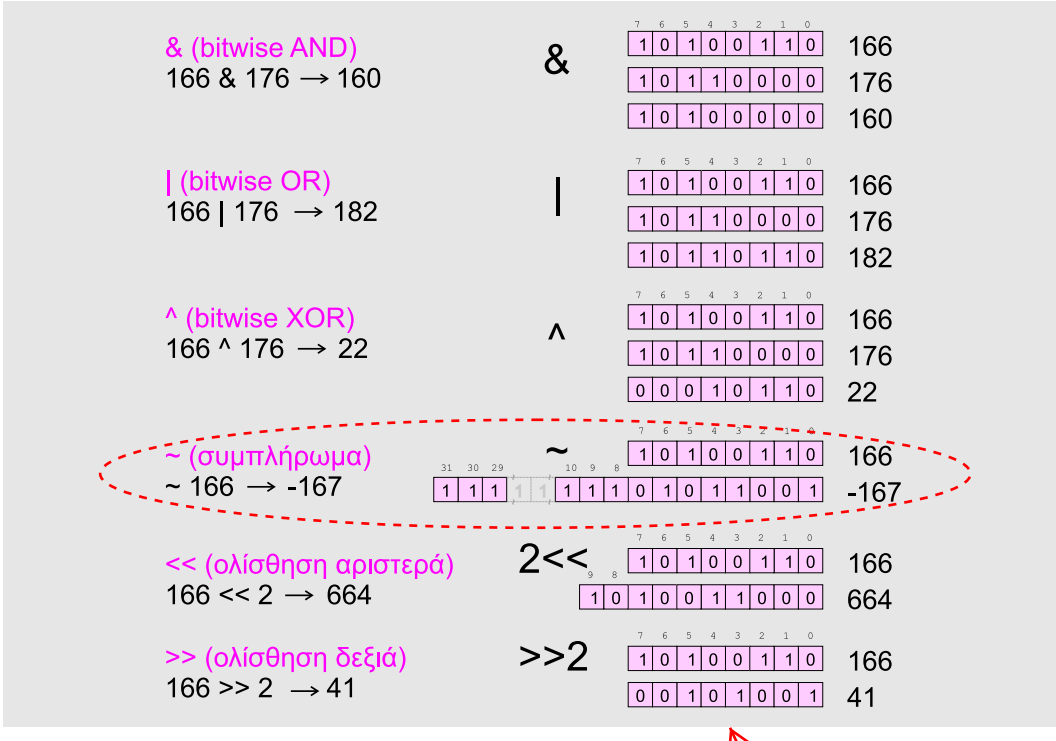


Σχήμα 4.2 *Παραδείγματα χρήσης των δυαδικών (bitwise) τελεστών της C*

Άλλαξε το σχήμα στο σημείο που σημειώνεται

- Οι δυαδικοί τελεστές εφαρμόζονται μόνο σε ακέραια μεγέθη
- Δεν πρέπει να γίνεται σύγχυση των λογικών τελεστών (&&, ||) μ… στές (&, |). Οι δυαδικοί τελεστές εκτελούν αριθμητικές, και όχι λ…
- Προσέξτε την εφαρμογή του τελε… του Σχήματος 4.2. Το αποτέλεσμα της πράξης 166 ^ 176 είνα… με την πράξη XOR στο 22 με τον ίδιο αριθμό, το 176, θα προκ… αρχικός αριθμός το 166 (22 ^ 176 → 166). Αυτή η ιδιότητα είναι χρήσιμη σε περιπτώσεις κωδικοποίησης και θα χρησιμοποιηθεί αργότερα σε αντίστοιχα παραδείγματα.

Προστέθηκε αυτή η παράγραφος

- Ο τελεστής συμπλήρωμα (~) αντιστρέφει όλα τα bit της αναπαράστασης ενός ακέραιου αριθμού. Επομένως και τα μηδενικά bit, που πιθανώς προηγούνται, από 0 γίνονται 1. Αυτό έχει ως αποτέλεσμα να επηρεαστεί και το πρώτο bit της αναπαράστασης (το 31ο) που αφορά στο πρόσημο του αριθμού. Έτσι το συμπλήρωμα ενός θετικού αριθμού είναι ένας αρνητικός αριθμός και αντίστροφα. Οι αρνητικοί αριθμοί αποθηκεύονται εσωτερικά στη μνήμη του Η/Υ με τη μέθοδο «συμπλήρωμα ως προς 2» (two's complement). Αν εφαρμόσουμε τον τελεστή συμπλήρωμα σε έναν ακέραιο Ν, με αποτέλεσμα να αντιστραφούν οι τιμές των επιμέρους bit του, προκύπτει ο αριθμός -(N+1).

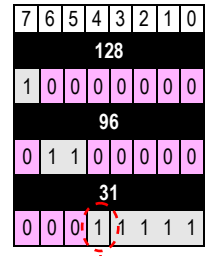
Π4.3 Έστω ότι μια Έγινε από τέσσερα πέντε ή μέτρησης, συνδεδεμένη με έναν Η/Υ, επιστρέφει τόσο την κατάσταση της όσο και τη μέτρηση σε έναν ακέραιο αριθμό ως εξής: Το bit 7 του αριθμού έχει τιμή 1 αν η συσκευή λειτουργεί σωστά. Τα bit 6 και 5 αναφέρουν το είδος της μέτρησης (0-Θερμοκρασία, 1-Πίεση, 2-Υγρασία, 3-Ταχύτητα ανέμου). Τα πέντε πρώτα bit (4, 3, 2, 1, και 0) αποτελούν την τιμή της μέτρησης. Τα υπόλοιπα bit του ακέραιου αριθμού (από το 8 μέχρι και το 31) δεν μεταφέρουν χρήσιμες πληροφορίες. Ο επόμενος κώδικας αναλύει τα bit του αριθμού και εμφανίζει την κατάσταση της συσκευής, το είδος της μέτρησης, και την τιμή της:

4_p4_3.c

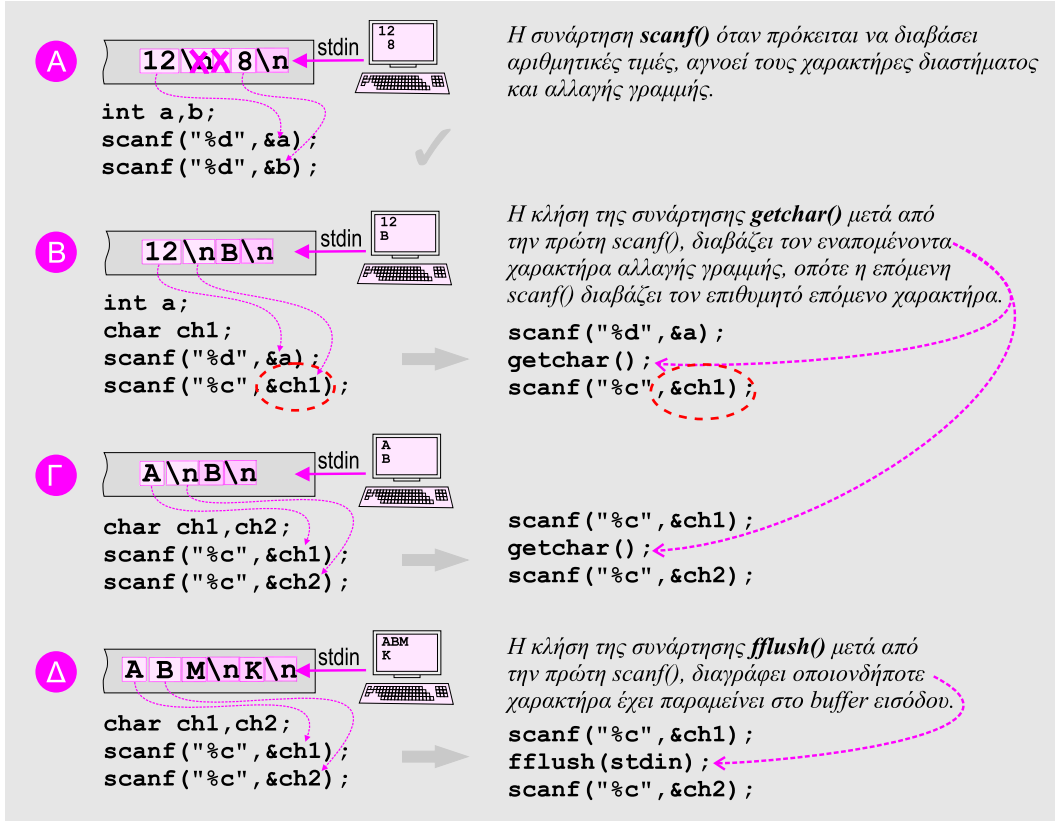
```
#include <stdio.h>

int main(void)
{
   int ar,ok,eidos,timi;
   printf("Δώσε αριθμό από τη συσκευή:");
   scanf("%d",&ar);
   ok=(ar&128)>>7;
   eidos=(ar&96)>>5;
   timi=ar&31;
   if (ok==1)
      printf("Είδος=%d Τιμή=%d\n",eidos,timi);
   else
      printf("Υπάρχει πρόβλημα στη συσκευή\n");
   return 0;
}
```

Αν η συσκευή λειτουργεί σωστά, εμφανίζει τα αποτελέσματα των μετρήσεων, διαφορετικά εμφανίζει κατάλληλο μήνυμα.

Η παράσταση ar&128 απομονώνει το bit 7 του αριθμού, και η εφαρμογή του τελεστή >>7 ολισθαίνει το bit 7 θέσεις δεξιά ώστε να είναι στην αρχή του byte και να αποδώσει τιμή 0 ή 1. Αυτό γίνεται διότι στη δυαδική του αναπαράσταση, ο αριθμός 128 έχει μόνο ενεργοποιημένο (1) το bit 7 (δείτε το διπλανό σχήμα).



Με ανάλογο τρόπο απομονώνονται και τα υπόλοιπα bit που μας ενδιαφέρουν. Ο τελεΈγινε από τέσσερα πέντε μαζί με κάποια σταθερή τιμή για να ένα bit του αριθμού. Π.χ. η παράσταση ar&96 απομονώνει τα bit 6 και 5 του ar, αφού στη δυαδική του αναπαράσταση ο αριθμός 96 έχει ενεργοποιημένα μόνο αυτά τα δύο bit. Η εφαρμογή του τελεστή >> (>>5) ολισθαίνει τα 2 bit κατά 5 θέσεις δεξιά αρχή του byte και να αποδώσουν τιμές 0 (00_2), 1 (01_2), 2 (1Στη θέση 4 το 0 έγινε 1 με γκρι φόντο η παράσταση ar&31 απομονώνει τα πέντε πρώτα bit του a του αναπαράσταση ο αριθμός 31 έχει ενεργοποιημένα μόνο αυτά τα bit.



Σχήμα 5.2 *Τα προβλήματα και οι λύσεις στη χρήση της scanf()*

Β. Εδώ αρχίζουν τα προβλήματα! Στην πρώτη κλήση της, η συνάρτηση scanf() περιμένει να διαβάσει έναν ακέραιο αριθμό και στη δεύτερη έναν χαρακτήρα. Ας υποθέσουμε ότι πληκτρολογούνται ο αριθμός 12, ένα[Έγιναν αλλαγές στο σχήμα στα σημεία που σημειώνονται]ς 'B', και άλλο ένα <enter>. Η πρώτη scanf() θα διαβάσει το 1[η μεταβλητή a, αλλά θα αφήσει το χαρακτήρα αλλαγής γραμμής[. Η επόμενη scanf() θα διαβάσει το χαρακτήρα αλλαγής γραμμ[υτόν στη μεταβλητή ch, αντί του χαρακτήρα 'B', όπως θα ήταν αναμενόμενο. Στον buffer εισόδου θα παραμείνει ο χαρακτήρας 'B' καθώς και ο χαρακτήρας αλλαγής γραμμής (το τελευταίο <enter>).

Η λύση βρίσκεται στην κλήση της getchar() μετά από την πρώτη scanf(). Η getchar() θα διαβάσει και θα απομακρύνει το χαρακτήρα αλλαγής γραμμής από το ρεύμα εισόδου, και η επόμενη scanf() θα διαβάσει κανονικά το χαρακτήρα 'B'.

Γ. Και εδώ γίνεται κάτι παρόμοιο! Στην πρώτη κλήση της η συνάρτηση scanf() περιμένει να διαβάσει ένα χαρακτήρα, όπως και στη δεύτερη. Ας υποθέσουμε ότι πληκτρολογούνται οι χαρακτήρες: 'A', <enter>, 'B', και <enter>. Η πρώτη scanf() θα διαβάσει το

Π7.3 Στη C, στη θέση οποιασδήποτε συνθήκης μπορεί να χρησιμοποιηθεί οποιαδήποτε παράσταση που αποδίδει αριθμητική τιμή. Αν η παράσταση αποδίδει τιμή 0 θεωρείται ψευδής, διαφορετικά θεωρείται αληθής. Αυτό γίνεται σαφές στο επόμενο πρόγραμμα.

7_p7_3.c

```
#include <stdio.h>

int main(void)
{
   int a=5,b=5,c;
   // Ελέγχεται η λογική παράσταση a==b
   if (a==b)
      printf("ίσα\n");
   else
      printf("άνισα\n");
   // Ελέγχεται η αριθμητική παράσταση a-b
   if (a-b)
      printf("άνισα\n");
   else
      printf("ίσα\n");
   // Ελέγχεται η αριθμητική παράσταση c=a
   if (c=a)
      printf("NAI\n");
   else
      printf("OXI\n");
   // Ελέγχεται η παράσταση printf("Μυτιλήνη\n")
   if (printf("Μυτιλήνη\n"))
      printf("NAI\n");
   else
      printf("OXI\n");
   // Ελέγχεται η παράσταση b
   if (b)
      printf("Τέλος\n");
   else
      printf("Αρχή\n");
   return 0;
}
```

Κανονική λογική παράσταση. Το αποτέλεσμά της είναι αληθές (5==5), οπότε εμφανίζεται η λέξη *ίσα*.

