζει ο προγραμματιστής. Αυτές οι λεκτικές μονάδες χρησιμοποιούνται συνήθως ως ονόματα που ο προγραμματιστής δίνει σε δικές του κατασκευές, όπως μεταβλητές, σταθερές, συναρτήσεις και δικούς του τύπους δεδομένων. Ένα όνομα προσδιορίζει μοναδιαία, από το σύνολο των κατασκευών του προγράμματος, την κατασκευή στην οποία αποδόθηκε, εξ ου και το όνομα αναγνωριστής. Περισσότερα στοιχεία για τους αναγνωριστές σημειώνονται στην §2.1.2.

1.7 Κανόνες δημιουργίας ευανώγνωστων προγραμμάτων

- Θα πρέπει να αποφεύγονται ονόματα ενός χαρακτήρα, όπως **i, j, x, y** (εκτός από ειδικές περιπτώσεις που θα εξετασθούν αργότερα).
- Τα ονόματα που χρησιμοποιούνται θα πρέπει να είναι εκφραστικά ονόματα. Συγκεκριμένα:
  1. Η μεταβλητή που αναπαριστά την ταχύτητα θα μπορούσε να ονομασθεί **velocity** και τη μέγιστη τιμή της **max_velocity** ή **maxVelocity**.
  2. Η συνάρτηση που εμφανίζει τα λάθη στην οθόνη μπορεί να λάβει τα ενδεικτικά ονόματα **display_error** ή **displayError**.
- Για καλύτερη αναγνωσιμότητα των μεταβλητών που αποτελούνται από δύο ή περισσότερες λέξεις ο προγραμματιστής θα πρέπει να αποφασίσει εάν θα χρησιμοποιήσει τη μορφή ***display_error*** ή τη μορφή ***displayError***. Η σύμβαση που θα επιλεγεί θα πρέπει να τηρηθεί σε όλο το πρόγραμμα.

- Θα πρέπει να χρησιμοποιούνται μικρά γράμματα για ονόματα μεταβλητών.

2. Μεταβλητές - σταθερές - I/O κονσόλας

2.1 Η έννοια της μεταβλητής

Ένα από τα βασικότερα πλεονεκτήματα του υπολογιστή είναι η δυνατότητά του να διαχειρίζεται πληροφορία, σε μορφή αριθμητικών δεδομένων, γραμμάτων ή ακολουθίας γραμμάτων. Τα δεδομένα αποθηκεύονται στη μνήμη και απαιτούν ένα μέσο για να κληθούν από τα προγράμματα, να εισαχθούν σε υπολογισμούς και να δημιουργήσουν νέα δεδομένα. Αρχικά οι προγραμματιστές χειρίζονταν τα δεδομένα δουλεύοντας με τις διευθύνσεις μνήμης στις οποίες ήταν αποθηκευμένα. Με την αύξηση του μεγέθους και της πολυπλοκότητας των προγραμμάτων, αυτός ο τρόπος διαχείρισης έθετε πολλούς περιορισμούς και αποτελούσε πηγή προβλημάτων.

Η λύση στο πρόβλημα της αναφοράς σε δεδομένα και στη διαχείρισή τους δόθηκε με την εισαγωγή της έννοιας των μεταβλητών, οι οποίες είναι φορείς δεδομένων. Το όνομα μίας μεταβλητής είναι άμεσα συνδεδεμένο με τη διεύθυνση στην οποία είναι αποθηκευμένο το δεδομένο. Με τον τρόπο αυτό ο προγραμματιστής μπορεί να χειρισθεί δεδομένα χωρίς να γνωρίζει την ακριβή διεύθυνση της μνήμης όπου αυτά τοποθετούνται.

Επιγραμματικά, η χρήση των μεταβλητών είναι ίδια με εκείνη της άλγεβρας, π.χ. η παράσταση **y=3x+5** ισχύει τόσο στα μαθηματικά όσο και στις γλώσσες προγραμματισμού, με τα **x** και **y** να είναι μεταβλητές. Ωστόσο στον προγραμματισμό η χρήση της είναι **γενικευμένη**: η μεταβλητή είναι μία θέση μνήμης για ένα δεδομένο. Δημιουργείται όταν δηλώνεται και η τιμή της μπορεί να είναι άγνωστη έως ότου χρησιμοποιηθεί από το πρόγραμμα.

2.1.1 Δήλωση μεταβλητής

Οι μεταβλητές δηλώνονται με ***πρόταση ορισμού***, η οποία τελειώνει πάντοτε με (**;**). Η μορφή της δήλωσης είναι: **data_type var, var, … ;**

π.χ. **int counter1, counter2;**

Οι μεταβλητές δηλώνονται στην αρχή μίας συνάρτησης, συνήθως αμέσως μετά το εισαγωγικό άγκιστρο (**{**), και οπωσδήποτε πριν από τη χρήση τους.

Η δήλωση γνωστοποιεί στο μεταγλωττιστή το όνομα και τις ιδιότητες της

μεταβλητής. Μία δήλωση έχει ως αποτέλεσμα τη σύνδεση του ονόματος της μεταβλητής με:

- τον ανάλογο τύπο δεδομένων, γεγονός που λαμβάνει χώρα στο χρόνο μεταγλώττισης (compile–time)
- μία θέση μνήμης κατάλληλου μεγέθους, γεγονός που λαμβάνει χώρα στο χρόνο εκτέλεσης (run–time)

2.1.2 Ονομασία μεταβλητής

Σε ό,τι αφορά την ονοματολογία, ακολουθούνται οι εξής κανόνες:

- Στην C τα ονόματα των μεταβλητών σχηματίζονται από:
  - α) τα γράμματα του αλφαβήτου
  - β) τα ψηφία 0 έως 9
  - γ) το χαρακτήρα υπογράμμισης __ (underscore)
- Το όνομα πρέπει να ξεκινά με γράμμα ή με χαρακτήρα υπογράμμισης (στη δεύτερη περίπτωση ο επόμενος χαρακτήρας πρέπει να είναι μικρό γράμμα).
- Το όνομα δεν πρέπει να είναι ίδιο με δεσμευμένη λέξη.
- Σημαντικοί είναι μόνο οι πρώτοι 31 χαρακτήρες του ονόματος. Οι υπόλοιποι δε λαμβάνονται υπόψη.
- Τα όνομα μεταβλητής πρέπει να είναι ενδεικτικό της ιδιότητάς της ή του τύπου δεδομένου που αντιπροσωπεύει, έτσι ώστε να δίνει πληροφορία στον προγραμματιστή.

Με βάση τα παραπάνω, ακολουθούν ενδεικτικές περιπτώσεις ονοματολογίας.

- Έγκυρα ονόματα μεταβλητών:

  totalArea **max_amount** **counter1**

  Counter1 **_temp_in_F**
- Μη έγκυρα ονόματα μεταβλητών:

  $product **total%** **3rd**
- Απαράδεκτα ονόματα μεταβλητών:

  l[1] **x2**

[1] Στην πορεία ανάγνωσης του κειμένου ο αναγνώστης θα παρατηρήσει ότι οι ονομασίες των μεταβλητών δε συνάδουν πάντοτε με τους κανόνες που αναφέρονται, καθώς χρησιμοποιούνται

maximum_number_of_students_in_my_class

2.2 Τύποι μεταβλητών

Οι μεταβλητές της γώσσας C ανήκουν σε δύο κατηγορίες. Η κατηγορία των βαθμωτών τύπων περιλαμβάνει τους ακέραιους (integers), οι οποίοι δηλώνονται με την κωδική λέξη **int**, τους πραγματικούς, οι οποίοι μερίζονται στους αριθμούς κινητής υποδιαστολής με κωδική λέξη **float** και τους αριθμούς διπλής ακρίβειας με κωδική λέξη **double**, τη μεταβλητή χαρακτήρα (character, **char**), τους δείκτες (pointers) και τον απαριθμητικό τύπο (enumerated, **enum**).

Στην κατηγορία των συναθροιστικών τύπων ανήκουν οι πίνακες, οι δομές (**struct**) και οι ενώσεις (**union**). Στη συνέχεια του κεφαλαίου γίνεται αναφορά στους βασικούς τύπους της C: **char**, **int**, **float**, **double** και σε επόμενο κεφάλαιο αναπτύσσονται οι πίνακες. Οι άλλοι τύποι μεταβλητών θα μελετηθούν αργότερα.

2.2.1 Ο τύπος του χαρακτήρα

Ο τύπος του χαρακτήρα (**char**) παριστάνει απλούς χαρακτήρες του αλφάβητου της γλώσσας. Βρίσκεται ανάμεσα σε <u>απλά</u> εισαγωγικά (π.χ. **'C'**, **'2'**, **'*'**, **')'**).

- Η δήλωση της μεταβλητής χαρακτήρα ακολουθεί τον εξής φορμαλισμό:

char όνομα_μεταβλητής; π.χ. **char choice**;

Υπάρχει η δυνατότητα ταυτόχρονα με τη δήλωση να αποδοθεί αρχική τιμή στη μεταβλητή:

char choice='A';

- Η εισαγωγή τιμών στις μεταβλητές χαρακτήρα γίνεται με χρήση της συνάρτησης ***scanf*** και του **προσδιοριστή** (specifier) **%c** (character). Η πρόταση

scanf("%c", &ch);

διαβάζει από την κύρια είσοδο (πληκτρολόγιο) ένα χαρακτήρα και τον αποδίδει στη μεταβλητή **ch**. Θα πρέπει να προσεχθεί η χρήση του **&** πριν από τη μεταβλητή. Ονομάζεται **τελεστής διεύθυνσης** και προηγείται πάντοτε των μεταβλητών στην εντολή ***scanf***.

μεταβλητές με ένα ή δύο γράμματα. Η χρήση τους γίνεται *συγγραφική αδεία*, επιβληθείσα για λόγους πρακτικούς, έτσι ώστε να μη διευρύνεται ο κώδικας.

- Η εκτύπωση μεταβλητών χαρακτήρα γίνεται με χρήση της συνάρτησης ***printf*** και του προσδιοριστή **%c**. Η πρόταση

printf("The character is %c\n", choice);

θα τυπώσει (θεωρώντας ότι στη **choice** εισήχθη ο **A**)

The character is A

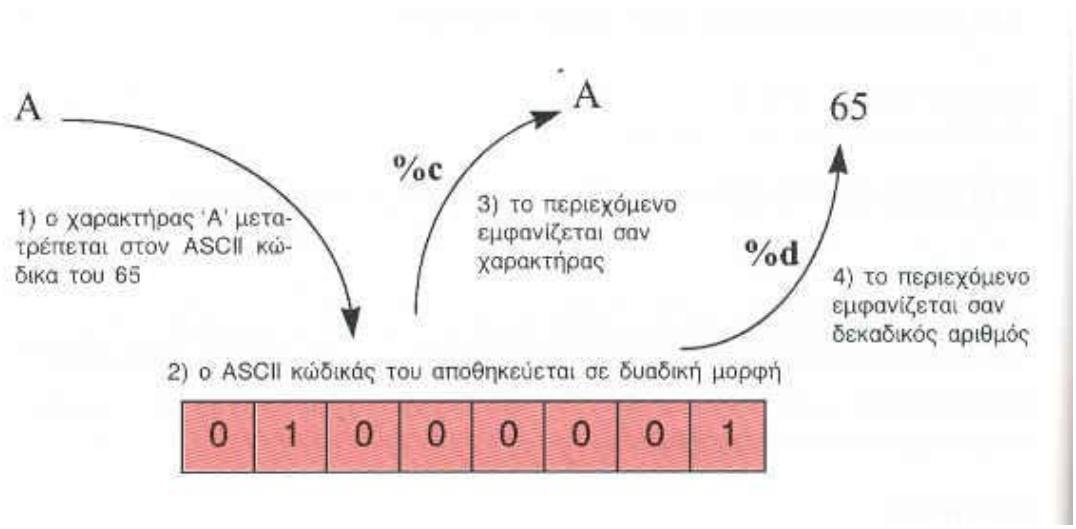
<u>Παρατήρηση:</u> Επειδή κάθε χαρακτήρας του κώδικα ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) αντιστοιχεί σε έναν οκταψήφιο δυαδικό αριθμό, εάν αντί του **%c** χρησιμοποιηθεί ο προσδιοριστής **%d** (decimal), η ***printf*** θα εμφανίσει τον ASCII κωδικό τού χαρακτήρα. Η πρόταση

printf("The ASCII Code of %c is %d\n", choice,choice);

θα τυπώσει

The ASCII Code of A is 65

όπου **65** είναι το δεκαδικό ισοδύναμο του δυαδικού κωδικού ASCII για το χαρακτήρα **A**.



Σχήμα 2-1. *Αποθήκευση και ανάκληση ASCII χαρακτήρα*

- Ο μεταγλωττιστής απαιτεί 1 byte μνήμης για την αποθήκευση της τιμής μίας μεταβλητής χαρακτήρα. Στο σχήμα 2-1 παρουσιάζονται οι διαδικασίες της αποθήκευσης και ανάκλησης για το χαρακτήρα A. Η τιμή της μεταβλητής μετατρέπεται σε ακέραιο (ενέργεια 1 του σχήματος 2-1), ο οποίος αποθηκεύεται (ενέργεια 2). Στην περίπτωση ανάκλησης της τιμής, εκτελείται η αντίστροφη διεργασία. Ο αριθμός μετατρέπεται σε χαρακτήρα μέσω του προσδιοριστή **%c** και

ακολούθως είτε τυπώνεται ο χαρακτήρας (ενέργεια 3) είτε τυπώνεται το δεκαδικό ισοδύναμο μέσω του προσδιοριστή **%d** (ενέργεια 4). Σε κάθε περίπτωση, ο μεταγλωττιστής είναι υπεύθυνος για το ότι ο υπολογιστής διαχειρίζεται τα bits και bytes σύμφωνα με τους τύπους που δηλώνονται.

2.2.2 Μη εκτυπούμενοι χαρακτήρες

Οι σταθερές τύπου χαρακτήρα ***«νέα γραμμή (new-line)»*** και ***«στηλοθέτης (tab)»*** ανήκουν στην κατηγορία των μη εκτυπούμενων χαρακτήρων, τους οποίους η C αναπαριστά με τις ***«ακολουθίες διαφυγής (escape sequences)»*** **\n** και **\t**, αντίστοιχα. Η παρακάτω πρόταση δίνεται ως παράδειγμα χρήσης χαρακτήρων διαφυγής:

printf("Write, \" a \\ is a backslash. \"\n");

Η πρόταση θα εμφανίσει στην κύρια έξοδο (οθόνη):

Write, " a \ is a backslash. "

Οι μη εκτυπούμενοι χαρακτήρες και οι αντίστοιχες ακολουθίες διαφυγής παρατίθενται στον πίνακα 2-1:

Χαρακτήρας	Ακολουθία	Χαρακτήρας	Ακολουθία
συναγερμός (κουδούνι)	**\a**	πλάγια γραμμή	\\
οπισθοχώρηση	**\b**	λατινικό ερωτηματικό	**\?**
αλλαγή σελίδας	**\f**	μονό εισαγωγικό	\’
νέα γραμμή	**\n**	διπλό εισαγωγικό	\”
επαναφορά κεφαλής	**\r**	οκταδικός αριθμός	**\ooo**
οριζόντιος στηλοθέτης	**\t**	δεκαεξαδικός αριθμός	**\xhhh**
κατακόρυφος στηλοθέτης	**\v**		

Πίνακας 2-1. *Μη εκτυπούμενοι χαρακτήρες και αντίστοιχες ακολουθίες διαφυγής*

Παράδειγμα 2.1

Να καταστρωθεί πρόγραμμα που να επιτελεί τα παρακάτω:

Ζήτησε από το χρήστη ένα χαρακτήρα

Πάρε από το χρήστη το χαρακτήρα

Τύπωσε το χαρακτήρα και τον ASCII κωδικό του

Βρες τον επόμενο χαρακτήρα

Τύπωσέ τον μαζί με τον κωδικό του

```
#include <stdio.h>
void main()
{
    char ch,next_ch;
    printf( "Write a character:\t" );
    scanf( "%c",&ch );
    printf( "The ASCII code of char %c is %d\n", ch, ch );
    next_ch=ch+1;     /* βρίσκει τον επόμενο χαρακτήρα */
    printf( "The ASCII code of char %c is %d\n", next_ch, next_ch );
}
```

2.2.3 Ο τύπος του ακεραίου

Ο τύπος του ακεραίου (**int** από τη λέξη integer) χρησιμοποιείται για να παραστήσει ακέραιους αριθμούς. Η περιοχή τιμών εξαρτάται από την αρχιτεκτονική του μηχανήματος. Για έναν υπολογιστή που διαθέτει 2 bytes (16 bits) για κάθε ακέραιο, το σύνολο των δυνατών τιμών είναι $2^{16} = 65536$. Εάν θεωρηθεί ότι τις μισές τιμές θα καταλάβουν αρνητικοί ακέραιοι και τις υπόλοιπες μισές θετικοί, η περιοχή τιμών του τύπου **int** είναι από -32768 έως +32767, με την ακόλουθη αντιστοιχία δυαδικής και δεκαδικής τιμής:

$$(0000000000000000)_2 \rightarrow (-32768)_{10}$$

$$(0111111111111111)_2 \rightarrow (-1)_{10}$$

$$(1000000000000000)_2 \rightarrow (0)_{10}$$

$$(1111111111111111)_2 \rightarrow (+32767)_{10}$$

- Η δήλωση της μεταβλητής ακεραίου ακολουθεί τον εξής φορμαλισμό:

int όνομα_μεταβλητής; π.χ. **int num**;

Υπάρχει η δυνατότητα ταυτόχρονα με τη δήλωση να αποδοθεί αρχική τιμή στη μεταβλητή:

int num=46;

- Εάν πριν από τη λέξη **int** τοποθετηθεί ο προσδιοριστής **long** τότε οι ακέραιοι **long int** εξασφαλίζουν αποθηκευτικό χώρο 32 bits. Αντίστοιχα, ο προσδιοριστής **unsigned** χρησιμοποιείται πριν από τη λέξη **int** για να χαρακτηρίσει τη μεταβλητή χωρίς πρόσημο, η οποία λαμβάνει τιμές από 0 έως 65535 για ακέραιο 16 bits.

Σε περιβάλλοντα 4 bytes (32 bits) όπως τα Windows XP, το σύνολο των δυνατών τιμών είναι $2^{32} = 4.294.967.296$. Έτσι οι προσημασμένοι ακέραιοι αποθηκεύουν τιμές στο διάστημα από -2.147.483.648 έως +2.147.483.647.

Ένας ακέραιος **short int** είναι τουλάχιστον 16 bits και ο **int** είναι <u>τουλάχιστον</u> τόσο μεγάλος όσο ο **short int** (συνήθως είναι 32 bits).

- Η εισαγωγή τιμών στις ακέραιες μεταβλητές γίνεται με χρήση της συνάρτησης ***scanf*** και του προσδιοριστή **%d**. Η πρόταση

scanf("%d", &num);

διαβάζει από το πληκτρολόγιο έναν ακέραιο και τον αποδίδει στη μεταβλητή **num**.

- Η εκτύπωση ακέραιων μεταβλητών γίνεται με χρήση της συνάρτησης ***printf*** και των προσδιοριστών **%d**, **%o**, **%x** για την εμφάνιση σε δεκαδική, οκταδική και δεκαεξαδική μορφή, αντίστοιχα. Οι προσδιοριστές **l** (long), **h** (short), και **u** (unsigned) τοποθετούνται πριν από τους **d**, **o**, **x**. Η πρόταση

printf("dec=%d, octal=%o, hex=%x", num,num,num);

θα τυπώσει (θεωρώντας ότι η **num** λαμβάνει την τιμή **46**):

dec=46, octal=56, hex=2e

<u>Παρατηρήσεις:</u>

1. Υπάρχει η δυνατότητα να καθορισθεί ο αριθμός των ψηφίων που θα τυπωθούν, τοποθετώντας τον επιθυμητό αριθμό ανάμεσα στο **%** και το **d**. Εάν ο αριθμός είναι μικρότερος από τον απαιτούμενο αριθμό ψηφίων του ακέραιου, η επιλογή δε θα ληφθεί υπόψη. Στην αντίθετη περίπτωση, στις πλεονάζουσες θέσεις θα τοποθετηθούν

κενά. Για το λόγο αυτό, ο αριθμός που τοποθετείται στον προσδιοριστή **%d** ονομάζεται **καθοριστικό ελάχιστου πλάτους πεδίου**. Με αυτόν τον τρόπο, σε διαδοχικές ***printf*** θα υπάρξει ευθυγράμμιση των αποτελεσμάτων κατά στήλες (βλ. Παράδειγμα 5.9). Οι προτάσεις

printf("dec=%1d, octal=%4o, hex=%4x", num,num,num);

printf("dec=%4d, octal=%4o, hex=%4", num,num,num);

θα τυπώσουν αντίστοιχα

dec=46, octal= 56, hex= 2e

dec= 46, octal= 56, hex= 2e

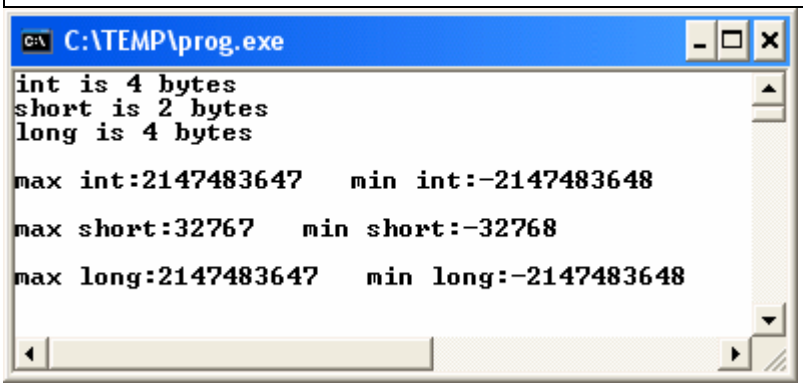
2. Όταν γράφεται ένας αριθμός στον πηγαίο κώδικα χωρίς δεκαδικό ή εκθετικό μέρος, ο μεταγλωττιστής το χειρίζεται ως **ακέραια σταθερά**. Η σταθερά **245** αποθηκεύεται ως **int**, ενώ η σταθερά **100000** αποθηκεύεται ως **long int**. Εάν ορισθεί η σταθερά **8965** ως **8965L**, ο μεταγλωττιστής αναγκάζεται να δεσμεύσει χώρο για **long int**.

Παράδειγμα 2.2

Να καταστρωθεί πρόγραμμα που να εξετάζει το μήκος του τύπου ακεραίου.

```
#include <stdio.h>
#include <climits.h>    // limits.h για παλαιότερα συστήματα
void main()
{
    int number_int=INT_MAX;  // Μέγιστος int: ορίζεται στο climits.h
    short int number_short=SHRT_MAX;  // Μέγιστος short int
    long int number_long=LONG_MAX;  // Μέγιστος long integer
    /* ο τελεστής sizeof δίνει το μέγεθος ενός τύπου δεδομένου
    ή μίας μεταβλητής */
    printf( "int is %d bytes\n",sizeof(int) );
    printf( "short is %d bytes\n",sizeof(short) );
    printf( "long is %d bytes\n",sizeof(long) );
```

```
        printf( "\nmax int:%d  min int:%d\n",number_int,INT_MIN );
        printf( "\nmax short:%d min short:%d\n",SHRT_MAX,SHRT_MIN
    );
        printf( "\nmax long:%d min long:%d\n",number_long,LONG_MIN );
    }
```



Από τα αποτελέσματα προκύπτει ότι ταυτίζονται τα bytes για **int** και **long**, γιατί στην έκδοση 5 της Borland C++ και με λειτουργικά συστήματα Windows 95 και μεταγενέστερα ο **int** καταλαμβάνει 4 bytes.

