

Κεφάλαιο

13

Τύποι δεδομένων οριζόμενοι από το χρήστη



Τύποι δεδομένων οριζόμενοι από το χρήστη (User-Defined data types)

Η C παρέχει πέντε διαφορετικούς τρόπους για να δημιουργήσουμε δικούς μας τύπους δεδομένων (πέρα από τους βασικούς τύπους που έχουμε ήδη γνωρίσει). Οι οριζόμενοι από το χρήστη τύποι δεδομένων (user defined data types) ανήκουν στις επόμενες κατηγορίες:

- **Δομές (Structures)**
Οι δομές είναι συλλογές από μεταβλητές, οι οποίες αναφέρονται με κοινή ονομασία.
- **Πεδία εύρους ενός (ή περισσότερων) bit (Bit fields)**
Είναι μια παραλλαγή δομής η οποία επιτρέπει εύκολη προσπέλαση στα bit ενός byte.
- **Ενώσεις (Unions)**
Οι ενώσεις επιτρέπουν σε δύο ή περισσότερες μεταβλητές, ακόμη και διαφορετικού τύπου, να μοιράζονται το ίδιο τμήμα μνήμης.
- **Απαριθμήσεις (Enumerations)**
Οι απαριθμήσεις είναι λίστες συμβόλων.
- **Typedef**
Η χρήση της **typedef** δημιουργεί ένα νέο όνομα για έναν **υπάρχοντα** τύπο δεδομένων. **Η typedef δεν δημιουργεί νέο τύπο δεδομένων.**

Δομές (structures)

Μια δομή στη C είναι μια συλλογή από μεταβλητές (ή πίνακες) κάθε τύπου, οι οποίες αναφέρονται με κοινό όνομα. Η δημιουργία μιας δομής δίνει τη δυνατότητα να οριστούν μεταβλητές αυτού του τύπου.

Μια δομή αποτελείται από επιμέρους πληροφορίες, οι οποίες καλούνται πεδία.

Μια δομή είναι συνήθως μια λογική ενότητα από πληροφορίες που σχετίζονται μεταξύ τους. Για παράδειγμα, το όνομα, η διεύθυνση, το τηλέφωνο, και η ηλι-

κία ενός ατόμου είναι ένα χαρακτηριστικό παράδειγμα μιας δομής με τέσσερα πεδία:

- το πεδίο του ονόματος,
- το πεδίο της διεύθυνσης,
- το πεδίο του τηλεφώνου, και
- το πεδίο της ηλικίας

Μια δομή ορίζεται με την πρόταση **struct**. Το επόμενο παράδειγμα ορίζει μία δομή με όνομα **stoxeia** με τα τέσσερα παραπάνω πεδία:

```
struct stoxeia
```

```
{
```


```
    char onoma[15];
```

```
    char address[20];
```

```
    char thl[13];
```

```
    int ilikia;
```

```
};
```

 Με την προηγούμενη πρόταση **δεν** δημιουργήθηκε καμία μεταβλητή, ούτε δεσμεύτηκε καθόλου μνήμη. Απλώς ορίστηκε ένας νέος τύπος δεδομένων με όνομα **stoxeia**. **ΠΡΟΣΟΧΗ:** η δομή **stoxeia** δεν είναι μεταβλητή, αλλά ένας τύπος δεδομένων όπως ο **char**, ο **int** κ.λπ.

 Το τέλος της δήλωσης **struct** οριοθετείται με ελληνικό ερωτηματικό (;).

Η δήλωση μεταβλητών της δομής stoxeia μπορεί να γίνει με δύο τρόπους:

α. Με την επόμενη πρόταση δήλωσης:

```
struct stoxeia pelatis,filos;
```

όπου ορίζονται δύο μεταβλητές του τύπου **stoxeia**, με ονόματα **pelatis** και **filos** αντίστοιχα.

β. Μαζί με τη δήλωση του τύπου:

```
struct stoxeia
```

```
{
```

```
    char onoma[15];
```

```
    char address[20];
```

```
    char thl[13];
```



```
int ilikia;
} pelatis, fillos;
```

όπου πάλι ορίζονται οι δύο μεταβλητές **pelatis** και **fillos**.

γ. Δηλώνοντας μόνο μεταβλητές αλλά όχι τύπο:

```
struct
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} pelatis, fillos;
```

όπου δηλώνονται οι δύο μεταβλητές με την προηγούμενη δομή, αλλά δεν δημιουργείται νέος τύπος δεδομένων (όπως ο **stoixeia**). Το επακόλουθο είναι να μην υπάρχει αργότερα η δυνατότητα να δηλωθούν με μια ξεχωριστή πρόταση μεταβλητές με την ίδια δομή.

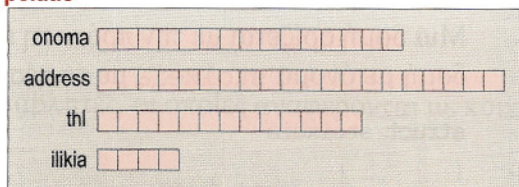
Με όποιον τρόπο και αν δηλωθούν, η κάθε μία από τις μεταβλητές **pelatis** και **fillos** καταλαμβάνει χώρο 52 byte στη μνήμη (15+20+13+4).

Ο τελεστής **sizeof** αποδίδει το συνολικό μέγεθος μιας δομής (σε byte) —όπως ακριβώς και στους άλλους τύπους δεδομένων.

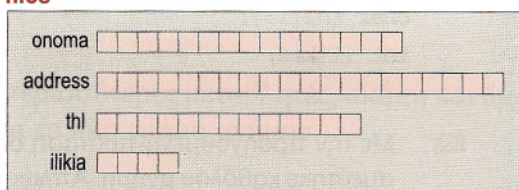
Η γενική μορφή δήλωσης ενός τύπου δομής είναι:

```
struct ονομασία.τύπου
{
    τύπος όνομα.πεδίου;
    τύπος όνομα.πεδίου;
    τύπος όνομα.πεδίου;
    .....
    τύπος όνομα.πεδίου;
} [μεταβλητές.δομής];
```

pelatis



fillos



Η δήλωση μιας δομής μπορεί να είναι καθολική (πριν από τη `main()`), οπότε ο ίδιος ο τύπος —αλλά και οι μεταβλητές που δηλώνονται μαζί του— έχουν εμβέλεια σε όλο το πρόγραμμα.

```
struct stoixeia
```

```
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
}filos;
```

Ο τύπος `stoixeia` έχει εμβέλεια σε όλο το πρόγραμμα.