Αριθμητική παράσταση. Το αποτέλεσμά της είναι 0 (5-5), οπότε θεωρείται ψευδής. Επομένως εμφανίζεται πάλι η λέξη *ίσα*.

Η παράσταση … γκριση (==) … μεταβλητή **c**. … αποτέλεσμα 5, οπότε θεωρείται αληθής. Επομένως εμφανίζεται η λέξη *NAI*.

Διαγράφηκαν τα ερωτηματικά στο τέλος δύο σχολίων

Η παράσταση **printf("Μυτιλήνη\n")** εμφανίζει τη λέξη *Μυτιλήνη* και επιστρέφει την τιμή 8 (το πλήθος χαρακτήρων που εμφάνισε), οπότε θεωρείται αληθής. Επομένως εμφανίζεται η λέξη *NAI*.

Η παράσταση **b** θεωρείται αληθής διότι η τιμή της μεταβλητής **b** είναι διάφορη του μηδενός (5). Επομένως θα εμφανιστεί η λέξη *Τέλος*.

☞ Στο διπλανό πλαίσιο εμφανίζονται τα αποτελέσματα του προγράμματος.

☞ Ο έλεγχος μη λογικών παραστάσεων είναι συνήθης πρακτική σε προγράμματα της C.

```
ίσα
ίσα
NAI
Μυτιλήνη
NAI
Τέλος
```

8.20 Με δεδομένη την εξίσωση $\mathbf{ax^2-bx+3=0}$, να γραφεί ένα πρόγραμμα που να εμφανίζει τις ρίζες της εξίσωσης για όλους τους συνδυασμούς των **a** και **b** στην περιοχή ακέραιων τιμών [-10 .. +10]. Να μη συμπεριλαμβάνονται τα ζεύγη τιμών για τους συντελεστές **a** και **b**, όταν έστω και ένας από τους δύο συντελεστές έχει τιμή 0. Σε κάθε περίπτωση, να γίνεται έλεγχος για πραγματικές ρίζες. ★★★

8.21 Ο αλγόριθμος του Ευκλείδη για τον υπολογισμό του μέγιστου κοινού διαιρέτη δύο θετικών ακεραίων **μ** και **ν** περιγράφεται ως εξής:

Βήμα 1: Θέσε στο **μ** τον μεγαλύτερο και στο **ν** τον μικρότερο αριθμό.

Βήμα 2: Στην περίπτωση που αρχικά το **ν** είναι 0, ο Μ.Κ.Δ είναι το **μ**.

Βήμα 3: Διαίρεσε το **μ** με το **ν** και ονόμασε **ρ** το υπόλοιπο.

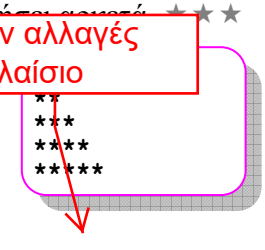
Βήμα 4: Αν **ρ=0**, ο υπολογισμός σταματά και το αποτέλεσμα είναι **ν**.

Βήμα 5: Αλλιώς, θέσε την τιμή του **μ** ίση με **ν** και την τιμή του **ν** ίση με **ρ** και πήγαινε στο βήμα 3.

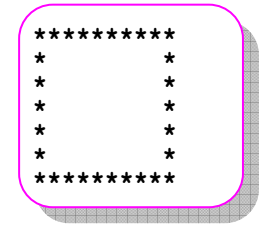
Να γραφεί ένα πρόγραμμα το οποίο να ζητάει δύο ακέραιους αριθμούς, και κατόπιν να υπολογίζει και να εμφανίζει τον μέγιστο κοινό διαιρέτη τους. ★★

8.22 Να γραφεί ένα πρόγραμμα το οποίο να μετράει τις λέξεις που δίνονται από το πληκτρολόγιο. Λέξη θεωρείται οποιαδήποτε ομάδα χαρακτήρων που δεν περιλαμβάνει κενά διαστήματα και χαρακτήρες αλλαγής γραμμής. Η διαδικασία θα σταματάει όταν διαβαστεί ο χαρακτήρας **EOF**. Το πρόγραμμα θα πρέπει να εμφανίζει το πλήθος των λέξεων που πληκτρολογήθηκαν. Μελετήστε τον κώδικα του προγράμματος **8_countlines.c**, το οποίο θα σας βοηθήσει σε αυτό. ★★★

8.23 Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει τ
τέλεσμα του πλαισίου που βρίσκεται στα δεξιά. Το πρόγραμμα θα πρέπει να χρησιμοποιεί επαναληπτική διαδικασία. ★★



8.24 Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει στην οθόνη ένα ορθογώνιο με αστεράκια, όπως βλέπετε στο πλαίσιο στα δεξιά. Το μέγεθος του ορθογωνίου θα πρέπει να είναι μεταβλητό. Το ύψος του σε γραμμές και το πλάτος του σε αστερίσκους θα δίνεται από τον χρήστη. Π.χ. το ορθογώνιο του παραδείγματος έχει ύψος 7 γραμμών και πλάτος 10 χαρακτήρων. ★★★



8.25 Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει στην οθόνη ένα δένδρο, όπως φαίνεται στο διπλανό πλαίσιο. Το ύψος του δένδρου (σε γραμμές) θα πρέπει να είναι μεταβλητό, και θα δίνεται από τον χρήστη. Για παράδειγμα, το δένδρο του πλαισίου έχει ύψος 6. Το μικρότερο επιτρεπτό ύψος του δένδρου να είναι 4 γραμμές και το μεγαλύτερο 20. ★★★

```
   *
  ***
 *****
*******
*********
   *
```

8.26 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★★

```
#include <stdio.h>
int main(void)
{
   int a=-10,b=5;
   for (printf("ΑΡΧΗ\n");a+b;printf("Λέσβος\n"))
   {
      a++;
      printf("ΑΙΓΑΙΟ\n");
   }
   printf("ΤΕΛΟΣ\n");
   return 0;
}
```

Έγιναν αλλαγές στο πλαίσιο

8.27 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★★

```
#include <stdio.h>
int main(void)
{
   int a=2,b=10,c;
   c=(b>a)+2;
   a=++b,b+c;
   for (a=c;a<10;a=a+3);
   printf("%d %d %d\n",a,b,c);
   return 0;
}
```

8.28 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★★

```
#include <stdio.h>
int main(void)
{
   int i,j,k;
   k=10;
   for (i=1;i<=10;i=i+2)
   {
      k=k-3;
      for (j=1;j<=4;++j)
         k=k+j;
```

9.11 Ο ΦΠΑ ενός προϊόντος μπορεί να ανήκει στις παρακάτω κατηγορίες:

Κατηγορία	Συντελεστής ΦΠΑ
1	0.00
2	0.06
3	0.13
4	0.19

Να γραφεί ένα πρόγραμμα το οποίο θα ζητάει να πληκτρολογήσουμε το πλήθος, την τιμή μονάδας, και την κατηγορία ΦΠΑ για 10 προϊόντα. Το πρόγραμμα θα πρέπει να εμφανίζει το συνολικό κόστος των προϊόντων, καθώς και το σύνολο του ΦΠΑ για όλα τα προϊόντα που αγοράσαμε. Ο υπολογισμός του ΦΠΑ πρέπει να υλοποιείται από μια συνάρτηση στην οποία θα μεταβιβάζεται το συνολικό ποσό ανά προϊόν, καθώς και η κατηγορία ΦΠΑ στην οποία ανήκει. ★★★

9.12 Να γραφεί μια συνάρτηση η οποία θα δέχεται δύο παραμέτρους: έναν ακέραιο αριθμό και έναν χαρακτήρα. Η συνάρτηση θα πρέπει να εμφανίζει στην οθόνη, σε μία γραμμή, τον χαρακτήρα της δεύτερης παραμέτρου τόσες φορές όσες είναι η τιμή της πρώτης παραμέτρου. Στο τέλος της ακολουθίας των χαρακτήρων θα πρέπει να υπάρχει αλλαγή γραμμής. ★★

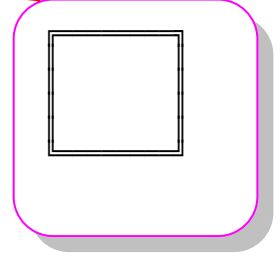
9.13 Να γραφεί ένα πρόγραμμα το οποίο θα χρησιμοποιεί τη συνάρτηση που γράψατε στην Άσκηση 9.12 και θα εμφανίζει το αποτέλεσμα που φαίνεται στο διπλανό πλαίσιο. ★★

```
**********
aaaaaaa
----------
```

9.14 Να γραφεί ένα πρόγραμμα το οποίο θα χ
νάρτηση που γράψατε στην Άσκηση 9.12 και θα εμφανίζει όλους τους κεφαλαίους χαρακτήρες του λατινικού αλφαβήτου (A~Z), από 20 φορές τον καθένα. ★★

Έγιναν αλλαγές στο πλαίσιο

9.15 Να γραφεί μια συνάρτηση η οποία θα δέχεται ως ορίσματα δύο ακέραιους αριθμούς. Η συνάρτηση πρέπει να εμφανίζει στην οθόνη ένα ορθογώνιο πλαίσιο με διπλή γραμμή. Το ύψος του ορθογωνίου σε γραμμές, και το πλάτος του σε χαρακτήρες, θα καθορίζεται από τις τιμές της πρώτης και της δεύτερης παραμέτρου αντίστοιχα. Το πλαίσιο με τη διπλή γραμμή θα πρέπει να υλοποιηθεί με χαρακτήρες που θα εντοπίσετε στον πίνακα ASCII. Γράψτε επίσης ένα πρόγραμμα το οποίο θα καλεί τη συνάρτηση ώστε να εμφανίζει ένα ορθογώνιο πλαίσιο διπλής γραμμής, ύψους 6 γραμμών και πλάτους 10 χαρακτήρων. ★★★



```
int add(int x, int y)
{
   int ss;
   ss=x+y;
   return ss;
}
```

Οι τυπικές παράμετροι παίζουν επίσης τον ρόλο τοπικών μεταβλητών.

Δήλωση της τοπικής μεταβλητής **ss**.

Η συνάρτηση **add()** επιστρέφει ως τιμή το περιεχόμενο της μεταβλητής **ss**.

Οι τυπικές παράμετροι παίζουν και αυτές τον ρόλο των τοπικών μεταβλητών, με τη διαφορά ότι σε αυτές αντιγράφονται οι τιμές των ορισμάτων με τα οποία καλείται η συνάρτηση.

Αν δεν γίνει κατανοητή η χρήση και η εμβέλεια των τυπικών παραμέτρων και των τοπικών μεταβλητών, είναι πολύ εύκολο να παρεισφρήσουν λάθη στα προγράμματά μας. Στη συνέχεια βλέπετε το πρ[illegible]γεμάτο ενδεχόμενα λάθη.

Προστέθηκαν τα εισαγωγικά (") που απουσίαζαν

```
#include <stdio.h>
int add(int x, int y);
int main(void)
{
   int sum;
   float a,b;
   printf("Δωσε δύο αριθμούς:");
   scanf("%f %f",&a,&b);
   sum=add(a,b);
   printf("ss=%d\n",ss);
   printf("Αθροισμα=%d\n",sum);
   return 0;
}

int add(int x, int y)
{
   int ss;
   ss=a+b;
   return ss;
}
```

Τα ορίσματα με τα οποία πρέπει να κληθεί η **add()** πρέπει να είναι τύπου **int**, και όχι **float**. Αυτό μπορεί να έχει συνέπεια την απώλεια πληροφοριών.

Η συνάρτηση **main()** αγνοεί την ύπαρξη της **ss**, η οποία είναι τοπική μεταβλητή της **add()**.

Η συνάρτηση **add()** αγνοεί την ύπαρξη των **a** και **b**, οι οποίες είναι τοπικές μεταβλητές της **main()**.

Στη συνέχεια βλέπετε ένα πιο ολοκληρωμένο πρόγραμμα με τέσσερις συναρτήσεις, εκτός από τη main(). Το πρόγραμμα ζητάει δύο ακέραιους αριθμούς και υπολογίζει το άθροισμα, το γινόμενο, και τον μέσο όρο τους.

10_emveleia.c

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

int add(int x, int y);
int gin(int x, int y);
float mo(int x, int y);
void display(int a, int b);
```

Προκαταβολικές δηλώσεις των συναρτήσεων

