2.1 Τελεστές και προτεραιότητα τελεστών

Μέχρι τώρα έχουμε χρησιμοποιήσει τους τελεστές: '+' (που χρησιμοποιείται για την πράξη της πρόσθεσης), '-' (που χρησιμοποιείται για την πράξη της αφαίρεσης), '*' (που χρησιμοποιείται για την πράξη του πολλαπλασιασμού), '/' (που χρησιμοποιείται για την πράξη της διαίρεσης), '%' (που χρησιμοποιείται για την εύρεση του υπολοίπου από ακέραια διαίρεση), '++' (που χρησιμοποιείται για την αύξηση κατά μία μονάδα) και '--' (που χρησιμοποιείται για την ελάτωση κατά μία μονάδα). Έχουμε επίσης χρησιμοποιήσει το σύμβολο '=' που έχει το ρόλο της εκχώρησης (ανάθεσης) τιμής σε μεταβλητή.

Στα παραδείγματα του προηγούμενου κεφαλαίου είχαμε χρησιμοποιήσει εκφράσεις (πάντοτε στα δεξιά του συμβόλου εκχώρησης) που είτε ήταν πολύ απλές (είχαν μόνο ένα τελεστή), είτε χρησιμοποιούσαν παρενθέσεις για να προσδιορίσουν τη σειρά εκτέλεσης. Η χρήση παρενθέσεων, όταν χρησιμοποιούμε εκφράσεις με περισσότερους από δύο τελεστές, δεν είναι υποχρεωτική, καθώς η σειρά εκτέλεσης καθορίζεται από τους κανόνες προτεραιότητας που διέπουν τους τελεστές. Η χρήση παρενθέσεων είναι επιβεβλημένη μόνο στην περίπτωση που η σειρά εκτέλεσης που προκύπτει από την προτεραιότητα των τελεστών είναι διαφορετική από αυτή που εμείς (οι προγραμματιστές) επιθυμούμε. Η προτεραιότητα των τελεστών, όταν υπάρχουν περισσότεροι από δύο σε μία έκφραση, παρουσιάζεται στον επόμενο πίνακα (όσο πιο επάνω τόσο μεγαλύτερη η προτεραιότητά του):

τελεστής	**λειτουργία**	**Πλήθος τελεστέων**	**Θέσεις τελεστέων**
(...)	Ομαδοποίηση υποεκφράσεων	1	ΕΣ
++	Αύξηση κατά ένα	1	ΑΡ ή ΔΕ
--	Μείωση κατά ένα	1	ΑΡ ή ΔΕ
*	πολλαπλασιασμός	2	ΑΡ και ΔΕ
/	διαίρεση	2	ΑΡ και ΔΕ
%	Υπόλοιπο ακέραιας διαίρεσης	2	ΑΡ και ΔΕ
+	Πρόσθεση	2	ΑΡ και ΔΕ
-	Αφαίρεση	2	ΑΡ και ΔΕ
=	Εκχώρηση / ανάθεση τιμής	2	ΑΡ και ΔΕ
Επεξηγήσεις συμβόλων: **ΑΡ**=αριστερά, **ΔΕ**=δεξιά, **ΕΣ**=εσωτερικά			

Έστω το επόμενο απόσπασμα προγράμματος στη γλώσσα προγραμματισμού C. Ποια τιμή θα έχει η μεταβλητή a μετά την εκτέλεσή του;

```
int a, b;
b=3;
a=b+2*5;
```

Η τιμή του a θα είναι 13. Αυτό θα συμβεί γιατί (σύμφωνα με τον πίνακα προτεραιοτήτων) πρώτα θα εκτελεστεί ο πολλαπλασιασμός (μεταξύ των τελεστέων του '*', δηλαδή θα γίνει πολλαπλασιασμός του 2 με το 5) και στη συνέχεια θα εκτελεσθεί η πρόσθεση (μεταξύ των τελεστέων του '+', δηλαδή μεταξύ της τιμής που περιέχει η μεταβλητή b και του αποτελέσματος του πολλαπλασιασμού). Αν θέλουμε να αλλάξουμε τη σειρά ώστε, πρώτα να γίνει το άθροισμα της τιμής της μεταβλητής b και του 2 και στη συνέχεια το αποτέλεσμα της πρόσθεσης να πολλαπλασιαθεί με το 5 θα πρέπει να χρησιμοποιήσουμε παρενθέσεις. Αυτό ακριβώς γίνεται στο επόμενο απόσπασμα:

```
int a, b;
b=3;
a=(b+2)*5;
```

Η τιμή του a μετά την εκτέλεσή του παραπάνω αποσπάσματος θα είναι 25.

2.2 Περισσότερο σύνθετες συντομεύσεις εντολών

Στο προηγούμενο κεφάλαιο είδαμε ότι η εντολή `a++;` είναι ισοδύναμη με την εντολή `a=a+1;` (Για όσους γνωρίζουν Pascal, αμφότερες οι προηγούμες δύο εντολές αντιστοιχούν στην εντολή `a:=a+1;` που χρησιμοποιούμε στην Pascal.)

Επίσης, στο προηγούμενο κεφάλαιο είδαμε ότι η εντολή `b--;` είναι ισοδύναμη με την εντολή `b=b-1;` (Για όσους γνωρίζουν Pascal, αμφότερες οι προηγούμες δύο εντολές αντιστοιχούν στην εντολή `b:=b-1;` που χρησιμοποιούμε στην Pascal.)

Πρέπει όμως να διευκρινήσουμε ότι η εντολή `b--;` δεν είναι πάντα ισοδύναμη με την εντολή `--b;` (και η τιμή που θα έχει τελικά η μεταβλητή `b` ε-

ξαρτάται από την έκφραση στην οποία εντάσεται). Ανάλογη διαφορά υφίσταται μεταξύ της πράξης `b++;` και `++b;` Στη συνέχεια εξετάζουμε κατάλληλα παραδείγματα (αποσπάσματα προγραμμάτων) για την κατανόηση της προ-αύξησης (`++b;`) και της μετά-αύξησης (`b++;`):

```
a=5;
b=7;
a=b++;
```

Η εκτέλεση του αποσπάσματος έχει σαν αποτέλεσμα, η μεταβλητή a να αποκτά την τιμή 7 και η μεταβλητή b να αποκτά την τιμή 8. Αυτό συμβαίνει γιατί η τελευταία εντολή του παραπάνω αποσπάσματος δίνει οδηγία να γίνει πρώτα εκχώρηση της τιμής της μεταβλητής b στην μεταβλητή a και στη συνέχεια να γίνει η αύξηση της μεταβλητής b.

```
a=5;
b=7;
a=++b;
```

Η εκτέλεση του αποσπάσματος έχει σαν αποτέλεσμα, η μεταβλητή a να αποκτά την τιμή 8 και η μεταβλητή b να αποκτά επίσης την τιμή 8. Αυτό συμβαίνει γιατί η τελευταία εντολή του παραπάνω αποσπάσματος δίνει οδηγία να γίνει πρώτα αύξηση της τιμής της μεταβλητής b και στη συνέχεια να γίνει η εκχώρηση της τιμής της μεταβλητής b στην μεταβλητή a.