Η μεταβλητή δομής `filos` έχει εμβέλεια σε όλο το πρόγραμμα.

```
main()
```

```
{
    struct stoixeia pelatis;
    .....
    .....
}
```

Η μεταβλητή δομής `pelatis` έχει εμβέλεια μόνο μέσα στη `main()`.

Όταν μια δομή δηλωθεί τοπικά, τότε ο τύπος της δομής είναι γνωστός μόνο μέσα στη συνάρτηση (ή σύνθετη πρόταση) στην οποία δηλώθηκε.

```
main()
```

```
{
    struct stoixeia
    {
        char onoma[15];
        char address[20];
        char thl[13];
        int ilikia;
    } pelatis;
    .....
    .....
}
```

Ο τύπος `stoixeia` έχει εμβέλεια μόνο μέσα στη `main()`.

Η μεταβλητή δομής `pelatis` έχει εμβέλεια μόνο μέσα στη `main()`.

Αναφορά στα πεδία μιας δομής

Η αναφορά στα πεδία μιας μεταβλητής δομής γίνεται μέσω του τελεστή **τελεία** (.). Έτσι, η επόμενη πρόταση,

```
pelatis.ilikia = 36;
```

αποθηκεύει τον αριθμό 36 στο πεδίο της ηλικίας της μεταβλητής **pelatis** (όπως έχει οριστεί στο προηγούμενο παράδειγμα).

Γενικά, η αναφορά σε ένα πεδίο μιας μεταβλητής δομής γίνεται χρησιμοποιώντας το όνομα της μεταβλητής ακολουθούμενο από μία τελεία και την ονομασία του πεδίου: **μεταβλητή_δομής.ονομασία_πεδίου**.

Ας θεωρήσουμε το προηγούμενο παράδειγμα με το οποίο δηλώνονται δύο μεταβλητές **pelatis** και **filos** που είναι τύπου **stoixeia** και την εξής δομή:

```
struct stoixeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} pelatis, filos;
```

Οι επόμενες προτάσεις έχουν αποτέλεσμα την καταχώριση των χαρακτήρων όπως εμφανίζονται στο διπλανό σχήμα:

```
strcpy(pelatis.onoma, "ΠΕΤΡΟΥ");
strcpy(pelatis.thl, "2651821");
strcpy(filos.onoma, "ΝΙΚΟΣ");
```

Η επόμενη πρόταση ζητάει να πληκτρολογηθεί η διεύθυνση του πελάτη και την αποθηκεύει στο πεδίο **address**:

```
gets(pelatis.address);
```

pelatis

onoma	Π Ε Τ Ρ Ο Υ \0
address	
thl	2 6 5 1 8 2 1 \0
ilikia	

filos

onoma	Ν Ι Κ Ο Σ \0
address	
thl	
ilikia	

Ο επόμενος κώδικας εμφανίζει στην οθόνη τους χαρακτήρες της διεύθυνσης του πελάτη **έναν-έναν**:

```
int t;
for (t=0; pelatis.address[t]!='\0'; t++) putchar(pelatis.address[t]);
```

Πίνακες από δομές

Με τον ίδιο τρόπο που δηλώνεται μία μεταβλητή τύπου δομής μπορεί να δηλωθεί και ένας πίνακας:

```
struct stoixeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} pelates[100];
```

Μετά από την παραπάνω δήλωση δημιουργείται ένας πίνακας 100 θέσεων με όνομα **pelates**, ο οποίος αποτελείται από 100 δομές τύπου **stoixeia**.

Η πρόσβαση στις διαφορετικές δομές που αποτελούν τον πίνακα γίνεται όπως σε όλους τους πίνακες: Για παράδειγμα, το **pelates[2]** αναφέρεται στην τρίτη δομή του πίνακα.

Η πρόσβαση σε κάποιο πεδίο μιας δομής γίνεται με τη βοήθεια του τελεστή τελεία (.) όπως στις απλές μεταβλητές δομής: το **pelates[2].ilikia** αναφέρεται στο πεδίο **ilikia** της τρίτης δομής του πίνακα.

Πίνακας **pelates**

pelates[0]

onoma	
address	
thl	
ilikia	

pelates[1]

onoma	ΠΕΤΡΟΥ
address	
thl	265182110
ilikia	

pelates[2]

onoma	
address	
thl	
ilikia	

onoma	
address	
thl	
ilikia	

pelates[98]

onoma	TOTTALE
address	
thl	
ilikia	

pelates[99]

onoma	
address	
thl	
ilikia	

Οι παρακάτω προτάσεις έχουν αποτέλεσμα την καταχώριση των χαρακτήρων όπως εμφανίζονται στο σχήμα της προηγούμενης σελίδας.

```
strcpy(pelates[1].onoma, "ΠΕΤΡΟΥ");  
strcpy(pelates[1].thl, "2651821");  
strcpy(pelates[98].onoma, "ΤΟΤΤΑΣ");
```

Ο επόμενος κώδικας καταχωρίζει στο πεδίο της διεύθυνσης και της ηλικίας (στο σύνολο των 100 δομών) το "ΜΥΤΙΑΗΝΗ" και το 30 αντίστοιχα.

```
for(i=0; i<100; i++)  
{  
    strcpy(pelates[i].address, "ΜΥΤΙΑΗΝΗ");  
    pelates[i].ilikia=30;  
}
```

Μεταβίβαση των πεδίων μιας δομής σε μια συνάρτηση

Μπορούμε να μεταβιβάζουμε τα πεδία μιας δομής σε μια συνάρτηση όπως κάνουμε με οποιονδήποτε άλλον τύπο δεδομένων.

Έστω, π.χ. μια μεταβλητή με όνομα `first` του τύπου δομής `test`:

```
struct test  
{  
    char x;  
    int y;  
    float z;  
    char s[10];  
} first;
```

Αν θεωρήσουμε μία συνάρτηση με όνομα `func()`, οι επόμενες κλήσεις της συνάρτησης μεταβιβάζουν τις τιμές του ορίσματος στην τυπική παράμετρο⁸ της συνάρτησης ως εξής:

```
func(first.x);      ➡ μεταβιβάζει την τιμή του πεδίου x
```

⁸ Υποθέτουμε ότι η τυπική παράμετρος της `func()` έχει τον κατάλληλο τύπο.

