C: Από τη Θεωρία στην Εφαρμογή

Κεφάλαιο 13ο
Δομές & Ενώσεις

Γ. Σ. Τσελίκης – Ν. Δ. Τσελίκας

Δομές (Structures)

- **Δομή (structure) στη C είναι μία συλλογή πεδίων, που συνήθως χρησιμοποιούνται για την ομαδοποίηση πληροφορίας που περιγράφει μία λογική οντότητα**

Π.χ. μία δομή μπορεί να περιέχει πληροφορίες για μία εταιρεία, όπως την επωνυμία της, το έτος ίδρυσης, το Α.Φ.Μ, τη διεύθυνσή της, το αντικείμενο εργασιών της, τον αριθμό των υπαλλήλων της, στοιχεία επικοινωνίας, κτλ...

Δήλωση Δομής (Ι)

- Μία δήλωση `struct` δημιουργεί έναν τύπο, ο οποίος ορίζεται από τον προγραμματιστή (user-defined type)
- Η γενική μορφή δήλωσης μίας δομής φαίνεται παρακάτω:

```
struct ετικέτα_δομής
{
        Δηλώσεις πεδίων;
} λίστα_μεταβλητών;
```

- Αν και η `ετικέτα_δομής` είναι προαιρετική, εμείς θα ονοματίζουμε κάθε τύπο δομής που δημιουργούμε, ώστε να μπορούμε να χρησιμοποιούμε την ετικέτα, όποτε θέλουμε, για να δηλώνουμε αντίστοιχες μεταβλητές

- Παραδείγματα δηλώσεων δομών:

```
struct Date
{
        int day;
        int month;
        int year;
};
```

```
struct Person
{
        char name[50];
        int tax_num;
        char addr[50];
};
```

```
struct Company
{
        char name[50];
        int start_year;
        int field;
        int tax_num;
        int num_empl;
        char addr[50];
        float balance;
};
```

Δήλωση Δομής (ΙΙ)

- Όπως φαίνεται από τα προηγούμενα παραδείγματα, η χρησιμότητα μίας δομής είναι η ομαδοποίηση διαφορετικού είδους πληροφορίας για την περιγραφή μίας οντότητας
- Το πόσα και ποια πεδία θα περιέχονται σε μία δομή εξαρτάται από τον προγραμματιστή και τους σκοπούς του προγράμματος
- Το πρώτο γράμμα της ετικέτας συνηθίζεται να είναι κεφαλαίο
- Αν, για παράδειγμα, το πρόγραμμα απαιτεί να υπάρχει ένα πεδίο που να δηλώνει το βάρος του ανθρώπου και ένα άλλο πεδίο που να δηλώνει το ύψος του, τότε η δομή **Person** του προηγούμενου παραδείγματος θα μπορούσε π.χ. να έχει τη μορφή που φαίνεται δίπλα

```
struct Person
{
        char name[50];
        int tax_num;
        char addr[50];
        float weight;
        float height;
};
```

Παρατηρήσεις (I)

- Η **συνήθης πρακτική** είναι η δήλωση της δομής μαζί με άλλες δηλώσεις, όπως πρωτότυπα συναρτήσεων και μακροεντολές, να **αποθηκεύονται σε ένα ξεχωριστό αρχείο επικεφαλίδας** το οποίο να συμπεριλαμβάνεται με την οδηγία `#include` όπου χρειάζεται

- Για απλότητα, στα επόμενα παραδείγματα θα δηλώνουμε τον κάθε τύπο δομής με καθολική εμβέλεια στην αρχή του αρχείου, ώστε όλες οι συναρτήσεις να μπορούν να τον χρησιμοποιήσουν

- Αν το πρότυπο μίας δομής δηλωθεί μέσα στο σώμα μίας συνάρτησης, τότε η εμβέλειά του – προφανώς – είναι **τοπική**

Η δήλωση μίας δομής **που δεν ακολουθείται από λίστα μεταβλητών** **δεν προκαλεί δέσμευση μνήμης** και **απλώς περιγράφει τη μορφή** της δομής

Παρατηρήσεις (II)