2.2.4 Τύποι πραγματικών αριθμών

Οι πραγματικοί αριθμοί είναι οι αριθμοί που διαθέτουν κλασματικό μέρος και εκφράζονται συνήθως στις ακόλουθες μορφές:

Αριθμός με δεκαδικά	*Επιστημονική σημειογραφία*	*Εκθετική σημειογραφία*
123.456	1.23456x102	1.23456e+02
0.00002	2.0x10-5	2.0e-5
50000.0	2.0x104	5.0e+04

Η γλώσσα C διαθέτει δύο τύπους για αναπαράσταση πραγματικών αριθμών. Τον τύπο **float** για αριθμούς κινητής υποδιαστολής απλής ακρίβειας και τον τύπο **double** για αριθμούς κινητής υποδιαστολής διπλής ακρίβειας.

- Η δήλωση της μεταβλητής float ή double ακολουθεί τον εξής φορμαλισμό:

float όνομα_μεταβλητής; π.χ. **float plank=6.63e-34**;

Η χρήση του προσδιοριστή **long** πριν από τον τύπο **double** χρησιμοποιείται για δήλωση μεταβλητής κινητής υποδιαστολής εκτεταμένης ακρίβειας, π.χ.

long double plank;

- Η εισαγωγή τιμών στις μεταβλητές κινητής υποδιαστολής γίνεται με χρήση της συνάρτησης ***scanf*** και του προσδιοριστή **%f** (float). Η πρόταση

scanf("%f", &num);

διαβάζει από το πληκτρολόγιο έναν πραγματικό αριθμό και τον αποδίδει στη μεταβλητή **num**.

- Η εκτύπωση πραγματικών μεταβλητών γίνεται με χρήση της συνάρτησης ***printf*** και των προσδιοριστών **%f** για εμφάνιση σε δεκαδική μορφή, **%e** για εμφάνιση σε εκθετική μορφή, και **%g** για να ανατεθεί στο σύστημα να επιλέξει μεταξύ των δύο προηγουμένων, με προτεραιότητα στη μορφή με το μικρότερο μέγεθος.

- Σε ό,τι αφορά το χώρο που καταλαμβάνουν στη μνήμη, ως συνηθισμένα μεγέθη αναφέρονται για τους μεν **float** τα 32 bits, για τους δε **double** τα 64 bits. Θα πρέπει να σημειωθεί ότι, σε αντιδιαστολή με τους ακέραιους αριθμούς, δεν υπάρχει αντιστοιχία ένα προς ένα ανάμεσα στους πραγματικούς αριθμούς και στις απεικονίσεις τους στις γλώσσες προγραμματισμού. Οι αριθμοί που αντιστοιχούν στους πραγματικούς αριθμούς είναι προσεγγίσεις αυτών και ονομάζονται **αριθμοί μηχανής**, εξαιτίας της ανάγκης να απεικονισθούν με πεπερασμένο αριθμό ψηφίων πραγματικοί αριθμοί που θεωρητικά μπορούν να περιέχουν άπειρο αριθμό κλασματικών ψηφίων. Σε μία μεταβλητή τύπου **float** των 32 bits τα 8 bits χρησιμοποιούνται για τον εκθέτη, ένα για το πρόσημο και τα υπόλοιπα 23 για το κλασματικό μέρος. Η μορφή του αριθμού είναι η ακόλουθη:

$$\pm(\ .\ d_1 d_2 ... d_{23}\ ) \cdot 2^e$$

όπου τα ψηφία $d_1 ... d_{23}$ είναι δυαδικά και το e είναι το δεκαδικό ισοδύναμο του οκταψήφιου δυαδικού εκθέτη. Κατά σύμβαση το $d_1 = 1$. Το δεκαδικό ισοδύναμο της

ανωτέρω μορφής είναι:

$$\pm\left[(d_1 \cdot 2^{-1}) + (d_2 \cdot 2^{-2}) + \ldots + (d_{23} \cdot 2^{-23})\right] \cdot 2^e = \pm 2^e \cdot \sum_{i=1}^{23} d_i \cdot 2^{-i}$$

Κατά συνέπεια, με βάση το παραπάνω σύστημα απεικόνισης το μέγεθος της κλασματικής ακρίβειας ενός αριθμού κινητής υποδιαστολής καθορίζεται από τον αριθμό των κλασματικών ψηφίων.

Ο οκταψήφιος δυαδικός εκθέτης αντιστοιχεί σε $2^8 = 256$ δυνατές τιμές. Από αυτές οι 127 δίνονται σε αρνητικούς ακέραιους και οι υπόλοιπες 129 σε θετικούς, με την ακόλουθη αντιστοιχία δυαδικής και δεκαδικής τιμής:

$$e_{\min} = (00000000)_2 \quad \rightarrow \quad e_{\min} = (-127)_{10}$$

$$e_{\max} = (11111111)_2 \quad \rightarrow \quad e_{\max} = (128)_{10}$$

Με βάση τα παραπάνω, η μέγιστη απόλυτη τιμή πραγματικών αριθμών που μπορεί να επιτευχθεί με μεταβλητή τύπου **float** των 32 bits είναι (βλ. Παράδειγμα 2.3):

$$|\max| = \left[(1 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + \ldots + (1 \cdot 2^{-23})\right] \cdot 2^{128} = 3.402823e + 38$$

Στους αριθμούς διπλής ακρίβειας δεσμεύονται 64 bits, εκ των οποίων τα 11 δίνονται στον εκθέτη, ένα στο πρόσημο και 52 στο κλασματικό μέρος.

<u>Παρατηρήσεις:</u>

1. Όπως σημειώθηκε στους ακέραιους, έτσι και στους πραγματικούς υπάρχει η δυνατότητα να καθορισθεί ο αριθμός των ψηφίων που θα εκτυπωθούν, τοποθετώντας τον επιθυμητό αριθμό ανάμεσα στο **%** και το **f**. Μάλιστα ο αριθμός θα είναι της μορφής **a.b**, με το **a** να δηλώνει το συνολικό αριθμό των ψηφίων – συμπεριλαμβανομένου του προσήμου – και το **b** να δηλώνει τον αριθμό των δεκαδικών ψηφίων. Οι προτάσεις

```
float num=46.37;
printf( "num=%8.4f,  num=%12.1f\n", num,num );
```

θα τυπώσουν

num= 46.3700, num= 46.4

Είναι φανερό ότι στην περίπτωση που ο αριθμός των δεκαδικών ψηφίων που ζητούνται είναι μικρότερος από τον απαιτούμενο γίνεται στρογγυλοποίηση (το **37** στρογγυλοποιήθηκε στο **40** και παρελήφθη το **0**).

2. Πραγματικοί αριθμοί όπως οι

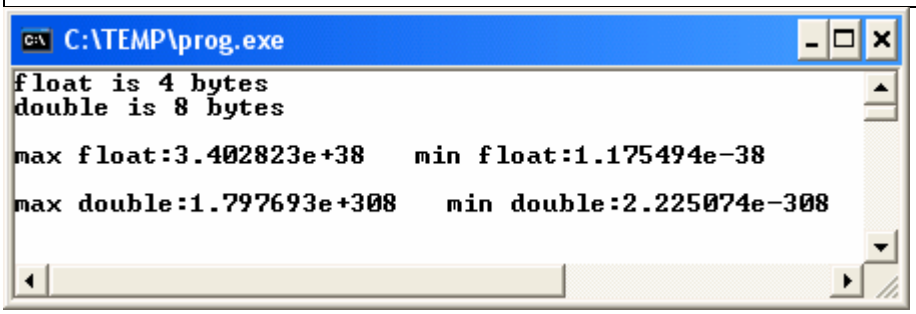
0.12 45.68 9e-5 24e09 0.0034e-08

όταν εμφανίζονται στον πηγαίο κώδικα αποτελούν τις **πραγματικές σταθερές**. Θεωρούνται από το μεταγλωττιστή ως **double** και δεσμεύουν τον αντίστοιχο χώρο.

Παράδειγμα 2.3

Να καταστρωθεί πρόγραμμα που να εξετάζει το μήκος του τύπου κινητής υποδιαστολής.

```
#include <stdio.h>
#include <float.h>      // για τα όρια του float
void main()
{
    float num_float=FLT_MAX;    // Μέγιστος float
    double num_double=DBL_MAX;  // Μέγιστος double
    /* ο τελεστής sizeof δίνει το μέγεθος ενός τύπου δεδομένου
    ή μίας μεταβλητής */
    printf( "float is %d bytes\n",sizeof(float) );
    printf( "double is %d bytes\n",sizeof(double) );
    printf( "\nmax float:%e min float:%e\n",num_float,FLT_MIN);
    printf( "\nmax double:%e,min double:%e\n"
num_double,DBL_MIN);
    printf( "\n\nPress any key to continue" );
}
```



2.3 I/O κονσόλας

Η I/O κονσόλας αναφέρεται στις λειτουργίες που γίνονται στο πληκτρολόγιο και στην οθόνη του υπολογιστή. Πέραν των **printf** και **scanf**, που χρησιμοποιήθηκαν προγουμένως (και αποτελούν τη μορφοποιούμενη I/O κονσόλας γιατί μπορούν να διαβάζουν και να τυπώσουν δεδομένα σε διάφορες μορφές), υπάρχει μία σειρά απλούστερων συναρτήσεων, που αναπτύσσεται στις επόμενες παραγράφους.

2.3.1 Οι συναρτήσεις getche, getch

Η συνάρτηση ***getche*** διαβάζει ένα χαρακτήρα από την κύρια είσοδο. Αναμένει έως ότου πατηθεί ένα πλήκτρο και στη συνέχεια επιστρέφει την τιμή του, εμφανίζοντας στην οθόνη το πλήκτρο που πατήθηκε. Το πρωτότυπο της ***getche*** είναι το ακόλουθο:

int getche(void);

και το αρχείο κεφαλίδας της συνάρτησης βρίσκεται στο ***conio.h***.

Η ***getche*** επιστρέφει μεν έναν ακέραιο αλλά το byte χαμηλής τάξης είναι αυτό που περιέχει τον χαρακτήρα. Η χρήση ακεραίων γίνεται για λόγους συμβατότητας με τον αρχικό μεταγλωττιστή της UNIX C.

Η συνάρτηση ***getch*** αποτελεί παραλλαγή της ***getche*** και βρίσκεται επίσης στο ***conio.h***. Λειτουργεί όπως ακριβώς η ***getche***, με τη διαφορά ότι η ***getch*** δεν εμφανίζει τον πληκτρολογηθέντα χαρακτήρα στην οθόνη.

2.3.2 Η συνάρτηση getchar

Η συνάρτηση ***getchar*** (get character) διαβάζει ένα χαρακτήρα από την κύρια είσοδο και τον επιστρέφει στο πρόγραμμα. Αποτελεί παραλλαγή της ***getche***. Είναι η αρχική συνάρτηση εισόδου χαρακτήρων και βασίζεται στο UNIX. Το πρόβλημα με τη συνάρτηση αυτή είναι ότι κρατά την είσοδο στην περιοχή προσωρινής αποθήκευσης μέχρι να δοθεί επαναφορά κεφαλής. Έτσι, μετά την επιστροφή της ***getchar*** περιμένουν ένας ή περισσότεροι χαρακτήρες στην ουρά εισόδου.

Το πρωτότυπο της ***getchar*** είναι το ακόλουθο:

int getchar(void);

και το αρχείο κεφαλίδας της συνάρτησης βρίσκεται στο ***stdio.h***.

2.3.3 Η συνάρτηση putchar

Η συνάρτηση ***putchar*** (put character) εμφανίζει στην οθόνη το χαρακτήρα που έχει ως όρισμα (π.χ. **c**), στην τρέχουσα θέση του δρομέα. Το πρωτότυπο της ***putchar*** είναι:

int putchar(int c);

Η ***putchar*** επιστρέφει μεν έναν ακέραιο αλλά το byte χαμηλής τάξης είναι αυτό που περιέχει το χαρακτήρα. Όπως συνέβη και με τις προηγούμενες συναρτήσεις, η χρήση ακεραίων γίνεται για λόγους συμβατότητας με τον αρχικό μεταγλωττιστή της UNIX C. Το αρχείο κεφαλίδας της συνάρτησης ***putchar*** βρίσκεται στο ***stdio.h***.

2.3.4 Η συνάρτηση kbhit

Η συνάρτηση ***kbhit*** (keyboard hit) ελέγχει κατά πόσον ο χρήστης έχει πατήσει κάποιο πλήκτρο. Εφόσον έχει πατήσει κάποιο πλήκτρο η συνάρτηση επιστρέφει ως αληθής, σε αντίθετη περίπτωση επιστρέφει ως ψευδής. Η συνάρτηση ***kbhit*** χρησιμοποιείται κυρίως για να διακόπτει ο χρήστης το πρόγραμμα κατά το δοκούν.

Το πρωτότυπο της ***kbhit*** είναι:

int kbhit(void);

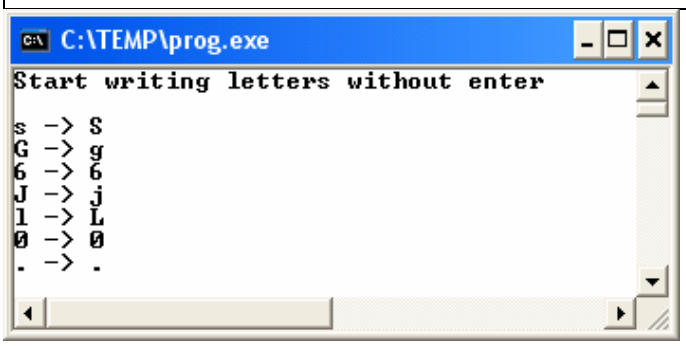
Το αρχείο κεφαλίδας της συνάρτησης ***kbhit*** βρίσκεται στο ***conio.h***. Παραδείγματα όπου γίνεται χρήση της ***kbhit***. θα παρουσιασθούν στη συνέχεια.

Παράδειγμα 2.4

Το ακόλουθο πρόγραμμα παίρνει χαρακτήρες από το πληκτρολόγιο και μετατρέπει τα κεφαλαία γράμματα σε μικρά και τούμπαλιν. Το πρόγραμμα τερματίζει μόλις πληκτρολογηθεί μία τελεία (**.**). Το αρχείο κεφαλίδας ***ctype.h*** απαιτείται για τη συνάρτηση ***islower***, που αληθεύει εάν το όρισμά της είναι σε μικρά γράμματα, και τις συναρτήσεις ***toupper***, ***tolower***, που μετασχηματίζουν τα γράμματα.[2]