`func(first.y);` ⇒ μεταβιβάζει την τιμή του πεδίου `y`
`func(first.z);` ⇒ μεταβιβάζει την τιμή του πεδίου `z`
`func(first.s);` ⇒ μεταβιβάζει τη διεύθυνση του πεδίου `s`
`func(first.s[3]);` ⇒ μεταβιβάζει την τιμή του 4ου χαρακτήρα του πεδίου `s`

Όταν θέλουμε να μεταβιβάσουμε τη διεύθυνση ενός πεδίου της δομής χρησιμοποιείται ο τελεστής `&`:

`func(&first.x);` ⇒ μεταβιβάζει τη διεύθυνση του πεδίου `x`
`func(&first.y);` ⇒ μεταβιβάζει τη διεύθυνση του πεδίου `y`
`func(&first.z);` ⇒ μεταβιβάζει τη διεύθυνση του πεδίου `z`
`func(first.s);` ⇒ μεταβιβάζει τη διεύθυνση του πεδίου `s`
`func(&first.s[3]);` ⇒ μεταβιβάζει τη διεύθυνση του 4ου χαρακτήρα του πεδίου `s`

- 🔍 Παρατηρούμε ότι το `&` προηγείται του ονόματος της δομής και όχι του πεδίου.
- 🔍 Τα πεδία που αναφέρονται σε πίνακα (όπως το `first.s`) δεν χρειάζονται τον τελεστή `&`, δεδομένου ότι ένας πίνακας φανερώνει, έτσι ή αλλιώς, μια διεύθυνση. Αν χρησιμοποιηθεί είναι λάθος.

Μεταβίβαση ολόκληρης της δομής σε συνάρτηση

Όταν χρησιμοποιείται ως παράμετρος μιας συνάρτησης μια ολόκληρη δομή, τότε οι τιμές όλων των πεδίων της δομής μεταβιβάζονται στα αντίστοιχα πεδία της τυπικής παραμέτρου της συνάρτησης. Αυτό φυσικά προϋποθέτει ότι η τυπική παράμετρος της συνάρτησης θα έχει την ίδια δομή με το όρισμα με το οποίο καλείται.

Στο παράδειγμα που ακολουθεί, η τυπική παράμετρος `p` είναι του ιδίου τύπου δομής (`test`) με το όρισμα `first` με το οποίο καλείται η συνάρτηση `func()`.

```
struct test
{
    char x;
    int y;
```

```

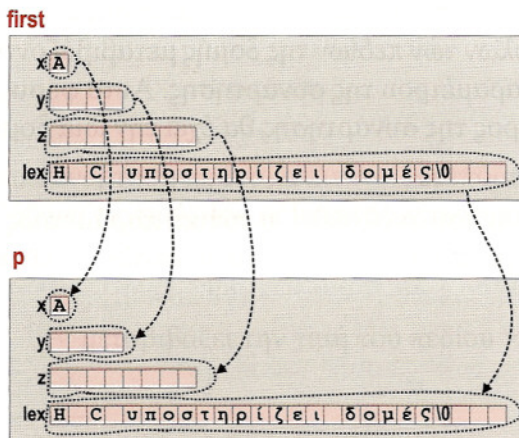
    double z;
    char lex[25];
};

main()
{
    struct test first;
    first.x='A';
    first.y=120;
    first.z=3.145678;
    strcpy(first.lex,"Η C υποστηρίζει δομές");
    func(first);
}

func(p)
struct test p;
{
    printf("%c\n",p.x);
    printf("%d\n",p.y);
    printf("%f\n",p.z);
    printf("%s\n",p.lex);
}

```

Οι τιμές των πεδίων της **first** αντιγράφονται στα αντίστοιχα πεδία της **p**:

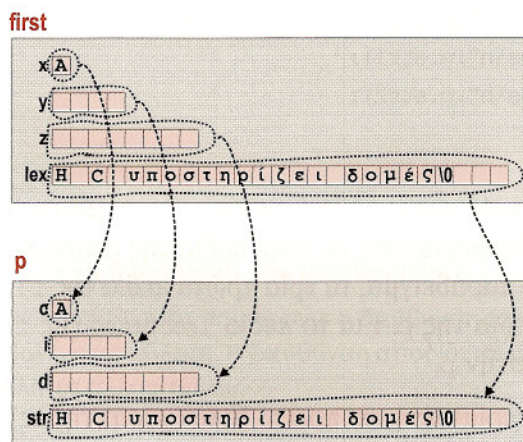


Η τυπική παράμετρος **p** θα μπορούσε να δηλωθεί ως δομή χωρίς τύπο με ίδιους τύπους και μεγέθη πεδίων, αλλά με διαφορετικά ονόματα:

```
func(p)
struct
{
    char c;
    int i;
    double d;
    char str[25];

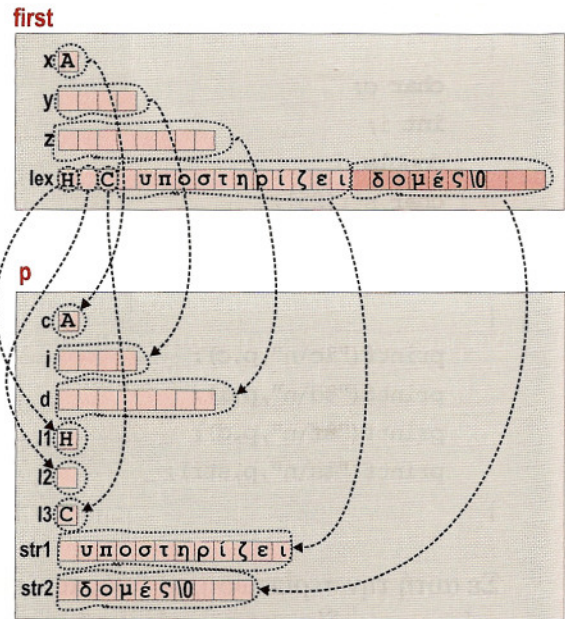
}p;
{
    printf("%c\n",p.c);
    printf("%d\n",p.i);
    printf("%f\n",p.d);
    printf("%s\n",p.str);
}
```

Σε αυτή την περίπτωση, οι τιμές των πεδίων της **first** αντιγράφονται στα αντίστοιχα πεδία της **p** χωρίς κανένα πρόβλημα.



