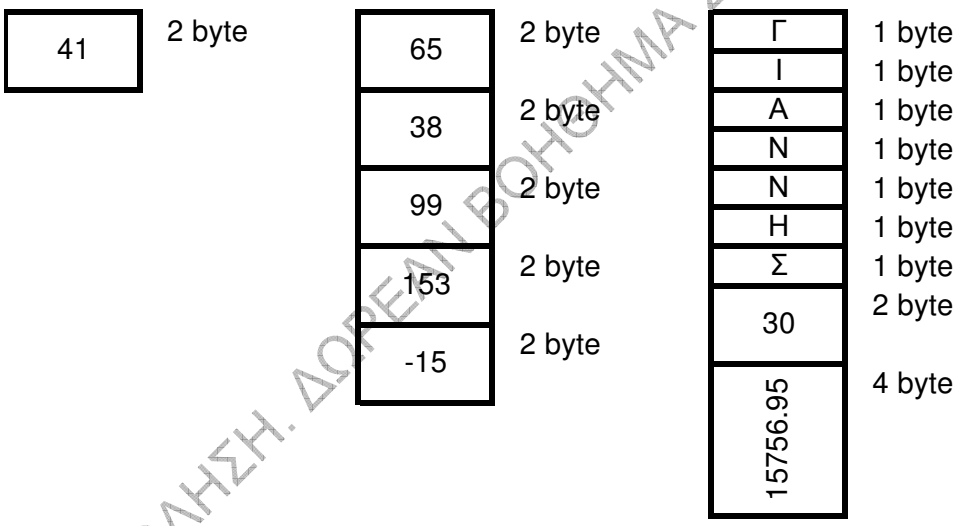
ΚΕΦΑΛΑΙΟ 3

ΔΟΜΕΣ και ΕΝΩΣΕΙΣ

3.1. ΕΙΣΑΓΩΓΗ ΣΤΙΣ ΔΟΜΕΣ - ΓΕΝΙΚΑ.

Συχνά θέλουμε να λειτουργούμε πάνω σε δεδομένα **διαφορετικού τύπου** μεταξύ τους, **τα οποίες να χειριζόμαστε σαν ομάδα**. Σε αυτή την περίπτωση δεν είναι κατάλληλες οι μεμονωμένες μεταβλητές, αλλά ούτε και οι πίνακες.



Σχ. 3.1

Για παράδειγμα, θέλουμε σε ένα πρόγραμμα να κρατάμε και να αποθηκεύουμε δεδομένα, τα οποία αφορούν τα στοιχεία ενός σπουδαστή. Θέλουμε να κρατάμε για τον κάθε σπουδαστή το ονοματεπώνυμό του (συμβολοσειρά), το εξάμηνο φοίτησης (ακέραιος), το έτος γέννησης (ακέραιος) και το μέσο όρο βαθμολογίας του (float). Όλα αυτά τα στοιχεία αποτελούν μια ομάδα για τον κάθε σπουδαστή. Δεν μπορούμε να τα αποθηκεύσουμε σε ένα πίνακα, αφού ο πίνακας κρατάει στοιχεία ίδιου τύπου, πράγμα που εδώ δεν ισχύει. Θα μπορούσαμε να χρησιμοποιήσουμε βέβαια αρκετούς διαφορετικούς πίνακες (ένα πίνακα συμβολοσειρών για τα ονόματα, ένα πίνακα

ακεραίων για τα εξάμηνα, ένα άλλο πίνακα ακεραίων για τα έτη γέννησης και ένα πίνακα float για τους μέσους όρους βαθμολογίας. Αυτό είναι κάτι άβολο, αφού τα χαρακτηριστικά για τον κάθε σπουδαστή παύουν να υπάρχουν σαν ομάδα και βρίσκονται πλέον διασκορπισμένα σε διάφορους πίνακες.

Για να λύσουμε τέτοια προβλήματα χρησιμοποιούμε ένα ειδικό τύπο δεδομένων, την **δομή (structure)**. **Μια δομή αποτελείται από ένα πλήθος στοιχείων δεδομένων, τα οποία δεν χρειάζεται να είναι του ίδιου τύπου**. Στο παράδειγμα που αναφέραμε πιο πάνω, η δομή θα αποτελείται από τα στοιχεία του σπουδαστή, όπως τα αναφέραμε, αλλά και από όσα άλλα στοιχεία θέλαμε. Στο παραπάνω σχήμα 3.1 φαίνονται σχηματικά όπως θα αποθηκευόταν στη μνήμη μια απλή ακέραια μεταβλητή, ένας πίνακας ακεραίων και μια δομή.