Ένας τελεστής που θα χρησιμοποιήσουμε για πρώτη φορά είναι ο τελεστής '+=' (που αποτελείται από δύο διαδοχικούς χαρακτήρες, το '+' και το '=' χωρίς να παρεμβάλεται κενός χρακτήρας). Θα εξηγήσουμε τη σημασία του νέου τελεστή, δίνοντας μία εντολή που τον χρησιμοποιεί και στη συνέχεια θα παραθέσουμε μια άλλη ισοδύναμη εντολή την οποία αντιλαμβανόμαστε καλά (χωρίς πιθανότητα παρερμηνείας). Η εντολή `a+=5;` που χρησιμοποιεί το νέο τελεστή είναι ισοδύναμη με την εντολή `a=a+5;` (και είναι επίσης ισοδύναμη με την εντολή `a:=a+5;` που χρησιμοποιούν όσοι γνωρίζουν Pascal). Στη συνέχεια εξετάζουμε ορισμένα παραδείγματα (αποσπάσματα προγραμμάτων) που χρησιμοποιούν το νέο τελεστή και προσομοιώνουμε σταδιακά την εκτέλεσή τους:

```
a=7; b=2;
a+=5;
```

είναι ισοδύναμο με `a=a+5;` δηλαδή `a=7+5;` δηλαδή `a=12;`

```
a=7; b=2;
a+=b*3;
```

είναι ισοδύναμο με `a=a+b*3;` δηλαδή `a=a+2*3;` δηλαδή `a=a+6;` δηλαδή `a=13;`

```
a=7; b=2;
a*=b+3;
```

εδώ γεννιέται το ερώτημα αν είναι ισοδύναμο με `a=a*b+3;` ή είναι ισοδύναμο με `a=a*(b+3);`

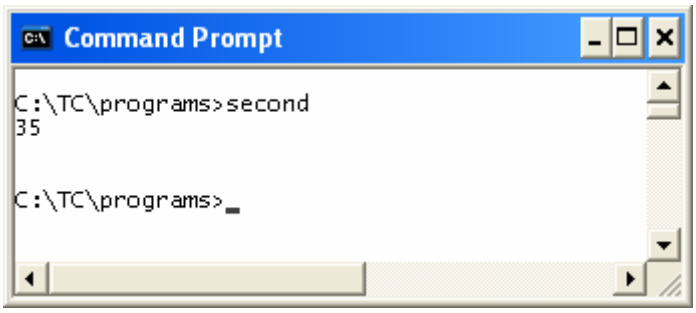
Αν ισχύει η πρώτη εκδοχή τότε σταδιακά έχουμε `a=a*2+3;` δηλαδή `a=7*2+3;` δηλαδή `a=14+3;` δηλαδή `a=17;`

Αν ισχύει η δεύτερη εκδοχή τότε σταδιακά έχουμε `a=a*(b+3);` δηλαδή `a=a*(2+3);` δηλαδή `a=a*5;` δηλαδή `a=7*5;` δηλαδή `a=35;`

Προτείνουμε στον αναγνώστη να δοκιμάσει το παραπάνω απόσπασμα στο δικό του υπολογιστή και στο δικό του compiler (αυτόν που χρησιμοποιεί). Για όσους δεν θα το δοκιμάσουν, το δοκιμάσαμε εμείς με το επόμενο (έστω second.c) πρόγραμμα:

```
1.  #include <stdio.h>
2.  main() {
3.     int a, b;
4.     a=7; b=2;
5.     a*=b+3;
6.     printf("%d\n",a);
7.     getchar();
8.  }
```

Η εκτέλεσή του εμφανίζεται στην επόμενη εικόνα:



Επομένως, η σωστή εκδοχή είναι η δεύτερη. Γενικά ισχύει:

```
a <τελεστής> = <έκφραση>;
```

είναι ισοδύναμο με

```
a = a <τελεστής> (<έκφραση>);
```

2.3 Επιπλέον βασικοί τύποι δεδομένων και εναλλακτικές μορφές αναπαράστασης αριθμητικών σταθερών

Στο προηγούμενο κεφάλαιο χρησιμοποιήσαμε μόνο μεταβλητές ακεραίων (int), πραγματικών (float) και χαρακτήρων (char). Στην παρούσα ενότητα θα εξετάσουμε επιπλέον τους ακεραίους μεγάλου μήκους (long int), τους ακεραίους μικρού μήκους (short int), τους μη προσημασμένους ακεραίους (unsigned int), τους πραγματικούς αριθμούς διπλής ακρίβειας (double) και θα δούμε εναλλακτικούς τρόπους να αναπαραστήσουμε αριθμητικές (ακέραιες και πραγματικές) σταθερές. Για το σκοπό αυτό δίνουμε κατάληλα παραδείγματα και στη συνέχεια τα ερμηνεύουμε. Στο πρώτο μας παράδειγμα ασχολούμαστε με μεταβλητές ακεραίων μεγάλου μήκους και με εναλλακτικές αναπαραστάσεις αριθμητικών σταθερών:

```
1.   main () {
2.      long int big;
3.
4.      int i=052;
5.      int j=0x14;
6.
7.      float unleaded_price=1.075;
8.      float super_price=0.1185e1;
```

```
9.
10.     big=65L;
11.  }
```

Μέχρι τώρα, σε όλα τα προηγούμενα προγράμματα, χρησιμοποιούσαμε δύο εντολές για να δηλώσουμε μία μεταβλητή και να τις αποδώσουμε κάποια τιμή. Αυτό δεν είναι υποχρεωτικό. Στη γλώσσα C έχουμε τη δυνατότητα να αναθέσουμε τιμή σε μια μεταβλητή κατά τη δήλωσή της. Τον κανόνα αυτό (δήλωση μεταβλητής με ανάθεση αρχικής τιμής) ακολουθούμε σε τέσσερις από τις μεταβλητές του παραπάνω προγράμματος (στις i, j, unleaded_price και super_price) αλλά δεν τον ακολουθούμε σε μία (στην big).

Στη γραμμή δύο (2) βλέπουμε πως δηλώνουμε μία μεταβλητή ακεραίου αριθμού μεγάλου μήκους (long int). Οι μεταβλητές τύπου long int χρησιμοποιούν περισσότερα (συνήθως διπλάσια) bytes από μία μεταβλητή τύπου int. Ετσι, οι μεταβλητές τύπου long int έχουν μεγαλύτερο εύρος τιμών. Στις πρώτες γεννιές μεταγλωτιστών οι μεταβλητές int χρησιμοποιούσαν δύο bytes (και είχαν εύρος τιμών από -32768 μέχρι 32767, δηλαδή 65536 διακριτές τιμές) και οι μεταβλητές long int χρησιμοποιούσαν τέσσερα bytes (και είχαν εύρος τιμών από -2147483648 μέχρι 2147483647, δηλαδή 4294967296 διακριτές τιμές). Σήμερα, το πλήθος σε bytes που χρησιμοποιούν οι μεταβλητές int και οι μεταβλητές long int εξαρτάται από τον μεταγλωτιστή (compiler) που έχουμε επιλέξει και αυτό μπορούμε να το πληροφορηθούμε από τα εγχειρίδια (manual) που συνοδεύουν το μεταγλωτιστή. Πάντοτε όμως ισχύει ότι: το μέγεθος ενός μικρού ακεραίου (short int) είναι μικρότερο ή ίσο από το μέγεθος ενός κανονικού ακεραίου (int) και ότι το μέγεθος ενός κανονικού ακεραίου (int) είναι μικρότερο ή ίσο από το μέγεθος ενός μεγάλου ακεραίου (long int).

Στή γραμμή τέσσερα (4) βλέπουμε πως αναπαριστούμε μια ακέραια αριθμητική σταθερά στο οκταδικό σύστημα. Μία ακέραια αριθμητική σταθερά στο οκταδικό σύστημα αρχίζει πάντα με το ψηφίο μηδέν (0) το οποίο ακολουθούν άλλα οκταδικά ψηφία (από 0 μέχρι 7). Η οκταδική τιμή 052 ισοδυναμεί με 5 * 8 + 2, δηλαδή με το δεκαδικό αριθμό 42.

Στή γραμμή πέντε (5) βλέπουμε πως αναπαριστούμε μια ακέραια αριθμητική σταθερά στο δεκαεξαδικό σύστημα. Μία ακέραια αριθμητική σταθερά στο δεκαεξαδικό σύστημα αρχίζει πάντα με τους διαδοχικούς χαρακτήρες 0x (ψηφίο μηδέν και λατινικό χαρακτήρα x) το οποίο ακολουθούν άλλα δε-

καεξαδικά ψηφία (από 0 μέχρι F). Η δεκαεξαδικά τιμή 0x14 ισοδυναμεί με 1 * 16 + 4, δηλαδή με το δεκαδικό αριθμό 20.

Στη γραμμή επτά (7) η σταθερά πραγματικού αριθμού που χρησιμοποιήσαμε έχει τη μορφή που είδαμε και στο προηγούμενο κεφάλαιο. Δηλαδή ακολουθεί τον κανόνα που χρησιμοποιούμε στην καθημερινότητα. Στην γραμμή οκτώ (8) χρησιμοποιούμε την επιστημονική γραφή πραγματικών αριθμών. Η σταθερά 0.1185e1 σημαίνει τον πραγματικό αριθμό 0.1185 πολλαπλασιασμένο με το 10 στην πρώτη δύναμη. Δηλαδή σημαίνει το 1.185 που χρησιμοποιούμε στην καθημερινότητα.