Η τυπική παράμετρος **p** θα μπορούσε ακόμη να δηλωθεί σαν δομή με διαφορετικούς τύπους και μεγέθη πεδίων:

```
func(p)
struct
{
    char c;
    int i;
    double d;
    char l1;
    char l2;
    char l3;
    char str1[12];
    char str2[10]
};
{
    printf("%c\n",p.c);
    printf("%d\n",p.i);
    printf("%f\n",p.d);
    printf("%c\n",p.l1);
    printf("%c\n",p.l2);
    printf("%c\n",p.l3);
    printf("%s\n",p.str1);
    printf("%s\n",p.str2);
}
```



Σε αυτή την περίπτωση, τα 38 byte της **first** αντιγράφονται ένα-ένα στην **p**, στην ουσία αδιαφορώντας, σε ποιο πεδίο της **p** καταλήγουν. Επομένως, για το συγκεκριμένο παράδειγμα, τα τρία πρώτα πεδία της **first** θα αντιγραφούν στα τρία πρώτα πεδία της **p**. Για το πεδίο **lex** όμως της **first**, δεν υπάρχει αντίστοιχο πεδίο στην **p**.

Σε αυτή την περίπτωση, το πρώτο byte (χαρακτήρας) της **lex** θα αντιγραφεί στο πεδίο **l1** της **p**, το δεύτερο byte (χαρακτήρας) της **lex** θα αντιγραφεί στο

πεδίο 12 της `p`, το τρίτο byte της `lex` στο πεδίο 13 της `p`, τα επόμενα 12 byte στο πεδίο `str1` και τα υπόλοιπα 10 στο `str2`.

Δείκτες σε δομές

Η C επιτρέπει δείκτες σε δομές όπως ακριβώς και σε οποιονδήποτε άλλο τύπο μεταβλητής.

Στο επόμενο παράδειγμα φαίνεται η δήλωση μιας μεταβλητής δείκτη `ptr` σε δομές τύπου `test`.

```
struct test
```

```
{
```

```
    char x;
```

```
    int y;
```

```
    double z;
```

```
    char lex[12];
```

```
};
```

```
main()
```

```
{
```

```
    struct test first, *ptr;
```

```
    first.x='A';
```

```
    strcpy(first.lex, "Μυτιλήνη");
```


```
    ptr=&first;
```

```
}
```

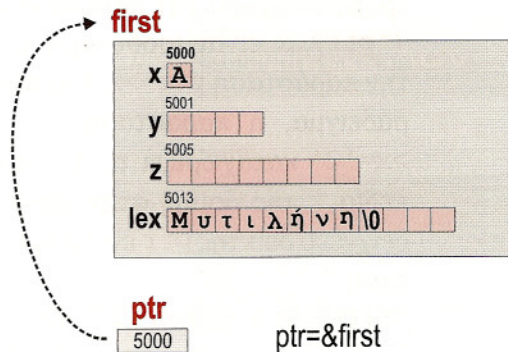
Με την πρόταση

```
ptr=&first;
```

η `ptr` παίρνει ως τιμή τη διεύθυνση της μεταβλητής `first`.

 Η διεύθυνση μιας μεταβλητής δομής είναι η διεύθυνση του πρώτου από τα byte που καταλαμβάνει η μεταβλητή. Η διεύθυνση αποδίδεται με τον τελεστή `&` όπως και στις μεταβλητές άλλου τύπου.

Η πρόσβαση στα πεδία μιας μεταβλητής δομής μέσω ενός δείκτη γίνεται με δύο τρόπους:



- Με χρήση του τελεστή *. Έτσι, η `(*ptr).x` αναφέρεται στο πεδίο `x` της μεταβλητής `first`, π.χ. η

`(*ptr).x='C';`

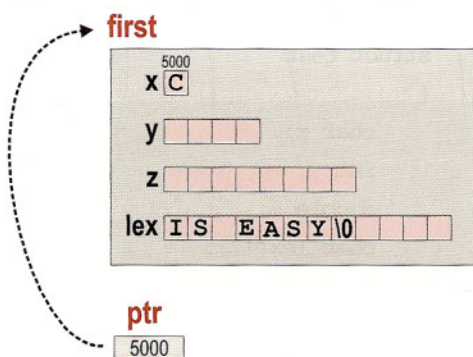
καταχωρίζει το 'C' στο πεδίο `x` της δομής που δείχνει ο δείκτης `ptr`.

Οι παρενθέσεις είναι υποχρεωτικές επειδή ο τελεστής τελεία (.) έχει μεγαλύτερη προτεραιότητα από τον (*). Ο τελεστής (*) σπάνια χρησιμοποιείται όταν πρόκειται για δείκτες σε δομές. Ο συνήθης τρόπος αναφοράς σε δομή μέσω δείκτη γίνεται με χρήση του τελεστή βέλους `->`.

- Με τον τελεστή βέλους `->`

Ο τελεστής βέλους `->` (παύλα και `>`) αντικαθιστά τον κλασικό τρόπο αναφοράς: Η παράσταση `(*ptr).x` είναι ισοδύναμη με την παράσταση `ptr->x`. Για παράδειγμα, η παράσταση `ptr->x='C'` καταχωρίζει το 'C' στο πεδίο `x` της δομής στην οποία δείχνει ο δείκτης `ptr` ενώ η πράξη

`strcpy(ptr->lex, "IS EASY")` καταχωρίζει τους χαρακτήρες "IS EASY" στο πεδίο `lex` της δομής στην οποία δείχνει ο δείκτης `ptr`.



- ☞ Χρησιμοποιούμε τον τελεστή τελεία (.) όταν θέλουμε να αναφερθούμε απευθείας σε ένα πεδίο μιας μεταβλητής δομής και τον τελεστή βέλους (`->`) όταν θέλουμε να αναφερθούμε σε ένα πεδίο μιας μεταβλητής δομής μέσω ενός δείκτη.

Η αριθμητική των δεικτών σε δομές ακολουθεί τη γενικότερη λογική της αριθμητικής των δεικτών που ισχύει και στους άλλους τύπους δεδομένων.