Μια πολύ σημαντική χρήση των δομών είναι η δημιουργία **νέων μορφών δεδομένων**, όπως λίστες, δέντρα, σωροί κλπ, πολλά από τα οποία θα εξετάσουμε αναλυτικότερα στη συνέχεια. Οι περισσότερες από αυτές τις νέες μορφές σχηματίζονται από δομές **συνδεδεμένες μεταξύ τους**.

3.2. ΜΙΑ ΑΠΛΗ ΔΟΜΗ.

Το παρακάτω πρόγραμμα χρησιμοποιεί μια απλή δομή που περιέχει δύο στοιχεία δεδομένων: μια μεταβλητή τύπου ακεραίου, την num και μια μεταβλητή τύπου χαρακτήρα, την ch.

```
void main(void)
{
    struct easy
    {
        int num;
        char ch;
    };

    struct easy dom;

    dom.num = 5;
    dom.ch  = 'A';
    printf("dom.num=%d, dom.ch=%c", dom.num,
            dom.ch);
}
```

Όταν εκτελεστεί το πρόγραμμα, στην οθόνη θα εμφανιστεί:

dom.num=5, dom.ch=A

Σχόλια:

1. Οι βασικοί τύποι δεδομένων της C (int, float κλπ) είναι γνωστοί στον compiler. Έτσι, χρησιμοποιώντας π.χ. έναν float, ο υπολογιστής ξέρει ότι θα δεσμευτεί γι' αυτόν χώρος 4 byte. Τέτοια «γνώση» δεν υπάρχει για τις δομές, αφού αυτές περιέχουν οποιοδήποτε πλήθος και είδος διαφορετικών στοιχείων. Έτσι, **πρέπει να πούμε στον compiler πώς είναι η δομή πριν χρησιμοποιηθούν μεταβλητές αυτού του τύπου**. Πρέπει δηλαδή να περιγράψουμε τη μορφή της δομής, να την μορφοποιήσουμε.

2. Η περιγραφή:

```
struct easy
{
   int num;
   char ch;
};
```

αναφέρεται σε ένα **νέο τύπο δεδομένων**, ο οποίος είναι **δομή** και λέγεται easy. Κάθε μεταβλητή αυτού του τύπου θα αποτελείται από δύο στοιχεία: μια ακέραια μεταβλητή και μια μεταβλητή τύπου χαρακτήρα. Θα λέμε σωστότερα ότι οι δομές του τύπου easy έχουν δυο **πεδία**. Σε όλες τις δομές αυτού του τύπου το ένα πεδίο θα είναι το πεδίο num της δομής, ενώ το άλλο θα είναι το πεδίο ch της δομής. Η παραπάνω περιγραφή δεν αναφέρεται σε συγκεκριμένες μεταβλητές, άρα **δεν δεσμεύει χώρο στη μνήμη**. **Λέει απλώς ποια είναι η μορφή των δεδομένων του είδους «δομή easy»**. Αυτό είναι αντίστοιχο με το εξής: λέγοντας για παράδειγμα int στις δηλώσεις, δεν δεσμεύεται χώρος. Δέσμευση γίνεται όταν αναφέρουμε και το όνομα της μεταβλητής.

3. Αφού περιγράψουμε την δομή, μπορούμε να δηλώσουμε μια ή περισσότερες μεταβλητές αυτού του τύπου. Στην δήλωση:

```
struct easy dom;
```

δηλώνεται μια μεταβλητή, η dom, που είναι του τύπου, του είδους struct easy. **Εδώ δεσμεύεται χώρος** στη μνήμη. Δεσμεύεται αρκετός χώρος, ώστε να χωράει όλα τα

στοιχεία της δομής, στην περίπτωσή μας δηλαδή **3 byte**, δύο για τον ακέραιο και ένα για τον χαρακτήρα. Η δήλωση αυτή είναι για παράδειγμα το αντίστοιχο μιας δήλωσης, όπως η: int ak;

4. Έστω ότι θέλω να αναφερθώ στα στοιχεία της δομής και συγκεκριμένα στο πεδίο num της δομής dom. Αυτό είναι το:

```
dom.num
```

Οι εντολές:

```
dom.num = 8;
dom.ch = 'K';
```

δίνουν την τιμή 8 στο πεδίο num της dom και την τιμή 'K' στο πεδίο ch της ίδιας δομής. Δηλαδή ο **τελεστής τελείας (.) ενώνει το όνομα μεταβλητής δομής με ένα πεδίο της δομής**.