```
   return 0;
}
int add()
{
   return x+y;
}
int gin()
{
   return x*y;
}
void out()
{
   printf("Αθροισμα=%d\n",aa);
   printf("Γινόμενο=%d\n",gg);
}
```

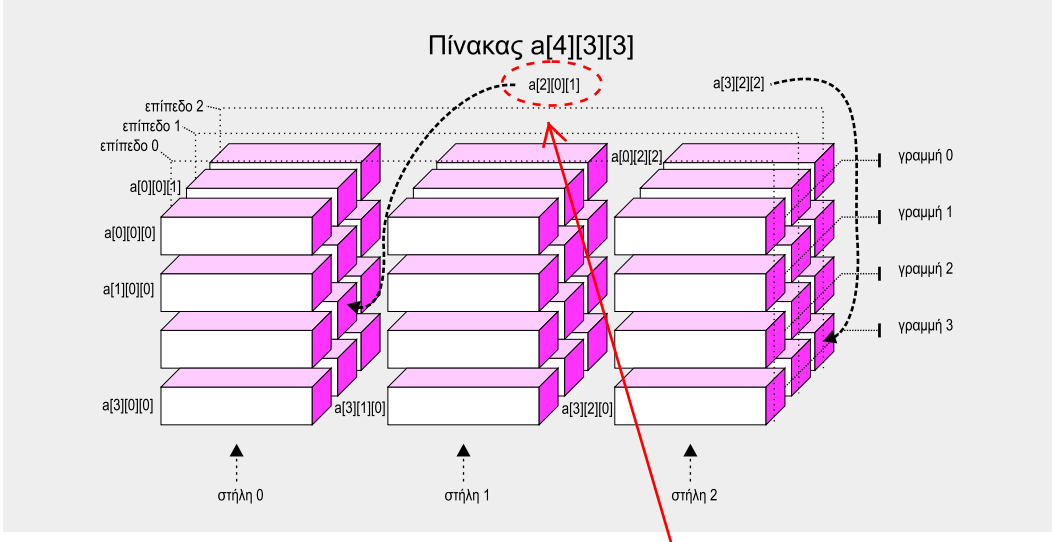
Έγιναν αλλαγές στο πλαίσιο

10.11 Πιστεύετε ότι είναι προτιμότερο να χρησιμοποιούμε καθολικές μεταβλητές για τη μεταβίβαση πληροφοριών σε συναρτήσεις, ή είναι καλύτερο η διαδικασία αυτή να υλοποιείται μέσω παραμέτρων; Να αιτιολογήσετε την απάντησή σας. ★

10.12 Να γραφεί μια συνάρτηση η οποία να εμφανίζει στην οθόνη ένα πλαίσιο το οποίο θα δημιουργείται από έναν χαρακτήρα και θα έχει συγκεκριμένο πλάτος (σε χαρακτήρες) και ύψος (σε γραμμές). Το ύψος, το πλάτος, και ο χαρακτήρας θα μεταβιβάζονται στη συνάρτηση ως ορίσματα. Η συνάρτηση θα πρέπει να έχει μια επί πλέον παράμετρο, η οποία στην περίπτωση που έχει τιμή 1 το πλαίσιο θα είναι γεμάτο με το χαρακτήρα και στην περίπτωση που έχει τιμή 0 θα εμφανίζεται μόνο το περίγραμμα. Επίσης, να γράψετε ένα πρόγραμμα το οποίο να καλεί τη συγκεκριμένη συνάρτηση και να εμφανίζει στην οθόνη τα πλαίσια που φαίνονται στο σχήμα. ★★★

```
aaaaaaaa
a      a
a      a
a      a
a      a
a      a
aaaaaaaa

****
****
****
****
****
```

10.13 Να γραφεί μια συνάρτηση η οποία θα επιστρέφει το παραγοντικό ενός ακέραιου αριθμού (n!). Ο αριθμός θα μεταβιβάζεται ως όρισμα στη συνάρτηση. Επίσης, γράψτε ένα πρόγραμμα το οποίο θα καλεί τη συγκεκριμένη συνάρτηση και θα εμφανίζει στην οθόνη το παραγοντικό των αριθμών από το 1 μέχρι το 10. ★★



Σχήμα 12.6 *Απεικόνιση πίνακα τριών διαστάσεων*

Στο Σχήμα 12.6, ο πίνακας a τριών διαστάσεων (a[4][3][3]) απεικονίζεται ως ένα σύνολο θέσεων μνήμης σε διάταξη τεσσάρων γραμμών, τριών στη[illegible]. Για να αναφερθούμε σε μια θέση μνήμης πρέπει να καθορίσο[illegible], και το επίπεδο στο οποίο βρίσκεται. Χρησιμοποιώντας τις τρ[illegible], μπορούμε να απεικονίσουμε πίνακες μέχρι τρεις διαστάσεις. Πώς μπορούμε όμως να αναπαραστήσουμε σχηματικά έναν πίνακα περισσότερων διαστάσεων; Είναι δυνατό να σχεδιάσουμε σε χαρτί έναν πίνακα τεσσάρων, πέντε, ή και περισσότερων διαστάσεων;

Έγιναν αλλαγές στο σχήμα. Το a[2][2][1] έγινε a[2][0][1]

Και βέβαια μπορούμε, αλλά θα πρέπει να ξεχάσουμε τον τρισδιάστατο χώρο που χρησιμοποιούσαμε μέχρι στιγμής, και να φανταστούμε τις θέσεις μνήμης σαν χωριά στα οποία για να φτάσουμε θα πρέπει να επιλέξουμε τον σωστό δρόμο σε κάθε σταυροδρόμι.

Έτσι, μπορούμε να φανταστούμε έναν πίνακα μίας διάστασης σαν μια συστοιχία δρόμων οι οποίοι ξεκινούν από μια πλατεία (Σχήμα 12.7α), με τον κάθε δρόμο να καταλήγει σε ένα χωριό (μια θέση μνήμης). Επομένως, για να φτάσουμε σε ένα χωριό (θέση μνήμης), θα πρέπει να γνωρίζουμε μόνο έναν αριθμό: τον δρόμο που πρέπει να ακολουθήσουμε από την πλατεία. Αντίστοιχα, έναν πίνακα δύο διαστάσεων μπορούμε να τον παρομοιάσουμε σαν μια συστοιχία δρόμων οι οποίοι ξεκινούν από μια πλατεία, με τον κάθε δρόμο να καταλήγει σε μια άλλη πλατεία από την οποία ξεκινούν οι δρόμοι για τα χωριά-θέσεις μνήμης (Σχήμα 12.7β). Τώρα, για να φτάσουμε σε ένα χωριό (θέση μνήμης), πρέπει να επιλέξουμε ποιον δρόμο θα ακολουθήσουμε στην πρώτη και ποιον δρόμο στη δεύτερη πλατεία. Έναν πίνακα τριών διαστάσεων, όπως τον πίνακα του Σχήματος 12.6, μπορούμε να τον παρομοιάσουμε και πάλι σαν μια πιο πολύπλοκη συστοιχία δρόμων όπου για να φτάσουμε σε κάποια θέση-χωριό πρέπει να περάσουμε από τρεις πλατείες (Σχήμα 12.7γ). Στη πρώτη πλατεία θα επιλέξουμε ανάμεσα σε τέσσερις δρόμους και στη δεύτερη

Οι παράμετροι str1 και str2 είναι δείκτες σε σύνολα χαρακτήρων.

☞ Πρέπει να γίνει κατανοητό ότι οι συμβολοσειρές συγκρίνονται αλφαβητικά, και όχι αριθμητικά. Για παράδειγμα, η πρόταση strcmp("ΝΙΚΟΣ","ΑΛΕΞΑΝΔΡΟΣ") έχει αποτέλεσμα 1 επειδή η συμβολοσειρά "ΝΙΚΟΣ" είναι **αλφαβητικά** μεγαλύτερη από τη συμβολοσειρά "ΑΛΕΞΑΝΔΡΟΣ". Για να γίνει πιο εύκολα αντιληπτή η αλφαβητική σύγκριση, φανταστείτε τις δύο συμβολοσειρές ως καταχωρίσεις σε έναν τηλεφωνικό κατάλογο. Αυτή που προηγείται είναι η μικρότερη.

☞ Για να είναι αλφαβητικά ίσες δύο συμβολοσειρές, πρέπει να περιέχουν ακριβώς τους ίδιους χαρακτήρες. Για παράδειγμα, οι συμβολοσειρές "ΝΙΚΟΣ" και "ΝΙΚΟΣ " δεν είναι ίσες, επειδή η δεύτερη περιέχει στο τέλος της δύο επιπλέον χαρακτήρες διαστήματος.

strcpy()

#include <string.h>

char *strcpy(char *str1, const char *str2)

Η συνάρτηση **strcpy()** αντιγράφει τη συμβολοσειρά στην οποία δείχνει ο δείκτης str2, μέσα στο χώρο μνήμης που δείχνει ο δείκτης str1. Η συνάρτηση επιστρέφει ως τιμή έναν δείκτη στο str1. Ο επόμενος κώδικας αντιγράφει τη συμβολοσειρά «Η γλώσσα C σε βάθος» στον πίνακα χαρακτήρων str.