Στη γραμμή δέκα (10) χρησιμοποιήσαμε την αριθμητική σταθερά 65L για να αποδώσουμε αρχική τιμή στη μεταβλητή big. Ο χαρακτήρας 'L' μετά το δεκαδικό αριθμό 65 επιβάλει τη μετατροπή του σε ακέραιο μεγάλου μήκους. Στο παράδειγμα αυτό θα μπορούσαμε να παραλείψουμε το 'L' γιατί η αυτόματη μετατροπή τύπου (που κάνει η γλώσσα C, και θα δούμε σε επόμενη ενότητα του παρόντος κεφαλαίου) είναι επαρκής.

Με βάση τις εξηγήσεις που δώσαμε, συνοψίζουμε τις τιμές που θα έχουν οι μεταβλητές στο τέλος του προγράμματος.

Μεταβλητή	**Τιμή**
big	65
i	42
j	20
unleaded_price	1.075
super_price	1.185

Στο επόμενο παράδειγμα θα ασχοληθούμε μεταβλητές που έχουν τύπους ακεραίου μικρού μήκους (short int), μη προσημασμένου ακεραίου (unsigned int) και πραγματικού αριθμού διπλής ακρίβειας (double):

```
1.   #include <stdio.h>
2.   main () {
3.      short int small;
4.      unsigned int uint;
5.      double area;
6.      float radius;
7.      double dpi=3.14159265368;
8.      float fpi=3.14159265368;
9.
```

```
10.     radius=5.45;
11.
12.     area=radius * radius * dpi;
13.     printf("To emvado (area) tou kyklou einai:
        %12.8f\n",area);
14.
15.     area=radius * radius * fpi;
16.     printf("To emvado (area) tou kyklou einai:
        %12.8f\n",area);
17.
18.     printf("\n");
19.     printf("H timi tis metavlitis dpi einai:
        %14.11f\n",dpi);
20.     printf("H timi tis metavlitis fpi einai:
        %14.11f\n",fpi);
21.  }
```

Στην γραμμή τρία (3) δηλώνουμε μία μεταβλητή με τύπο ακεραίου μικρού μήκους. Την μεταβλητή αυτή δεν την αξιοποιούμε στη συνέχεια του προγράμματος. Αυτό δεν απαγορεύεται αλλά είναι σπατάλη και πρέπει να αποφεύγεται. Εδώ χρησιμοποιήθηκε για να υποδείξει πως δηλώνουμε μεταβλητές ακεραίου μικρού μήκους.

Στην γραμμή τέσσερα (4) δηλώνουμε μία μεταβλητή με τύπο μη προσημασμένου ακεραίου. Τη μεταβλητή αυτή (όπως και την προηγούμενη) δεν την αξιοποιούμε στη συνέχεια του προγράμματος.

Στη γραμμή πέντε (5) δηλώνουμε μία μεταβλητή με τύπο πραγματικού αριθμού διπλής ακρίβειας. Στη μεταβλητή αυτή θα αποθηκεύσουμε το εμβαδό του κύκλου.

Στη γραμμή έξι (6) δηλώνουμε μία μεταβλητή με τύπο πραγματικού αριθμού απλής ακρίβειας. Στη μεταβλητή αυτή θα αποθηκεύσουμε την ακτίνα του κύκλου.

Στη γραμμή επτά (7) δηλώνουμε μία μεταβλητή με τύπο πραγματικού αριθμού διπλής ακρίβειας και την αρχικοποιούμε (της αναθέτουμε τιμή) ταυτοχρόνως. Πρόκειται για τη γεωμετρική σταθερά π που χρησιμοποιούμε για την εύρεση του εμβαδού, της διαμέτρου και άλλων μεγεθών του κύκλου.

Στη γραμμή οκτώ (8) κάνουμε το ανάλογο με τη γραμμή επτά (7) με τη διαφορά ότι η μεταβλητή τώρα είναι απλής ακριβείας.

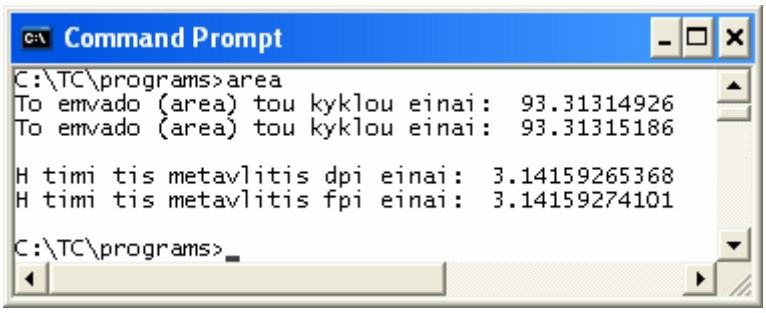
Στη γραμμή δέκα (10) δίνουμε (αναθέτουμε / εκχωρούμε) τιμή στη μεταβλητή radius. Η μεταβλητή radius, όπως προκύπτει και από το όνομά της, αντιπροσωπεύει (περιέχει) την ακτίνα του κύκλου για τον οποίο θα υπολογίσουμε το εμβαδό.

Στη γραμμή δώδεκα (12) υπολογίζουμε το εμβαδό του κύκλου και αναθέτουμε το αποτέλεσμα του υπολογισμού στην μεταβλητή area.

Στη γραμμή δεκατρία (13) εμφανίζουμε (στην οθόνη του H/Y) το εμβαδό του κύκλου. Το καινούργια στοιχείο είναι ότι έχουμε αντικαταστήσει τον προσδιοριστή τύπου δεδομένων `%g` (που χρησιμοποιήσαμε σε προηγούμενες χρήσεις της εντολής printf για την εμφάνιση πραγματικών αριθμών) με το `%12.8f`. Στην πραγματικότητα τόσο το `%g`, όσο και το `%12.8f` δεν είναι απλώς προσδιοριστές τύπου δεδομένων (της μεταβλητής ή έκφρασης από την οποία θα αντληθεί ή θα προκύψει η τιμή που θα εμφανίσει η printf) αλλά είναι επιπλέον προσδιοριστές της μορφής εξόδου του αποτελέσματος (format). Ο δεύτερος προσδιοριστής (`%12.8f`) που χρησιμοποιούμε στο παράδειγμά μας, ορίζει ότι θα εμφανισθεί ένας πραγματικός αριθμός και για την εμφάνισή του θα χρησιμοποιηθούν οκτώ (8) δεκαδικά ψηφία και δώδεκα (12) συνολικά χαρακτήρες (για ολόκληρο τον αριθμό: ακέραιο μέρος, διαχωριστικό χαρακτήρα για το δεκαδικό μέρος και δεκαδικό μέρος).

Στο ζεύγος εντολών των γραμμών δεκαπέντε (15) και δεκαέξι (16) επαναλαμβάνουμε ότι και στο ζεύγος εντολών των γραμμών δώδεκα (12) και δεκατρία (13), με μοναδική διαφορά ότι στο τελευταίο ζεύγος εντολών χρησιμοποιούμε την μεταβλητή dpi αντί της μεταβλητής fpi που χρησιμοποιήσαμε στο προηγούμενο ζεύγος εντολών. Θα περίμενε κανείς τα αποτελέσματα από τα δύο ζευγάρια εντολών να είναι τα ίδια, καθώς στο μοναδικό σημείο που διαφέρουν έχει χρησιμοποιηθεί η ίδια αριθμητική σταθερά (αμφότερες οι μεταβλητές dpi και fpi έχουν λάβει την ίδια τιμή). Στην πραγματικότητα όμως, όπως προκύπτει και από την εκτέλεση του προγράμματος (βλέπε εικόνα παρακάτω), τα δύο αποτελέσματα διαφέρουν από το πέμπτο δεκαδικό ψηφίο και μετά. Αυτό συμβαίνει γιατί ο πρώτος υπολογισμός χρησιμοποίησε μια μεταβλητή (dpi) με διπλή ακρίβεια και ικανή να συγκρατήσει την ακριβή τιμή που της εκχωρήσαμε. Αντίθετα ο δεύτερος υπολογισμός χρησιμοποίησε μια μεταβλητή (fpi) με απλή ακρίβεια και ανίκανη να συγκρατήσει με απόλυτη την ακριβεια την τιμή που της εκχωρήσαμε. Αυτή η αδυναμία γίνεται αντιληπτή με τις εντολές των γραμμών δεκαεννέα (19) και (20)

και τα αντίστοιχα αποτελέσματα που αυτές εμφανίζουν. Ακολουθεί το πρόγραμμα (έστω area.c) κατά την εκτέλεσή του:



2.4 Περισσότερα για το format (προσδιοριστή μορφής) παρουσίασης δεδομένων που χρησιμοποιούμε στην printf

Μέχρι τώρα έχουμε χρησιμοποιήσει τους προσδιοριστές μορφής: `%d`, `%g`, `%c`, `%12.8f` και `%14.11f`, στις εντολές printf που έχουμε χρησιμοποιήσει. Οι προσδιοριστές ακολουθούν μία συγκεκριμένη δομή συνταξής τους. Αυτή είναι:

```
%{flag}{minimum width}{precision spec}
{long size spec}conversion
```

Υποχρεωτικά είναι μόνο το πρώτο (`%`) και το τελευταίο (`conversion`) συνθετικό της δομής του προσδιοριστή μορφής. Οι πιο γνωστές τιμές που χρησιμοποιούμε για το `conversion` είναι:

conversion	**Τύπος δεδομένων μεταβλητής ή έκφρασης από την οποία θα αντληθεί η τιμή και τρόπος που θα εμφανισθεί η τιμή**
`d`	προσημασμένη δεκαδική μετατροπή ακεραίου (int)
`u`	μη προσημασμένη δεκαδική μετατροπή μη προσημασμένου ακεραίου (unsigned)
`o`	μη προσημασμένη οκταδική μετατροπή μη προσημασμένου ακεραίου (unsigned)
`x`	μη προσημασμένη δεκαεξαδική μετατροπή μη προσημασμένου ακεραίου (unsigned)
`c`	μετατροπή απλού χαρακτήρα

s	μετατροπή αλφαριθμητικής ακολουθίας (string)
f	προσημασμένη δεκαδική μετατροπή πραγματικού αριθμού (floating point)
e	προσημασμένη δεκαδική μετατροπή πραγματικού αριθμού (floating point)
g	προσημασμένη δεκαδική μετατροπή πραγματικού αριθμού (floating point)
p	pointer (βλέπε επόμενα κεφάλαια, κυρίως κεφάλαιο 7)

Τα στοιχεία `minimum width` και `precision spec` τα χρησιμοποιούμε για περισσότερο ακριβή καθορισμό της μορφής εμφάνισης. Στο παράδειγμα χρήσης του `%14.11f` το `14` έχει τη θέση του `minimum width` (πόσους χαρακτήρες, κατ' ελάχιστο, θα καταλάβει ο αριθμός) και το `.11` έχει τη θέση του `precision spec` (πόσα δεκαδικά ψηφία θα εμφανισθούν). Το στοιχείο `long size spec` χρησιμοποιείται όταν θέλουμε να υποδείξουμε ότι θα αντλήσουμε την πληροφορία που θα εμφανίσουμε από μία μεταβλητή μεγάλου μήκους. Για παράδειγμα για να εμφανίσουμε τα δεδομένα μιας μεταβλητής τύπου `long int` θα χρησιμοποιήσουμε `%ld` που χρησιμοποιεί το `l` στη θέση του `long size spec` και το `d` στη θέση του `conversion`. Το στοιχείο `flag` θα το εξετάσουμε αργότερα.

Στη συνέχεια θα επεκτείνουμε το πρόγραμμα που παρουσιάσαμε στην αρχή της προηγούμενης ενότητας προκειμένου να εμφανίζει τις τιμές που έχουν οι μεταβλητές του. Στην νέα του μορφή, το πηγαίο (έστω third.c) πρόγραμμα είναι:

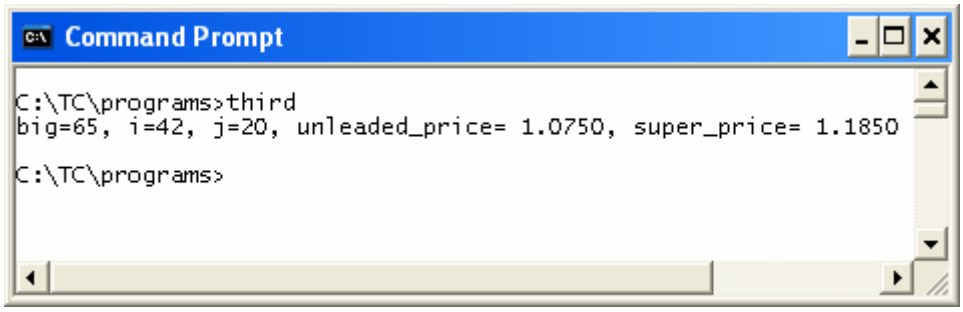
```
1.   #include <stdio.h>
2.      main () {
3.      long int big;
4.
5.      int i=052;
6.      int j=0x14;
7.
8.      float unleaded_price=1.075;
9.      float super_price=0.1185e1;
10.
11.     big=65L;
12.
```

```
13.     printf("big=%ld, i=%d, j=%d, unleaded_price=%7.4f,
        super_price=%7.4f", big, i, j, unleaded_price,
        super_price);
14.
15.     getchar();
16.  }
```

Για να μην υπάρχει παρανόηση, στο παραπάνω πρόγραμμα, η γραμμή 13 καταλαμβάνει μόνο μία γραμμή στο πηγαίο αρχείο (παρότι παραπάνω εμφανίζεται να καταλαμβάνει χώρο τριών γραμμών). Η διευκρίνηση αυτή δε σημαίνει ότι υπάρχει περιορισμός που επιβάλει ότι μία εντολή της γλώσσας C πρέπει να καταλαμβάνει υποχρεωτικά μία γραμμή του πηγαίου προγράμματος. Η γλώσσα προγραμματισμού C επιτρέπει την στοιχειοθέτηση μίας εντολής σε περισσότερες από μία γραμμές του πηγαίου προγράμματος. Επίσης η γλώσσα προγραμματισμού C επιτρέπει την στοιχειοθέτηση περισσοτέρων από μία εντολών στην ίδια γραμμή του πηγαίου προγράμματος (αρκεί να τις διαχωρίζει με semicolon). Εδώ η διευκρίνιση, σχετικά με τη γραμμή 13, έγινε για να υποδείξει (επεξηγήσει) το συμβολισμό. Δηλαδή όταν χρησιμοποιούμε έναν αριθμό γραμμής τότε υπονοούμε ότι η εντολή (έστω και αν στο βιβλίο εμφανίζεται σε περισσότερες από μία γραμμές) καταλαμβάνει μία γραμμή στο πηγαίο πρόγραμμα.

Από την εντολή printf της γραμμής 13, γίνεται αντιληπτό ότι στο format (πρώτο όρισμα) της εντολής μπορούμε να χρησιμοποιήσουμε περισσότερους από ένα προσδιοριστές μορφής. Βέβαια, για να μην έχουμε απρόβλεπτα αποτελέσματα, πρέπει το format (πρώτο όρισμα) να το ακολουθούν τόσα επιπλέον ορίσματα, όσοι και οι προσδιοριστές μορφής του format.

Η εκτέλεση του προγράμματος έχει το αποτέλεσμα που εμφανίζεται στην επόμενη εικόνα:



2.5 Ειδικές ακολουθίες χαρακτήρων που χρησιμοποιούμε στην printf αλλά και στις αλφαριθμητικές ακολουθίες