5. Όπως μπορεί να υπάρχουν περισσότερες από μια μεταβλητές τύπου int, float κλπ σε ένα πρόγραμμα, μπορεί επίσης να υπάρχει οποιοδήποτε πλήθος μεταβλητών ενός δεδομένου είδους δομής. Πιο κάτω δηλώνονται δύο τέτοιες μεταβλητές, οι dom και str:

```
struct easy dom;
struct easy str;
```

Οι εντολές:

```
dom.num = 6;
dom.ch = 'F';
str.num = 32;
str.ch = 'P';
```

δίνουν την τιμή 6 στο πεδίο num της dom, την τιμή 'F' στο πεδίο ch της dom, την τιμή 32 στο πεδίο num της str και την τιμή 'P' στο πεδίο ch της str.

6. Μπορούμε να συνδυάσουμε σε μια εντολή την περιγραφή του είδους της δομής και των μεταβλητών δομής. Οι δηλώσεις δηλαδή των dom και str της πιο πάνω παραγράφου, μπορούν να γραφούν και έτσι:

```
struct easy
{
    int num;
    char ch;
} dom, str;
```

3.3. ΔΕΔΟΜΕΝΑ ΣΤΙΣ ΔΟΜΕΣ.

3.3.1. Εισαγωγή.

Στο παρακάτω πρόγραμμα οι δομές persa και persb παίρνουν τιμές από το πληκτρολόγιο, κατόπιν δε τα περιεχόμενά τους εμφανίζονται στην οθόνη:

```
void main(void)
{
   struct person
   {
      char name[30];
      int code;
   };
   struct person persa, persb;
   char strcode[10];

   gets (persa.name);
   gets (strcode);
   persa.code = atoi (strcode);
   gets (persb.name);
   gets (strcode);
   persb.code = atoi (strcode);
   puts ("ΠΕΡΙΕΧΟΜΕΝΑ ΔΟΜΩΝ");
   printf ("%s\n", persa.name);
   printf ("%05d\n", persa.code);
   printf ("%s\n", persb.name);
   printf ("%05d\n", persb.code);
}
```

Στο πρόγραμμα γίνεται χρήση της συνάρτησης atoi( ). Οι αριθμοί μπορούν να αποθηκευτούν είτε σε αριθμητική μορφή, είτε ως συμβολοσειρές. Π.χ., ο αριθμός 356 μπορεί να αποθηκευτεί σε ένα πίνακα χαρακτήρων, ως ψηφία:

'3', '5', '6', '\0'

Η συνάρτηση **atoi( )** δέχεται ως **όρισμα** μια **συμβολοσειρά** και επιστρέφει την αντίστοιχη **ακέραια τιμή**. Άλλες αντίστοιχες συναρτήσεις είναι η **atof( )** που μετατρέπει μια συμβολοσειρά σε **double** και η **atol( )** που μετατρέπει μια συμβολοσειρά σε **long**. Παρατηρείστε ότι το διάβασμα των πεδίων code των persa και persb θα μπορούσε να γίνει και **με τη χρήση της scanf** ως εξής:

```
scanf ("%d\n", &persa.code);
scanf ("%d\n", &persb.code);
```

Το «μηδέν» στο **%05d** κάνει τον ακέραιο να γραφεί σε χώρο 5 διαστημάτων, ως γνωστό, με τα διαστήματα όμως που δεν χρησιμοποιούνται να εμφανίζονται στην οθόνη σαν μηδέν.

3.3.2. Αρχικές τιμές και απόδοση τιμής.

Όπως και οι απλές μεταβλητές, οι δομές μπορούν να πάρουν **αρχικές τιμές κατά τη δήλωσή τους.** Π.χ.:

```
struct person persa = {“ΓΙΑΝΝΗΣ“, 15};
struct person persb = {“ΝΙΚΟΣ“, 10};
```

Εξ άλλου, η παρακάτω εντολή είναι έγκυρη, αν και δεν ίσχυε στις αρχικές εκδόσεις της C:

```
persa = persb;
```

Η δυνατότητα αυτή είναι χρησιμότατη αφού μπορούμε **να αναθέσουμε με μια εντολή τα στοιχεία μιας δομής σε μια άλλη**, ακόμη κι αν πρόκειται για παράδειγμα για μεγάλους πίνακες.

3.4. ΦΩΛΙΑΣΜΕΝΕΣ ΔΟΜΕΣ.