Η αύξηση ενός δείκτη (που δείχνει σε δεδομένα τύπου δομής) κατά 1, αυξάνει την τιμή του δείκτη τόσο, όσο το μέγεθος του συγκεκριμένου τύπου δομής σε byte.

Έτσι, στο συγκεκριμένο παράδειγμα ο δείκτης `ptr` δείχνει σε δεδομένα τύπου δομής `test`, το μέγεθος της οποίας είναι 25 byte (1+4+8+12). Επομένως μια πρόταση `ptr++`, θα έχει ως αποτέλεσμα την αύξηση της τιμής του δείκτη `ptr` κατά 25!

Πίνακες από δομές και δείκτες

Ο ορισμός ενός πίνακα από δομές δημιουργεί ταυτόχρονα και ένα δείκτη με όνομα το όνομα του πίνακα, ο οποίος δείχνει στο πρώτο στοιχείο (δομή) του πίνακα.

Στο διπλανό σχήμα, ο δείκτης `res` δείχνει στην πρώτη θέση του πίνακα `res[0]`, το `res+1` στη δεύτερη θέση `res[1]`, κ.ο.κ.

Η πρόσβαση στα στοιχεία του πίνακα μπορεί να γίνει με δύο τρόπους. Είτε με τη συνήθη χρήση των πινάκων, είτε με τη χρήση της λογικής των δεικτών.

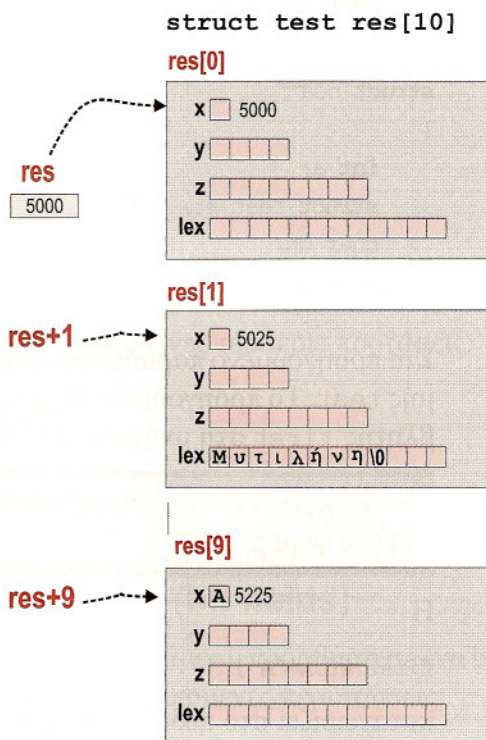
Οι προτάσεις π.χ.

```
strcpy(res[1].lex, "Μυτιλήνη");
res[9].x='A';
```

και οι προτάσεις

```
strcpy((res+1)->lex, "Μυτιλήνη");
(res+9)->x='A';
```

έχουν ακριβώς το ίδιο αποτέλεσμα. Και στις δύο περιπτώσεις η καταχώριση των χαρακτήρων γίνεται όπως στο προηγούμενο σχήμα.



Δομές μέσα σε δομές

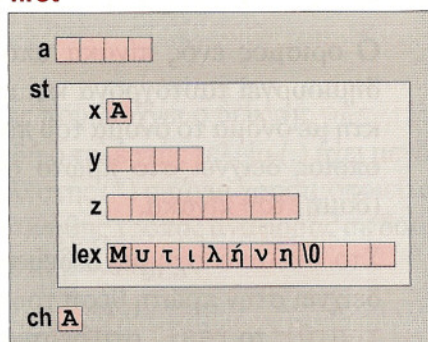
Ένα πεδίο μιας δομής μπορεί να είναι μια άλλη δομή.

```
struct test
{
    char x;
    int y;
```

```
double z;
char lex[12];
};

struct nea
{
    int a;
    struct test st;
    char ch;
} first;
```

first



Στο προηγούμενο παράδειγμα, ένα πεδίο της δομής **nea**, το **st**, είναι τύπου δομής **test**. Το προηγούμενο σχήμα δείχνει την εσωτερική απεικόνιση της μεταβλητής **first** στη μνήμη του H/Y.

Η αναφορά σε ένα πεδίο μιας ένθετης δομής γίνεται με τη χρήση του τελεστή τελεία (.) δύο φορές:

Η

```
first.st.x='A';
```

καταχωρίζει στο πεδίο **x** της ένθετης δομής το 'A'

ενώ η

```
strcpy(first.st.lex, "Μυτιλήνη");
```

καταχωρίζει τους χαρακτήρες "Μυτιλήνη" στο πεδίο **lex** της ένθετης δομής **st** και η,

```
first.ch='A';
```

καταχωρίζει στο πεδίο **ch** της δομής **first** το χαρακτήρα 'A'.

Και όμως γίνεται...

Όταν έχουμε δύο μεταβλητές του ίδιου τύπου δομής, μπορούμε να καταχωρίσουμε τα περιεχόμενα των πεδίων της μιας στην άλλη με μια πρόταση της μορφής:

```
st1=st2;
```

Για παράδειγμα, για τις μεταβλητές `st1` και `st2` που έχουν δηλωθεί όπως παρακάτω:

```
struct test
{
    char x;
    int y;
    double z;
    char lex[12];
}st1,st2;
```

η πρόταση `st1=st2` είναι ισοδύναμη με τις επόμενες τέσσερις προτάσεις:

```
st1.x=st2.x;
st1.y=st2.y;
st1.z=st2.z;
strcpy(st1.lex,st2.lex);
```

Πεδία εύρους ενός (ή περισσότερων) bit (bit fields)

Η C προσφέρει ενσωματωμένη δυνατότητα για την άμεση πρόσβαση σε ένα bit ενός byte. Η δυνατότητα αυτή είναι χρήσιμη στις επόμενες περιπτώσεις:

- Για λόγους οικονομίας μνήμης μπορούν μέσα σε ένα byte να αποθηκευθούν πολλές λογικές μεταβλητές (αλήθεια-1 ή ψέμα-0).
- Υπάρχουν περιφερειακές συσκευές που μεταδίδουν πληροφορίες κωδικοποιημένες σε bit μέσα σε ένα byte.
- Υπάρχουν μέθοδοι κωδικοποίησης που είναι αναγκαίο να έχουν πρόσβαση στα επιμέρους bit ενός byte.