Σχεδόν σε όλα τα προηγούμενα προγράμματα, στις εντολές printf χρησιμοποιούσαμε την ειδική ακολουθία χαρακτήρων '\n', για την οποία έχουμε εξηγήσει ότι προκαλεί τη δημιουργία μιας αλλαγής (προώθησης) γραμμής στην έξοδο των αποτελεσμάτων. Υπάρχουν αρκετές ειδικές ακολουθίες χαρακτήρων που μπορούμε να χρησιμοποιήσουμε στην printf (και όχι μόνο σε αυτή). Στην πραγματικότητα παρότι χρησιμοποιούμε (συνήθως) δύο χαρακτήρες για να ορίσουμε μία ειδική ακολουθία χαρακτήρων, η επίδραση που έχει αυτή στην έξοδο των αποτελεσμάτων είναι να παράγει έναν μόνο ειδικό χαρακτήρα. Χρησιμοποιούμε τη φράση ειδικό χαρακτήρα καθώς πρόκειται για τους χαρακτήρες που έχουν τακτικούς αριθμούς από μηδέν (0) μέχρι τριανταένα (31), δεν έχουν συγκεκριμένο σύμβολο το οποίο εμφανίζουν (όπως για παράδειγμα ο χαρακτήρας με τακτικό αριθμό 65 εμφανίζει το κεφαλαίο λατινικό άλφα) και η επίδρασή τους σε μία απλή συσκευή εξόδου (είτε κλασσικό σειριακό τερματικό χωρίς μνήμη και επεξεργαστή – μόνο οθόνη με πληκτρολόγιο, είτε εκτυπωτής ακίδων) έχει σα συνέπεια τη συνολική μορφοποίηση της εξόδου πάρα την εμφάνιση μεμονωμένου χαρακτήρα. Για παράδειγμα ο χαρακτήρας form feed (με τακτικό αριθμό 12) προωθεί το χαρτί (σε ένα εκτυπωτή ακίδων) μία σελίδα (δηλαδή αφήνει κενή την υπόλοιπη σελίδα και αρχίζει να γράφει πάλι στην επόμενη σελίδα). Ένα υποσύνολο αυτών των χαρακτήρων τους κωδικοποιεί (αφού δεν έχουν ορισμένο / συγκεκριμένο σύμβολο) η γλώσσα C με τις ειδικές ακολουθίες χαρακτήρων που αποτελούνται από δύο σύμβολα (δύο χαρακτήρες), εκ' των οποίων ο πρώτος είναι πάντα ο χαρακτήρας back slash (ανάποδη κοψιά, '\'). Η σημασία αυτών για την έξοδο από μια εντολή printf, συνοψίζεται στον επόμενο πίνακα:

σύμβολο	**σημασία**
\n	χαρακτήρας αλλαγής γραμμής – new line
\t	χαρακτήρας στηλοθέτησης – tab
\b	χαρακτήρας οπισθοδρόμησης – backspace
\r	επαναφορά στο αριστερό περιθώριο της σελίδας – carriage return
\f	αλλαγή σελίδας – form feed
\'	μονό εισαγωγικό – single quote

\"	διπλό εισαγωγικό – double quote
\\	Χαρακτήρας ανάποδη κοψιά – backslash

Βλέπουμε ότι εκτός από την κωδικοποίηση ειδικών χαρακτήρων, κωδικοποιούνται και άλλοι περισσότερο συνηθισμένοι χαρακτήρες (μονό εισαγωγικό, διπλό εισαγωγικό και backslash). Αυτό γίνεται καθώς οι τελευταίοι (παρότι είναι απλοί χαρακτήρες) χρησιμοποιούνται με ειδική σημασία στο format (πρώτο όρισμα) της εντολής printf . Με τον τρόπο ανερούμε την ειδική τους σημασία και τους χρησιμοποιούμε με την αυθεντική τους σημασία.

Τα επόμενα δύο αποσπάσματα προγράμματος θα διευκολύνουν την κατανόηση των ειδικών ακολουθιών χαρακτήρων σε σχέση με την εντολή printf. Πρώτο παράδειγμα:

```
printf("Καλημέρα\nΣπύρο\n\nΚαλημέρα Ελλη\n");
printf("Καλημέρα \"Ελλη\"\n");
```

θα εμφανίσει:

```
Καλημέρα
Σπύρο

Καλημέρα Ελλη
Καλημέρα "Ελλη"
```

Δεύτερο παράδειγμα:

```
printf("ΑΜ\tΒαθμός\n");
printf("021352\t8,0\n");
printf("000132\t4,5\n");
printf("095123\t7,5\n");
printf("\n");
printf("Όνομα\t\tΒαθμός\n");
printf("Παναγιώτης\t6,5\n");
printf("Γεράσιμος\t8,0\n");
printf("Κωνσταντίνος\t7,5\n");
```

θα εμφανίσει:

```
ΑΜ      Βαθμός
021352  8,0
000132  4,5
095123  7,5

Όνομα           Βαθμός
Παναγιώτης      6,5
Γεράσιμος       8,0
Κωνσταντίνος    7,5
```

Ειδικά για το χαρακτήρα στηλοθέτησης, διευκρινίζουμε ότι η συμπεριφορά του είναι να συμπληρώνει τα απαιτούμενα κενά, ώστε τα στοιχεία να εμφανίζονται μετά τον επόμενο στηλοθέτη. Η εξ' ορισμού θέση των στηλοθετών είναι κάθε οκτώ χαρακτήρες. Για το λόγο αυτό μετά τη λέξη `Όνομα` βάλαμε δύο ειδικές ακολουθίες χαρακτήρων `\t`. Διαφορετικά η λέξη `Βαθμός` θα εμφανιζόταν από τον έννατο ($9^{ο}$) χαρακτήρα και μετά (δηλαδή πάνω από το 'η' του `Παναγιώτης`) αντί από τον δέκατο-έβδομο ($17^{ο}$) χαρακτήρα και μετά (δηλαδή πάνω από το '6' του `6,5`) που είναι το επιθυμητό.

2.6 Σχόλια

Συχνά θέλουμε να ενσωματώσουμε, σε ένα πηγαίο πρόγραμμα, πληροφορίες που ενώ δεν επηρεάζουν τον τρόπο λειτουργίας του, είτε βοηθούν στην αναγνωσιμότητά του από ανθρώπους (όχι από τη μηχανή – δηλαδή τον compiler) είτε ενημερώνουν για το σκοπό, το χρόνο και τους ανθρώπους (προγραμματιστές) που το συνέταξαν. Οτιδήποτε θέλουμε να γράψουμε και να μη γίνει αντικείμενο επεξεργασίας του compiler, θα πρέπει να το βάλουμε σε σχόλια. Στη γλώσσα C τα σχόλια ξεκινούν με το ζεύγος των χαρακτήρων slash ('/') και αστερίσκος ('*'). Τα σχόλια μπορούν να εκτείνονται σε μία γραμμή αλλά μπορούν επίσης να επεκταθούν σε πολλές γραμμές. Για τους γνώστες της γλώσσας Pascal, τα /* ..... */ που χρησιμοποιούμε όταν γράφουμε σχόλια στη γλώσσα C είναι ισοδύναμα με τα (* ..... *) και τα { ..... } που χρησιμοποιούμε στην Pascal. Για παράδειγμα το πρώτο μας πρόγραμμα (που παρουσιάσθηκε στο προηγούμενο κεφάλαιο θα μπορούσε να εμπλουτισθεί με σχόλια, ως εξής:

```
1.  /* programmer Nikitas N. Karanikolas */
```

```
2.  /* date of completion: March 24, 2009 */
3.  #include <stdio.h>
4.  main () {
5.     printf("Hello World !!!\n");
6.     /* remove next command when program runs under DOS */
7.     getchar();
8.  }
```

2.7 Μετατροπές τύπων κατά την εκτέλεση μιας έκφρασης και κατά την εκχώρηση τιμής έκφρασης σε μεταβλητή

Η γλώσσα προγραμματισμού C δεν είναι αυστηρή, όσο είναι άλλες γλώσσες υψηλού επιπέδου, και επιτρέπει την ανάμειξη διαφορετικών τύπων στις εκφράσεις, ανάμειξη τύπων κατά την ανάθεση / εκχώρηση τιμών σε μεταβλητές, και άλλες αναμείξεις δεδομένων. Βέβαια για όλα υπάρχουν κανόνες (σημασιολογικοί) που προβλέπουν τι γίνεται σε κάθε περίπτωση ανάμειξης. Δηλαδή δεν είναι εξαρτημένη η συμπεριφορά από την κάθε υλοποίηση της γλώσσας C αλλά είναι εκ των προτέρων καθορισμένη. Στη συνέχεια, με τη βοήθεια ενός προγράμματος (έστω fourth.c) θα αναδείξουμε τους σημαντικότερους από αυτούς τους κανόνες:

```
1.   #include <stdio.h>
2.   main () {
3.      float r=5.0;
4.      int a,b,d;
5.      float c1, c2, e;
6.
7.      a=17;
8.      b=5;
9.      c1=a/b;       /* 1 */
10.     c2=a/r;       /* 2 */
11.     d=a/r;        /* 3 */
12.     e=a/(float)b; /* 4 */
13.
14.     printf ("c1=%g, c2=%g, d=%d, e=%g\n", c1, c2, d, e);
15.  }
```

Ας σχολιάσουμε τις παραπάνω τέσσερις αριθμημένες (με τη βοήθεια σχολίων) εντολές:

Ο τελεστής slash (‘/’) όταν εφαρμόζεται σε τελεστέους με τύπους ακεραίου αριθμού (int, long int, κλπ) συνεπαγεται (όπως αναφέραμε και σε ανάλογο

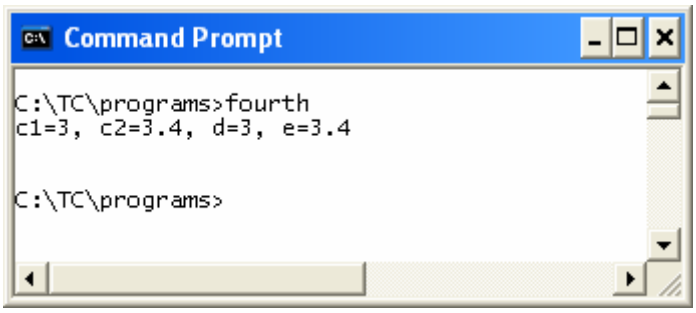
παράδειγμα του προηγούμενου κεφαλαίου) την εκτέλεση ακέραιας διαίρεσης (για όσους γνωρίζουν Pascal ισοδυναμεί με τον τελεστή div της Pascal). Με βάση τον κανόνα αυτό η έκφραση στο δεξί μέλος της εντολής που φέρει αρίθμηση ένα (1) θα έχει την ακέραια τιμή 3. Η εκχώρηση μιας ακέραιας τιμής σε μια μεταβλητή με τύπο πραγματικού αριθμού έχει σα συνέπεια την εκχώρηση του πραγματικού ισοδύναμου της ακέραιας τιμής στη μεταβλητή. Έτσι η μεταβλητή c1 θα αποκτήσει την πραγματική αριθμητική τιμή 3.0.

Όταν έστω και ένας από τους τελεστέους του τελεστή slash (‘/’) είναι πραγματικός αριθμός τότε έχουμε πραγματική διαίρεση. Επομένως το αποτέλεσμα της έκφρασης στο δεξί μέλος της εντολής που φέρει αρίθμηση δύο (2) θα έχει την πραγματική αριθμητική τιμή 3.4. Η εκχώρηση πραγματικής τιμής σε μεταβλητή με τύπο πραγματικού αριθμού είναι άμεση και δεν επιβάλει καμμία (υπονοούμενη) μετατροπή. Επομένως η μεταβλητή c2 θα αποκτήσει την πραγματική αριθμητική τιμή 3.4.

Η εκχώρηση μιας πραγματικής αριθμητικής τιμής σε μια ακέραια μεταβλητή συνεπάγεται τον υποβιβασμό (μετατροπή) της πραγματικής αριθμητικής τιμής σε ακέραια αριμητική τιμή, πριν την εκχώρηση. Επομένως, παρότι το αποτέλεσμα της έκφρασης στο δεξί μέλος της εντολής που φέρει αρίθμηση τρία (3) θα έχει την πραγματική αριθμητική τιμή 3.4, θα γίνει υποβιβασμός του (με αποκοπή των δεκαδικών του ψηφίων, όχι με στρογγυλοποίηση) στην ακέραια τιμή 3 και στη συνέχεια θα πραγματοποιηθεί η εκχώρηση. Ετσι η μεταβλητή d θα αποκτήσει την ακέραια αριθμητική τιμή 3. Προσέξτε ότι μιλάμε για αποκοπή και όχι για στρογγυλοποίηση.

Κάνοντας αλλαγή τύπου (καλουπώνοντας, cast) έναν τελεστέο ώστε να μετασχηματισθεί σε πραγματικό αριθμό τότε η πράξη slash (‘/’) στην οποία συμμετέχει ο μετασχηματισμένος αριθμός, γίνεται πραγματική διαίρεση. Αυτό συμβαίνει στο δεξί μέλος της εντολής που φέρει αρίθμηση τέσσερα (4). Άρα και το απότέλεσμα της πραγματικής πλέον διαίρεσης, δηλαδή η πραγματική τιμή 3.4, εκχωρείται στην πραγματική μεταβλητή e.

Όλα τα παραπάνω επαληθεύονται στην επόμενη εικόνα, όπου βλέπουμε το εκτελέσιμο του παραπάνω προγράμματος σε πλήρη εξέλιξη:



2.8 cast (αλλαγή τύπου)

Ένα παράδειγμα επιβολής (από τον προγραμματιστή) αλλαγής τύπου (cast) είδαμε στην τέταρτη αριθμημένη εντολή της προηγούμενης ενότητας. Η αλλαγή τύπου δεν είναι πάντα μια εφική λειτουργία και πρέπει να γίνεται με μεγάλη προσοχή. Οι περισσότερο συνηθισμένες αλλαγές τύπου είναι: μεταξύ χαρακτήρων (char) και ακεραίων (int), μεταξύ ακεραίων (int) και μεγάλων ακεραίων (long int), μεταξύ ακεραίων (int) και πραγματικών (float) και μεταξύ πραγματικών (float) και πραγματικών διπλής ακριβείας (double). Για αυτές τις μετατροπές, οι μεταγλωτιστές της C γνωρίζουν πολύ καλά τι πρέπει να κάνουν. Αλλές αλλαγές τύπου δεν είναι πάντα λογικές. Για παράδειγμα η μετατροπή (αλλαγή τύπου) από χαρακτήρα σε πραγματικό αριθμό μοιάζει να μην έχει φυσική εξήγηση. Ακόμη περισσότερο ακατανόητη μοιάζει μια μετατροπή από σύνθετο τύπο που αρίζει ο χρήστης (struct, που θα εξετάσουμε σε επόμενο κεφάλαιο) σε ακέραιο (int).

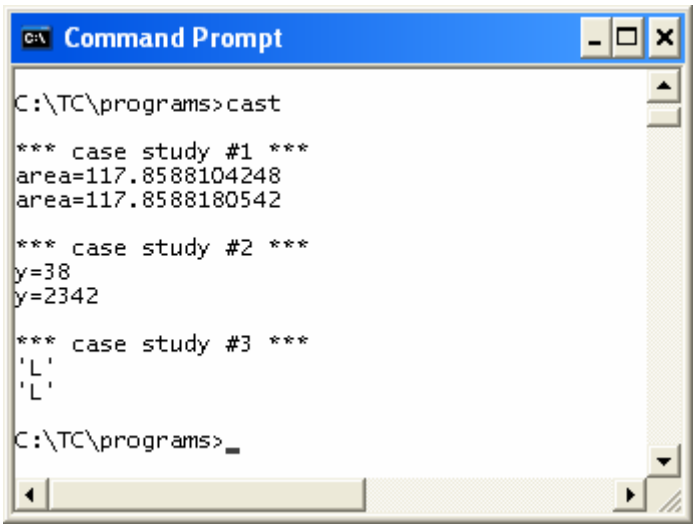
Ο κανόνας που μπορούμε να δώσουμε με ασφάλεια, για την αλλαγή (cast) τύπου είναι: μεταβολή από μικρότερο σε μεγαλύτερο τύπο γίνεται χωρίς καμμία απώλεια πληροφορίας και μεταβολή από μεγαλύτερο σε μικρότερο τύπο μπορεί να συνεπάγεται την απώλεια πληροφορίας. Για να είναι πλήρης ο τελευταίος κανόνας θα πρέπει να ορίσουμε ποια είναι η διάταξη των τύπων (ποιος είναι μεγαλύτερος και ποιος μικρότερος τύπος). Μικρότερος από όλους είναι ο τύπος char, μετά (ανεβαίνοντας προς μεγαλύτερους σε διάταξη τύπους) είναι ο τύπος int, μετά ο τύπος long int, μετά είναι ο τύπος float και ανώτερος όλων είναι ο double.