```
char str[80];
strcpy(str,"Η γλώσσα C σε βάθος");
```

Η καταχώριση ενός συγκεκριμένου συνόλου χαρακτήρων μέσα σε έναν πίνακα μπορεί να γίνει μόνο με την strcpy(). Ένα σύνηθες λάθος που γίνεται είναι η χρήση του τελεστή ανάθεσης = για τη καταχώριση μιας συμβολοσειράς σε έναν πίνακα:

```
char lex[10];
lex="Νίκος";
```

Εσφαλμένη πρόταση καταχώρισης. Για την καταχώριση συγκεκριμένων χαρακτήρων μέσα σε έναν πίνακα πρέπει να χρησιμοποιείται η **strcpy()**.

Παρόλο που αρχικά φαίνεται να δουλεύει σωστά, ο παραπάνω κώδικας είναι αιτία πολλών προβλημάτων που εντοπίζονται δύσκολα (δείτε το Παράδειγμα Π12.7). Δεν πρέπει να συγχέουμε τη σωστή καταχώριση αρχικής τιμής σε έναν πίνακα χαρακτήρων στην πρόταση δήλωσής του:

```
char lex[10]="Νίκος";
```

Ο αριθμός του παραδείγματος από 12.6 έγινε 12.7

strcat()

#include <string.h>

char *strcat(char *str1, const char *str2)

Η συνάρτηση **strcat()** προσθέτει στο τέλος της συμβολοσειράς str1 τη συμβολοσειρά str2. Στη συνάρτηση μεταβιβάζονται οι διευθύνσεις των δύο συμβολοσειρών.

```
   int found=0;
   printf("Δώσε όνομα:");
   gets(ep);
   fp=fopen("sxoleio","rb");
   while (!feof(fp))
   {
      fread(&m,sizeof(struct stoixeia),1,fp);
      if (strcmp(m.eponymo,ep)==0)
      {
         found++;
         printf("Επώνυμο:%s\n",m.eponymo);
         printf("Τάξη:%s\n",m.taxi);
         printf("ΜΟ:%f\n",m.mesos_oros);
         printf("Ηλικία:%d\n",m.ilikia);
      }
   }
   fclose(fp);
   if (found==0)
      printf("Δεν βρέθηκαν μαθητές με αυτό το επώνυμο\n");
   else
      printf("Βρέθηκαν %d μαθητές\n",found);
   return 0;
}
```

Η **fread()** διαβάζει ένα τμήμα (μια εγγραφή) από τόσα byte όσο το μέγεθος της δομής **stoixeia** και το καταχωρίζει στη μεταβλητή **m**. Η **fread()** χρειάζεται τη διεύθυνση της **m**, η οποία αποδίδεται από το **&m**.

Ελέγχει αν το επώνυμο της εγγραφής που διαβάστηκε είναι το επώνυμο που αναζητούμε. Αν είναι, αυξάνει τον μετρητή **found** κατά 1 και εμφανίζει τα στοιχεία του μαθητή.

Προστέθηκε η πρόταση

Π14.6 Το επόμενο πρόγραμμα χρησιμοποιείται για την κρυπτογράφηση και αποκρυπτογράφηση ενός αρχείου οποιουδήποτε τύπου. Η κρυπτογράφηση βασίζεται στον τελεστή bit «συμπληρώματος» (~). Ο τελεστής αυτός αντιστρέφει τα bit ενός byte με τη λογική που έχουμε εξηγήσει στο κεφάλαιο 4. Για παράδειγμα, αν ένα byte έχει τιμή 10110100, το συμπλήρωμά του είναι 01001011. Αν τώρα εφαρμόσουμε πάλι τον τελεστή του συμπληρώματος στο αποτέλεσμα, έχουμε ως αποτέλεσμα το αρχικό byte. Το πρόγραμμα που ακολουθεί διαβάζει ένα προς ένα τα byte από το αρχείο εισόδου, υπολογίζει το συμπλήρωμά τους, και τα γράφει στο αρχείο εξόδου. Και τα δύο αρχεία ανοίγουν ως δυαδικά.

```
~ 10110100 → 01001011
~ 01001011 → 10110100
```

14_p14_6.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   FILE *fp1,*fp2;
   char file_in[30],file_out[30],ch;
   printf("Δώσε όνομα αρχείου εισόδου:");
   gets(file_in);
   printf("Δώσε όνομα αρχείου εξόδου:");
   gets(file_out);
   fp1=fopen(file_in,"rb");
```

Άνοιγμα του αρχείου εισόδου σε δυαδική μορφή για ανάγνωση δεδομένων.

Έλεγχος των περιεχομένων ενός δείκτη προς συνάρτηση

Για να ελέγξουμε σε ποια συνάρτηση δείχνει ένας δείκτης, απλώς χρησιμοποιούμε τον τελεστή σύγκρισης ==, συγκρίνοντας τον δείκτη είτε με τη διεύθυνση είτε με το όνομα της συνάρτησης:

```
if (ptr1==&func1)
   printf("Ο δείκτης ptr1 δείχνει στη συνάρτηση func1()\n");
else if (ptr1==func2)
   printf("Ο δείκτης ptr1 δείχνει στη συνάρτηση func2()\n");
else
   printf("Ο δείκτης ptr1 δεν δείχνει σε καμία από τις δύο συναρτήσεις\n");
```

Η λογική παράσταση ptr1==&func1 θα μπορούσε να γραφεί ως ptr1==func1, χωρίς τον τελεστή διεύθυνσης. Οι δύο παραστάσεις είναι ισοδύναμες, όμως οι περισσότεροι μεταγλωττιστές προτιμούν τον πρώτο τρόπο σύνταξης.