Η μέθοδος που ακολουθεί η C για να έχει πρόσβαση στα bit ενός byte βασίζεται στις δομές.





Ο επόμενος τύπος δομής `test` ορίζει και τρία πεδία bit (bit fields) με ονόματα `odigos`, `andras`, και `kapnistis` αντίστοιχα.

```
struct test
{
    unsigned int odigos:1;
```



```
unsigned int andras:1;
unsigned int kapnistis:1;
char ch;
int a;
double mo;
}first;
```

Και τα τρία αυτά πεδία μαζί καταλαμβάνουν χώρο ενός byte (και μάλιστα "περισσεύουν" 5 bits). Επομένως, η μεταβλητή `first` της παραπάνω δομής καταλαμβάνει χώρο 14 byte (1+1+4+8).

-  Ο τύπος ενός πεδίου bit μπορεί να είναι είτε `int` είτε `unsigned int`.
-  Τα πεδία bit εύρους ενός bit **πρέπει** να είναι τύπου `unsigned int`.
-  Σε ένα πεδίο bit εύρους ενός bit μπορεί να αποθηκευτεί μόνο το 0 ή το 1.
-  Η πρόσβαση σε ένα πεδίο bit μιας δομής γίνεται κανονικά, όπως σε οποιοδήποτε άλλο πεδίο της: π.χ. `first.andras=1;`

Ένα πεδίο bit μπορεί να έχει εύρος περισσότερο από ένα bit.

Στο επόμενο παράδειγμα, το πεδίο `odigos` έχει εύρος 3 bit, το `andras` 4, και το `kapnistis` 2 bit:

```
struct test
{
    int odigos:3;
    int andras:4;
    int kapnistis:2;
    char ch;
    int a;
    float mo;
}first;
```

Σε αυτή την περίπτωση, τα τρία πεδία bit καταλαμβάνουν 2 byte και "περισσεύουν" 7 bit. Το συνολικό μέγεθος της `first` σε αυτή την περίπτωση είναι 11 byte (2+1+4+4).

Ο αριθμός των bit που απαρτίζουν ένα πεδίο bit, καθώς και ο τύπος του (`unsigned int` ή `int`) καθορίζει το μέγιστο ακέραιο αριθμό που μπορεί να καταχωριστεί σε αυτό το πεδίο.

Ενώσεις (unions)

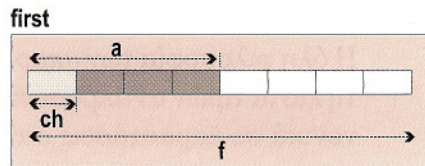
Μια **ένωση** (union) είναι στη C μια θέση στη μνήμη, η οποία χρησιμοποιείται από διαφορετικές μεταβλητές που μπορεί να είναι ακόμα και διαφορετικού τύπου.

Ο ορισμός μιας **ένωσης** γίνεται όπως και ο ορισμός μιας δομής:

```
union ονομασία.τύπου
{
    τύπος όνομα.πεδίου;
    τύπος όνομα.πεδίου;
    .....
} [μεταβλητές];
```

Στο επόμενο παράδειγμα ορίζεται ένας τύπος `union` και μία μεταβλητή αυτού του τύπου με όνομα `first`:

```
union test
{
    int a;
    double f;
    char ch;
} first;
```



Και τα τρία πεδία "μοιράζονται" τον ίδιο χώρο μνήμης, ξεκινώντας από την ίδια διεύθυνση. Η μεταβλητή `first` καταλαμβάνει 8 byte —όσο είναι το μεγαλύτερο σε μέγεθος πεδίο της (το `f` που είναι τύπου `double`). Μπορούμε όμως μέσω των στοιχείων `a` και `ch` να έχουμε πρόσβαση είτε στα 4 πρώτα bytes (με το `a`) είτε μόνο στο πρώτο byte (με το `ch`).

Με δεδομένη π.χ. τη μεταβλητή `first` όπως δηλώθηκε προηγουμένως,

- με την πρόταση `first.ch='A'` καταχωρίζουμε στο πρώτο από τα 8 byte της `first` το χαρακτήρα 'A'.
- με την πρόταση `first.a=345` καταχωρίζουμε στα πρώτα 4 byte της `first` τον ακέραιο αριθμό 345, αντικαθιστώντας τα περιεχόμενα των 4 πρώτων byte.
- με την πρόταση `first.f=123.3467821` καταχωρίζουμε στα 8 byte της `first` τον αριθμό διπλής ακρίβειας 123.3467821, αντικαθιστώντας τα περιεχόμενα και των 8 byte.

Απαριθμήσεις (Enumerations)

Ένας τύπος **απαρίθμησης** (enumeration) είναι ένα σύνολο από τιμές που μπορεί να πάρει μια μεταβλητή αυτού του τύπου.

Ο ορισμός μιας **απαρίθμησης** γίνεται ως εξής:

enum ονομασία.τύπου {τιμή1, τιμή2,} [μεταβλητές];

Για παράδειγμα, η

```
enum days {mon,tue,wed,thu,fri,sat,sun} birth_day,today;
```

Ορίζει δύο μεταβλητές `birth_day` και `today`, τύπου `days`, που μπορούν να πάρουν τιμές από τη λίστα τιμών της απαρίθμησης.

Η όλη φιλοσοφία των απαριθμήσεων στηρίζεται στο γεγονός ότι κάθε τιμή από τη λίστα τιμών αντιπροσωπεύει έναν ακέραιο αριθμό και μπορεί να μετέχει κανονικά σε παραστάσεις σαν μια **σταθερά** τύπου `int`.

Αν δεν ορίζεται διαφορετικά, το πρώτο σύμβολο της λίστας έχει τιμή 0, το δεύτερο 1, κ.ο.κ. Στο επόμενο τμήμα κώδικα η μεταβλητή `today` θα πάρει την τιμή `thu` (ουσιαστικά, το 3), η `birth_day` την `fri` (δηλαδή το 4) και, τέλος, η `today` θα πάρει την τιμή `wed` (2):

```
today=thu;  
birth_day=today+1;  
today++;
```

 Τα σύμβολα της λίστας τιμών πρέπει να είναι επιτρεπτά ονόματα της C.