[2] Ο αναγνώστης καλείται να παρακάμψει την επαναληπτική πρόταση και την υπό συνθήκη διακλάδωση. Θα μελετηθούν εκτενώς αργότερα.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
void main()  {
        char ch;
        printf( "Start writing letters without enter\n\n" );
        do  {
                ch=getche();
                printf( " -> ",ch );
                if (islower(ch)) putchar(toupper(ch));
                else putchar(tolower(ch));
                printf( "\n" );
        }
        while (ch!='.');        // τέλος προγράμματος με '.'
}
```



3. Τελεστές - εκφράσεις

3.1 Ορισμοί - σημειογραφίες

Οι τελεστές (operators) είναι σύμβολα ή λέξεις που αναπαριστούν συγκεκριμένες διεργασίες, οι οποίες εκτελούνται επί ενός ή περισσότερων δεδομένων. Τα δεδομένα καλούνται **τελεστέοι** (operands) και μπορούν να είναι μεταβλητές, σταθερές ή ακόμη και κλήσεις συναρτήσεων.

Οι τελεστές χρησιμοποιούνται για το σχηματισμό εκφράσεων. Μία έκφραση, εν γένει, αποτελείται από έναν ή περισσότερους τελεστέους και από έναν ή περισσότερους τελεστές. Κάθε έκφραση έχει μία τιμή, η οποία υπολογίζεται με ορισμένους κανόνες. Για παράδειγμα, στην έκφραση **num+12** ο χαρακτήρας + αναπαριστά τη διεργασία της πρόσθεσης των δύο τελεστέων, οι οποίοι είναι η μεταβλητή **num** και η σταθερά **12**.

Οι τελεστές ταξινομούνται, ανάλογα με τον αριθμό των τελεστέων στους οποίους δρουν, σε μοναδιαίους (unary), δυαδικούς (binary) και τριαδικούς (ternary). Μία δεύτερη κατηγοριοποίηση επιτελείται με βάση τη διεργασία που εκτελούν, οδηγώντας στις κατηγορίες του πίνακα 3-1:

Κατηγορία	Ενδεικτικοί τελεστές
Αριθμητικοί	+ - * /
Λογικοί	&& || !
Συσχετιστικοί	> >= == !=
Διαχείρισης bits	>> & ! ^
Διαχείρισης μνήμης	& [] . ->

Πίνακας 3-1. *Κατηγορίες τελεστών*

Τα σύμβολα των συνηθέστερων δυαδικών τελεστών στη C παρατίθενται στον πίνακα που ακολουθεί:

Δυαδικός τελεστής	Σύμβολο	Δυαδικός τελεστής	Σύμβολο
Μικρότερο	<	πρόσθεση	+
μικρότερο ή ίσο	<=	αφαίρεση	-
Ίσο	= =	πολλαπλασιασμός	*
Διάφορο	!=	διαίρεση πραγματικών	/
Μεγαλύτερο	>	πηλίκο διαίρεσης ακεραίων	/
Μεγαλύτερο ή ίσο	>=	υπόλοιπο διαίρεσης ακεραίων	%

Πίνακας 3-2. *Σύμβολα δυαδικών τελεστών*

Στη συνέχεια του κεφαλαίου θα μελετηθούν οι συνηθέστεροι των τελεστών ενώ για ενδελεχή μελέτη ο αναγνώστης μπορεί να ανατρέξει στην αναφορά [3].

3.1.1 Σημειογραφία

Η γλώσσα C δίνει τη δυνατότητα να επιτυγχάνονται διαφορετικές λειτουργίες στην ίδια έκφραση ανάλογα με τη θέση των τελεστών ανάμεσα στους τελεστέους. Για το λόγο αυτό έχουν αναπτυχθεί τρεις σημειογραφίες για τους δυαδικούς τελεστές:

- Η σημειογραφία ***ένθεσης*** ή ***ένθετου τελεστή*** (infix notation), όταν ο τελεστής τοποθετείται <u>μεταξύ</u> των τελεστέων στους οποίους ενεργεί, όπως στην έκφραση **x+y**.
- Η σημειογραφία ***πρόθεσης*** ή ***προπορευόμενου τελεστή*** (prefix notation), όταν αυτός τοποθετείται <u>πριν</u> από τους τελεστέους, όπως στην έκφραση **+x y**.
- Η σημειογραφία ***παρελκόμενου τελεστή*** (postfix notation), όταν ο τελεστής τοποθετείται <u>μετά</u> από τους τελεστέους, όπως στην έκφραση **x y+**.

<u>Παρατηρήσεις:</u>

1. Οι τελεστές είναι προτιμότερο να μη χρησιμοποιούνται σε μεικτούς τύπους. Για παράδειγμα, η έκφραση

out_int = my1_int + my2_int

δεν παρουσιάζει κανένα πρόβλημα, σε αντιδιαστολή με τον ακόλουθο μεικτό τύπο

out_float = my_double / my_int

2. Στη C υπάρχει διάκριση ανάμεσα στη διαίρεση ακεραίων και στη διαίρεση αριθμών κινητής υποδιαστολής. <u>Στη διαίρεση ακεραίων το αποτέλεσμα είναι το</u>

πηλίκο, π.χ. **5/2=2**. Για να ληφθεί ως αποτέλεσμα αριθμός κινητής υποδιαστολής, τουλάχιστον ένας από τους τελεστέους πρέπει να είναι αριθμός κινητής υποδιαστολής: **5.0/2 υπολογίζεται ως 2.5**

3.2 Κατηγορίες εκφράσεων της C - προτεραιότητα και προσεταιριστικότητα

Οι εκφράσεις τής C μπορούν να καταταγούν στις παρακάτω κατηγορίες:

- ***Σταθερές εκφράσεις***. Είναι εκφράσεις που περιέχουν μόνο σταθερές τιμές.
- ***Ακέραιες εκφράσεις και εκφράσεις κινητής υποδιαστολής***. Είναι εκφράσεις, οι οποίες μετά από όλες τις άμεσες και έμμεσες μετατροπές τύπων δίνουν αποτέλεσμα ακέραιου τύπου ή τύπου κινητής υποδιαστολής, αντίστοιχα.
- ***Εκφράσεις δείκτη***. Είναι εκφράσεις με τιμή μία διεύθυνση. Περιλαμβάνουν μεταβλητές δείκτη, τον τελεστή διεύθυνσης (**&**), αλφαριθμητικές σταθερές και ονόματα πινάκων.

Ο υπολογισμός μίας έκφρασης δεν είναι πάντοτε απλή υπόθεση, ιδιαίτερα στην περίπτωση που υπάρχουν ένθετες (nested) εκφράσεις, δηλαδή εκφράσεις που είναι φωλιασμένες μέσα σε άλλες. Στην έκφραση

(((n+5)<=a) && q)

η έκφραση **n+5** είναι φωλιασμένη στην έκφραση **(n+5)<=a**, η οποία με τη σειρά της είναι φωλιασμένη στη συνολική έκφραση. Μία άλλη περίπτωση δυσχέρειας στον υπολογισμό είναι η διαδοχική παράθεση τελεστών: **7*8-2**, η οποία μπορεί να υπολογισθεί είτε ως **(7*8)-2=54** είτε ως **7*(8-2)=42**, οδηγώντας σε διαφορετικά αποτελέσματα.

Για να αντιμετωπισθούν οι ανωτέρω δυσχέρειες έχει υιοθετηθεί μία σειρά εφαρμογής των τελεστών, η επονομαζόμενη **εφαρμοστική σειρά** (applicative order), η οποία στηρίζεται στις έννοιες της **προτεραιότητας** (precedence) και της **προσεταιριστικότητας** (associativity) των τελεστών.

Οι τελεστές ταξινομούνται σε επίπεδα προτεραιότητας, με τη σύμβαση ότι οι τελεστές υψηλότερου επιπέδου προτεραιότητας δρούν επί των τελεστέων πριν από τους τελεστές χαμηλότερου επιπέδου.

Η ύπαρξη περισσότερων τελεστών στο ίδιο επίπεδο προτεραιότητας επιβάλλει τον προσδιορισμό της **κατεύθυνσης εφαρμογής**, με την κατεύθυνση από τα αριστερά

προς τα δεξιά να είναι ευρύτερα χρησιμοποιούμενη. Ένας τελεστής καλείται **αριστερής προσεταιριστικότητας** (left associative), όταν σε εκφράσεις που περιέχουν πολλά στιγμιότυπα του τελεστή η ομαδοποίηση γίνεται από τα αριστερά προς τα δεξιά. Έτσι, η έκφραση **10-8-2** υπολογίζεται ως **(10-8)-2**. Οι τελεστές +,-,*,/ είναι όλοι αριστερής προσεταιριστικότητας.

Για τη C παράδειγμα **δεξιάς προσεταιριστικότητας** αποτελεί η ύψωση σε δύναμη και ο τελεστής ανάθεσης (=). Στην έκφραση **num1=num2=10** εφαρμόζεται πρώτα ο δεξιός τελεστής ανάθεσης, με αποτέλεσμα η **num2** να αποκτήσει την τιμή 10, και ακολούθως εφαρμόζεται ο αριστερός τελεστής ανάθεσης, έτσι ώστε η **num1** εξισώνεται με τη **num2** και αποκτά την τιμή **10**.

Η προτεραιότητα και το είδος προσεταιριστικότητας των τελεστών παρατίθενται στον πίνακα 3-3, όπου οι τελεστές έχουν τεθεί με σειρά φθίνουσας προτεραιότητας:

Τελεστές	**Είδος προσεταιριστικότητας**
() [] ->	από αριστερά προς τα δεξιά
! ˜ ++ - - + - * & (τύπος) sizeof	από δεξιά προς τα αριστερά
*** / % *(αριθμητικοί τελεστές)***	από αριστερά προς τα δεξιά
+ - *(αριθμητικοί τελεστές)*	»
<< >>	»
< <= > >=	»
= = ! =	»
&	»
^	»
|	»
&&	»
||	»
?:	από δεξιά προς τα αριστερά
= += -= *= %= &= ^= |= <<= >>=	»
,	από αριστερά προς τα δεξιά

Πίνακας 3-3. *Προτεραιότητα και προσεταιριστικότητα τελεστών*

Με βάση τα προαναφερθέντα, είναι προφανές ότι με τους κανόνες προτεραιότητας και προσεταιριστικότητας δεν είναι απαραίτητη η χρήση

παρενθέσεων για τον προσδιορισμό του τρόπου υπολογισμού της τιμής των εκφράσεων. Ωστόσο, οι παρενθέσεις χρησιμοποιούνται για τους ακόλουθους λόγους:

- Για να προσδιορισθεί συγκεκριμένη σειρά εφαρμογής, όπως στην έκφραση **(2-3)*4**.
- Για να καταστεί μία έκφραση ευανάγνωστη, όπως στην έκφραση **2-(3*4)**, παρά το γεγονός ότι στην τελευταία περίπτωση αποτελεί πλεονασμό.

Στην περίπτωση ένθετων παρενθέσεων ο μεταγλωττιστής εφαρμόζει πρώτα τις εσωτερικές παρενθέσεις. Για παρενθέσεις όμως που βρίσκονται στο ίδιο βάθος ένθεσης, δεν ορίζεται η σειρά υπολογισμού.

Παράδειγμα 3.1

Με χρήση του πίνακα 3-3 να υπολογισθούν οι εκφράσεις:

α) **x = 17 - 2 * 8**

β) **y = 17 - 2 - 8**

α) **x =17-(2*8) , x=1**

β) **y = (17-2)-8, y=7**

3.3 Τελεστές αύξησης και μείωσης

Ο τελεστής αύξησης (increment operator) συμβολίζεται ++. Με χρήση αυτού του τελεστή η έκφραση **num = num + 1;** γίνεται **num ++;**

Αντίστοιχα, ο τελεστής μείωσης (decrement operator) συμβολίζεται **--** και η έκφραση **num = num - 1;** γίνεται **num --;**

Παράδειγμα 3.2

Προπορευόμενοι και παρελκόμενοι τελεστές μοναδιαίας αύξησης και μείωσης: να υπολογισθούν οι τιμές των **x** και **y** στις ακόλουθες διαδοχικές εκφράσεις.

Πρόταση	τιμή x	Τιμή y
int x = 10, y = 20;	**10**	**20**
++ x;	**11**	**20**
y = --x;	**10**	**10**
y = x-- + y;	**9**	**20**
y = y – x++;	**10**	**11**

Παράδειγμα 3.3

Να προσδιορισθεί η τιμή των **x** και **z** μετά την εκτέλεση κάθε μίας από τις παρακάτω προτάσεις, θεωρώντας ότι πριν την εκτέλεση της κάθε πρότασης οι τιμές των **x** και **y** είναι το **10** και το **20** αντίστοιχα.

α) **z = ++x + y;**

β) **z = --x + y;**

γ) **z = x++ + y;**

δ) **z = x-- + y;**

Στην περίπτωση του προπορευόμενου τελεστή, το σύστημα πρώτα εκτελεί την αύξηση ή μείωση και μετά χρησιμοποιεί τη νέα τιμή της μεταβλητής στον υπολογισμό της τιμής της έκφρασης (προτάσεις *α* και *β*). Αντίθετα, στην περίπτωση του παρελκόμενου τελεστή το σύστημα πρώτα χρησιμοποιεί την τιμή της μεταβλητής για τον υπολογισμό της τιμής της έκφρασης και μετά εκτελεί την αύξηση ή μείωση της τιμής της μεταβλητής (προτάσεις *γ* και *δ*).

Πρόταση	τιμή x	τιμή z
Z = ++x + y;	**11**	**31**
Z = --x + y;	**9**	**29**
Z = x++ + y;	**11**	**30**
Z = x-- + y;	**9**	**30**

3.4 Τελεστές ανάθεσης

Οι τελεστές ανάθεσης (assignment operators) εκτελούν κάποια πράξη ανάμεσα στους τελεστέους και εκχωρούν το αποτέλεσμα σε έναν από τους τελεστέους:

- **x* = 10;** εκτελεί την πράξη του πολλαπλασιασμού μεταξύ των **x** και **10** και εκχωρεί το αποτέλεσμα στον τελεστέο **x**. Αντιστοιχεί στην πρόταση **x = x * 10;**
- **x* = y + 1;** Αντιστοιχεί στην πρόταση **x = x * (y + 1);** κι <u>όχι</u> στην πρόταση **x = x * y + 1;**

Τελεστές ανάθεσης δημιουργούν κι οι τελεστές διαχείρισης δυαδικών ψηφίων (bitwise operators). Οι τελεστές αυτοί είναι: **>>=** **<<=** **&=** **^=** **|=** .

Οι τελεστές ανάθεσης μαζί με τους τελεστές αύξησης/μείωσης γίνονται αιτία δημιουργίας παρενεργειών (side effects), για το λόγο αυτό αναφέρονται και ως **παρενεργοί τελεστές** (side effect operators). Οι παρενέργειες αυτές έχουν ως αποτέλεσμα την απροσδιόριστη συμπεριφορά του συστήματος ως προς τον τρόπο υπολογισμού της τιμής της μεταβλητής **i** σε εκφράσεις όπως: **i = n[i++];** ή **i = ++i + 1;**

3.5 Συσχετιστικοί τελεστές

Οι συσχετιστικοί τελεστές (relational operators) συγκρίνουν δύο τελεστέους. Οι βασικοί τελεστές της κατηγορίας αυτής παρατίθενται στον πίνακα 3-4:

Τελεστής	Δράση
<	μικρότερο από
>	μεγαλύτερο από
<=	μικρότερο ή ίσον από
>=	μεγαλύτερο ή ίσον από
==	Ίσο
!=	Διάφορο

Πίνακας 3-4. *Συσχετιστικοί τελεστές*

Το αποτέλεσμα της χρήσης των συσχετιστικών τελεστών είναι είτε **ΑΛΗΘΕΣ** (true) είτε **ΨΕΥΔΕΣ** (false). Για παράδειγμα, η τιμή της έκφρασης **(3 < 2)** είναι

ψευδής ενώ η τιμή της έκφρασης **(2 = = 2)** είναι αληθής.

Στη C (και σε πολλές άλλες γλώσσες προγραμματισμού) η τιμή ΑΛΗΘΗΣ αντιστοιχεί στον ακέραιο **1** και η τιμή ΨΕΥΔΗΣ αντιστοιχεί στον ακέραιο **0**.

Παρατήρηση: Συγκρίνοντας τους αριθμητικούς με τους συσχετιστικούς τελεστές προκύπτει ότι και οι δύο χρησιμοποιούν αριθμητικές εισόδους, π.χ. **(num + 10)** και **(num < 10)**, όπου **num** είναι μία ακέραια μεταβλητή. Ωστόσο οι αριθμητικοί τελεστές μπορούν να δώσουν ως έξοδο οποιοδήποτε αριθμό (για κάθε τιμή του **num** η πρόταση **(num + 10)** δίνει μία άλλη τιμή) ενώ οι συσχετιστικές τελεστές έχουν δίτιμη έξοδο (για κάθε τιμή του **num** μικρότερη του 10 η πρόταση **(num < 10)** δίνει TRUE (1) και για όλες τις άλλες τιμές του num δίνει FALSE (0)).

3.6 Λογικοί τελεστές

Οι λογικοί τελεστές δρουν επί ενός ή δύο τελεστέων και λειτουργούν με βάση τη δίτιμη άλγεβρα Boole. Τόσο οι είσοδοι όσο και οι έξοδοι μπορούν να λάβουν μόνο δύο τιμές, TRUE και FALSE. Οι τελεστές της κατηγορίας αυτής παρατίθενται στον πίνακα 3-5 ενώ στον πίνακα 3-6 περιγράφεται ο τρόπος λειτουργίας τους (πίνακας αληθείας):

Τελεστής	Δράση
&&	Λογικό AND
||	Λογικό OR
!	Λογικό NOT

Πίνακας 3-5. *Λογικοί τελεστές*

p	q	p&&q	p||q	!p
T	**T**	**T**	**T**	**F**
T	**F**	**F**	**T**	
F	**T**	**F**	**T**	**T**
F	**F**	**F**	**F**	

Πίνακας 3-6. *Πίνακας αληθείας*

Παράδειγμα 3.4

Για x = 10 και y = -8 να υπολογισθούν οι εκφράσεις:

α) **(x+5) < (12-y)**

β) **(x>5) || (y>10)**

Αντιακθιστώντας τις αριθμητικές τιμές προκύπτει:

α) **(10+5) < (12- (-8)) → 15< 20 → TRUE**

β) **(10>5) || (-8>10) → (TRUE) || (FALSE) → TRUE**

3.7 Μετατροπές τύπων

Όταν ένας τελεστής έχει τελεστέους διαφορετικών τύπων δεδομένων, αυτοί μετατρέπονται σε ενιαίο τύπο. Η μετατροπή είτε γίνεται αυτόματα από τον υπολογιστή, οπότε καλείται **υπονοούμενη** (implicit conversion), είτε άμεσα από τον προγραμματιστή, οπότε καλείται **ρητή** (explicit conversion).

3.7.1 Υπονοούμενες μετατροπές

Οι υπονοούμενες μετατροπές διευκολύνουν την εργασία του προγραμματιστή, ο οποίος όμως θα πρέπει σε κάθε περίπτωση να γνωρίζει τις συνέπειες μίας μετατροπής. Για παράδειγμα, η έκφραση

3.0 + 1/2

δε δίνει τιμή **3.5**, όπως πιθανόν να ήταν αναμενόμενο, αλλά **3.0**.

Η διαδικασία της αυτόματης μετατροπής στηρίζεται στους ακόλουθους κανόνες:

- Σε κάθε πράξη που υπάρχουν δύο τύποι δεδομένων, ο στενότερος τύπος μετατρέπεται στον ευρύτερο χωρίς να υπάρχει απώλεια πληροφορίας.
- Οι τύποι της γλώσσας ταξινομούνται ανάλογα με το μέγεθος της μνήμης που απαιτούν για αποθήκευση, όπως παρακάτω

char < int < long < float < double

Ο τύπος **unsigned** ακολουθεί τον αντίστοιχο προσημασμένο τύπο.

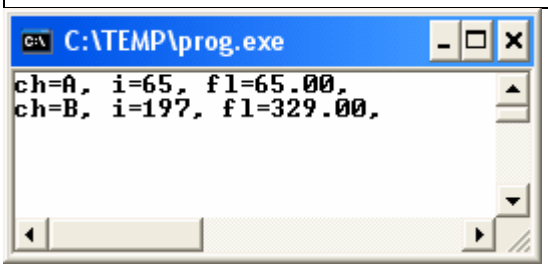
- Όλοι οι μεταγλωττιστές της C όταν υπολογίζουν αριθμητικές εκφράσεις

μετατρέπουν αυτόματα τον τύπο **char** σε **int** και το **float** σε **double**.

Παράδειγμα 3.5

Να μελετηθεί το ακόλουθο πρόγραμμα:

```
#include <stdio.h>
void main()
{
    char ch;
    int i;
    float fl;
    fl=i=ch='A';  // (1)
    printf( "ch=%c, i=%d, fl=%2.2f,\n",ch,i,fl );
    ch=ch+1;  // (2)
    i=fl+2*ch;  // (3)
    fl=2.0*ch+i;  // (4)
    printf( "ch=%c, i=%d, fl=%2.2f,\n",ch,i,fl );
}
```



(1): Ο χαρακτήρας **'A'** αποθηκεύεται ως χαρακτήρας στη μεταβλητή **ch**. Η μεταβλητή **i** λαμβάνει την τιμή του ακέραιου από τη μετατροπή του **'A'** (**65**), ενώ η μεταβλητή **fl** λαμβάνει την τιμή του αριθμού κινητής υποδιαστολής που προέρχεται από το **65** (**65.00**).

(2): Η πρόσθεση της μονάδας γίνεται στην ακέραια τιμή του **'A'**. Το αποτέλεσμα (**66**) αντιστοιχεί στο χαρακτήρα **'B'**, ο οποίος αποθηκεύεται στη μεταβλητή **ch**.

(3): Η πράξη δίνει **2*66+65.00=197.00**. Το αποτέλεσμα μετατρέπεται σε **int** (**197**) και αποθηκεύεται στη μεταβλητή **i**.

(4): Η πράξη δίνει **2.0*66+197=329.00** (οι αριθμοί int μετατρέπονται σε **float**). Το αποτέλεσμα αποθηκεύεται στη μεταβλητή **fl**.

3.7.2 Ρητές μετατροπές - τελεστής typecast

Εκτός από τις αυτόματες μετατροπές η C επιτρέπει ρητές μετατροπές μίας τιμής σε ένα διαφορετικό τύπο δεδομένων. Η διαδικασία ονομάζεται **προσαρμογή** ή **εκμαγείο** (casting) και ο τελεστής μετατροπής τύπου ή ***cast*** τελεστής, όπως αποκαλείται, είναι μοναδιαίος κι έχει τη μορφή ***(τύπος δεδομένων)***, π.χ. ***(float)***. Τοποθετείται μπροστά από μία έκφραση για να μετατρέψει την τιμή της στον περικλειόμενο σε παρενθέσεις τύπο:

```
int i,j;
float f1,f2,f3;
i=5; j=2;
fl = i/j + 0.5;                /* Αποτέλεσμα: 2.5 */
f2 = (float)i/(float)j + 0.5;  /* Αποτέλεσμα: 3.0 */
f3 = i/j + 0.5;                /* Αποτέλεσμα: 2.5 */
```

3.8 Τελεστής sizeof

Ο τελεστής ***sizeof*** είναι μοναδιαίος και δρα σε δύο τύπους δεδομένων:

α) σε έκφραση, π.χ. **sizeof(x+y)**

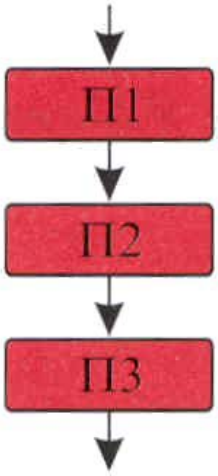
β) σε τύπο δεδομένων, πχ. **sizeof(int)**

Σε κάθε περίπτωση επιστρέφει τον αριθμό των bytes που η τιμή τής έκφρασης ή ο τύπος των δεδομένων καταλαμβάνει στη μνήμη. Προσοχή θα πρέπει να δοθεί στο γεγονός ότι το σύστημα δεν υπολογίζει την τιμή της έκφρασης κι έτσι πιθανή ύπαρξη παρενεργειών από τους τελεστές δε δημιουργεί παρενέργειες στη λειτουργία της ***sizeof***.

4. Έλεγχος ροής - προτάσεις υπό συνθήκη διακλάδωσης

4.1 Έλεγχος ροής

Τα προγράμματα αποτελούνται από προτάσεις, οι οποίες εκτελούνται με κάποια σειρά. Ο πιο συνηθισμένος τρόπος εκτέλεσης είναι ο **ακολουθιακός**: δύο ή περισσότερες προτάσεις βρίσκονται διατεταγμένες η μία μετά την άλλη και εκτελούνται διαδοχικά, όπως φαίνεται στο σχήμα 4.1.



Σχήμα 4-1. *Ακολουθιακή εκτέλεση προτάσεων*

Ωστόσο ορισμένες φορές επιβάλλεται να γίνουν λογικές επιλογές (με χρήση λογικών τελεστών και τελεστών συσχέτισης). Εάν π.χ. περιγραφόταν η σωστή συμπεριφορά ενός πεζού μπροστά σε ένα φωτεινό σηματοδότη, θα προέκυπτε η ακόλουθη πρόταση:

ΕΑΝ στο σηματοδότη βρίσκεται ο ΓΡΗΓΟΡΗΣ
ΤΟΤΕ μπορείς να διασχίσεις την οδό
ΑΛΛΙΩΣ περίμενε αλλαγή του σηματοδότη

Για να επιτευχθεί οποιαδήποτε διαφοροποίηση από την ακολουθιακή εκτέλεση απαιτούνται ειδικές κατασκευές. Ορισμένες από αυτές τις κατασκευές διασφαλίζουν ταυτόχρονα τη δόμηση του προγράμματος, με κύριο στόχο: η δομή του πηγαίου κώδικα να μας βοηθά να κατανοήσουμε τι κάνει το πρόγραμμα. Οι κατασκευές

διακρίνονται σε δύο βασικές κατηγορίες:

1) την ***υπό συνθήκη διακλάδωση*** (conditional branching)
2) την ***επανάληψη*** (looping)

Η πρώτη κατηγορία θα αποτελέσει αντικείμενο του παρόντος κεφαλαίου ενώ η δεύτερη κατηγορία θα μελετηθεί στο επόμενο κεφάλαιο.

4.2 Επιλεκτική εκτέλεση προτάσεων

Στις γλώσσες προγραμματισμού μία πρόταση διακλάδωσης περιέχει έναν αριθμό υποπροτάσεων, από τις οποίες επιλέγεται για εκτέλεση μόνο μία. Η πρόταση ***if***, που συναντάται σε πολλές γλώσσες προγραμματισμού, είναι η πλέον γνωστή πρόταση αυτής της κατηγορίας κι έχει την παρακάτω μορφή:

if E then Π1 else Π2

Στο σχήμα 4-2α αναπαριστάται η παραπάνω πρόταση. Είναι προφανές ότι είναι πρόταση μίας εισόδου – μίας εξόδου. Ο έλεγχος του προγράμματος εισέρχεται από το σημείο στην κορυφή, υπολογίζεται η τιμή της λογικής πρότασης **E** και, έαν είναι αληθής, επιλέγεται για εκτέλεση η πρόταση **Π1**, διαφορετικά η **Π2**. Σε κάθε περίπτωση, ο έλεγχος μεταφέρεται στο ένα και μοναδικό σημείο εξόδου στο κάτω μέρος του διαγράμματος.

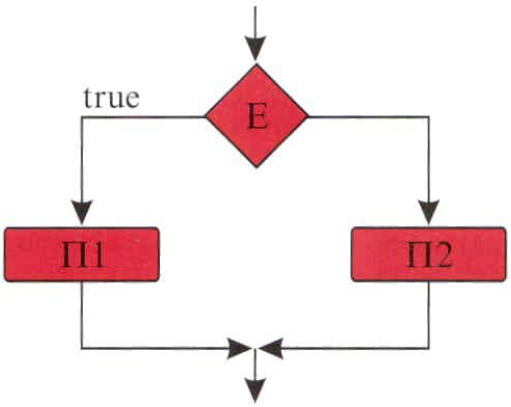
Στο σχήμα 4-2β παρουσιάζεται η γενίκευση της προηγούμενης περίπτωσης, η επιλεκτική εκτέλεση πολλών προτάσεων, όπου μάλιστα υπάρχει ένθεση, δηλαδή υπάρχει διακλάδωση μέσα σε διακλάδωση. Ο φορμαλισμός για την περίπτωση αυτή είναι ο ακόλουθος:

if E1 then Π1
else if E2 then Π2
else Π0

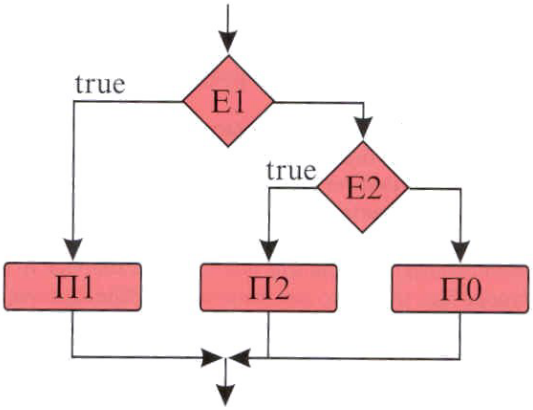
Οι λογικές εκφράσεις υπολογίζονται σειριακά και η πρώτη που θα δώσει αληθή τιμή οδηγεί στην εκτέλεση της αντίστοιχης πρότασης. Εάν καμία από τις λογικές εκφράσεις δε δώσει αληθή τιμή, εκτελείται η **Π0**.

Η πρόταση ***if*** βασίζει την επιλογή της σε λογική έκφραση. Υπάρχουν κατασκευές στις οποίες η απόφαση επιλογής και εκτέλεσης πρότασης βασίζεται σε άλλου τύπου εκφράσεις, όπως συμβαίνει με τη ***switch*** της C, στην οποία η επιλογή γίνεται μέσα από ένα σύνολο αμοιβαία αποκλειόμενων επιλογών, όπως θα αναλυθεί

σε επόμενη παράγραφο.



(α) Δύο προτάσεις



(β) Πολλές προτάσεις με ένθεση

Σχήμα 4-2. *Επιλεκτική εκτέλεση προτάσεων*

Παράδειγμα 4.1

Να περιγραφεί με ψευδοκώδικα η διεργασία που πρέπει να ακολουθήσει ο υπολογιστής για να διαπιστώσει κατά πόσο ένα δεδομένο έτος είναι δίσεκτο ή όχι. Να χρησιμοποιηθεί η κατασκευή **if – else**.

Εάν αναπαρασταθεί το έτος με την ακέραια μεταβλητή **year** και ο τελεστής υπολοίπου (modulo) με το σύμβολο **%**, η περιγραφή μπορεί να γίνει ως ακολούθως:

IF ((year % 400) == 0) THEN το έτος είναι δίσεκτο
ELSE IF ((year % 100) == 0) THEN το έτος δεν είναι δίσεκτο
ELSE IF ((year % 4) == 0) THEN το έτος είναι δίσεκτο

ELSE το έτος δεν είναι δίσεκτο

4.3 Υπό συνθήκη διακλάδωση if - else

Στη C η υπό συνθήκη διακλάδωση ***if*** έχει στη γενική περίπτωση την ακόλουθη σύνταξη:

```
if (συνθήκη)
{
    προτάσεις;
}
else
{
    προτάσεις;
}
```

Η ***if*** αποτελείται από τρία τμήματα:

- **Το τμήμα της συνθήκης**, που ακολουθεί τη λέξη ***if***.
- **Το αληθές τμήμα**, που ακολουθεί τη λέξη ***if*** και εκτελείται όταν η συνθήκη είναι αληθής.
- **Το ψευδές τμήμα** – εφόσον υπάρχει – που ακολουθεί τη λέξη ***else*** και εκτελείται όταν η συνθήκη είναι ψευδής.

Όταν τα ***if***, ***else*** ακολουθεί μία μόνο πρόταση, τα άγκιστρα περιττεύουν, ωστόσο είναι ορθή προγραμματιστική τακτική να τοποθετούνται πάντοτε, αφενός μεν για να καταστήσουν τον κώδικα ευανάγνωστο αφετέρου δε για να αποτρέψουν λάθη σε περίπτωση που προστεθούν κι άλλες προτάσεις στο αληθές ή το ψευδές τμήμα.

<u>Παρατηρήσεις:</u>

1. Μερικές φορές δεν υπάρχει ***else***, δηλαδή δεν υπάρχει ψευδές τμήμα:

```
if (gas_tank_empty == TRUE) fill_up_tank();
```

Εάν η συνθήκη είναι ψευδής (π.χ. το ντεπόζιτο της βενζίνης είναι άδειο) δε γίνεται καμία ενέργεια.

2. Όταν υπάρχουν **περισσότερα** από δύο τμήματα και απαιτούνται ένθετες προτάσεις **if/else**, το ζεύγος

else { if (*συνθήκη*) { *προτάσεις;* } }

μπορεί να ατικατασταθεί με την περισσότερο ευανάγνωστη μορφή:

else if (*συνθήκη*) { *προτάσεις;* }

Η ανωτέρω μορφή ονομάζεται **κλίμακα *if – else – if***. Στη γενική περίπτωση έχει την ακόλουθη σύνταξη:

```
if (συνθήκη)
{
    προτάσεις;
}
else if (συνθήκη)
{
    προτάσεις;
}
. . . . . .
else if (συνθήκη)
{
    προτάσεις;
}
else
{
    προτάσεις;
}
```

Παράδειγμα 4.2

Να ελεγχθεί κατά πόσον τα ακόλουθα τμήματα κώδικα είναι λειτουργικά ισοδύναμα.