Δείκτες προς συναρτήσεις ως παράμετροι

Μια συνάρτηση μπορεί να διαθέτει παραμέτρους που να είναι δείκτες προς άλλες συναρτήσεις:

```
void call_ptr(int (*p)(int x, int y))
{
   printf("Κλήση της συνάρτησης μέσω του δείκτη p : %d\n",p(10,100));
}

int main(void)
{
   int (*ptr1)(int x, int y), (*ptr2)(int x, int y);
   ptr1=&func1;
   ptr2=&func2;
   call_ptr(&func1);
   call_ptr(ptr2);
   return 0;
}
```

Η παράμετρος **p** είναι ένας δείκτης προς συναρτήσεις τύπου **int** με δύο παραμέτρους επίσης τύπου **int**.

Καλείται η συνάρτηση στην οποία δείχνει ο δείκτης **p**.

Καλείται η συνάρτηση **call_ptr()** με όρισμα τη διεύθυνση της συνάρτησης **func1()**. Η **call_ptr** με τη σειρά της θα καλέσει τη **func1()** μέσω της παραμέτρου της **p**.

Καλείται η συνάρτηση **call_ptr()** με όρισμα το δείκτη **ptr2**. Η **call_ptr** με τη σειρά της θα καλέσει τη **func2()** διότι ο δείκτης **ptr2** περιέχει τη διεύθυνση της **func2()**.

Πίνακες δεικτών προ

Είναι προφανές ότι μπορ ακες δεικτών προς συναρτήσεις. Κάθε στοιχείο του πίνακα θα ι ιας συνάρτησης. Η παρακάτω πρόταση δηλώνει έναν πίνακα δεικτών προς συναρτήσεις, με 10 θέσεις και όνομα ptr:

```
int (*ptr[10])(int x, int y);
```

Η χρήση ενός πίνακα δεικτών προς συναρτήσεις γίνεται εμφανής στο Παράδειγμα Π15.5.

Διορθώθηκε ο αριθμός του παραδείγματος από Π15.4 σε Π15.5

Πρέπει να έχουμε υπόψη ότι δεν είναι δυνατό να μεταβάλουμε το μέγεθος ενός πίνακα. Ο πίνακας έχει πάντα το μέγεθος που ορίζουμε με την πρόταση δήλωσής του.

Ένα άλλο πρόβλημα προκύπτει όταν δεν γνωρίζουμε το μέγεθος του πίνακα κατά τη συγγραφή του κώδικα, αλλά το γνωρίζουμε κατά την ώρα εκτέλεσης του προγράμματος. Για παράδειγμα, ίσως το πρόγραμμα να ζητάει το πλήθος των μαθητών μιας τάξης και έπειτα να δημιουργείται ένας πίνακας με το πλήθος των μαθητών. Στις αρχικές εκδόσεις της C, δεν μπορούσαμε να δηλώσουμε έναν πίνακα που οι διαστάσεις του να καθορίζονται κατά τη διάρκεια εκτέλεσης. Στο πρότυπο C99 και τους πίνακες μεταβλητού μήκους που ήδη έχουμε συναντήσει, προτάσεις όπως η επόμενη

Εγινε αλλαγή στη σελίδα παραπομπής. Η σωστή σελίδα είναι 382

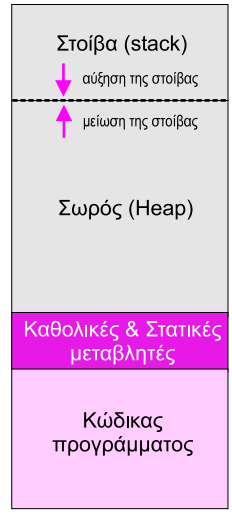
int num[a];

είναι έγκυρες, και μάλιστα υπάρχει η δυνατότητα καθορισμού των διαστάσεων, και επομένως του μεγέθους ενός πίνακα, από τιμές μεταβλητών κατά τη στιγμή της δήλωσης του πίνακα (δείτε την ενότητα *Πίνακες μεταβλητού μήκους* στη σελίδα 382). Και πάλι, όμως, από τη στιγμή που θα δηλωθεί ο πίνακας με συγκεκριμένες διαστάσεις, δεν μπορούμε να μεταβάλουμε το μέγεθός του αργότερα.

Η λύση βρίσκεται στον μηχανισμό δυναμικής κατανομής μνήμης που διαθέτει η C.

Η C διαθέτει έναν μηχανισμό **δυναμικής κατανομής** μνήμης και σχετικές συναρτήσεις για την υλοποίησή του. Μπορούμε να δεσμεύουμε περισσότερη μνήμη όταν την έχουμε ανάγκη, ή να αποδεσμεύουμε μνήμη την οποία δεν χρειαζόμαστε πλέον.

Δυναμική κατανομή μνήμης



Στο διπλανό σχήμα απεικονίζεται ο τρόπος διαχείρισης της μνήμης από τη C. Ένα τμήμα μνήμης χρησιμοποιείται για την αποθήκευση του κώδικα του προγράμματος, και ένα γειτονικό τμήμα καταλαμβάνουν οι καθολικές και οι στατικές μεταβλητές.

Όπως έχει ήδη αναφερθεί, η **στοίβα** (stack) είναι ένας χώρος μνήμης στον οποίο αποθηκεύονται οι τοπικές μεταβλητές, καθώς και οι πληροφορίες που αφορούν κλήσεις των συναρτήσεων, όπως η διεύθυνση επιστροφής στον κώδικα από τον οποίον κλήθηκε μια συνάρτηση.

Η στοίβα αυξάνεται ή μειώνεται ανάλογα με τις ανάγκες του προγράμματος.

Ο υπόλοιπος αδιάθετος χώρος μνήμης μεταξύ της στοίβας και του χώρου των γενικών μεταβλητών λέγεται **σωρός** (heap).

Κατά τον χρόνο εκτέλεσης (run-time) του προγράμματος, μπορούν να διατεθούν τμήματα μνήμης (memory blocks) από τον σωρό με δυναμική κατανομή,