Μπορούμε να αναθέσουμε σε κάποιο από τα σύμβολα μία συγκεκριμένη τιμή, οπότε η αρίθμηση των υπόλοιπων συμβόλων ξεκινάει από αυτή τη νέα τιμή:

```
enum days {mon,tue,wed,thu=12,fri,sat,sun} birth_day,today;
```


Τώρα, τα σύμβολα **thu**, **fri**, **sat** και **sun** αντιπροσωπεύουν τους ακέραιους 12, 13, 14, και 15 αντίστοιχα. Με το

```
int a;
```

```
a=fri;
```

η **a** θα πάρει τιμή 13.

 **ΠΡΟΣΟΧΗ:** Μια πρόταση της μορφής **fri=8;** είναι λάθος επειδή το σύμβολο **fri** δεν είναι μεταβλητή.

 Οι μεταβλητές τύπου **enum** είναι, στην ουσία, μεταβλητές τύπου **int**.

Η C δε δεσμεύεται για τις τιμές που μπορεί να πάρει μια μεταβλητή ενός τύπου **enum**. Για παράδειγμα, η πρόταση **today=23;** είναι απόλυτα σωστή από τη στιγμή που η **today** είναι ουσιαστικά μία μεταβλητή τύπου **int**.

Εύλογα αναδύεται τώρα η ερώτηση: Ποιο είναι το νόημα της λίστας των τιμών, αφού μπορούμε να δώσουμε όποια ακέραια τιμή θέλουμε σε μία μεταβλητή τύπου **enum**; Η απάντηση απλή: **Κανένα !!!** Απλώς το πρόγραμμα γίνεται πιο ευανάγνωστο.

Άλλο να γράφουμε π.χ.

```
if (today==mon) ... και άλλο
```

```
if (today==0) ...
```

Η χρήση της typedef

Η C επιτρέπει να ορίσουμε δικά μας ονόματα για τους βασικούς τύπους μεταβλητών χρησιμοποιώντας το διακριτικό **typedef**. Η σύνταξη της πρότασης **typedef** είναι:

typedef **τύπος** **όνομα**;

όπου **τύπος** ένας επιτρεπόμενος τύπος και **όνομα** το νέο όνομα αυτού του τύπου. Το νέο όνομα δεν καταργεί το παλιό αλλά χρησιμοποιείται επιπρόσθετα. Για παράδειγμα, η

```
typedef float pososto;
```

ορίζει το όνομα **pososto** για τον τύπο **float**, οπότε οι παρακάτω προτάσεις είναι εντελώς ισοδύναμες:

```
pososto a,b;
```

```
float a,b;
```

Παραδείγματα

- Π.1** Υποθέτουμε ότι έχουμε έναν πίνακα δομών με τη διπλανή γραμμογράφιση. Στον πίνακα είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Η επόμενη συνάρτηση εντοπίζει τη θέση μνήμης του πίνακα η οποία περιέχει τα στοιχεία του μαθητή με συγκεκριμένο όνομα. Η συνάρτηση επιστρέφει έναν δείκτη στη θέση (δομή) στην οποία εντόπισε το όνομα. Αν το όνομα δεν εντοπιστεί επιστρέφει τιμή NULL.

```
struct stoixeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];
```

```
struct stoixeia *find_onom(math, onoma)
```

```
struct stoixeia math[];
```

```
char onoma[];
```

```
{
```

```
    int i;
```

```
    for(i=0; i<100; i++)
```

```
    {
```

```
        if(strcmp(math[i].eponymo, onoma)==0)
```

```
            return &math[i];
```

```
    }
```

```
    return NULL;
```

```
}
```

Η strcmp() συγκρίνει για κάθε μια θέση μνήμης του πίνακα το πεδίο **eponymo** με το **onoma** που δόθηκε για αναζήτηση.

Αν εντοπιστεί το όνομα τότε επιστρέφει τη διεύθυνση της θέσης στην οποία βρέθηκε το όνομα, διαφορετικά επιστρέφει NULL.

Η κλήση της συνάρτησης για εντοπισμό ενός ονόματος μέσα στον πίνακα **mathites** γίνεται με τον παρακάτω κώδικα:

```
struct stoixeia *ptr;
char lex[30];
.....
printf("Δώσε όνομα για έρευνα:");
gets(lex);
ptr=find_onom(mathites,lex);
if(ptr==NULL)
{
    puts("Το όνομα δε βρέθηκε");
}
else
{
    puts("Στοιχεία μαθητή");
    puts("=====");
    printf("Επώνυμο:%s\n",ptr->eponymo);
    printf("Τάξη:%s\n",ptr->taxi);
    printf("ΜΟ:%f\n",ptr->mesos_oros);
    printf("Ηλικία:%d\n",ptr->ilikia);
}
```

Η συνάρτηση επιστρέφει ένα δείκτη ο οποίος καταχωρίζεται στη μεταβλητή δείκτη ptr.

Π.2 Υποθέτουμε ότι έχουμε τον πίνακα δομών **mathites** του προηγούμενου παραδείγματος. Ο επόμενος κώδικας εμφανίζει τα στοιχεία του μαθητή με το μεγαλύτερο μέσο όρο, καθώς και το μέσο όρο των ηλικιών των μαθητών.

```
int i,sum=0,thesi=0;
float max,mo_il;
.....
max=mathites[0].mesos_oros;
for(i=0;i<100;i++)
{
    if(mathites[i].mesos_oros>max)
    {
```

Θέτουμε στη max το μέσο όρο του πρώτου μαθητή (από την πρώτη θέση μνήμης του πίνακα).

Αν εντοπίσουμε μέσο όρο μεγαλύτερο από το max, τον καταχωρίζουμε στη θέση max. Επίσης, καταχωρίζουμε στη μεταβλητή thesi τη θέση (το i) που τον εντοπίσαμε.


```

        max=mathites[i].mesos_oros;
        thesi=i;
    }
    sum=sum+mathites[i].ilikia;
}
mo_il=sum/100.0;
puts("Στοιχεία μαθητή με το μεγαλύτερο μέσο όρο");
puts("=====");
printf("Επώνυμο:%s\n",mathites[thesi].eponymo);
printf("Τάξη:%s\n",mathites[thesi].taxi);
printf("Μέσος όρος %f\n",mathites[thesi].mesos_oros);
puts("=====");
printf("Μέσος όρος ηλικιών:%f\n",mo_il);

```

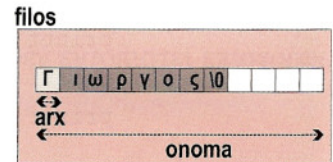
Στη θέση μνήμης sum προσθέτουμε μι-
α-μια τις ηλικίες των μαθητών. Τέλος
υπολογίζουμε το μέσο όρο των ηλικιών
διαίρωντας δια 100.