```
int maria, petros;                    int maria, petros;
maria = 15;                           maria = 15;
petros = maria +3;                    petros = maria +3;
if ( maria < 2 )     maria *=2;       if ( maria < 20 )   maria *=2;
else petros = 1700;                   if ( maria >= 20 ) petros = 1700;
printf( "petros = %d\n",petros );     printf( "petros = %d\n",petros );
```

Οι πρώτες πέντε γραμμές των δύο τμημάτων κώδικα είναι ίδιες, στο τέλος των οποίων η μεταβλητές **maria** και **petros** έχουν λάβει τις τιμές **30** και **18**, αντίστοιχα. Στην έκτη γραμμή υπάρχει διαφοροποίηση: στο πρώτο τμήμα κώδικα το ***else*** δε θα εκτελεσθεί, καθώς η συνθήκη του ***if*** ήταν αληθής, και η μεταβλητή **petros** θα διατηρήσει την τιμή της (**18**). Στο δεξί τμήμα κώδικα όμως, η διακλάδωση ***if*** είναι καινούρια, η συνθήκη είναι αληθής (**maria=30>20**) και η μεταβλητή **petros** θα λάβει νέα τιμή (**1700**). Κατά συνέπεια τα δύο τμήματα κώδικα δεν είναι λειτουργικά ισοδύναμα.

4.4 Ο υποθετικός τελεστής

Ο υποθετικός τελεστής (**?:**) αποτελείται από δύο σύμβολα. Ανήκει στην κατηγορία των τελεστών που αποτελούνται από συνδυασμό συμβόλων και δεν ακολουθούν καμία από τις postfix, prefix ή infix σημειογραφίες. Όταν τα σύμβολα ή οι λέξεις του τελεστή είναι διάσπαρτα στους τελεστέους στους οποίους εφαρμόζεται ο τελεστής, λέμε ότι ο τελεστής είναι σε **μεικτή σημειογραφία** (mixfix notation).

Η έκφραση που σχηματίζει ο υποθετικός τελεστής έχει τη μορφή:

εκφρ1 ? εκφρ2 : εκφρ3

Ουσιαστικά ο υποθετικός τελεστής υλοποιεί μία υποθετική πρόταση. Η τιμή της παραπάνω έκφρασης είναι η τιμή της **εκφρ2**, εάν η **εκφρ1** είναι αληθής, αλλιώς είναι η τιμή της **εκφρ3**.

Η **εκφρ1** αποτελεί τη συνθήκη ελέγχου. Έτσι η έκφραση

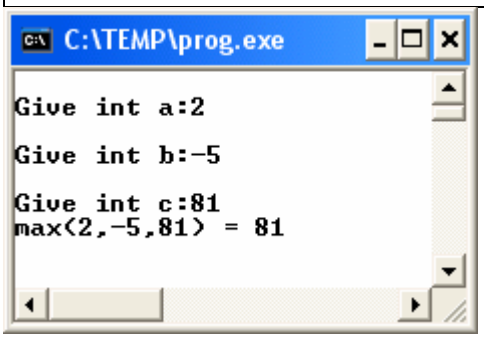
x>z ? x : z

Έχει τιμή **x**, εάν το **x>z** είναι αληθές, διαφορετικά έχει τιμή **z**.

Παράδειγμα 4.3

Να γραφεί πρόγραμμα που υπολογίζει το μέγιστο ανάμεσα σε τρεις ακέραιους. Ακολούθως να τροποποιηθεί ο κορμός του προγράμματος κάνοντας χρήση του υποθετικού τελεστή.

```
#include <stdio.h>
void main()
{
    int a,b,c;
    printf( "\nGive int a:" );     scanf("%d",&a);
    printf( "\nGive int b:" );     scanf("%d",&b);
    printf( "\nGive int c:" );     scanf("%d",&c);
    if (a>b)
    {
        if (a>c) printf( "max(%d,%d,%d) = %d\n",a,b,c,a );
        else printf( "max(%d,%d,%d) = %d\n",a,b,c,c );
    }
        else if (b>c) printf( "max(%d,%d,%d) = %d\n",a,b,c,b );
        else printf( "max(%d,%d,%d) = %d\n",a,b,c,c );
}
```



Χρησιμοποιώντας τον υποθετικό τελεστή, οι προτάσεις διακλάδωσης συνοψίζονται στην ακόλουθη πρόταση:

```
max=(a>b?a:b) > c ? (a>b?a:b):c;
printf( "max(%d,%d,%d) = %d\n",a,b,c, max );
```

4.5 Υπό συνθήκη διακλάδωση switch

Αν και η κλίμακα ***if – else – if*** μπορεί να πραγματοποιεί ελέγχους διαφόρων ειδών, είναι δύσχρηστη καθώς δεν παρέχει εποπτεία στον προγραμματιστή και καθυστερεί στην εκτέλεση. Για τις περιπτώσεις πολλαπλής διακλάδωσης η C διαθέτει την πολυκλαδική εντολή ***switch***, η οποία έχει την ακόλουθη σύνταξη:

```
switch(έκφραση)
{
    case (σταθ.-έκφρ. 1):
        προτάσεις;
    break;
    case (σταθ.-έκφρ. 2):
        προτάσεις;
    break;
    . . . . . .
    case (σταθ.-έκφρ. N):
        προτάσεις;
    break;
    default:
        προτάσεις;
    break;
}
```

Η πρόταση ***switch*** επιτρέπει τον προσδιορισμό απεριόριστου αριθμού διαδρομών, ανάλογα με την τιμή της έκφρασης. Υπολογίζεται η έκφραση και η τιμή της συγκρίνεται διαδοχικά με τις σταθερές εκφράσεις (***σταθ.-έκφρ. 1, σταθ.-έκφρ. 2,***

…). Ο έλεγχος μεταφέρεται στις προτάσεις που είναι κάτω από τη ***σταθ.-έκφρ.*** με την οποία ισούται η τιμή της ***έκφρασης***. Εάν δεν ισούται με καμία από τις σταθερές εκφράσεις, ο έλεγχος μεταφέρεται στις προτάσεις που ακολουθούν την ετικέτα ***default***, εάν βέβαια αυτή υπάρχει, αλλιώς στην πρόταση που ακολουθεί το σώμα της ***switch***.

Η πρόταση ελέγχου ***break***, η οποία υποδηλώνει άμεση έξοδο από τη ***switch***, είναι προαιρετική. Εάν αυτή λείπει, μετά την εκτέλεση των προτάσεων που ακολουθούν την επιλεγείσα ετικέτα ***case*** θα ακολουθήσει η εκτέλεση των προτάσεων και των επόμενων ***case*** ετικετών. Βέβαια στην πράξη η ***break*** συναντάται σχεδόν πάντοτε, ακόμη και μετά τις προτάσεις της ετικέτας ***default***. Το τελευταίο γίνεται για να προστατευθούμε από το δύσκολο στην ανεύρεση σφάλμα που θα προκύψει εάν προστεθεί μελλονικά μία νέα ***case*** και ταυτόχρονα παραληφθεί να προστεθεί πριν από αυτή η ***break***.

Θα πρέπει να σημειωθεί ότι η ***switch*** διαφέρει από την ***if*** στο ότι ελέγχει μόνο την ισότητα, ενώ η παράσταση με συνθήκη της ***if*** μπορεί να είναι οιουδήποτε τύπου.

Η λειτουργία της ***switch*** διέπεται από το ακόλουθο σύνολο κανόνων:

- Κάθε ***case*** πρέπει να έχει μία ***int*** ή ***char*** σταθερά έκφραση.
- Δύο ***case*** δεν μπορούν να έχουν την ίδια τιμή.
- Οι προτάσεις κάτω από την ετικέτα ***default*** εκτελούνται όταν δεν ικανοποιείται καμία από τις ***case*** ετικέτες.
- Η ***default*** δεν είναι απαραίτητα η τελευταία ετικέτα.

Παράδειγμα 4.4

Να γραφεί πρόγραμμα, το οποίο να δίνει τη δυνατότητα στο χρήστη να εισάγει δύο αριθμούς και στη συνέχεια να εκτελεί επί αυτών επιλεκτικά μία από τις τέσσερις αριθμητικές πράξεις.

Χρησιμοποιώντας δομημένα Ελληνικά η διεργασία περιγράφεται ως εξής:

Πάρε δύο αριθμούς

Ενημέρωσε το χρήστη για δυνατές επιλογές

Πάρε την επιλογή του χρήστη

Ανάλογα με την επιλογή

Εκτέλεσε την αντίστοιχη πράξη

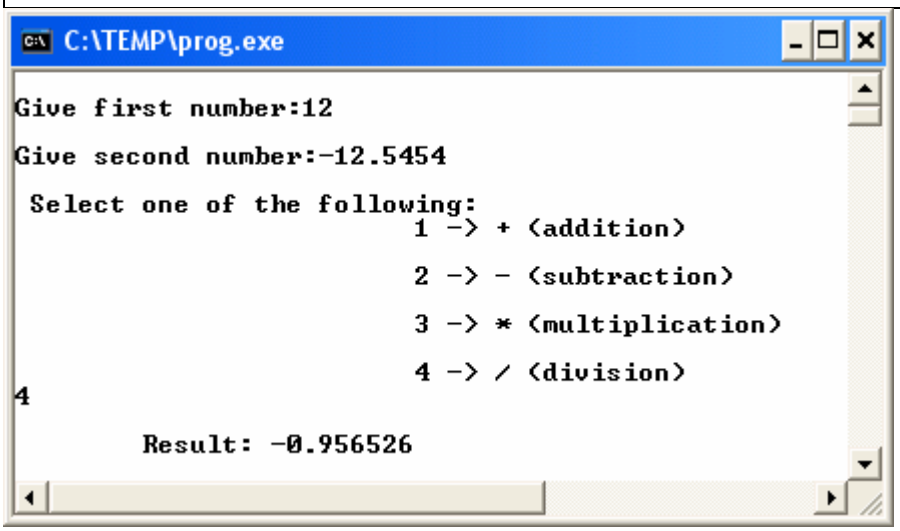
Εμφάνισε το αποτέλεσμα

Τερμάτισε

Ο κώδικας του προγράμματος είναι ο ακόλουθος:

```
#include <stdio.h>
#include <conio.h>
#define ADD 1
#define SUB 2
#define MUL 3
#define DIV 4
void main()  {
    float num1, num2,result;
    int choice;
    printf( "\nGive first number:"); scanf("%f",&num1 );
    printf( "\nGive second number:"); scanf("%f",&num2 );
    printf( "\n Select one of the following:" );
    printf( "\n\t\t\t 1 -> + (addition)\n" );
    printf( "\n\t\t\t 2 -> - (subtraction)\n" );
    printf( "\n\t\t\t 3 -> * (multiplication)\n" );
    printf( "\n\t\t\t 4 -> / (division)\n" );
    scanf( "%d",&choice );
    switch(choice)
    {
        case 1:
            result=num1+num2;
        break;
        case 2:
            result=num1-num2;
        break;
        case 3:
            result=num1*num2;
        break;
```

```
            case 4:
                if (num2) result=num1/num2; // num2 != 0
                else printf( "\t\t ERROR: division by 0" );
            break;
            default:
            printf( "This selection is not supported" );
            break;
        }  // Τέλος της switch
        printf( "\n\tResult: %f\n",result );
    }  // Τέλος της main
```

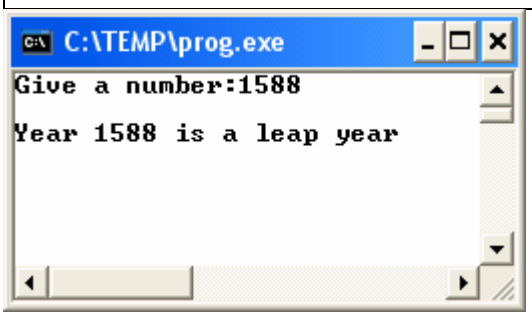


Παράδειγμα 4.5

Να γραφεί πρόγραμμα που να υλοποιεί το παράδειγμα 4.1.

```
    #include <stdio.h>
    void main()  {
        int year;
        printf( "Give a number:" );
        scanf( "%d",&year );
        if ( ((year%400)==0) || ( !((year%100)==0) && ((year%4)==0) ) )
```

```
                    printf( "\nYear %d is a leap year",year );
            else printf( "\nYear %d is not a leap year",year );
    }
```



5. Προτάσεις επανάληψης - βρόχοι

5.1 Γενικά

Οι προτάσεις επανάληψης αποτελούν ένα ισχυρό εργαλείο προγραμματισμού καθώς μπορούν να κωδικοποιήσουν και να συμπυκνώσουν επαναλαμβανόμενες λειτουργίες, δημιουργώντας ένα βρόχο (loop). Ορίζονται ως προτάσεις που **επαναλαμβάνουν ένα μπλοκ εντολών είτε για όσες φορές το επιθυμούμε είτε έως ότου πληρωθεί μία συνθήκη τερματισμού**. Η πλήρωση του κριτηρίου τερματισμού οδηγεί στην περάτωση του βρόχου.

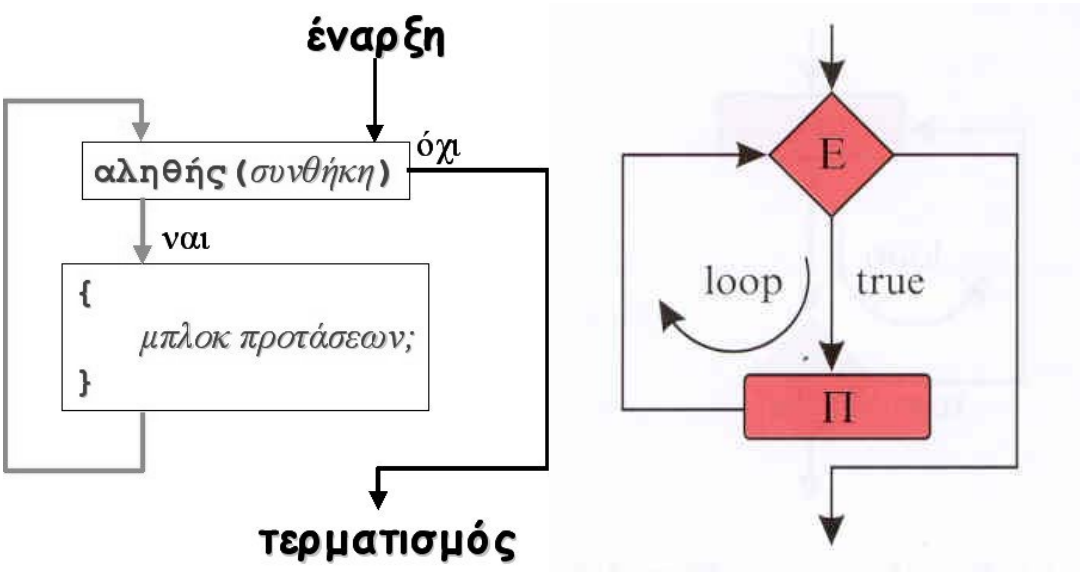
Εάν δεν υπάρχει συγκεκριμένος αριθμός επαναλήψεων ή συνθήκη τερματισμού, ο βρόχος θα εκτελείται αενάως και θα καλείται **ατέρμων βρόχος** (infinite loop), γεγονός που οδηγεί ως επί το πλείστον σε σφάλμα.

Εάν ο βρόχος τελειώνει μετά το πέρας ενός ορισμένου αριθμού επαναλήψεων τότε καλείται **βρόχος οδηγούμενος από μετρητή**. Εάν περατώνεται με την πλήρωση ενός κριτηρίου τερματισμού ονομάζεται **βρόχος οδηγούμενος από γεγονός**. Επιπρόσθετα, στις περισσότερες γλώσσες προγραμματισμού υπάρχει μία δεύτερη κατηγοριοποίηση των προτάσεων επανάληψης: *α)* εκείνες στις οποίες ο έλεγχος του κριτηρίου τερματισμού γίνεται στην αρχή του βρόχου, επονομαζόμενες **βρόχοι με συνθήκη εισόδου** (pre-test loops), και *β)* εκείνες στις οποίες ο έλεγχος του κριτηρίου τερματισμού γίνεται στο τέλος του βρόχου, επονομαζόμενες **βρόχοι με συνθήκη εξόδου** (post-test loops).

Στις παραγράφους που ακολουθούν πρώτα θα παρουσιασθούν οι μορφές βρόχων που κυριαρχούν στις γλώσσες προγραμματισμού και στη συνέχεια θα εστιασθούμε στις επαναληπτικές προτάσεις που χρησιμοποιούνται στη C.

5.1.1 Βρόχος while-do

Ο βρόχος ***while-do*** είναι βρόχος με συνθήκη εισόδου, δυνάμενος να οδηγείται τόσο από μετρητή όσο και από γεγονός. Η λειτουργία του απεικονίζεται στο μπλοκ διάγραμμα του σχήματος 5-1α ενώ το διάγραμμα ροής παρατίθεται στο σχήμα 5-1β.



(α) Μπλοκ διάγραμμα ***(β)*** Διάγραμμα ροής

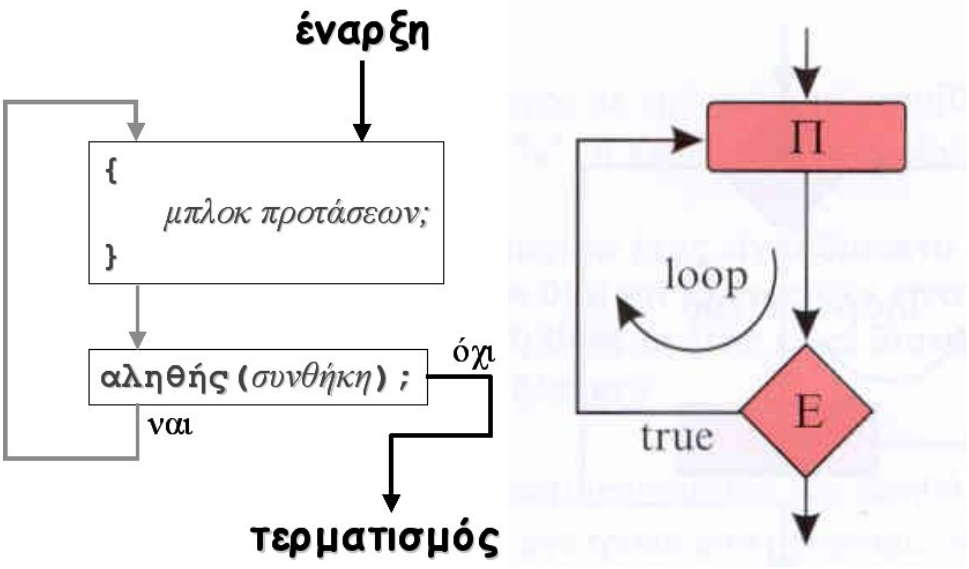
Σχήμα 5-1. *Βρόχος while-do*

Όπως προκύπτει από το σχήμα 5-1, ο βρόχος θα εκτελείται – δηλαδή η πρόταση Π – για όσες επαναλήψεις η συνθήκη Ε είναι αληθής. Κατά συνέπεια απαιτείται: *α)* πριν την πρώτη επανάληψη η συνθήκη Ε να είναι αληθής και *β)* κατά τη διάρκεια εκτέλεσης του βρόχου να υπάρχει η δυνατότητα, μέσω της Π, να μπορεί να καταστεί η Ε ψευδής για να τερματισθεί ο βρόχος.

5.1.2 Βρόχος do-while

Ο βρόχος ***do-while*** είναι βρόχος με συνθήκη εξόδου, δυνάμενος κι αυτός να οδηγείται τόσο από μετρητή όσο και από γεγονός. Στα σχήματα 5-2α και 5-2β παρουσιάζονται το μπλοκ διάγραμμα και το διάγραμμα ροής αντίστοιχα.

Από το σχήμα 5-2 καθίσταται φανερό ότι ο βρόχος ***do-while*** διαφέρει από το βρόχο ***while-do*** στο σημείο ελέγχου της συνθήκης τερματισμού. Ο βρόχος ***do-while*** επιτρέπει την εκτέλεση της πρώτης επανάληψης πριν προχωρήσει στον έλεγχο της συνθήκης, γεγονός που σημαίνει ότι δεν απαιτείται να είναι αληθής η συνθήκη πριν από την εκτέλεση του βρόχου. Μπορεί να γίνει αληθής μέσα στην Π και, φυσικά, πρέπει να υπάρχει η δυνατότητα, μέσω της Π, να μπορεί να καταστεί η Ε ψευδής για να τερματισθεί ο βρόχος.



(α) *Μπλοκ διάγραμμα* **(β)** *Διάγραμμα ροής*

Σχήμα 5-2. *Βρόχος do-while*

5.2 Βρόχοι με συνθήκη εισόδου στη C

5.2.1 Βρόχος while

Ο βρόχος ***while*** είναι βρόχος με συνθήκη εισόδου, οδηγούμενος από γεγονός. Η λειτουργία του περιγράφεται εποπτικά από το σχήμα 5-1α και η σύνταξή του είναι η ακόλουθη:

```
while (συνθήκη)
{
    προτάσεις, μέσα στις οποίες θα αλλάζει η συνθήκη;
}
```

Η λειτουργία της πρότασης επανάληψης ***while*** μπορεί να μορφοποιηθεί σε δομημένα Ελληνικά ως εξής:

Έλεγξε τη συνθήκη.
Εάν είναι αληθής
    Προχώρησε στις προτάσεις
    Ξεκίνησε από την αρχή
Αλλιώς σταμάτησε

Ο βρόχος ***while*** είναι κατάλληλος στις περιπτώσεις που δεν είναι γνωστός εκ των προτέρων ο αριθμός των επαναλήψεων. Εκτελείται καθόσον η συνθήκη παραμένει αληθής. Όταν η συνθήκη καταστεί ψευδής, ο έλεγχος του προγράμματος παρακάμπτει το περιεχόμενο του βρόχου και προχωρά στην επόμενη εντολή. Επιπρόσθετα, για το βρόχο while ισχύουν οι παρατηρήσεις της §5.1.1.

Θα πρέπει να σημειωθεί ότι εάν το σώμα του βρόχου αποτελείται από μία πρόταση, δεν απαιτούνται **{}**.

Παράδειγμα 5.1

Να περιγραφεί η λειτουργία του ακόλουθου τμήματος κώδικα και να δοθεί το διάγραμμα ροής της πρότασης επανάληψης.