Όπως υπάρχουν πίνακες πινάκων, έτσι μπορεί να υπάρχουν και **δομές που περιέχουν άλλες δομές**. Π.χ.:

```
struct person
{
    char name[30];
    int code;
};
struct couple
{
    struct person chief;
    struct person memb;
} first, second;
```

Η δομή couple είναι οργανωμένη έτσι, ώστε να περιέχει **δύο στοιχεία, καθένα από τα οποία είναι μια δομή τύπου person**. Το πρώτο από τα πεδία της first (η οποία όπως βλέπουμε είναι μια μεταβλητή δομή τύπου couple) είναι το:

```
first.chief
```

Αυτό όμως είναι μεταβλητή δομή τύπου person, της οποίας το δεύτερο πεδίο για παράδειγμα είναι το:

```
first.chief.code
```

Μπορώ προφανώς να πώ:

```
first.chief.code = 158;
```

3.5. ΔΟΜΕΣ ΚΑΙ ΣΥΝΑΡΤΗΣΕΙΣ.

Μια συνάρτηση μπορεί να έχει τιμή επιστροφής ένα τύπο δομής. Η συνάρτηση newpers( ) (βλ. παρακάτω), διαβάζει από το πληκτρολόγιο στοιχεία μιας μεταβλητής δομής και τα καταχωρεί σε μεταβλητές της main( ). Συγκεκριμένα, η συνάρτηση διαβάζει τα στοιχεία της μεταβλητής και **επιστρέφει την τιμή που διάβασε, δηλαδή ολόκληρη τη μεταβλητή δομής**. Όπως λοιπόν μια συνάρτηση που επιστρέφει ακέραια τιμή είναι και αυτή τύπου int, έτσι και **μια συνάρτηση που έχει τιμή επιστροφής δομή, είναι και αυτή του ίδιου τύπου**.

```
#include <stdio.h>
struct person
{
    char name[30];
    int code;
};
struct person newpers( );      /* Δήλωση της newpers( ) */
void disppers(struct person);      /*Δήλωση της disppers( )*/
void main(void)
{
    struct person persa;
    struct person persb;

    persa = newpers( );
    persb = newpers( );
    dispers (persa);
    dispers (persb);
}
struct person newpers( )              /* Ορισμός newpers( ) */
{
    struct person temp;
    char  ch;

    gets (temp.name);
    scanf ("%d", &temp.code);
    scanf ("%c", &ch);
    return temp;
}
void disppers(struct person datom) /* Ορισμός disppers( ) */
{
    puts ("Στοιχεία προσώπου: ");
    printf ("Όνομα : %s\n", datom.name);
    printf ("Κωδικός : %05d\n", datom.code);
}
```

Στο παράδειγμά μας, η newpers( ) είναι τύπου struct person. Η temp είναι μια μεταβλητή δομή του τύπου person, ορισμένη ως τοπική μεταβλητή της συνάρτησης newpers( ). Μέσα στη συνάρτηση διαβάζονται τα περιεχόμενά της και τελικά η τιμή της είναι αυτή που επιστρέφεται στη main( ).

Μια συνάρτηση μπορεί να δέχεται δομές ως ορίσματα. Στην περίπτωση αυτή τα ορίσματα δηλώνονται με τον τρόπο που γνωρίζουμε ότι δηλώνεται οποιαδήποτε μεταβλητή. Η συνάρτηση disppers( ) στο πιο πάνω παράδειγμα δέχεται ως όρισμα μια δομή, της οποίας εμφανίζει τα στοιχεία στην οθόνη.

Και οι συναρτήσεις και το κυρίως πρόγραμμα πρέπει να ξέρουν τη μορφή της δομής person, για τον λόγο δε αυτό η περιγραφή της έχει δοθεί **εξωτερικά**, δηλαδή πριν από τη main( ), **έξω από όλες τις συναρτήσεις**.

Η scanf("%c", &ch); μέσα στην newpers( ) χρησιμοποιείται για την **«κατανάλωση» του Enter** μετά το διάβασμα του κωδικού. Το ίδιο θα έκανε το \n στην scanf( ) που ακολουθεί:

```
scanf (“%d\n”, &temp.code);
```

3.6. ΠΙΝΑΚΕΣ ΔΟΜΩΝ.