Π.3 Στο επόμενο πρόγραμμα η μεταβλητή **filos** είναι μεταβλητή τύπου **union stoixeia**. Το πεδίο **arx** και ο πρώτος χαρακτήρας του πεδίου **onoma** μοιράζονται την ίδια θέση μνήμης.

```

union stoixeia
{
    char arx;
    char onoma[10];
}filos;

```



```

main()
{
    gets(filos.onoma);
    printf("Όνομα:%s\n",filos.onoma);
    printf("Αρχικός χαρακτήρας:%c\n",filos.arx);
}

```

Το πεδίο **onoma** της μεταβλητής **filos**,
περιέχει το όνομα που πληκτρολογήσαμε,
ενώ το πεδίο **arx** περιέχει τον πρώτο του
χαρακτήρα,

Ανασκόπηση Κεφαλαίου 13

- Η δομή είναι ο καλύτερος τρόπος για να αποθηκεύσουμε πληροφορίες που σχετίζονται μεταξύ τους, π.χ. τα στοιχεία ενός ατόμου.
- Η δήλωση μιας δομής γίνεται με την πρόταση **struct**.
- Η πρόσβαση σε ένα πεδίο μιας δομής γίνεται με τον τελεστή τελεία (.).
- Ένας δείκτης σε μια δομή περιέχει τη διεύθυνση του πρώτου byte του πρώτου πεδίου της δομής.
- Η πρόσβαση σε ένα πεδίο μιας δομής μέσω ενός δείκτη γίνεται με τον τελεστή βέλους →.
- Ένα πεδίο μιας δομής μπορεί να είναι μια άλλη δομή.
- Ένα πεδίο μιας δομής μπορεί να έχει μέγεθος συγκεκριμένα bit (πεδία bit).
- Μια ένωση αποτελείται από πεδία τα οποία μοιράζονται τις ίδιες θέσεις μνήμης.
- Μια απαρίθμηση δηλώνει τιμές τις οποίες μπορούν να λάβουν μεταβλητές αυτού του τύπου.
- Οι τιμές μιας απαρίθμησης είναι στην ουσία αριθμητικές σταθερές.
- Η **typedef** δηλώνει ένα "ψευδώνυμο" για έναν υπάρχοντα τύπο δεδομένων.

Ασκήσεις Κεφαλαίου 13

- 13.1 Να γραφεί πρόγραμμα το οποίο να καταχωρίζει σε μία μεταβλητή δομής τα στοιχεία ενός αυτοκινήτου (αριθμός κυκλοφορίας, χρώμα, κατασκευαστής, κυβικά, ιπποδύναμη). Να επιλέξετε εσείς το σωστό τύπο δομής που πρέπει να χρησιμοποιήσετε. ★ ★
- 13.2 Να γραφεί πρόγραμμα το οποίο να καταχωρίζει σε έναν πίνακα από δομές τα στοιχεία μέχρι 100 το πολύ αυτοκινήτων (αριθμός κυκλοφορίας, χρώμα, κατασκευαστής, κυβικά, ιπποδύναμη). Να επιλέξετε εσείς το σω-

στό τύπο δομής που πρέπει να χρησιμοποιήσετε. Η διαδικασία καταχώρισης να σταματάει όταν δοθεί κενό για τον αριθμό κυκλοφορίας. ★ ★ ★

- 13.3** Υποθέτουμε ότι έχουμε έναν πίνακα δομών με τη διπλανή γραμμογράφιση. Στον πίνακα αυτό είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Να γραφεί συνάρτηση η οποία να εμφανίζει τη λίστα των μαθητών. Η κάθε γραμμή να περιέχει το επώνυμο, την τάξη, και το μέσο όρο του κάθε μαθητή (με ένα δεκαδικό ψηφίο). ★ ★

```
struct stoxeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];
```

- 13.4** Να γραφεί ακόμη μια συνάρτηση, σύμφωνα με τα δεδομένα της προηγούμενης άσκησης, η οποία να επιστρέφει ως τιμή τη μεγαλύτερη από τις ηλικίες των 100 μαθητών. ★ ★

- 13.4** Εντοπίστε τα λάθη στο επόμενο πρόγραμμα: ★

```
enum days {mon,tue,wed,thu,fri,sat,sun} birth_day,today;
typedef int meres;
main()
{
    meres a,b;
    days my_date;
    a=1;
    my_date=mon;
    wed=5;
    meres=23;
    today=3;
    birth_day=fri;
}
```


13.5 Ποια από τα επόμενα αληθεύουν: ★

- ❑ Μια δομή δεν μπορεί να περιέχει δύο πεδία με το ίδιο όνομα.
- ❑ Σε μια μεταβλητή τύπου `enum` δεν μπορούμε να καταχωρίσουμε άλλη τιμή πέρα από τις τιμές που ορίσαμε στη δήλωση του τύπου `enum` στον οποίο ανήκει.
- ❑ Για να έχουμε πρόσβαση στο πεδίο μιας δομής μέσω ενός δείκτη χρησιμοποιούμε τον τελεστή βέλους `→`.
- ❑ Όταν μεταβιβάζουμε μια δομή ως παράμετρο σε μια συνάρτηση, η παράμετρος **πρέπει** να είναι του ίδιου τύπου δομής με το όρισμα που θα της μεταβιβάσουμε.
- ❑ Όταν έχουμε δύο μεταβλητές του ίδιου τύπου δομής, μπορούμε με τον τελεστή `=` να καταχωρίσουμε όλα τα περιεχόμενα της μιας στην άλλη. Για παράδειγμα, η `filos=pelatis` καταχωρίζει όλα τα πεδία της μεταβλητής δομής `pelatis` στα αντίστοιχα πεδία της μεταβλητής δομής `filos`.