Όπως προαναφέραμε, η τέταρτη αριθμημένη εντολή του προγράμματος της προηγούμενης ενότητας περιελάμβανε μία αλλαγή τύπου. Αυτή η αλλαγή τύπου ανήκε στην κατηγορία μεταβολής από μικρότερο σε μεγαλύτερο τύπο

δεδομένων. Στη συνέχεια, με τη βοήθεια ένος προγράμματος (έστω cast.c) θα μελετήσουμε τη μεταβολή από μεγαλύτερο σε μικρότερο τύπο δεδομένων:

```
1.   #include <stdio.h>
2.   main () {
3.      int x, y;
4.      double pi;
5.      float area, radius;
6.
7.      printf("\n*** case study #1 ***\n");
8.      pi=3.14159265368;
9.      radius=6.125;
10.     area=radius*radius*pi;
11.     printf("area=%12.10f\n",area);
12.     area=radius*radius*(float)pi;
13.     printf("area=%12.10f\n",area);
14.
15.     printf("\n*** case study #2 ***\n");
16.     x=2341;
17.     y=(char)x + 1;
18.     printf("y=%d\n", y);
19.     y=x + 1;
20.     printf("y=%d\n", y);
21.
22.     printf("\n*** case study #3 ***\n");
23.     x=76;
24.     printf("\'%c\'\n", (char) x);
25.     printf("\'%c\'\n", x);
26.  }
```

Η έξοδος του ιδίου προγράμματος εμφανίζεται στην επόμενη εικόνα:



Στην γραμμή δέκα (10) υπολογίζουμε το εμβαδό ενός κύκλου και στην γραμμή ένδεκα (11) το εμφανίζουμε. Το ίδιο επαναλαμβάνουμε στις γραμμές δώδεκα (12) και δεκατρία (13). Στη δεύτερη περίπτωση υποβιβάζουμε την τιμή της μεταβλητής pi από double σε float, πριν τη χρησιμοποιήσουμε στην έκφραση (στο δεξί μέλος της εκχώρησης, της γραμμής 12). Καθώς η έκφραση αυτή δεν έχει άλλο στοιχείο με τύπο double, όλα τα στάδια υπολογισμού της έκφρασης μεταπίπτουν σε πράξη float. Η float πλέον τιμή του pi περιέχει μια προσέγγιση της αρχικά εκχωρηθείσας τιμής (3.14159265368) και η εκτέλεση της πράξης με την προσεγγιστική τιμή οδηγεί στο ελαφρώς διαφορετικό αποτέλεσμα (διαφορά από το έκτο δεκαδικό ψηφίο, όπως προκύπτει από την έξοδο του προγράμματος κάτω από την επικεφαλίδα “case study #1”).

Το ζεύγος των εντολών στις γραμμές δεκαεπτά (17) και δεκαοκτώ (18) επαναλαμβάνεται στις γραμμές δεκαεννέα (19) και είκοσι (20). Η μοναδική διαφορά ευρίσκεται μεταξύ των εντολών στη δεκαεπτά και στη δεκαεννέα γραμμή, όπου στην πρώτη εξ’ αυτών γίνεται αλλαγή τύπου σε char στην τιμή της μεταβλητής x. Ο υποβιβασμός του τύπου δημιουργεί εμφανές αποτέλεσμα στην έξοδο του προγράμματος (βλέπε έξοδο κάτω από την επικεφαλίδα “case study #2”). Η εναλλαγή μεταξύ ακεραίων και χαρακτήρων δεν δημιουργεί καταστάσεις σαν και αυτή που μόλις είδαμε, αν οι τιμές του ακεραίου ευρίσκονται εντός των ορίων μηδέν (0) έως διακόσια-πενήντα-πέντε (255). Το προαναφερθέν εύρος αριθμών (0 ... 255) αποτελεί το εύρος

των τακτικών αριθμών των μη προσημασμένων χαρακτήρων (unsigned char) και επιπλέον αποτελεί τον τρόπο που οι χαρακτήρες αναπαρίστανται στη μνήμη του Η/Υ (δηλαδή καταλαμβάνουν ένα byte, 8 bits). Η μετατροπή ενός ακεραίου σε χαρακτήρα, συνεπάγεται την απώλεια του πιο σημαντικού από τα δύο byte που αποτελούν τον ακέραιο. Έτσι στο παράδειγμά μας η μεταβλητή x είχε την επόμενη αναπαράσταση στη μνήμη του Η/Υ:

Μεταβλητή x															
Σημαντικότερο byte								Λιγότερο σημαντικό byte							
0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	1

και μετά την αλλαγή τύπου, τη θέση της στην έκφραση (της γραμμής δεκαεπτά) έλαβε ο επόμενος χαρακτήρας:

μοναδικό byte							
0	0	1	0	0	1	0	1

στη συνέχεια η πράξη της πρόσθεσης έγινε πράξη μεταξύ χαρακτήρων (bytes):

0	0	1	0	0	1	0	1
				+			
0	0	0	0	0	0	0	1
				=			
0	0	1	0	0	1	1	0

Με την εκχώρηση (ανάθεση τιμής στη μεταβλητή y) έγινε μετατροπή (από τον compiler, όχι τον προγραμματιστή) προς τα πάνω (από char σε int). Το αποτέλεσμα της εκχώρησης είναι οτι η μεταβλητή y απέκτησε την επόμενη μορφή στη μνήμη του Η/Υ:

Μεταβλητή y															
Σημαντικότερο byte								Λιγότερο σημαντικό byte							
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0

Πρόκειται δηλαδή για την εσωτερική αναπαράσταση του ακέραιου αριθμού τριάντα-οκτώ (38) που εμφανίστηκε με την εντολή printf της γραμμής δεκαοκτώ.

Στην περίπτωση της εντολής δεκαεννέα έγινε πράξη ακεραίων και προέκυψε η έξοδος της τιμής δύο χιλιάδες τριακόσια σαράντα δύο (2342).

Το τελευταίο μέρος του προγράμματος (γραμμές είκοσι δύο μέχρι είκοσι πέντε) προστέθηκε προκειμένου να αναδείξει ότι η αλλαγή τύπου της τιμής μιας μεταβλητής (ή μιας έκφρασης) από int σε char γίνεται, είτε το απαιτήσει ο χρήστης (όπως στην γραμμή είκοσι τέσσερα) είτε αυτόματα (από τον compiler) όταν ο προσδιοριστής μορφής εξόδου (στοιχείο conversion στο πρώτο όρισμα της printf) είναι c (δηλαδή char). Από το παράδειγμα προκύπτει ότι ο τακτικός αριθμός που αντιστοιχεί στο χαρακτήρα 'L' είναι το νούμερο 76.

Προφανώς για κάποιους αναγνώστες θα έχει προκύψει το ερώτημα: ποια είναι η προτεραιότητα του τελεστή αλλαγής τύπου σε σχέση με τους άλλους τελεστές; Η απάντηση είναι ότι η αλλαγή τύπου (cast) προηγείται των αριθμητικών πράξεων. Εξαίρεση αποτελούν οι τελεστές '++' και '--' που προηγούνται της αλλαγής τύπου.

Οι υπόλοιπες παράγραφοι της παρούσας ενότητας, μπορούν να παραλειφθούν στην πρώτη ανάγνωση του βιβλίου. Μπορείτε να επανέλθετε, όταν θα έχετε γίνει περισσότερο έμπειροι προγραμματιστές.

Αναφέραμε παραπάνω ότι το οι τακτικοί αριθμοί των μη προσημασμένων χαρακτήρων (unsigned char) είναι από 0 μέχρι 255 (0 ... 255). Ποιο είναι το εύρος των τακτικών αριθμών των απλών (δηλαδή προσημασμένων) χαρακτήρων; Το εύρος των απλών χαρακτήρων είναι από –128 μέχρι 127 (-128 ... 127). Και ποια είναι η αντιστοιχία των τακτικών αριθμών των μη προσημασμένων χαρακτήρων με τους τακτικούς αριθμούς των προσημασμένων χαρακτήρων; Σε αυτό το ερώτημα δίνει απάντηση το επόμενο πρόγραμμα (έστω charord.c):

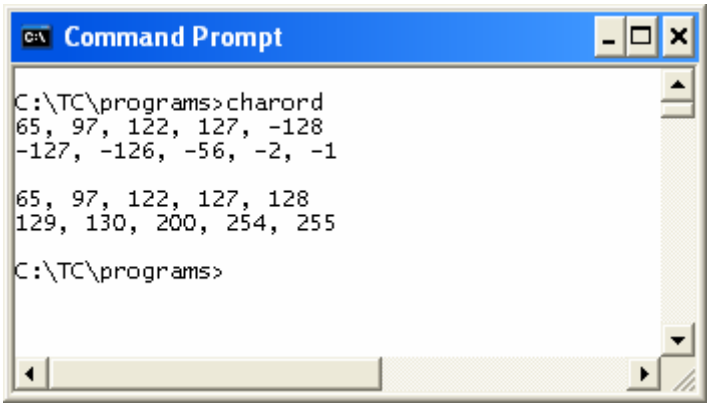
```
#include <stdio.h>

main() {
   char c1, c2, c3, c4, c5, c6, c7, c8, c9, c10;
   c1='A';
   c2='a';
   c3='z';
```

```
    c4=c3+5;
    c5=c4+1;
    c6=c5+1;
    c7=130;
    c8=200;
    c9=254;
    c10=255;

    printf("%d, %d, %d, %d, %d\n", c1, c2, c3, c4, c5);
    printf("%d, %d, %d, %d, %d\n", c6, c7, c8, c9, c10);
    printf("\n");
    printf("%d, %d, %d, %d, %d\n", (unsigned char) c1,
        (unsigned char) c2, (unsigned char) c3,
        (unsigned char) c4, (unsigned char) c5);
    printf("%d, %d, %d, %d, %d\n", (unsigned char) c6,
        (unsigned char) c7, (unsigned char) c8,
        (unsigned char) c9, (unsigned char) c10);
}
```