```
int count=30;
int limit=40;
while (count<limit)
{
  count++;
  printf( "count is %d\n",count );
}
<επόμενη πρόταση>;
```

Στις πρώτες δύο γραμμές δηλώνονται και αρχικοποιούνται οι ακέραιες μεταβλητές **count** και **limit**. Ακολουθεί η πρόταση επανάληψης ***while***, η οποία καθορίζει ως συνθήκη η μεταβλητή **count** να είναι μικρότερη της **limit**, γεγονός που αληθεύει. Κατά συνέπεια ο έλεγχος εισέρχεται στο βρόχο, η μεταβλητή **count** αυξάνεται κατά μία μονάδα και τυπώνεται στην οθόνη η φράση

count is 31

Ο βρόχος εκτελείται συνολικά 10 φορές:

count is 31

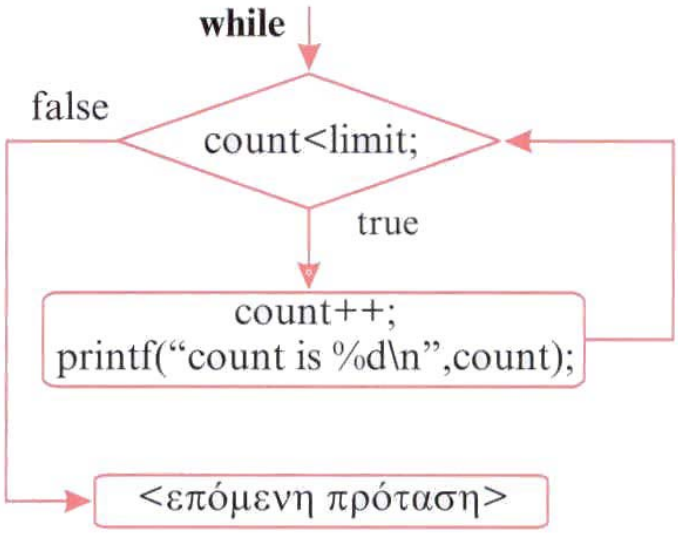
count is 32

. . . . . .

count is 40

Στην ενδέκατη επανάληψη η **count** έχει λάβει την τιμή 40 και είναι ίση με τη **limit**, οπότε η συνθήκη καθίσταται ψευδής και ο έλεγχος προσπερνά το βρόχο και προχωρά στην επόμενη πρόταση. Θα πρέπει να σημειωθεί ότι εάν η συνθήκη ήταν εξαρχής ψευδής (π.χ. **count=45**), ο βρόχος δε θα εκτελείτο ούτε μία φορά.

Το διάγραμμα ροής απεικονίζεται στο σχήμα 5-3:



Σχήμα 5-3. *Διάγραμμα ροής της πρότασης επανάληψης*

Παράδειγμα 5.2

Δίνονται οι παρακάτω δύο προτάσεις:

α) **while (++count<12) Π1**

β) **while (count++<12) Π1**

Να περιγραφεί ο τρόπος με τον οποίο ο υπολογιστής τις εκτελεί, εντοπίζοντας τη διαφορά τους, εάν υπάρχει.

Υπάρχει διαφορά μεταξύ των προτάσεων κι αυτή εντοπίζεται στον αριθμό επαναλήψεων. Η *(α)* χρησιμοποιεί την προθεματική σημειογραφία ενώ η *(β)* τη μεταθεματική. Στην πρόταση *(α)* πρώτα αυξάνεται η τιμή τής **count** και η νέα τιμή της συγκρίνεται με το **12**, ενώ στη *(β)* πρώτα συγκρίνεται η τιμή τής **count** με το **12** και στη συνέχεια αυξάνεται η τιμή της. Αυτό σημαίνει πως η πρόταση Π1 θα

εκτελεσθεί μία φορά παραπάνω στην περίπτωση *(β)*.

5.2.2 Βρόχος for

Ο βρόχος ***for*** είναι βρόχος με συνθήκη εισόδου, οδηγούμενος από μετρητή. Η λειτουργία του περιγράφεται εποπτικά από το σχήμα 5-1α και η σύνταξή του είναι η ακόλουθη:

```
for (αρχική τιμή μετρητή; συνθήκη; βήμα μετρητή)
{
    προτάσεις;
}
```

Η λειτουργία της πρότασης επανάληψης ***for*** μπορεί να μορφοποιηθεί σε δομημένα Ελληνικά ως εξής:

Αρχικοποίησε το μετρητή
Έλεγξε τη συνθήκη
Εάν είναι αληθής
    Εκτέλεσε τις προτάσεις
    Ενημέρωσε το μετρητή
    Επάνελθε στον έλεγχο της συνθήκης
Αλλιώς ενημέρωσε το μετρητή και σταμάτησε

Μία τυπική εκτέλεση του βρόχου ***for*** είναι η ακόλουθη, κατά την οποία σε κάθε επανάληψη θα τυπώνεται η μεταβλητή **n**:

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

Ωστόσο ο βρόχος ***for*** στη γλώσσα C παρέχει μεγάλη ευελιξία καθώς οι εκφράσεις μέσα στις παρενθέσεις μπορούν να έχουν πολλές παραλλαγές. Ενδεικτικά αναφέρονται μερικές από τις παραλλαγές ενώ για ενδελεχή μελέτη ο αναγνώστης μπορεί να ανατρέξει στην αναφορά [18].

- Μπορεί να χρησιμοποιηθεί ο τελεστής μείωσης για μέτρηση προς τα κάτω:

```
for (n=10; n>0; n- -) printf( "n=%d\n",n );
```

- Το βήμα καθορίζεται από το χρήστη:

```
for (n=0; n<60; n=n+13) printf( "n=%d\n",n );
```

- Χρησιμοποιώντας την ιδιότητα ότι κάθε χαρακτήρας του κώδικα ASCII έχει μία ακέραια τιμή, ο μετρητής μπορεί να είναι μεταβλητή χαρακτήρα. Το παρακάτω τμήμα κώδικα θα τυπώνει τους χαρακτήρες από το '**a**' έως το '**z**' μαζί με τον ASCII κωδικό τους:

  for (n='a'; n<'z'; n++) printf("n=%d, the ASCII value is %d\n",n,n);

- Ο μετρητής μπορεί να αυξάνει κατά γεωμετρική πρόοδο:

  for (n=0; n<60.0; n=1.2*n) printf("n=%f\n",n);

Θα πρέπει να σημειωθεί ότι εάν το σώμα του βρόχου αποτελείται από μία πρόταση, δεν απαιτούνται **{}**.

Παράδειγμα 5.3

Να περιγραφεί η λειτουργία του ακόλουθου τμήματος κώδικα και να δοθεί το διάγραμμα ροής της πρότασης επανάληψης.

```
int count, max_count=30;
for (count=0; count<max_count; count++)
{
    printf( "count is %d\n",count );
    <άλλες προτάσεις>;
}
<επόμενη πρόταση>;
```

Στην πρώτη γραμμή δηλώνονται οι ακέραιες μεταβλητές **count** και **max_count**, και αποδίδεται τιμή στη **max_count**. Ακολουθεί η πρόταση επανάληψης ***for***, η οποία έχει μετρητή τη μεταβλητή **count**, βήμα τη μονάδα και θα εκτελείται καθόσον ο μετρητής έχει τιμή μικρότερη της **max_count**.

Ο βρόχος εκτελείται συνολικά 30 φορές:

count is 0

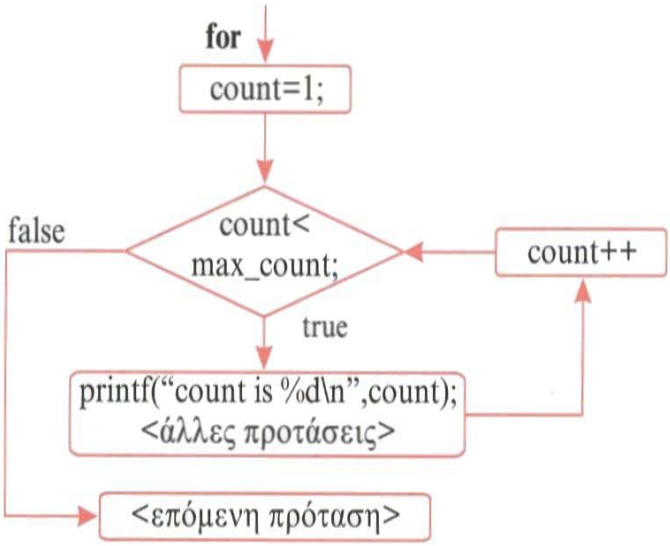
count is 1

. . . . . .

count is 29

Στο τέλος της τριακοστής επανάληψης ο μετρητής έχει λάβει την τιμή 30 και στον έλεγχο της συνθήκης στην τριακοστή πρώτη επανάληψη η τελευταία είναι ψευδής, οπότε ο έλεγχος προσπερνά το βρόχο και προχωρά στην επόμενη πρόταση.

Το διάγραμμα ροής απεικονίζεται στο σχήμα 5-4:



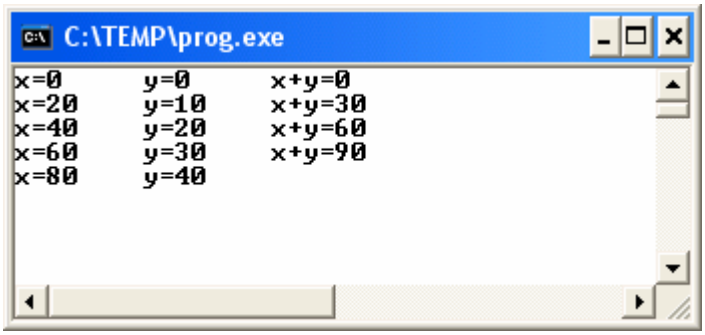
Σχήμα 5-4. *Διάγραμμα ροής της πρότασης επανάληψης*

Παράδειγμα 5.4

Η πρόταση επανάληψης ***for*** μπορεί να χρησιμοποιεί περισσότερες της μίας μεταβλητές ελέγχου του βρόχου. Στο ακόλουθο πρόγραμμα τόσο η μεταβλητή **x** όσο και η **y** ελέγχουν το βρόχο:

```
#include <stdio.h>
void main()
{
    int x,y;
    for (x=0,y=0; x+y<100; x=x+20,y=y+10)
            printf( "x=%d\ty=%d\tx+y=%d\n",x,y,x+y );
    printf( "x=%d\ty=%d ",x,y );
}
```

Το παραπάνω πρόγραμμα τυπώνει τους αριθμούς 0 έως 90 σε βήματα του 30. Σε κάθε εκτέλεση του βρόχου το **x** αυξάνει κατά 20 και το **y** κατά 10. Μετά το τέλος του βρόχου τα **x** και **y** έχουν διατηρήσει τις τιμές που τους δόθηκαν πριν τερματίσει ο βρόχος, όπως φαίνεται στα αποτελέσματα.



5.2.3 Ο τελεστής κόμμα (,)

Στο παράδειγμα 5.4 οι μεταβλητές για το έλεγχο της πρότασης επανάληψης διαχωρίζονταν με τον **τελεστή κόμμα (,)**. Ο τελεστής κόμμα επιτρέπει την παράθεση περισσότερων της μίας εκφράσεων σε θέσεις όπου επιτρέπεται μία έκφραση. Η τιμή της έκφρασης είναι η τιμή της δεξιότερης των εκφράσεων. Συνήθως περιπλέκει τον κώδικα και για αυτό το λόγο η χρήση του είναι περιορισμένη, εκτός από την πρόταση ***for***, στην οποία συνηθίζεται να χρησιμοποιείται ως συνθετικό των εκφράσεων αρχικοποίησης και ανανέωσης. Για παράδειγμα, η πρόταση

for (i=0,j=10; i<8; i++,j++) t[j]=s[i];

αντιγράφει τα οκτώ πρώτα στοιχεία του πίνακα **s** στον **s**, ξεκινώντας από το ενδέκατο στοιχείο του.

Θα πρέπει να αποφεύγονται προτάσεις όπως η

for (ch=getchar(),j=0; ch!= '.'; j++,putchar(ch),ch=getchar())

Η πρόταση, αν και είναι συμπαγής ως προς τον κώδικα, μειώνει σε μεγάλο βαθμό την αναγνωσιμότητά του.

5.2.4 Μετασχηματισμός βρόχων while - for

Ένας βρόχος ***for*** μπορεί να μετασχηματισθεί σε βρόχο ***while*** και τανάπαλιν με βάση την ακόλουθη φόρμα μετασχηματισμού:

```
for (αρχική τιμή μετρητή; συνθήκη; βήμα μετρητή)
{
    προτάσεις;
}
```

↓

```
αρχική τιμή μετρητή;
while (συνθήκη)
{
    προτάσεις;
    βήμα μετρητή;
}
```

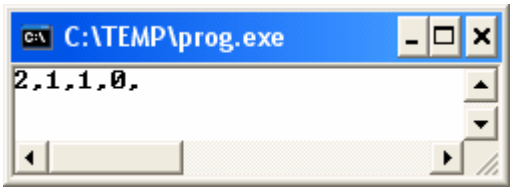
Παράδειγμα 5.5

Τι εμφανίζεται στην οθόνη του υπολογιστή από την εκτέλεση του βρόχου:

for (n = 4; n > 0; n - -) printf("%d,",n/2);

Να γραφεί εκ νέου ο παραπάνω κώδικας αντικαθιστώντας τη ***for*** με ***while***.

Ο παραπάνω κώδικας εκτελεί ένα βρόχο ***for*** με φθίνον βήμα τη μονάδα, σε κάθε επανάληψη του οποίου τυπώνεται το πηλίκο διαίρεσης ακεραίων. Τα αποτελέσματα είναι τα εξής:



Με βάση τη φόρμα μετασχηματισμού της §5.2.4, ο κώδικας με χρήση του βρόχου ***while*** είναι ο ακόλουθος:

```
n=4;
while (n>0)
{
    printf( "%d,",n/2 );
    n- -;
}
```

5.3 Βρόχος με συνθήκη εξόδου στη C (do - while)

Ο βρόχος ***do–while*** είναι βρόχος με συνθήκη εξόδου. Η λειτουργία του περιγράφεται εποπτικά στο σχήμα 5-2α και η σύνταξή του είναι η ακόλουθη:

```
do
{
    προτάσεις, μέσα στις οποίες θα αλλάζει η συνθήκη;
}
while (συνθήκη);
```

Η λειτουργία της πρότασης επανάληψης ***do–while*** μπορεί να μορφοποιηθεί σε δομημένα Ελληνικά ως εξής:

Εκτέλεσε τις προτάσεις
Έλεγξε τη συνθήκη
Εάν είναι αληθής
    Ξεκίνησε από την αρχή
Αλλιώς σταμάτησε

Είναι φανερό ότι ο βρόχος ***do–while*** εκτελείται τουλάχιστον μία φορά, καθώς ο έλεγχος της συνθήκης έπεται του σώματος του βρόχου. Δεν είναι συχνή η χρήση του – στατιστικά χρησιμοποιείται μόνο στο 5% των περιπτώσεων χρήσης βρόχου – καθώς αφενός μεν είναι προτιμότερο να εξετάζεται ένας βρόχος προτού εκτελεσθεί παρά μετά, αφετέρου δε σε πολλές χρήσεις είναι σημαντικό να μπορεί να παραληφθεί τελείως ο βρόχος εφόσον δεν ικανοποιείται εξαρχής ο έλεγχος. Επιπρόσθετα, για το βρόχο while ισχύουν οι παρατηρήσεις της §5.1.2.

Θα πρέπει να σημειωθεί ότι εάν το σώμα του βρόχου αποτελείται από μία πρόταση, δεν απαιτούνται {}.

Παράδειγμα 5.6

Να περιγραφεί η λειτουργία του ακόλουθου τμήματος κώδικα και να δοθεί το διάγραμμα ροής της πρότασης επανάληψης.

```
int count=30;
int limit=40;
do
{
  count++;
  printf( "count is %d\n",count);
}
while (count<limit);
<επόμενη πρόταση>;
```

Στις πρώτες δύο γραμμές δηλώνονται και αρχικοποιούνται οι ακέραιες μεταβλητές **count** και **limit**. Ακολουθεί η πρόταση επανάληψης ***do–while***, η οποία καθορίζει ως συνθήκη η μεταβλητή **count** να είναι μικρότερη της **limit**, γεγονός που αληθεύει. Κατά συνέπεια ο έλεγχος εισέρχεται στο βρόχο, η μεταβλητή **count** αυξάνεται κατά μία μονάδα, τυπώνεται στην οθόνη η φράση

count is 31

και στη συνέχεια διενεργείται ο έλεγχος της συνθήκης. Ο βρόχος εκτελείται συνολικά 10 φορές:

count is 31

count is 32

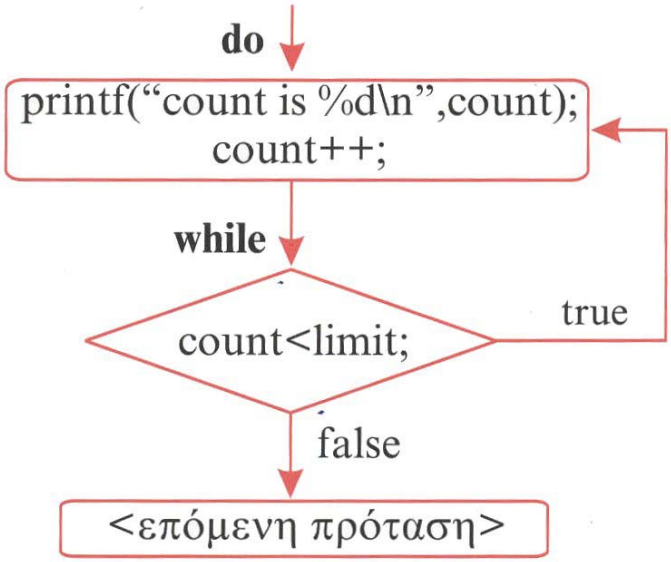
. . . . . .

count is 40

Στο τέλος της δέκατης επανάληψης η συνθήκη είναι ψευδής και ο βρόχος τερματίζεται. Θα πρέπει να σημειωθεί ότι εάν η συνθήκη ήταν εξαρχής ψευδής (π.χ.

count=45), ο βρόχος θα εκτελείτο μία φορά.

Το διάγραμμα ροής απεικονίζεται στο σχήμα 5-5:



Σχήμα 5-5. *Διάγραμμα ροής της πρότασης επανάληψης*

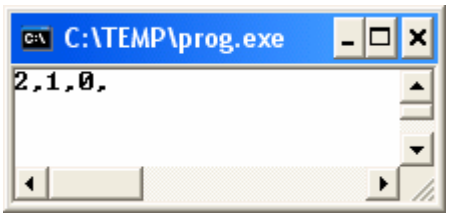
Παράδειγμα 5.7

Τι εμφανίζεται στην οθόνη του υπολογιστή από την εκτέλεση του βρόχου:

for (n=4; n > 1; n- -) printf("%d,",10%n);

Να γραφεί εκ νέου ο παραπάνω κώδικας αντικαθιστώντας τη ***for*** με ***do–while***.

Ο παραπάνω κώδικας εκτελεί ένα βρόχο ***for*** με φθίνον βήμα τη μονάδα, σε κάθε επανάληψη του οποίου τυπώνεται το υπόλοιπο διαίρεσης ακεραίων. Τα αποτελέσματα είναι τα εξής:



Ο κώδικας με χρήση του βρόχου ***do–while*** είναι ο ακόλουθος:

```
n=4;
do
```

```
{
    printf( "%d,",10%n );
    n- -;
}
while (n-1);
```

Θα πρέπει να σημειωθεί ότι στη συνθήκη ελέγχου του βρόχου ο μετρητής δεν είναι το **n** αλλά το **n-1** (δηλαδή η **n-1** να είναι αληθής, επομένως **n-1>0⇔n>1**), καθώς πρέπει να ληφθεί υπόψη η πρώτη επανάληψη, η οποία εκτελείται ούτως ή άλλως.

5.4 Ένθετοι βρόχοι

Ένθετος ή φωλιασμένος βρόχος (nested loop) ονομάζεται ο βρόχος που περικλείεται σε έναν άλλο. Ο εσωτερικός βρόχος λογίζεται ως μία πρόταση του εσωτερικού, κατά συνέπεια πρώτα θα εκτελείται ολόκληρος ο εσωτερικός βρόχος και μετά θα εκτελείται η επόμενη επανάληψη του εξωτερικού. Η C δε θέτει κανένα περιορισμό στην ένθεση των προτάσεων ελέγχου ροής, επιτρέποντας την πολλαπλή ένθεση. Ο συνολικός αριθμός επαναλήψεων σε έναν πολλαπλό βρόχο είναι το γινόμενο του αριθμού των επαναλήψεων όλων των επιμέρους βρόχων.

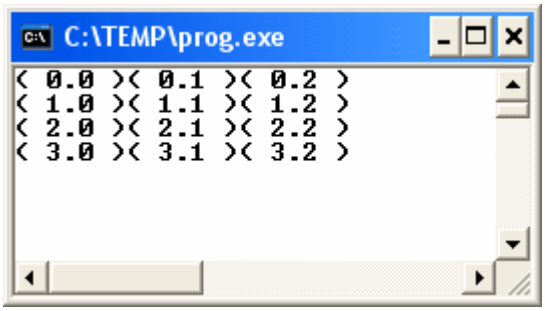
Παράδειγμα 5.8

Να περιγραφεί η λειτουργία του ακόλουθου τμήματος κώδικα.