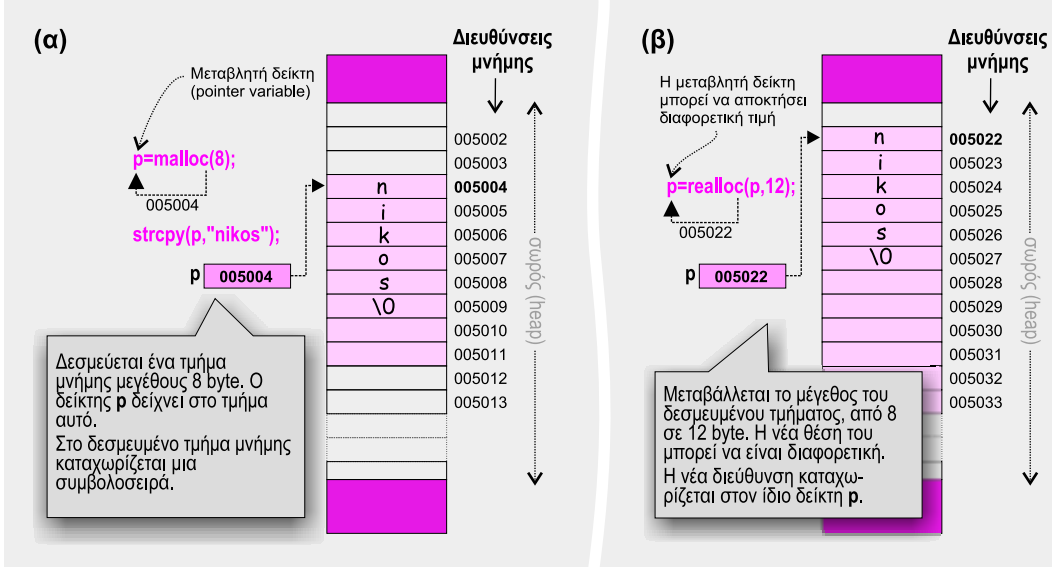
malloc() και η calloc(). Η πρώτη παράμετρος **ptr** είναι ένας δείκτης ο οποίος δείχνει στο υπάρχον δεσμευμένο τμήμα μνήμης, και η παράμετρος **size** καθορίζει το νέο μέγεθος που θα αποκτήσει το τμήμα. Η συνάρτηση επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του νέου τμήματος. Αν δεν υπάρχει αρκετή διαθέσιμη μνήμη για το νέο μέγεθος του μπλοκ, η realloc() επιστρέφει έναν δείκτη NULL χωρίς να επηρεάσει τα περιεχόμενα του αρχικού μπλοκ. **Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση**.

☞ Για να μπορέσει η C να αλλάξει το μέγεθος ενός τμήματος, ίσως χρειαστεί να μετακινήσει το τμήμα σε άλλη θέση, ακόμη και αν το νέο μέγεθος είναι μικρότερο από το αρχικό. Τα περιεχόμενα του ε… …άφονται αυτόματα στη νέα θέση. Επομένως, η διεύθυνση του … …ο δείκτης που επιστρέφεται) μπορεί να είναι διαφορετική από … …ατος.

Έγινε αλλαγή της διεύθυνσης σε 005004

Η συνάρτηση realloc() διαθέτει δύο παραμέτρους τύπου δείκτη void και size_t αντίστοιχα. Η τιμή που επιστρέφει η συνάρτηση είναι ένας δείκτης τύπου void.

Στο παράδειγμα του Σχήματος 17.2 φαίνεται η λειτουργία της realloc(). Με τη συνάρτηση malloc() δεσμεύουμε ένα τμήμα 8 θέσεων μνήμης, την αρχική διεύθυνση του οποίου (έστω η 005004) καταχωρίζουμε στον δείκτη p (Σχήμα 17.2α). Με χρήση της συνάρτησης strcpy() αντιγράφουμε το σύνολο χαρακτήρων «nikos» μέσα στο δεσμευμένο τμήμα μνήμης. Η κλήση της realloc() που ακολουθεί έχει ως αποτέλεσμα την αύξηση του δεσμευμένου τμήματος από 8 σε 12 θέσεις μνήμης (Σχήμα 17.2β).



Σχήμα 17.2 *Χρήση της συνάρτησης realloc()*

```
   neos->next=*plh;
   *plh=neos;
   return 1;
}

void display_all_nodes(struct node *lh)
{
   struct node *p;
   p=lh;
   while (p!=NULL)
   {
      printf("%s %d\n",p->onoma,p->ilikia);
      p=p->next;
   }
}
```

Η πρόταση αυτή καταχωρίζει στον χειριστή της λίστας τη διεύθυνση του νέου κόμβου. Μην ξεχνάμε ότι ο δείκτης **plh** δείχνει στον δείκτη **list_head**!

Αντικαταστάθηκε η πρόταση printf(...)

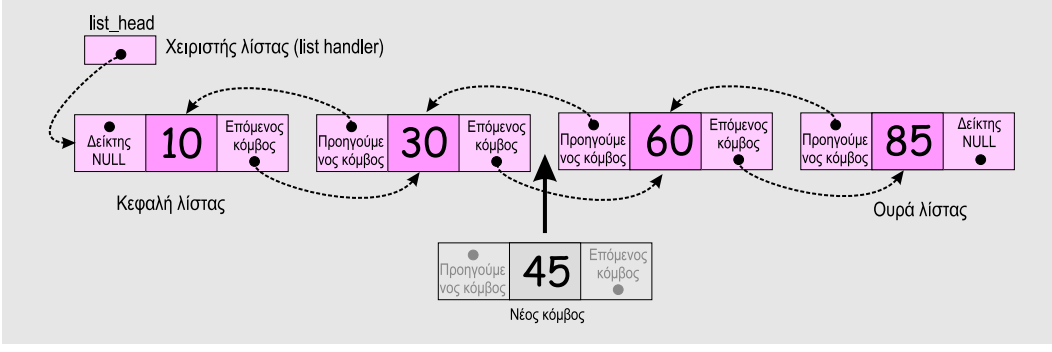
Διασχίζει όλη τη λίστα ξεκινώντας από τον κόμβο στον οποίο δείχνει η παράμετρος **lh** και εμφανίζει τα δεδομένα κάθε κόμβου. Όταν στην παράμετρο **lh** μεταβιβαστεί ο χειριστής (κεφαλή) της λίστας, η συνάρτηση εμφανίζει όλα της τα περιεχόμενα.

Διατεταγμένη συνδεδεμένη λίστα

Σε μια διατεταγμένη (ταξινομημένη) συνδεδεμένη λίστα, οι κόμβοι είναι διατεταγμένοι (ταξινομημένοι) με τη σειρά, με βάση κάποιο στοιχείο της λίστας. Το στοιχείο που χρησιμοποιείται για τη διάταξη των κόμβων λέγεται «κλειδί». Η προσθήκη ενός νέου κόμβου στη λίστα δεν γίνεται πλέον στην κεφαλή ή στην ουρά της λίστας, αλλά σε συγκεκριμένο σημείο, έτσι ώστε η λίστα να διατηρηθεί ταξινομημένη (Σχήμα 18.4). Ο πιο κατάλληλος τύπος για την υλοποίηση μιας διατεταγμένης λίστας είναι η διπλά συνδεδεμένη λίστα. Η όλη διαδικασία υλοποίησης μιας διατεταγμένης λίστας επικεντρώνεται στον εντοπισμό της θέσης στην οποία πρέπει να παρεμβληθεί ο νέος κόμβος. Έστω ότι έχουμε μια διπλά συνδεδεμένη λίστα με κόμβους της παρακάτω δομής:

```
struct node
{
   int data;
   struct node *next;
   struct node *previous;
};
```

Το τμήμα δεδομένων αποτελείται μόνο από το πεδίο **data**, στο οποίο καταχωρίζεται ένας ακέραιος αριθμός.



Σχήμα 18.4 *Προσθήκη νέου κόμβου σε διατεταγμένη λίστα*