Όπως δηλώνουμε πίνακες ακεραίων, πίνακες float κλπ, μπορούμε να δηλώσουμε και πίνακα, **κάθε στοιχείο** του οποίου **είναι μια δομή**. Η μεταβλητή pinax που δηλώνεται παρακάτω είναι ένας πίνακας δομών 30 θέσεων, κάθε μια από τις οποίες είναι του τύπου employee. Δεσμεύονται δηλαδή 30 θέσεις στη μνήμη, λέγονται και οι 30 με το όνομα pinax και **σε κάθε μια θέση θα μπει μια δομή** τύπου employee:

```
struct employee
{
   char name[30];
   int   code;
   float pos;
};
struct employee pinax[30];
```

Η παρακάτω εντολή διαβάζει από το πληκτρολόγιο μια συμβολοσειρά και την καταχωρεί στο στοιχείο name της τρίτης θέσης του πίνακα pinax:

```
gets (pinax[2].name);
```

3.7. ΔΕΙΚΤΕΣ ΚΑΙ ΔΟΜΕΣ.

Όπως ξέρουμε, κάθε δείκτης είναι στην πράξη η διεύθυνση μιας απλής μεταβλητής. Μέσω των δεικτών γνωρίζουμε λοιπόν τις διευθύνσεις των μεταβλητών στη μνήμη, καθώς και τις διευθύνσεις των πρώτων στοιχείων των πινάκων που έχουμε δηλώσει. Αντίστοιχα, οι δείκτες «κρατούν» και τις διευθύνσεις των δομών που δηλώνουμε.

Όταν έχουμε δηλώσει μια μεταβλητή δομή κάποιου είδους, άρα **όταν γνωρίζουμε το όνομά της**, η προσπέλαση των πεδίων της γίνεται με χρήση του **τελεστή τελεία (.)**. Όταν όμως θέλουμε **να προσπελάσουμε τα πεδία μιας δομής** όχι μέσω του ονόματός της (το οποίο πιθανόν και να μη γνωρίζουμε), αλλά **μέσω ενός δείκτη που δείχνει στην δομή**, τότε χρησιμοποιούμε τον **τελεστή βέλος (->)**.

Στο παράδειγμα που ακολουθεί έχει δηλωθεί μια μεταβλητή δομή, η dok, στην οποία προφανώς δείχνει ο δείκτης &dok. Στην ίδια δομή τοποθετείται να δείχνει και ο δείκτης ptr, ο οποίος έχει δηλωθεί ως δείκτης σε δομές του είδους simple.

```
void main(void)
{
   struct simple
   {
      int   num;
      char ch;
   };
   struct simple  dok;
   struct simple  *ptr;

   ..............................
   ptr = &dok;
   ptr -> num = 303;
   ptr -> ch = 'Q';
   ..............................
}
```


Σχόλια:

1. Στο πιο πάνω παράδειγμα, η εντολή:

   ```
   ptr -> num = 303;
   ```

   κάνει το πεδίο num της δομής, στην οποία δείχνει ο ptr, ίσο με 303.

2. Ο συμβολισμός:

   ```
   ptr.num
   ```

προφανώς δεν είναι έγκυρος, διότι αριστερά του τελεστή τελεία (.) πρέπει να υπάρχει όνομα δομής, ενώ ο ptr είναι δείκτης σε δομή.

3. Είναι σωστός ο συμβολισμός:

   (*ptr).num

   Μέσα στις παρενθέσεις υπάρχουν τα περιεχόμενα του ptr, δηλαδή μια μεταβλητή δομή, άρα σωστά χρησιμοποιείται ο τελεστής τελεία. Οι παρενθέσεις εξ άλλου είναι αναγκαίες, αφού ο τελεστής τελεία (.) έχει μεγαλύτερη προτεραιότητα από τον τελεστή *.

4. Αντίστοιχα είναι σωστός και ο συμβολισμός:

   (&ptr) -> num

   αφού μέσα στις παρενθέσεις υπάρχει τώρα ένας δείκτης σε δομή, άρα σωστά χρησιμοποιείται ο τελεστής βέλος. Οι παρενθέσεις και εδώ είναι αναγκαίες, αφού ο τελεστής βέλος (->) έχει μεγαλύτερη προτεραιότητα από τον τελεστή &.