```
for ( i=0; i<4; i++ )
{
    for ( j=0; j<3; j++ )
    {
```

```
            printf( "( %d.%d )", i, j );
        }
        printf( "\n" );
    }
```

Ο κώδικας παρουσιάζει ένα διπλό βρόχο. Ο εξωτερικός βρόχος ***for*** έχει μετρητή το **i** και σώμα που αποτελείται από δύο προτάσεις: *α)* τον εσωτερικό βρόχο ***for*** με μετρητή το **j** και *β)* την πρόταση ***printf("\n");***. Σε κάθε επανάληψη του εσωτερικού βρόχου θα εκτελούνται όλες οι επαναλήψεις του ένθετου και στη συνέχεια θα εκτελείται η ***printf("\n");***, όπως προκύπτει από τα αποτελέσματα:

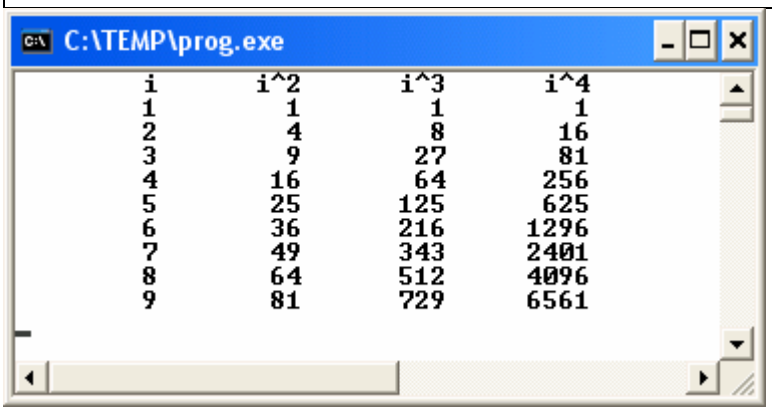


Παράδειγμα 5.9

Το πρόγραμμα που ακολουθεί εμφανίζει τις τέσσερις πρώτες ακέραιες δυνάμεις των αριθμών έως 9.

```
# include <stdio.h>
void main()  {
    int i,j,k,temp;
    printf( "      i     i^2     i^3     i^4\n" );
    for ( i=1; i<10; i++ )  {
        for ( j=1; j<5; j++ )  {
            temp=1;
            for ( k=0; k<j; k++ )  temp=temp*i;
            printf( "%9d",temp );
```

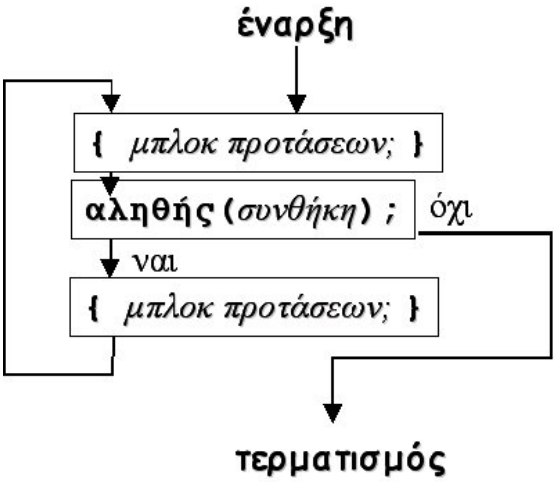
```
            }   // Τέλος του βρόχου j
            printf( "\n" );
        }   // Τέλος του βρόχου i
    }
```



5.5 Διακοπτόμενοι βρόχοι στη C

5.5.1 Η κωδική λέξη break

Στο προηγούμενο κεφάλαιο η λέξη ***break*** χρησιμοποιήθηκε στην πρόταση διακλάδωσης ***switch***. Όμως πέραν αυτής, η ***break*** έχει και δεύτερη χρήση, η οποία σχετίζεται με τις προτάσεις επανάληψης. Χρησιμοποιείται για να τερματίζει αμέσως την εκτέλεση ενός βρόχου, μεταβιβάζοντας τον έλεγχο του προγράμματος στην εντολή που βρίσκεται αμέσως μετά το βρόχο. Εάν η ***break*** βρίσκεται μέσα σε ένθετο βρόχο, τότε επηρεάζεται μόνο ο εσώτερος βρόχος. Η λειτουργία της ***break*** περιγράφεται εποπτικά στο σχήμα 5-6:



Σχήμα 5-6. *Μπλοκ διάγραμμα της break*

Παράδειγμα 5.10

Το πρόγραμμα που ακολουθεί εμφανίζει στην οθόνη τους αριθμούς 1 έως 10 και στη συνέχεια τερματίζεται γιατί η ***break*** υπερφαλαγγίζει τη συνθήκη ελέγχου του βρόχου **t<100**.

```
# include <stdio.h>
void main()
{
    int t;
    for ( t=0; t<100; t++ )  {
        printf( "%d ",t );
        if (t==10) break;
    }
}
```

5.5.2 Η πρόταση continue

Η εντολή ***continue*** μεταφέρει τον έλεγχο της ροής στην αρχή του βρόχου, παραλείποντας την εκτέλεση του υπόλοιπου τμήματος του σώματος του βρόχου και προχωρώντας στην επόμενη επανάληψη.

Στους βρόχους ***while*** και ***do–while*** η εντολή ***continue*** υποχρεώνει τον έλεγχο του προγράμματος να περάσει κατευθείαν στη συνθήκη ελέγχου και να προχωρήσει κατόπιν στην επεξεργασία του βρόχου. Στην περίπτωση της ***for*** ο υπολογιστής εκτελεί πρώτα το τμήμα του βρόχου και κατόπιν τη συνθήκη ελέγχου, προτού συνεχισθεί η εκτέλεση του βρόχου.

Παράδειγμα 5.11

Το πρόγραμμα που ακολουθεί εμφανίζει στην οθόνη μόνο τους άρτιους αριθμούς.

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

Κάθε φορά που παράγεται ένας περιττός αριθμός ενεργοποιείται η εντολή διακλάδωσης και εκτελείται η ***continue***, παρακάμπτεται η ***printf*** και οι υπόλοιπες προτάσεις, οπότε ο έλεγχος προχωρά στην επόμενη επανάληψη.

5.5.3 Η πρόταση goto

Η πρόταση **ρητής διακλάδωσης**

goto <ετικέτα>;

μεταφέρει τον έλεγχο στην πρόταση που σημειώνεται με την ετικέτα ως

<ετικέτα>: πρόταση

Η εντολή ***goto*** **πρέπει να αποφεύγεται** γιατί οδηγεί σε κώδικα «σπαγγέτι» και αίρει τα πλεονεκτήματα του δομημένου προγραμματισμού. Μπορεί να χρησιμοποιηθεί σε περιπτώσεις εξόδου από πολύ βαθιά ενσωματωμένη δομή, όπου μία προσεκτική χρήση της ***goto*** μπορεί να δώσει πιο συμπαγή κώδικα. Θα πρέπει να σημειωθεί ότι η C είναι δομημένη κατά τρόπο ώστε να μην απαιτεί ποτέ τη χρήση τής ***goto***, σε αντιδιαστολή με άλλες γλώσσες προγραμματισμού, όπως η FORTRAN και η BASIC, στις οποίες επιβάλλεται η χρήση της ***goto*** σε ορισμένες περιπτώσεις.

Παράδειγμα 5.12

Έστω το ακόλουθο τμήμα κώδικα:

```
for (. . .)  {
    for (. . .)  {
        while (. . .)  {
            if (. . .)  goto label13;
            . . . . . . .
        }
    }
}
label13:  printf( "ERROR!!!" );
```

Η απαλοιφή της ***goto*** θα υποχρέωνε τον κώδικα να εκτελέσει έναν αριθμό από πρόσθετους ελέγχους. Η χρήση μίας ***break*** δε θα ήταν αρκετή γιατί θα προκαλούσε έξοδο μόνο από τον εσώτερο βρόχο. Εάν τοποθετούνταν έλεγχοι σε όλους τους βρόχους, ο κώδικας θα έπαιρνε την ακόλουθη σωστή αλλά δύσχρηστη μορφή:

```
done=0;
for (. . .) {
    for (. . .) {
        while (. . .) {
            if (. . .) {
                done=1;
                break;   // έξοδος από τη while
            }  // τέλος του if
            . . . . . .
        }  // τέλος της while
        if (done) break;
    } // τέλος του εσωτερικού βρόχου for
    if (done) break;
    }  // τέλος του εξωτερικού βρόχου for
```

5.6 Κανόνες για τη χρήση των προτάσεων επανάληψης

1. Τοποθετείτε πάντοτε το σώμα των προτάσεων επανάληψης μία θέση στηλογνώμονα δεξιότερα, για αύξηση της αναγνωσιμότητας του κώδικα. Στην περίπτωση δε που το σώμα αποτελείται από περισσότερες της μίας προτάσεις, περικλείετε αυτές σε άγκιστρα.

2. Αποφεύγετε τη χρήση της πρότασης διακλάδωσης ***goto***. Καταστρέφει τη δόμηση του προγράμματος και τις περισσότερες φορές προδίδει αδυναμία κατασκευής δομημένου κώδικα.

3. Προτιμήστε το βρόχο επανάληψης συνθήκης εισόδου (***while***) από τον αντίστοιχο συνθήκης εξόδου (***do-while***) γιατί οδηγεί σε πιο ευανάγνωστο κώδικα.

4. Αποφεύγετε κατά το δυνατόν τη χρήση των ***break*** και ***continue*** σε βρόχους επανάληψης, επειδή διακόπτουν την κανονική ροή ελέγχου και καθιστούν την παρακολούθησή της δύσκολη.

5. Ελέγξτε σχολαστικά και βεβαιωθείτε ότι κάθε συνθήκη βρόχου επανάληψης οδηγεί στην έξοδο μετά από πεπερασμένες επαναλήψεις, έτσι ώστε να μη δημιουργούνται ατέρμονες βρόχοι.

6. Πίνακες - αλφαριθμητικά

6.1 Μονοδιάστατοι πίνακες

Ο πίνακας είναι μία συλλογή μεταβλητών ίδιου τύπου, οι οποίες είναι αποθηκευμένες σε διαδοχικές θέσεις μνήμης. Χρησιμοποιείται για την αποθήκευση και διαχείριση δεδομένων κοινού τύπου και αποτελεί, μαζί με τους δείκτες, από τα πλέον ισχυρά εργαλεία της γλώσσας C.

- Η δήλωση του πίνακα ακολουθεί τον εξής φορμαλισμό:

τύπος_δεδομένου όνομα_πίνακα[μέγεθος]

Διακρίνονται τρία τμήματα: *α)* ο τύπος δεδομένου (float, int, char, double), *β)* το όνομα του πίνακα και *γ)* ο αριθμός των στοιχείων που απαρτίζουν τον πίνακα. Έτσι, μία τυπική δήλωση ενός πίνακα 31 στοιχείων κινητής υποδοαστολής είναι η ακόλουθη:

float temp[31];

- Η αναφορά σε στοιχείο πίνακα γίνεται με συνδυασμό του ονόματος και ενός δείκτη (index), ο οποίος εκφράζει τη σειρά τού στοιχείου μέσα στον πίνακα:

temp[0]: ***πρώτο*** στοιχείο του πίνακα

temp[1]: ***δεύτερο*** στοιχείο του πίνακα

temp[30]: ***τελευταίο*** (τριακοστό πρώτο) στοιχείο του πίνακα

- Η απόδοση αρχικής τιμής κατά τη δήλωση του πίνακα γίνεται με χρήση του τελεστή ανάθεσης ως εξής:

float temp[5] = {1,2,-4.2,6,8}: αρχικοποιούνται και τα 5 στοιχεία του πίνακα **temp**.

float temp[5] = {1,2,-4.2}: αρχικοποιούνται τα 3 πρώτα στοιχεία του πίνακα **temp**, δηλαδή τα **temp[0]**, **temp[1]**, **temp[2]**.

- Η ανάγνωση και εκτύπωση ενός πίνακα γίνονται κατά στοιχείο, με τους κανόνες που ισχύουν για κάθε τύπο δεδομένου:

```
for ( i=0; i<ar_size; i++ )
{
    scanf( "%f",&ar[i] );
    printf( "ar[%d]=%f",i,ar[i] );
```

}

<u>Παρατηρήσεις</u>:

1) Όταν αποδίδονται αρχικές τιμές μπορεί να παραληφθεί το μέγεθος του πίνακα. Ο υπολογιστής θα υπολογίσει αυτόματα πόσα είναι τα στοιχεία του πίνακα από τον αριθμό των αρχικών τιμών που δίδονται. Η παρακάτω δήλωση

char d_ar[ ] = {'a', 'b', 'c', 'd'};

έχει ως αποτέλεσμα τη δημιουργία ενός πίνακα χαρακτήρων (***char***) τεσσάρων στοιχείων με αρχικές τιμές:

d_ar[0] = 'a'

d_ar[1] = 'b'

d_ar[2] = 'c'

d_ar[3] = 'd'

2) Το γεγονός ότι οι δείκτες των στοιχείων ενός πίνακα ξεκινούν από το **0** κι όχι από το **1** μπορεί αρχικά να προκαλέσει σύγχυση αλλά αντανακλά τη φιλοσοφία της C, η οποία επιδιώκει να παραμείνει ο προγραμματισμός κοντά στην αρχιτεκτονική του υπολογιστή. Το **0** αποτελεί το σημείο εκκίνησης για τους υπολογιστές. Εάν η αρίθμηση των στοιχείων πίνακα ξεκινούσε από το **1**, όπως π.χ. συμβαίνει στη FORTRAN, ο μεταγλωττιστής θα έπρεπε να αφαιρέσει τη μονάδα από κάθε αναφορά σε δείκτη στοιχείου για να ληφθεί η πραγματική διεύθυνση ενός στοιχείου. Επομένως, η επιλογή της C παράγει πιο αποτελεσματικό κώδικα.

3) Υπάρχει διαφορά ανάμεσα στη δήλωση πίνακα και στην αναφορά στοιχείου πίνακα. Σε μία δήλωση, ο δείκτης καθορίζει το μέγεθος του πίνακα. Σε μία αναφορά στοιχείου πίνακα, ο δείκτης προσδιορίζει το στοιχείο του πίνακα στο οποίο αναφερόμαστε. Π.χ. στη δήλωση **int temp[31];** το **31** δηλώνει τον αριθμό των στοιχείων του πίνακα. Αντίθετα στη **temp[13]=21;** το **13** δηλώνει το συγκεκριμένο στοιχείο (14°) του πίνακα, στο οποίο αναφερόμαστε και αποδίδουμε την τιμή **21**.

4) Μπορεί να βρεθεί το μέγεθος σε bytes ενός πίνακα χρησιμοποιώντας τον τελεστή ***sizeof***. Για παράδειγμα, εάν θεωρηθεί ο πίνακας **int ar[5];** η έκφραση **sizeof(ar)** δίνει τιμή **20** επειδή ο πίνακας αποτελείται από 5 ακεραίους των 4 bytes.

Στη ***sizeof*** θα πρέπει να περιλαμβάνεται μόνο το όνομα του πίνακα. Αν περιληφθεί δείκτης ενός στοιχείου, τότε θα εξαχθεί το μέγεθος του στοιχείου. Για

παράδειγμα, η έκφραση **sizeof(ar[0])** δίνει τιμή **4**.

Χρησιμοποιώντας ένα συνδυασμό των παραπάνω μπορεί να βρεθεί ο αριθμός των στοιχείων του πίνακα. Η έκφραση **sizeof(ar)/sizeof(ar[0])** δίνει **5**, τον αριθμό δηλαδή των στοιχείων του πίνακα **ar**.

6.2 Πολυδιάστατοι πίνακες

Οι πολυδιάστατοι πίνακες είναι πίνακες, τα στοιχεία των οποίων είναι επίσης πίνακες. Η πρόταση **int array[4][12];** δηλώνει τη μεταβλητή **array** ως πίνακα 4 στοιχείων, όπου το κάθε στοιχείο είναι πίνακας 12 στοιχείων ακεραίων. Η C δε θέτει περιορισμό στον αριθμό των διαστάσεων των πινάκων.

Αν και ο πολυδιάστατος πίνακας αποθηκεύεται στη μνήμη ως μία ακολουθία στοιχείων μίας διάστασης, μπορούμε να το θεωρούμε ως πίνακα πινάκων. Για παράδειγμα, έστω το επόμενο «μαγικό τετράγωνο», του οποίου οι γραμμές οριζόντια, κάθετα και διαγώνια δίνουν το ίδιο άθροισμα:

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Για να αποθηκευθεί το τετράγωνο αυτό σε πίνακα θα μπορούσε να γίνει η ακόλουθη δήλωση:

```
int magic[5][5]= {    {17, 24, 1, 8, 15},
                      {23, 5, 7, 14, 16},
                      {4, 6, 13, 20, 22},
                      {10, 12, 19, 21, 3},
                      {11, 18, 25, 2, 9}
                 };
```

Από τον προηγούμενο κώδικα γίνεται αντιληπτό ότι στην απόδοση των αρχικών

τιμών οι τιμές των στοιχείων κάθε γραμμής περικλείονται σε άγκιστρα.

- Για την αναφορά σε στοιχείο ενός πολυδιάστατου πίνακα θα πρέπει να καθορισθούν τόσοι δείκτες όσοι είναι αναγκαίοι. Έτσι, η έκφραση

magic[1]

αναφέρεται στη δεύτερη γραμμή του πίνακα, ενώ η

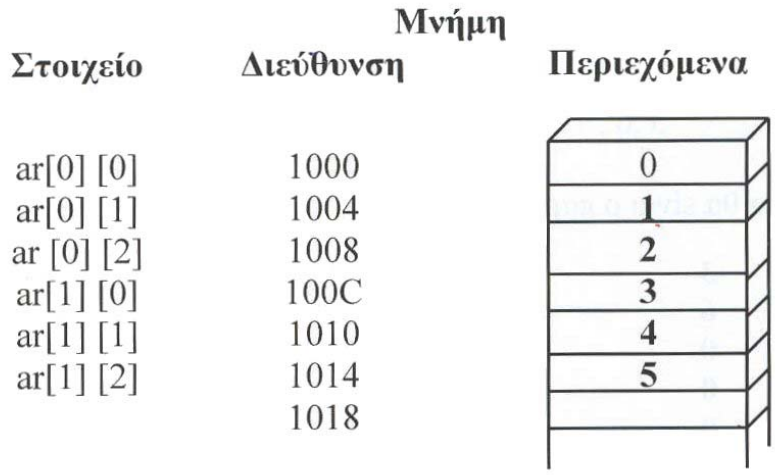
magic[1][3]

αναφέρεται στο τέταρτο στοιχείο της δεύτερης γραμμής του πίνακα.

- Οι πολυδιάστατοι πίνακες αποθηκεύονται κατά γραμμές, που σημαίνει ότι ο τελευταίος δείκτης θέσης μεταβάλλεται ταχύτερα κατά την προσπέλαση των στοιχείων. Για παράδειγμα, ο πίνακας που δηλώνεται ως:

```
int ar[2][3]= {     {0, 1, 2},
                    {3, 4, 5}
              };
```

αποθηκεύεται όπως φαίνεται ακολούθως:



Σχήμα 6-1. *Αποθήκευση πολυδιάστατου πίνακα*

6.3 Αρχικοποίηση πολυδιάστατου πίνακα

Για την αρχικοποίηση ενός πολυδιάστατου πίνακα <u>κάθε γραμμή αρχικών τιμών περικλείεται σε άγκιστρα</u>. Εάν δεν υπάρχουν οι αναγκαίες αρχικές τιμές, τα επιπλέον στοιχεία λαμβάνουν αρχική τιμή 0. Έτσι, στη δήλωση και αρχικοποίηση του

ακόλουθου πίνακα

```
int ar[5][3]= {     {1, 2, 3},
                    {4},
                    {5, 6, 7}
              };
```

η μεταβλητή ***ar*** δηλώνεται ως πίνακας 5 γραμμών και 3 στηλών. Ωστόσο έχουν αποδοθεί τιμές μόνο για τις 3 πρώτες γραμμές του πίνακα και μάλιστα για τη δεύτερη γραμμή έχει ορισθεί η τιμή μόνο του πρώτου στοιχείου. Η παραπάνω δήλωση έχει ως αποτέλεσμα τη δημιουργία του ακόλουθου πίνακα:

1	**2**	**3**
4	**0**	**0**
5	**6**	**7**
0	**0**	**0**
0	**0**	**0**

Εάν δε συμπεριληφθούν τα ενδιάμεσα άγκιστρα:

```
int ar[5][3]= {     1, 2, 3,
                    4,
                    5, 6, 7 };
```

το αποτέλεσμα είναι ο ακόλουθος πίνακας, που είναι σαφώς διαφορετικός, οπότε δημιουργείται πρόβλημα:

1	**2**	**3**
4	**5**	**6**
7	**0**	**0**
0	**0**	**0**
0	**0**	**0**

<u>Παρατηρήσεις:</u>

1. Όπως και με τους πίνακες μίας διάστασης, έτσι και στους πολυδιάστατους πίνακες εάν αμεληθεί να δοθεί το μέγεθος του πίνακα, ο μεταγλωττιστής θα το καθορίσει αυτόματα με βάση τον αριθμό αρχικών τιμών που παρουσιάζονται. Ωστόσο στους πολυδιάστατους πίνακες μπορεί να παραληφθεί ο αριθμός των στοιχείων μόνο της πρώτης διάστασης, καθώς ο μεταγλωττιστής μπορεί να τον υπολογίσει από τον αριθμό των αρχικών τιμών που διατίθενται. Η παρακάτω δήλωση αξιοποιεί τη δυνατότητα αυτή του μεταγλωττιστή:

```
int ar[ ][3][2]=    {    {    {1, 1}, {0, 0}, {1, 1}    },
                         {    {0, 0}, {1, 2}, {0, 1}    }
                    };
```

Με την παραπάνω δήλωση ο **ar** δηλώνεται αυτόματα ως πίνακας 2x3x2.

2. Η παρακάτω δήλωση:

int ar[ ][ ]={ 1, 2, 3, 4, 5, 6};

είναι ανεπίτρεπτη επειδή ο μεταγλωττιστής δεν μπορεί να γνωρίζει τι είδους θα ήταν αυτός ο πίνακας. Θα μπορούσε να το θεωρήσει είτε πίνακα 2x3 είτε 3x2.

3. Η παρακάτω δήλωση:

printf("%d",array[1,2]);

είναι λανθασμένη στη γλώσσα C αλλά δεν εντοπίζεται από το μεταγλωττιστή και οδηγεί σε ανεπιθύμητα αποτελέσματα.. Η σωστή είναι

printf("%d",array[1][2]);

Παράδειγμα 6.1

Έστω πίνακας που αναπαριστά τις μέσες θερμοκρασίες των μηνών των τελευταίων τριών ετών. Να δοθούν: *(α)* βρόχος για τον υπολογισμό των μέσων ετήσιων θερμοκρασιών των τριών ετών, *(β)* βρόχος για τον υπολογισμό των μέσων μηνιαίων θερμοκρασιών των τριών ετών.

α)

```
#define MONTHS 12
#define YEARS 3
. . . . . . .
int year, month;
float subtotal, temp[YEARS][MONTHS], avg_temp[YEARS];
. . . . . . .
// ο temp θεωρείται αρχικοποιημένος
for ( year=0; year<YEARS; year++ )
```