Η επόμενη εικόνα περιέχει την έξοδό του:



Θα πρέπει να διευκρινήσουμε ότι η εσωτερική αναπαράσταση των χαρακτήρων στη μνήμη δεν έχει καμία διαφοροποίηση είτε η μεταβλητή στην οποία φυλάγεται ο χαρακτήρας έχει απλό τύπο χαρακτήρα (char), είτε έχει μη προσημασμένο τύπο χαρακτήρα (unsigned char). Δηλαδή τα ίδια bit (δυαδικά ψηφία) ενεργοποιούνται στη μνήμη που καταλαμβάνει ο χαρακτήρας ανεξάρτητα αν έχει δηλωθεί ως char ή ως unsigned char. Ο τύπος unsigned char είναι απλά πιο βολικός σε χειρισμούς (εκφράσεις) που αναμειγνύουν πράξεις μεταξύ ακεραίων και χαρακτήρων. Όταν δηλώνουμε με-

ταβλητές απλού χαρακτήρα (char) και αναμειγνύουμε πράξεις μεταξύ ακεραίων και χαρακτήρων θα πρέπει να πληροφορούμε το μεταγλωττιστή πως να τους χειριστεί. Αυτό γίνεται με αλλαγή τύπου (cast), όπως στις δύο τελευταίες εντολές του προηγούμενου προγράμματος.

2.9 Είσοδος δεδομένων με scanf

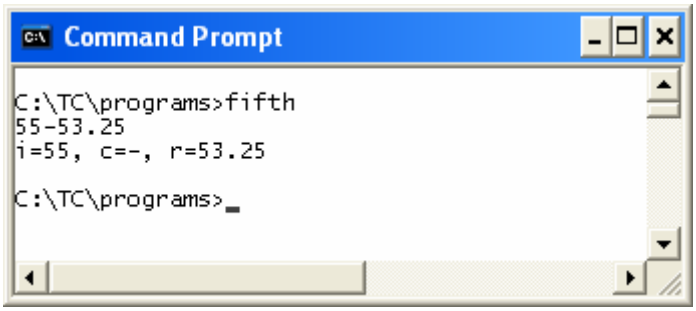
Είναι γνωστό ότι τα προγράμματα εκτός από επεξεργασία και έξοδο αποτελεσμάτων, περιλαμβάνουν και ένα στάδιο εισόδου δεδομένων, δηλαδή πληροφοριών επί των οποίων θα γίνει η επεξεργασία. Μέχρι τώρα το βήμα αυτό το προσπερνάγαμε και στη θέση του χρησιμοποιούσαμε εντολές εκχώρησης (ανάθεσης) για να ορίσουμε τις τιμές των μεταβλητών που στη συνέχεια συμμετείχαν στην επεξεργασία. Τώρα μπορούμε να δούμε εντολές εισόδου για να ολοκληρώσουμε τα στάδια του βασικότερου κύκλου προγραμματισμού (είσοδος – επεξεργασία – έξοδος). Θα εξετάσουμε την εντολή scanf (που προέρχεται από το scan formatted). Θα ξεκινήσουμε με ένα παράδειγμα (έστω fifth.c) και θα ερμηνεύσουμε στη συνέχεια:

```
1.  #include <stdio.h>
2.  main () {
3.     int i;
4.     float r;
5.     char c;
6.
7.     scanf("%d",&i);
8.     scanf("%c%g",&c,&r);
9.  }
```

Η εντολή scanf (εκτός από το πρώτο όρισμά της που προσδιορίζει τους τύπους των μεταβλητών που θα ακολουθήσουν) δέχεται επιπλέον ορίσματα (μεταβλητές) στις οποίες επιστρέφει τα στοιχεία που διάβασε από το τερματικό (και το πληκτρολόγιο του χειριστή). Οι μεταβλητές αυτές, διαβιβάζονται στην scanf με την by reference τεχνική, προκειμένου να είναι δυνατή η μεταβολή τους από τη συνάρτηση scanf. (Για τους γνώστες της Pascal, μιλαμε για μεταβλητές VAR.) Ομως η γλώσσα C δεν μεταβιβάζει αυτόματα τις διευθύνσεις των by reference μεταβλητών και περιμένει από τον προγραμματιστή να κάνει αυτή την ενέργεια. Αυτό γίνεται με την προσθήκη του τελεστή διεύθυνσης μεταβλητής (δηλαδή του συμβόλου ‘&’) πριν από κάθε μεταβλητή με απλό τύπο. (Όλοι οι τύποι δεδομένων που έχουμε μέχρι

τώρα συναντήσει είναι απλοί τύποι.) Όπως προαναφέραμε, το πρώτο όρισμα της εντολής είναι το format ανάγνωσης και προσδιορίζει τον τρόπο που θα ερμηνευθούν τα δεδομένα (που εισαγάγει ο χρήστης από το τερματικό) και κατ' επέκταση ποιοι είναι οι τύποι των μεταβλητών που ακολουθούν. Οι προσδιοριστές μορφής που είδαμε με την printf χρησιμοποιούνται και στη scanf. Έτσι στην εντολή της γραμμής επτά (7) του παραπάνω προγράμματος ζητάμε να διαβάσει η εντολή ένα ακέραιο αριθμό. Στην εντολή της γραμμής οκτώ (8) ζητάμε να διαβάσει πρώτα ένα χαρακτήρα και μετά ένα πραγματικό αριθμό.

Στη συνέχεια βλέπουμε το εκτελέσιμο του προηγούμενου πηγαίου προγράμματος, σε πλήρη εξέλιξη:



Όπως βλέπουμε ο χειριστής πληκτρολόγησε `54-53.25`. Οι μεταβλητές του προγράμματος ενημερώθηκαν από τη scanf και έλαβαν τις εξής τιμές:

μεταβλητή	**τιμή**	**τύπος τιμής**
`i`	54	ακέραια τιμή
`c`	'-'	χαρακτήρα
`r`	53.25	πραγματική τιμή

2.10 Ανανεωμένη προτεραιότητα τελεστών

Με βάση τα στοιχεία που αναπτύξαμε στο παρόν κεφάλαιο ο πίνακας προτεραιότητας τελεστών ανανεώνεται ως εξής:

τελεστής	**λειτουργία**	**Πλήθος τελεστέων**	**Θέσεις τελεστέων**
`( ... )`	Ομαδοποίηση υποεκφράσεων με παρενθέσεις	1	ΕΣ
`++`	Αύξηση κατά ένα	1	ΑΡ ή ΔΕ
`--`	Μείωση κατά ένα	1	ΑΡ ή ΔΕ
`&`	**Διεύθυνση μεταβλητής (κλήση by reference)**	**1**	**ΔΕ**
`(τύπος)`	**Αλλαγή τύπου (cast)**	**1**	**ΔΕ**
`*`	πολλαπλασιασμός	2	ΑΡ και ΔΕ
`/`	διαίρεση	2	ΑΡ και ΔΕ
`%`	Υπόλοιπο ακέραιας διαίρεσης	2	ΑΡ και ΔΕ
`+`	Πρόσθεση	2	ΑΡ και ΔΕ
`-`	Αφαίρεση	2	ΑΡ και ΔΕ
`=`	Εκχώρηση / ανάθεση τιμής	2	ΑΡ και ΔΕ
`*= /= %= += -=`	Πράξη και ανάθεση τιμή	2	ΑΡ και ΔΕ
Επεξηγήσεις συμβόλων: **ΑΡ**=αριστερά, **ΔΕ**=δεξιά, **ΕΣ**=εσωτερικά			