Παράδειγμα:

Στο παρακάτω πρόγραμμα γίνεται απόδοση τιμών στα διάφορα πεδία μιας δομής:

```
struct person
{
   char name[30];
   int code;
};
struct couple
{
   struct person chief;
   struct person memb;
};
void main(void)
{
   struct couple frost;
   struct couple *sand;

   sand = &frost;
   frost.chief.name = "ΓΙΑΝΝΗΣ";          /* 1 */
   frost.chief.code = 1999;               /* 2 */
   sand ->chief.name = "ΜΑΡΙΑ";           /* 3 */
   sand ->chief.code = 1958;              /* 4 */
   frost.memb->name = "ΚΩΣΤΑΣ";           /* 5 */
   sand ->memb->name = "ΝΙΚΟΛΑΟΣ";        /* 6 */
}
```

Στο πρόγραμμα αυτό:

- Οι γραμμές με αριθμούς 2 και 4 είναι σωστές συντακτικά.
- Οι γραμμές 1 και 3 είναι λάθος, αφού η αντιγραφή συμβολοσειράς σε συμβολοσειρά γίνεται με τη χρήση της συνάρτησης strcpy( ) και όχι με τη χρήση του τελεστή απόδοσης τιμής (=).
- Η γραμμή 5 είναι λάθος, διότι το frost.memb που βρίσκεται στα αριστερά του name είναι δομή και όχι δείκτης σε δομή, άρα είναι λάθος η χρήση του τελεστή βέλους. Και επιπλέον φυσικά υπάρχει το λάθος που αναφέρθηκε παραπάνω σχετικά με την αντιγραφή μιας συμβολοσειράς σε άλλη.
- Η γραμμή 6 είναι λάθος, διότι το sand->memb που βρίσκεται στα αριστερά του name είναι δομή και όχι δείκτης σε δομή, άρα είναι λάθος η χρήση του τελεστή βέλους πριν το name. Επίσης και εδώ υπάρχει το λάθος ως προς τον τρόπο αντιγραφής μιας συμβολοσειράς σε άλλη.

3.8. ΕΝΩΣΕΙΣ.

Χρησιμοποιούνται όπως και οι δομές για την **ομαδοποίηση μεταβλητών διαφορετικών τύπων**. Όπως ξέρουμε, οι δομές μας επιτρέπουν να χειριζόμαστε σα μια μονάδα ένα πλήθος διαφορετικών μεταβλητών, αποθηκευμένων σε διαφορετικές θέσεις μνήμης. Οι ενώσεις μας επιτρέπουν να χειριζόμαστε **τον ίδιο χώρο μνήμης σαν** ένα πλήθος από **διαφορετικές μεταβλητές**, δηλαδή **ένα κομμάτι μνήμης αντιμετωπίζεται ως μεταβλητή ενός τύπου σε μια περίπτωση και μεταβλητή διαφορετικού τύπου σε άλλη περίπτωση**. Έτσι, μόνο ένα από τα μέλη (τα πεδία) της ένωσης μπορεί να χρησιμοποιηθεί κάθε φορά.

Μια ένωση περιγράφεται με τον ίδιο τρόπο που περιγράφεται και μια δομή, με τη διαφορά ότι **χρησιμοποιείται η λέξη-κλειδί union**, αντί για την λέξη-κλειδί struct. Η δήλωση εξ άλλου μιας μεταβλητής ένωσης γίνεται με αντίστοιχο τρόπο με αυτόν που γίνεται η δήλωση μιας μεταβλητής δομής.

Στο παρακάτω παράδειγμα το μέγεθος των ενώσεων του είδους mixed είναι **4 byte**, παρά το ότι περιέχει ένα ακέραιο και ένα float. Αυτό γιατί, οι δυο μεταβλητές

καταλαμβάνουν τον ίδιο χώρο στη μνήμη, οπότε τελικά **δεσμεύεται ο χώρος που χρειάζεται η μεγαλύτερη** από αυτές. Στο παράδειγμα αυτό, στη γραμμή 13, το πρόγραμμα θα εμφανίσει στην οθόνη «σκουπίδια», διότι θα προσπαθήσει να γράψει ένα float σαν ακέραιο.

Παράδειγμα:

```
void main(void)
{
  union mixed
  {
      int incase;
      float flocase;
  } unmet;
  printf ("Μέγεθος ενότητας = %5d byte\n", sizeof(union mixed));
  unmet.incase = 159;
  printf ("Ακέραια μεταβλητή = %5d\n", unmet.incase);
  scanf ("%f", &unmet.flocase);
  printf ("Float μεταβλητή = %8.2f\n", unmet.flocase);
  printf ("Ακέραια μεταβλητή = %5d\n", unmet.incase);        /* 13 */
}
```

Με τις ενώσεις χρησιμοποιείται ο τελεστής -> με τον ίδιο τρόπο που χρησιμοποιείται και με τις δομές.