```
{
    subtotal=0.0;
    for ( month=0 ; month<MONTHS ; month++ )
            subtotal+=temp[year][month];
    avg_temp[year]=subtotal/MONTHS;
}
.......
```

β)

```
#define YEARS 3
#define MONTHS 12
.......
int year, month;
float subtotal, temp[YEARS][MONTHS], avg_temp[MONTHS];
.......
// ο temp θεωρείται αρχικοποιημένος
for ( month=0; month<MONTHS; month++ )
{
    subtotal=0.0;
    for ( year=0; year<YEARS; year++ )
    subtotal+=temp[year][month];
    avg_temp[month]=subtotal/YEARS;
}
.......
```

6.4 Αποθήκευση των πινάκων στη μνήμη

Η πρόταση δήλωσης **int ar[5];** έχει ως αποτέλεσμα τη δέσμευση χώρου στη μνήμη για την αποθήκευση 5 μεταβλητών ακέραιου τύπου. Έστω ότι μέσα στον

κώδικα του προγράμματος υπάρχουν οι παρακάτω προτάσεις ανάθεσης, που αποδίδουν τιμές σε στοιχεία του πίνακα:

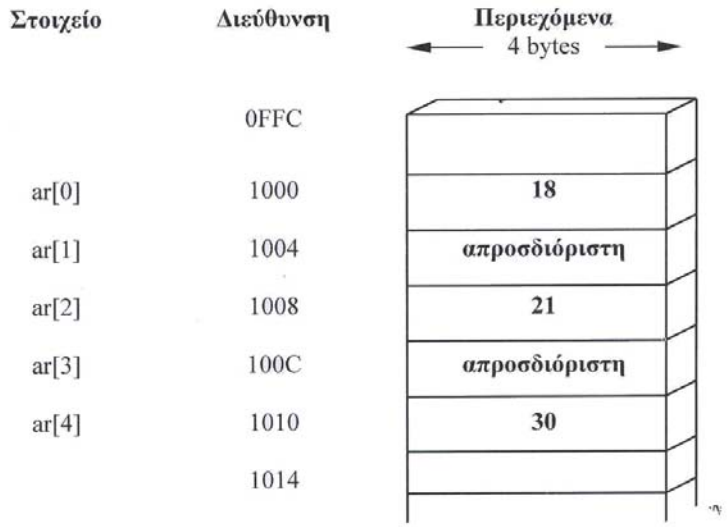
ar[0] = 18;

ar[2] = 21;

ar[4] = ar[0] + 12;

Στο σχήμα 6-2 φαίνεται η μορφή που έχει η μνήμη, η οποία έχει δεσμευτεί για την αποθήκευση του πίνακα **ar**, θεωρώντας 32 bits για κάθε ακέραιο και πως η πρώτη διεύθυνση είναι η 1000. Γίνεται αντιληπτό ότι τα **ar[1]** και **ar[3]** έχουν απροσδιόριστες τιμές. Τα περιεχόμενα αυτών των θέσεων μνήμης είναι ο,τιδήποτε έχει μείνει στις θέσεις αυτές από προηγούμενη αποθήκευση. Οι απροσδιόριστες τιμές αναφέρονται ως «σκουπίδια» (junk) και πολλές φορές αποτελούν αιτία πρόκλησης λαθών.

Για την αποφυγή προβλημάτων αυτής της μορφής πρέπει να αποδίδονται αρχικές τιμές στους πίνακες ή να υπάρχει συνεχής αντίληψη του τι περιλαμβάνει ένας πίνακας. Θα πρέπει να σημειωθεί ότι η ANSI C διασφαλίζει ότι οι καθολικές μεταβλητές (αρχικοποιούνται από το σύστημα με την τιμή 0. Αυτό συμβαίνει βέβαια και για όλα τα στοιχεία πίνακα, ο οποίος έχει δηλωθεί ως καθολική μεταβλητή.



Σχήμα 6-2. *Αποθήκευση πίνακα στη μνήμη*

6.5 Το αλφαριθμητικό

Το αλφαριθμητικό ή **συμβολοσειρά (string)** είναι ένας πίνακας χαρακτήρων που

τερματίζει με το **μηδενικό (null) χαρακτήρα**. Ο μηδενικός χαρακτήρας έχει ASCII κωδικό **0** και αναπαρίσταται από την ακολουθία διαφυγής **\0**.

- Η δήλωση του αλφαριθμητικού ακολουθεί τον εξής φορμαλισμό:

char όνομα[μήκος]

Διακρίνονται τρία τμήματα: *α)* ο τύπος δεδομένου, ο οποίος είναι πάντοτε char, *β)* το όνομα του αλφαριθμητικού και *γ)* το μήκος του. Έτσι, μία τυπική δήλωση ενός αλφαριθμητικού 30 χαρακτήρων έχει την ακόλουθη μορφή:

char book_title[30]

- Τα αλφαριθμητικά μπορούν να εμφανίζονται μέσα στον κώδικα όπως οι αριθμητικές σταθερές, αποτελώντας τις **αλφαριθμητικές σταθερές**. Η αλφαριθμητική σταθερά απαντήθηκε σε προηγούμενο κεφάλαιο, όπου και σημειώθηκε ότι οι χαρακτήρες της περικλείονται σε διπλά εισαγωγικά. Για την αποθήκευσή της χρησιμοποιείται ένας πίνακας χαρακτήρων, με το μεταγλωττιστή να θέτει <u>αυτόματα</u> στο τέλος του αλφαριθμητικού ένα μηδενικό χαρακτήρα για να προσδιορίσει το τέλος του. Έτσι, η αλφαριθμητική σταθερά **"Hello"** απαιτεί για αποθήκευση 6 bytes, όπως φαίνεται παρακάτω:

'H' 'e' 'l' 'l' 'o' '\0'

<u>Παρατήρηση:</u> Θα πρέπει να σημειωθεί η διαφορά ανάμεσα στη σταθερά χαρακτήρα **'A'** και την αλφαριθμητική σταθερά **"A"**. Η πρώτη απαιτεί 1 byte για αποθήκευση, ενώ η δεύτερη απαιτεί ένα byte για το χαρακτήρα **A** κι ένα byte για το **null**.

6.6 Αρχικοποίηση αλφαριθμητικού

Η ανάθεση τιμής με τη δήλωση ακολουθεί το γενικό κανόνα απόδοσης αρχικής τιμής σε πίνακα:

char isbn[ ] = {'0', '-', '4', '9', '-', '7', '4', '3', '-', '3', '\0'};

Στην πράξη χρησιμοποιείται η ακόλουθη εναλλακτική και πιο συμπαγής μορφή με χρήση αλφαριθμητικής σταθεράς:

char isbn[ ] = "0-49-743-3";

Θα πρέπει να προσεχθεί ότι στη δήλωση με λίστα στοιχείων ο προγραμματιστής πρέπει να περιλάβει ως τελευταία τιμή το **null**. Στο δεύτερο τρόπο απόδοσης αρχικής τιμής, αυτή την εργασία την εκτελεί αυτόματα ο μεταγλωττιστής.

6.7 Είσοδος - έξοδος αλφαριθμητικών

6.7.1 Εισαγωγή αλφαριθμητικού

Η εισαγωγή αλφαριθμητικού από την κύρια είσοδο γίνεται με τη μορφοποιούμενη συνάρτηση ***scanf*** και τον προσδιοριστή **%s**. Η πρόταση

scanf("%s", isbn);

διαβάζει την κύρια είσοδο ως αλφαριθμητικό και αποθηκεύει την τιμή στη μεταβλητή **isbn**. Δε χρειάζεται ο τελεστής **&** πριν από το όνομα της μεταβλητής **isbn** όπως συνέβαινε με τους άλλους τύπους δεδομένων, γιατί το όνομα του αλφαριθμητικού αναπαριστά τη διεύθυνση του πρώτου στοιχείου του.

Εναλλακτικά, η εισαγωγή αλφαριθμητικού μπορεί να γίνει με χρήση της συνάρτησης ***gets***, η γενική μορφή της οποίας είναι

gets(όνομα_πίνακα_χαρακτήρων)

Καλείται η ***gets*** με το όνομα του πίνακα χαρακτήρων ως όρισμα, χωρίς δείκτη. Με την επιστροφή από τη ***gets*** το αλφαριθμητικό θα έχει περασθεί στον πίνακα χαρακτήρων. Η ***gets*** θα διαβάζει χαρακτήρες από το πληκτρολόγιο έως ότου πατηθεί το ENTER.

Παρατηρήσεις:

1. Το αλφαριθμητικό αποθηκεύεται σε έναν πίνακα χαρακτήρων μαζί με το τελικό μηδενικό χαρακτήρα. Κατά συνέπεια, όταν δηλώνεται το μέγεθος του πίνακα, έστω N, σε αυτόν μπορεί να αποθηκευθεί αλφαριθμητικό μέγιστου μήκους N-1.

2. Θα πρέπει να σημειωθεί ότι τόσο η ***scanf*** όσο και η ***gets*** δεν εκτελούν έλεγχο ορίων στον πίνακα χαρακτήρων με τον οποίο καλούνται. Εάν π.χ. δηλωθεί **char isbn[30]** και το αλφαριθμητικό είναι μεγαλύτερο από το μέγεθος του **isbn**, ο πίνακας θα ξεπερασθεί.

6.7.2 Εκτύπωση αλφαριθμητικού

Η εκτύπωση αλφαριθμητικής σταθεράς γίνεται με την ***printf*** χωρίς τη χρήση προσδιοριστή. Απλώς της δίνεται η προς εκτύπωση αλφαριθμητική σταθερά:

printf("Hello");

Η εκτύπωση αλφαριθμητικού γίνεται με την ***printf*** χρησιμοποιώντας τον προσδιοριστή **%s**. Η παρακάτω πρόταση

printf("The ISBN code is: %s", isbn);

θα έχει ως αποτέλεσμα να τυπωθεί στην οθόνη η πρόταση

The ISBN code is: 0-49-743-3

Εναλλακτικά, η εκτύπωση αλφαριθμητικής σταθεράς και αλφαριθμητικού μπορεί να γίνει με χρήση της συνάρτησης ***puts***, η γενική μορφή της οποίας είναι

puts(όνομα_πίνακα_χαρακτήρων)

Καλείται η ***puts*** με το όνομα του πίνακα χαρακτήρων ως όρισμα, χωρίς δείκτη, π.χ. **puts(isbn)**. Βέβαια, η ***puts*** παρουσιάζει το μειονέκτημα ότι δεν παρέχει δυνατότητες μορφοποίησης της εξόδου.

Παράδειγμα 6.2

Στο ακόλουθο πρόγραμμα γίνεται εισαγωγή και εκτύπωση αλφαριθμητικών με όλους τους τρόπους που περιγράφηκαν ανωτέρω. Θα πρέπει να προσεχθεί η χρήση της ***define*** για την εισαγωγή αλφαριθμητικής σταθεράς.

```
#include <stdio.h>
#define STA "Hello"
void main()  {
    char str1[ ]= "First String";
    char str2[81];
    puts(STA);
    printf( "\nstr1 is: %s\nGive str2:",str1 );
    gets(str2);
    printf( "\nstr2 is: %s\nGive another str2:",str2 );
    scanf( "%s",str2 );
```

```
        printf( "\nNew str2 is: " );
        puts(str2);
    }
```

6.8 Συναρτήσεις αλφαριθμητικών

Η C υποστηρίζει μία ποικιλία συναρτήσεων για το χειρισμό των αλφαριθμητικών. Οι συναρτήσεις αυτές βρίσκονται στο αρχείο κεφαλίδας **<string.h>**. Οι πιο συνηθισμένες παρουσιάζονται στον ακόλουθο πίνακα, στον οποίο η δεύτερη και η τρίτη στήλη περιέχουν τα ονόματα των συναρτήσεων όταν η λειτουργία τους επιδρά σε ολόκληρο το αλφαριθμητικό ή στους πρώτους *n* χαρακτήρες, αντίστοιχα:

Λειτουργία	**Όλοι οι χαρακτήρες**	**Οι *n* πρώτοι χαρακτήρες**
Εύρεση μήκους string	**strlen()**	
Αντιγραφή string	**stcpy()**	**strncpy()**
Συνένωση 2 strings	**strcat()**	**strncat()**
Σύγκριση 2 strings	**strcmp()**	**strncmp()**
Εύρεση χαρακτήρα σε string	**strchr()**	**strrchr()**
Εύρεση string σε string	**strstr()**	

6.8.1 Η συνάρτηση μήκους αλφαριθμητικού

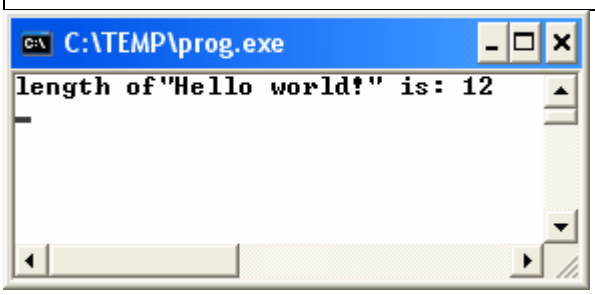
Η συνάρτηση ***strlen()*** επιστρέφει τον αριθμό χαρακτήρων του αλφαριθμητικού, χωρίς να συμπεριλαμβάνει το μηδενικό χαρακτήρα. Το παρακάτω τμήμα κώδικα

```
char name[12] = "abcd";
printf( "%d\n", strlen(name) );
```

θα τυπώσει τον αριθμό των χαρακτήρων του αλφαριθμητικού **name**, δηλαδή 4 κι όχι 12, που είναι ο αριθμός των στοιχείων του πίνακα χαρακτήρων **name**.

Παράδειγμα 6.3

```
#include <stdio.h>
#include <string.h>
void main(
{
    int cnt;
    char msg[20] = { "Hello world!" };
    cnt = strlen(msg);
    printf( "length of\"%s\" is: %d\n",msg,cnt );
}
```



6.8.2 Η συνάρτηση αντιγραφής αλφαριθμητικού

Η συνάρτηση ***strcpy()*** αντιγράφει ένα αλφαριθμητικό από έναν πίνακα σε έναν άλλο. Δέχεται δύο ορίσματα που είναι τα ονόματα των αλφαριθμητικών. ***Το όνομα του πίνακα προορισμού πρέπει να είναι το πρώτο όρισμα, ενώ το δεύτερο όρισμα προσδιορίζει τον πίνακα πηγής***. Στο παρακάτω τμήμα κώδικα

```
char name1[12] = "abcd";
```

```
char name2[12] = "ef";
strcpy(name1,name2);
printf( "%s\n", name1 );
```

η πρόταση **strcpy(name1,name2);** αντιγράφει το περιεχόμενο του πίνακα **name2** στον πίνακα **name1**. Έτσι στην οθόνη θα εμφανισθεί το «**ef**».

Για να αντιγραφούν οι πρώτοι *n* χαρακτήρες του **name2** χρησιμοποιείται η σύνταξη **strncpy(name1,name2,n)**, όπου το τρίτο όρισμα είναι ο αριθμός των προς αντιγραφή χαρακτήρων.

6.8.3 Η συνάρτηση συνένωσης αλφαριθμητικών

Η συνάρτηση ***stcat()*** δέχεται δύο ορίσματα που είναι τα ονόματα των αλφαριθμητικών, τα οποία και συνενώνει. Συγκεκριμένα, ***προσθέτει στο τέλος του αλφαριθμητικού, που προσδιορίζεται από το πρώτο όρισμα, τα στοιχεία του αλφαριθμητικού που προσδιορίζεται από το δεύτερο όρισμα***. Στο παρακάτω τμήμα κώδικα

```
char name1[12] = "abcd";
char name2[12] = "ef";
strcat(name1,name2);
printf( "%sn", name1 );
```

προστίθεται στο τέλος του πίνακα **name1** το περιεχόμενο του πίνακα **name2**. Έτσι στην οθόνη θα εμφανισθεί το «**abcdef**».

Για να προστεθούν οι πρώτοι ***n*** χαρακτήρες του **name2** χρησιμοποιείται η σύνταξη **strncat(name1,name2,n)**, όπου το τρίτο όρισμα είναι ο αριθμός των προστιθέμενων χαρακτήρων.

Παράδειγμα 6.4

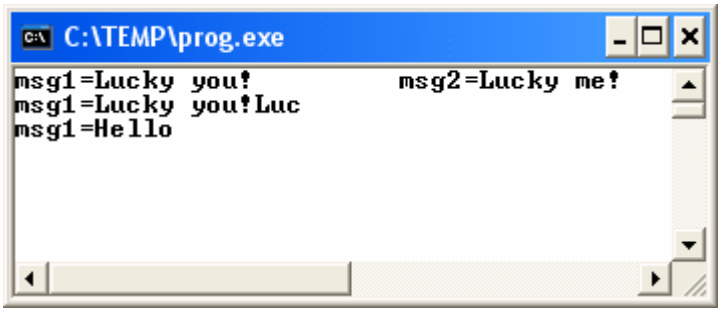
Να περιγραφεί η λειτουργία του ακόλουθου προγράμματος:

```
#include <stdio.h>
#include <string.h>
void main()
```

```
{
    char msg1[20] ,msg2[20];
    strcpy( msg1, "Lucky you!" );
    strcpy( msg2, "Lucky me!" );
    printf( "msg1=%s\t\tmsg2=%s\n", msg1,msg2 );
    strncat( msg1,msg2,3 );
    printf( "msg1=%s\n", msg1 );
    strcpy( msg1,"Hello" );
    printf( "msg1=%s\n", msg1 );
}
```

Δηλώνονται δύο πίνακες χαρακτήρων 20 θέσεων και με χρήση της συνάρτησης αντιγραφής αλφαριθμητικών τους αποδίδονται τα περιεχόμενα **"Lucky you!"** και **"Lucky me!"**, αντίστοιχα.

Ακολούθως οι τρεις πρώτοι χαρακτήρες του **msg2** προσαρτώνται στο τέλος του **msg1**. Τέλος, στο **msg1** αντιγράφεται το αλφαριθμητικό **"Hello"**.



6.8.4 Η συνάρτηση σύγκρισης αλφαριθμητικών

Η συνάρτηση ***strcmp(name1,name2)*** δέχεται δύο ορίσματα που είναι τα ονόματα των αλφαριθμητικών, τα οποία και συγκρίνει. Η έξοδός της είναι ένας ακέραιος αριθμός, ο οποίος λαμβάνει την τιμή **0** εφόσον τα αλφαριθμητικά είναι όμοια.

Για να συγκριθούν οι πρώτοι ***n*** χαρακτήρες των αλφαριθμητικών χρησιμοποιείται η σύνταξη **strncmp(name1,name2,n)**, όπου το τρίτο όρισμα είναι ο αριθμός των προς σύγκριση χαρακτήρων.

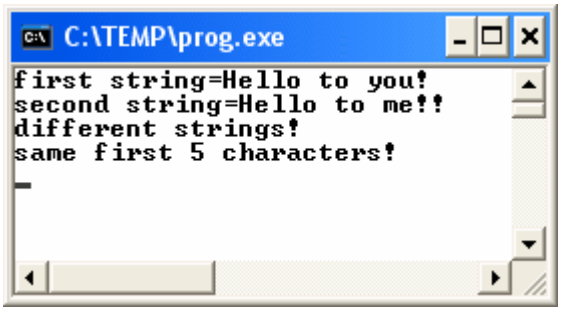
Παράδειγμα 6.5

Να περιγραφεί η λειτουργία του ακόλουθου προγράμματος:

```
#include <stdio.h>
#include <string.h>
#define N 5
void main()
{
    char msg1[81] = {"Hello to you"};
    char msg2[81] = {"Hello to me!"};
    int diff;
    printf( "first string=%s!\n",msg1 );
    printf( "second string=%s!\n",msg2 );
    diff = strcmp(msg1,msg2);
    if(diff==0) printf( "same strings!\n" );
    else printf( "different strings!\n" );
    diff = strncmp(msg1,msg2,N);
    if(diff==0) printf( "same first %d characters!\n",N );
    else printf( "different first %d characters!\n",N );
}
```

Δηλώνονται δύο πίνακες χαρακτήρων 81 θέσεων, στους οποίους αποδίδονται τα περιεχόμενα **"Hello to you"** και **"Hello to me!"**, αντίστοιχα.

Ακολούθως συγκρίνονται τα δύο αλφαριθμητικά με χρήση της ***strcmp()*** και κατόπιν συγκρίνονται οι πέντε πρώτοι χαρακτήρες τους με χρήση της ***strncmp()***.


C:\TEMP\prog.exe
first string=Hello to you!
second string=Hello to me!!
different strings!
same first 5 characters!

7. Δημιουργία προγραμμάτων - παραδείγματα

7.1 Γενικά

Στο παρόν κεφάλαιο θα χρησιμοποιηθεί το σύνολο των στοιχείων της C, που παρατέθηκαν στα προηγούμενα κεφάλαια, με άμεσο στόχο την υλοποίηση απλών προβλημάτων και απώτερο την εισαγωγή στη λογική της προγραμματιστικής ανάλυσης. Επιπρόσθετα, τα παραδείγματα που θα αναπτυχθούν στη συνέχεια αποτελούν μία ενδεικτική αναφορά για τον αναγνώστη, καθώς αποτελούν επιλεγμένα θέματα εξετάσεων.

Πρόβλημα 7.1

Να γραφεί πρόγραμμα, το οποίο να δέχεται ως είσοδο κείμενο, να απαριθμεί τις εμφανίσεις των ψηφίων 0-9, τα λευκά διαστήματα και τους υπόλοιπους χαρακτήρες και στη συνέχεια να τυπώνει τα αποτελέσματα. Το πρόγραμμα ολοκληρώνεται όταν δοθεί ο χαρακτήρας τελεία '**.**'.

Το παραπάνω πρόβλημα μπορεί να μορφοποιηθεί σε δομημένα Ελληνικά ως εξής:

Για κάθε χαρακτήρα του κειμένου που είναι διάφορος της '. '
ελέγχεται ο τύπος του χαρακτήρα
αν είναι ένας από τους ' ', '\t', '\n'
αυξάνεται κατά μία μονάδα ο απαριθμητής των διαστημάτων
αν είναι ψηφίο
αυξάνεται κατά μία μονάδα ο απαριθμητής που αντιστοιχεί στο ψηφίο
σε κάθε άλλη περίπτωση
αυξάνεται κατά μία μονάδα ο απαριθμητής των υπόλοιπων χαρακτήρων

Αναπαράσταση δεδομένων:

- Όσον αφορά στις μεταβλητές, απαιτείται ένας απαριθμητής για τα διαστήματα, τον οποίο ονομάζουμε **n_white**, ένας απαριθμητής για τους λοιπούς χαρακτήρες, ο **n_other**, και δέκα απαριθμητές για τα ψηφία. Για την τελευταία περίπτωση μπορούμε να δηλώσουμε δέκα ανεξάρτητες μεταβλητές αλλά είναι προτιμότερο να επιλεχθεί ένας πίνακας δέκα θέσεων, όπως ο **int n_digit[10]**, ο οποίος οδηγεί σε πιο συμπαγή και δομημένο κώδικα.

- Για την εκτύπωση των αριθμών των εμφανίσεων των δέκα ψηφίων, στην περίπτωση του πίνακα απαριθμητών, ο αντίστοιχος κώδικας θα έχει τη μορφή

```
for (i=0;i<10;i++)
        printf( "Digit %d has been encountered \t %d \t times\n",i,n_n_digit[i]
);
```

Αναπαράσταση διεργασιών:

- Για τη διεργασία «πάρε χαρακτήρα» χρησιμοποιείται η συνάρτηση ***getchar***, η οποία επιστρέφει το χαρακτήρα που διαβάζει από την κύρια είσοδο. Αυτός ο χαρακτήρας πρέπει να αποθηκευθεί σε μία μεταβλητή τύπου χαρακτήρα για περαιτέρω επεξεργασία.

Τα παραπάνω οδηγούν στη δήλωση

char ch;

και στην έκφραση

ch=getchar();

η τιμή της οποίας είναι η τιμή του αριστερού τελεστέου της έκφρασης.

- Για την ανίχνευση του τέλους της εισαγωγής χαρακτήρων χρησιμοποιούμε το συσχετιστικό τελεστή **!=** οδηγούμαστε στην έκφραση

(ch=getchar())!='.'

Η έκφραση γίνεται ψευδής όταν αναγνωσθεί ο χαρακτήρας τελεία. Μπορεί επομένως να χρησιμοποιηθεί ως έκφραση μίας πρότασης ***while***, που θα οδηγεί στην επανάληψη του συνόλου των ενεργειών που το πρόγραμμα πρέπει να εκτελεί για κάθε χαρακτήρα.

<u>Διαμόρφωση της ροής ελέγχου:</u>

Με βάση τα προηγούμενα, η περιγραφή διαμορφώνεται ως εξής:

while ((ch=getchar())!='.')
{
ελέγχεται ο τύπος του χαρακτήρα
αν είναι ένας από τους ' ', '\t', '\n'
αυξάνεται κατά μία μονάδα ο απαριθμητής των διαστημάτων
αν είναι ψηφίο
αυξάνεται κατά μία μονάδα ο απαριθμητής που αντιστοιχεί στο ψηφίο
σε κάθε άλλη περίπτωση
αυξάνεται κατά μία μονάδα ο απαριθμητής των υπόλοιπων χαρακτήρων
}

Το σώμα της ***while*** αποτελεί κλασική περίπτωση επιλογής από αμοιβαία αποκλειόμενες ενέργειες, γεγονός που οδηγεί στη χρήση της πρότασης ***switch***. Η έκφραση ανάλογα με την τιμή της οποίας θα γίνει η επιλογή της κατάλληλης ενέργειας είναι η απλή έκφραση **ch**.

A) **εάν είναι ένας από τους ' ', '\t', '\n'**
αυξάνεται κατά μία μονάδα ο απαριθμητής n_white

ή εκφρασμένη στη C

```
case ' ':
case '\t':
case '\n':
    n_white++;
break;
```

B) **εάν ο χαρακτήρας είναι ψηφίο**
αυξάνεται κατά μία μονάδα ο απαριθμητής που αντιστοιχεί στο ψηφίο

μία πρώτη μορφή του κώδικα είναι η παρακάτω

```
case '0':
    n_digit++;
break;
. . . . . . . . . .
case '9':
    n_white++;
break;
```

Ο κώδικας αυτός δεν εκμεταλλεύεται τη δήλωση των απαριθμητών των ψηφίων ως πίνακα χαρακτήρων. Για το λόγο αυτό, θα προσπαθήσουμε να ενοποιήσουμε τα ***case*** ώστε να έχουν μία παραμετρική πρόταση, που σε κάθε περίπτωση θα οδηγεί στην αύξηση του κατάλληλου απαριθμητή. Θεωρούμε την έκφραση

ch-'o'

και εξετάζουμε την τιμή της για τιμές τής **ch** από το σύνολο {'0', '1', …, '9'}. Είναι προφανές ότι, εάν το **ch** είναι **'0'**, η έκφραση έχει τιμή 0 οπότε και η πρόταση

n_digit[ch-'0']++;

έχει ως αποτέλεσμα την αύξηση του απαριθμητή που αντιστοιχεί στο ψηφίο **'0'**.

Αντίστοιχα, η παραπάνω πρόταση για **ch='8'** θα αυξήσει τον απαριθμητή που αντιστοιχεί στο **'0'**. Κατά αυτόν τον τρόπο οδηγούμαστε στον ακόλουθο συμπαγή κώδικα:

```
case '0':
case '1':
. . . . . . . . . ....
case '9':
    n_digit[ch-'0']++;
break;
```

Γ) Για τα υπόλοιπα στοιχεία έχουμε την κλασική περίπτωση χρήσης της εντολής ***default***, οπότε προκύπτει:

```
default:
    n_other++;
break;
```

Το τελευταίο σημείο που πρέπει να προσεχθεί είναι η αρχικοποίηση των μεταβλητών.

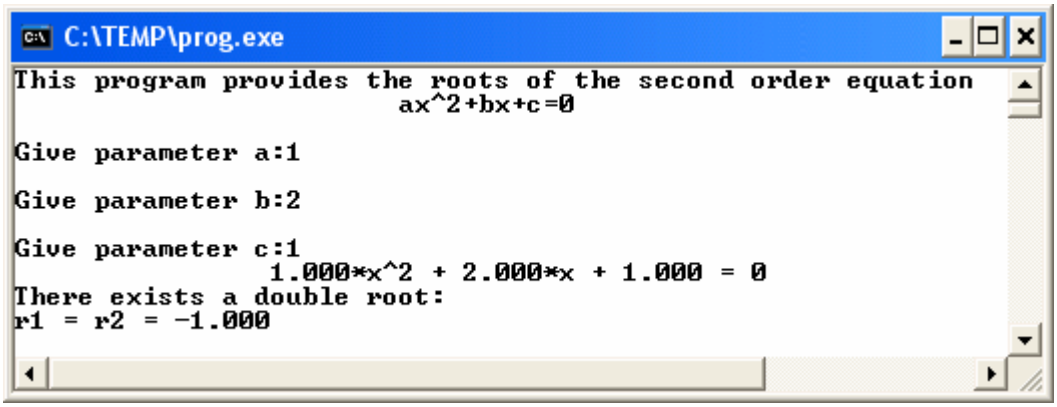
Πρόβλημα 7.2

Να γραφεί πρόγραμμα που επιλύει δευτεροβάθμιες εξισώσεις. Να δέχεται ως είσοδο τους συντελεστές της εξίσωσης και να ελέγχει το είδος των ριζών (απλές πραγματικές, συζυγείς μιγαδικές ή διπλή πραγματική). Θεωρείστε την περίπτωση πραγματικών συντελεστών.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>            // για sqrt(), fabs()
void main()
{
    float a,b,c, D, r1,r2, r,im;
    printf( "This program provides the roots of the second order
            equation\n" );
    printf( "\t\t\tax^2+bx+c=0\n" );
    printf( "\nGive parameter a:");     scanf("%f",&a );
    if (a==0) // Έλεγχος για α=0, οπότε η εξίσωση γίνεται α/θμια
    {
        printf( "For a 2nd order equation, a!=0. Try again\n" );
        printf( "\nGive parameter a:");     scanf("%f",&a );
    }  //τέλος του if
    printf( "\nGive parameter b:");     scanf("%f",&b );
    printf( "\nGive parameter c:");     scanf("%f",&c );
    printf( "\t\t%.3f*x^2 + %.3f*x + %.3f = 0\n",a,b,c ) ;

    D = b*b-4*a*c;    // Διακρίνουσα
```

```
        if(D<0)  // Εάν Δ<0, οι ρίζες είναι συζυγείς μιγαδικές
        {
            printf( "There exist two conjugate complex roots:\n" );
            r=-b/(2*a);
            im=sqrt(-D)/(2*a);
            printf( "r1 = %.3f + j%.3f\n",r,im );
            printf( "r2 = %.3f - j%.3f",r,im );
        }  //τέλος του if
        else if (fabs(D)<1e-10)    // fabs → float,   abs → integer
        {                    // Εάν Δ=0, υπάρχει διπλή ρίζα
            printf( "There exists a double root:\n" );
            printf( "r1 = r2 = %.3f\n", -b/(2*a) );
        } //τέλος του else if
        else // Σε κάθε άλλη περίπτωση υπάρχουν δύο πραγματικές ρίζες
        {
            printf( "There exist two real roots:\n" );
            r1=(-b+sqrt(D))/(2*a);
            r2=(-b-sqrt(D))/(2*a);
            printf("r1=%.3f\nr2=%.3f\n",r1,r2 );
        } //τέλος της else
    } //τέλος της main
```

<u>Αποτέλεσμα για διπλή πραγματική ρίζα:</u>



Αποτέλεσμα για συζυγείς μιγαδικές ρίζες:

```
C:\TEMP\prog.exe
This program provides the roots of the second order equation
                        ax^2+bx+c=0

Give parameter a:1

Give parameter b:1

Give parameter c:2
                 1.000*x^2 + 1.000*x + 2.000 = 0
There exist two conjugate complex roots:
r1 = -0.500 + j1.323
r2 = -0.500 - j1.323
```

Αποτέλεσμα για απλές πραγματικές ρίζες:

```
C:\TEMP\prog.exe
This program provides the roots of the second order equation
                        ax^2+bx+c=0

Give parameter a:1

Give parameter b:1

Give parameter c:-4
                 1.000*x^2 + 1.000*x + -4.000 = 0
There exist two real roots:
r1=1.562
r2=-2.562
```

Αποτέλεσμα για α=0:

```
C:\TEMP\prog.exe
This program provides the roots of the second order equation
                        ax^2+bx+c=0

Give parameter a:0
For a second order equation, a should not be 0. Try again

Give parameter a:_
```

Πρόβλημα 7.3

Να γραφεί πρόγραμμα, χωρίς χρήση της εντολής ***goto***, με το οποίο θα εισάγονται από το πληκτρολόγιο δύο ζεύγη πραγματικών αριθμών (*x1*,*y1*), (*x2*,*y2*). Από τα ζεύγη

αυτά θα υπολογίζονται οι συντελεστές *a*, *b* της εξίσωσης της ευθείας *y=ax+b*. Θα πρέπει να λαμβάνεται πρόνοια ώστε σε περίπτωση που *x2=x1* να ζητείται εκ νέου το σημείο (*x2*,*y2*). Οι συντελεστές *a*, *b* να εμφανίζονται στην οθόνη.

```
#include <stdio.h>
void main()
{
    float x1,y1,x2,y2,a,b;
    printf( "\nGive x1:"); scanf( "%f",&x1 );
    printf( "\nGive y1:"); scanf( "%f",&y1 );
    printf( "\nGive x2:"); scanf( "%f",&x2 );
    while (x2==x1)
    {
        printf( "n\ERROR! x2=x1, give x2 again, x2:");
        scanf( "%f",&x2 );
    }
    printf( "\nGive y2:"); scanf( "%f",&y2 );
    a=(y2-y1)/(x2-x1);
    b=y2-x2*a;
    printf( "y=ax+b, a=%f, b=%f\n",a,b );
} //τέλος της main
```

Πρόβλημα 7.4

Να γραφεί πρόγραμμα, το οποίο θα δέχεται από το πληκτρολόγιο διαδοχικά 6 ακέραιους αριθμούς του ΛΟΤΤΟ. Θα πρέπει να λαμβάνεται πρόνοια ώστε όταν είτε ένας αριθμός δεν ανήκει στο [1 49] είτε επαναλαμβάνεται αριθμός που δόθηκε προηγουμένως, να ζητείται νέα τιμή γι' αυτόν. Οι αριθμοί θα αποθηκεύονται σε πίνακα ακεραίων και θα εμφανίζονται στην οθόνη μετά το πέρας της εισαγωγής τους.

```
#include <stdio.h>
#define N 6
void main()
{
    int lotto[6];
    int j,deikt,i=0;
    while (i<N)
    {
        deikt=0;
        printf( "\nNumber %d, enter your choice:",i+1 );
        scanf( "%d",&lotto[i] );
        if ((lotto[i]<1) || (lotto[i]>49)) deikt++;   // [1,49]
        for (j=0;j<i;j++) if (lotto[j]==lotto[i]) deikt++;  //Να μην
        επαναληφθεί
                                                    //προηγούμενος αριθμός
        if (deikt) printf( "\nInvalid choice!! Try again\n" );
        else i++;
    }  //τέλος της while
    for (i=0;i<N;i++) printf( "\nNumber %d is set to %d",i+1,lotto[i] );
}  //τέλος της main
```

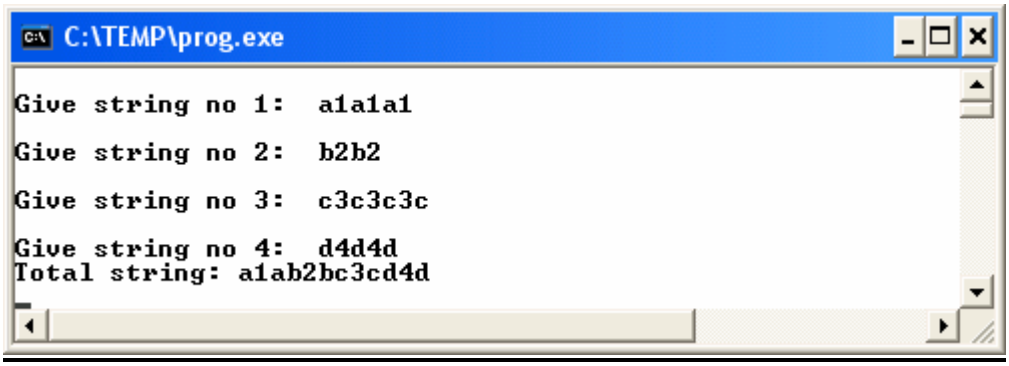
```
C:\TEMP\prog.exe

Number 1, enter your choice:45

Number 2, enter your choice:67

Invalid choice!! Try again

Number 2, enter your choice:-1

Invalid choice!! Try again

Number 2, enter your choice:2

Number 3, enter your choice:2

Invalid choice!! Try again

Number 3, enter your choice:5

Number 4, enter your choice:7

Number 5, enter your choice:8

Number 6, enter your choice:9

Number 1 is set to 45
Number 2 is set to 2
Number 3 is set to 5
Number 4 is set to 7
Number 5 is set to 8
Number 6 is set to 9
```

Πρόβλημα 7.5

Να γραφεί πρόγραμμα, με το οποίο θα εισάγονται από το πληκτρολόγιο 4 αλφαριθμητικά σε πίνακα αλφαριθμητικών, μήκους 7 χαρακτήρων το καθένα. Στη συνέχεια θα λαμβάνονται οι τρεις πρώτοι χαρακτήρες κάθε αλφαριθμητικού και θα συνενώνονται σε ένα νέο αλφαριθμητικό, το οποίο και θα τυπώνεται.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main()
{
    int i;
    char str[4][8], str_total[13];     //στις διαστάσεις των πινάκων θα πρέπει
να
                                       //ληφθεί υπόψη και ο '\0'
    for (i=0;i<4;i++)
    {
```

```
            printf( "\nGive string no %d:  ",i+1 );
            scanf( "%s",str[i] );
            if (i==0) strncpy(str_total,str[i],3);
            else strncat(str_total,str[i],3);
        }
        printf( "Total string: %s\n",str_total );
    } //τέλος της main
```

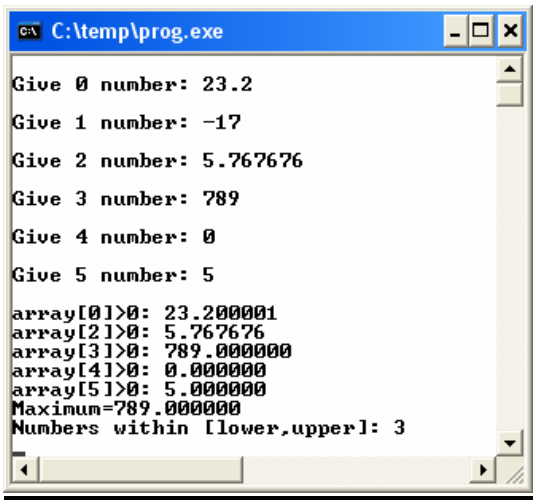


Πρόβλημα 7.6

Να γραφεί πρόγραμμα με το οποίο θα εισάγονται 6 πραγματικοί αριθμοί από το πληκτρολόγιο, θα αποθηκεύονται στον πίνακα **array[]** και θα τυπώνονται: α) οι θετικοί εξ αυτών, β) ο μεγαλύτερος, γ) ο αριθμός των στοιχείων του **array[]**, τα οποία έχουν τιμές στο διάστημα [1.05 50.8].

```
    #include <stdio.h>
    #include <conio.h>
    #include <math.h>
    #define N 6
    #define lower 1.05
    #define upper 50.8
    void main()
    {
```

```
        float array[N],maxim;
        int i,count=0;
        for (i=0;i<N;i++)
        {
            printf( "\nGive %d number: ",i );
            scanf( "%f",&array[i] );
        }
        maxim=array[0];
        printf( "\n" );
        for (i=0;i<N;i++)    /* i=0 για να μπουν όλα σε ένα βρόχο, μόνο για το
                                μέγιστο θα ήταν i=1 */
        {
            if (array[i]==fabs(array[i])) printf( "array[%d]>0: %f\n",i,array[i]
);
            if (array[i]>maxim) maxim=array[i];
            if ((array[i]>=lower) && (array[i]<=upper)) count++;
        }
        printf( "Maximum=%f\n",maxim );
        printf( "Numbers within [lower,upper]: %d\n",count );
    }  //τέλος της main
```



Πρόβλημα 7.7

Να γραφεί πρόγραμμα με το οποίο θα εισάγονται 6 χαρακτήρες από το πληκτρολόγιο, θα αποθηκεύονται στον πίνακα **array[]** και θα τυπώνονται διαδοχικά τα ακόλουθα:

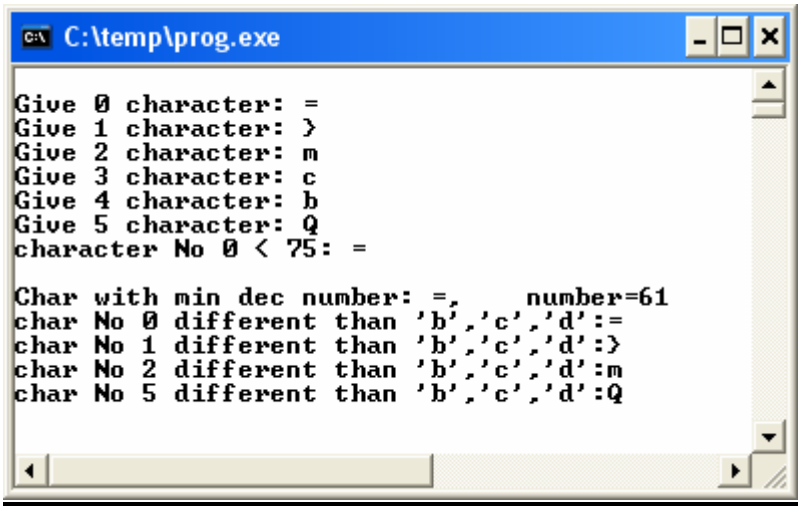
α) οι χαρακτήρες με δεκαδικό ισοδύναμο μικρότερο του 75

β) ο χαρακτήρας με το μικρότερο δεκαδικό ισοδύναμο

γ) όσοι χαρακτήρες είναι διάφοροι των χαρακτήρων '**b**', '**c**', '**d**' (υλοποίηση αποκλειστικά με χρήση της εντολής ***switch-case***)

```
#include <stdio.h>
#include <conio.h>
#define N 6
void main()
{
    char array[N];
    int minim,i;
    for (i=0;i<N;i++)
    {
        printf( "\nGive %d character: ",i );
        array[i]=getche();
    }
    printf( "\n" );
    minim=array[0];
    for (i=0;i<N;i++)
    {
        if (array[i]<75) printf("character No %d < 75: %c\n",i,array[i] );
        if (array[i]<minim) minim=array[i];
    }
    printf( "Char with min dec number: %c, number=%d\n",minim,minim);
    for (i=0;i<N;i++)
    {
```

```
                switch (array[i])
                {
                    case 'b':
                    case 'c':
                    case 'd':
                    break;
                    default:
                      printf("char No %d different than
'b','c','d':%c\n",i,array[i]);
                    break;
                } //τέλος της switch
            } // τέλος της for
        } //τέλος της main
```



Βιβλιογραφία

Α. Έντυπη

1 *Modern C++ Design: Generic Programming and Design Patterns*, A Alexandrescu, Addison-Wesley, 2001.

2 *C Προγραμματισμός*, H M Deitel, Prentice-Hall, 2003.

3 *C: How to Program*, H M Deitel, P J Deitel, Εκδόσεις Γκιούρδα, 1996.

4 *The Pragmatic Programmer: From Journeyman to Master*, A Hunt, D Thomas, Addison-Wesley, 1999.

5 *Διαδικαστικός Προγραμματισμός - C (Τόμος Α)*, Κλ Θραμπουλίδης, Εκδόσεις Τζιόλα, 2002.

6 *A Book on C*, A Kelley. I Pohl, Addison-Wesley, 1997.

7 *The Practice of Programming*, B W Kernighan, Addison-Wesley, 1999.

8 *Η γλώσσα προγραμματισμού C*, B W Kernighan. D M Ritchie, Εκδόσεις Κλειδάριθμος, 1990.

9 *C Programming: A Modern Approach*, K N King, W.W. Norton & Company, 1996.

10 *The Art of Computer Programming*, D E Knuth, Addison-Wesley, 1997.

11 *Χρήση και Προγραμματισμός Turbo C++*, R Lafore, Εκδόσεις Γκιούρδα, 1992.

12 *Αντικειμενοστρεφής Προγραμματισμός με την Turbo C++*, R Lafore, Εκδόσεις Κλειδάριθμος, 1992.

13 *Go To*, S Lohr, Basic Books, 2001.

14 *Προγραμματισμός Ι: Παρουσιάσεις Διαλέξεων*, Π Μαστοροκώστας, ΑΤΕΙ Σερρών, 2004.

15 *C++ Primer Plus*, S Prata, SAMS, 2002.

16 *Εγχειρίδιο εκμάθησης Turbo C*, H Schildt, Εκδόσεις Κλειδάριθμος, 1989.

17 *The C++ Programming Language*, B Stroustrup, Addison-Wesley, 1997.

18 *Expert C Programming*, P Van der Linden, Prentice Hall, 1994.

19 *C Βήμα προς Βήμα*, M Waite, S Prata, Εκδόσεις Γκιούρδα, 1996.

20 *Πλήρης Οδηγός Χρήσης της C*, M Waite, S Prata, D Martin, Εκδόσεις Γκιούρδα, 2000.

B. Ηλεκτρονική

1 http://www.cs.northwestern.edu/academics/courses/110/

2 http://www.stanford.edu/class/cs106a/

3 http://gd.tuwien.ac.at/languages/c/programming-bbrown/

4 http://gd.tuwien.ac.at/languages/c/programming-bbrown/

5 http://www.csd.uoc.gr/~hy150/EssentialC.pdf

6 http://www.cuj.com/

7 http://pw1.netcom.com/~tjensen/ptr/pointers.htm

8 http://www.acm.uiuc.edu/webmonkeys/book/c_guide/