- Αν η δομή έχει ετικέτα, μπορούμε να τη χρησιμοποιήσουμε για δηλώσεις μεταβλητών, π.χ. :

```
struct Person p1, p2;
```

- Για να είναι ξεκάθαρο, η δήλωση της δομής `Person` **ορίζει έναν τύπο δεδομένων με αυτό το όνομα**, δεν προκαλεί δέσμευση μνήμης και ούτε η `Person` είναι μεταβλητή
- Η `Person` **απλά περιγράφει τη μορφή** που θα έχουν οι μεταβλητές της δομής όταν αυτές δηλωθούν
- Οι μεταβλητές `p1` και `p2` δηλώνονται σαν δομές τύπου `Person`, η κάθε μία έχει τα δικά της πεδία και ο μεταγλωττιστής δεσμεύει μνήμη για αυτές
- Εναλλακτικά, μπορούμε να δηλώσουμε μεταβλητές στη `λίστα_μεταβλητών`, π.χ.:

```
struct Book
        char title[100];
        int year;
        float price;
} b1, b2;
```

Παρατηρήσεις (III)

- Όπως με κάθε μεταβλητή, όταν δηλώνεται μία δομή ο μεταγλωττιστής δεσμεύει μνήμη για την αποθήκευση των πεδίων της με τη σειρά που εμφανίζονται
- Π.χ. το επόμενο πρόγραμμα εμφανίζει πόσες οκτάδες δεσμεύτηκαν για τη δομή `d`
- Συνεπώς, η έξοδος του προγράμματος αναμένεται να είναι `12`...

```c
#include <stdio.h>

struct Date
{
        int day;
        int month;
        int year;
};

int main(void)
{
        struct Date d;

        printf("%u\n", sizeof(d));
        return 0;
}
```

...και όντως, είναι `12` !!

Παρατηρήσεις (IV)

Συνεπώς, ποια αναμένεται να είναι η έξοδος του παρακάτω προγράμματος???

```c
#include <stdio.h>

struct Date
{
        char day;
        int month;
        int year;
};

int main(void)
{
        struct Date d;

        printf("%u\n", sizeof(d));
        return 0;
}
```

Απαντήσατε 9
αλλά εμφανίστηκε ξανά...12 ???
Don't panic...

Για την αποφυγή λαθών στον υπολογισμό της μνήμης που καταλαμβάνει μία δομή να χρησιμοποιείτε πάντα τον τελεστή `sizeof` και να μην μπαίνετε στη διαδικασία να προσθέτετε τη μνήμη που δεσμεύουν ξεχωριστά τα πεδία της

Παρατηρήσεις (V)

- Επειδή συχνά ο μεταγλωττιστής απαιτεί πιο γρήγορη πρόσβαση στα πεδία της δομής, κάθε πεδίο συνήθως αποθηκεύεται σε μία διεύθυνση μνήμης που να είναι πολλαπλάσια κάποιου αριθμού (συνήθως του 4)

- Αν υποθέσουμε ότι το πεδίο `month` πρέπει να αποθηκευτεί σε μία διεύθυνση πολλαπλάσια του 4, ο μεταγλωττιστής θα δεσμεύσει τρεις πρόσθετες οκτάδες αμέσως μετά το πεδίο `day`

- Τέτοιες «οπές» μπορεί να υπάρχουν μόνο μεταξύ των πεδίων και στο τέλος της δομής, όχι, όμως, στην αρχή

Παραδείγματα

- Ποια αναμένεται να είναι η έξοδος των παρακάτω προγραμμάτων ???

```c
#include <stdio.h>
/* Δήλωση προτύπου δομής με όνομα Test. */
struct Test
{
    double x;
    int y;
    int z;
    char w;
};
int main(void)
{
    printf("Size = %d\n",sizeof(Test));
    return 0;
}
```

Έξοδος: `Size = 24`

```c
#include <stdio.h>
/* Δήλωση προτύπου δομής με όνομα Test. */
struct Test
{
    int y;
    int arr[4];
    char pin[5];
};
int main(void)
{
    printf("Size = %d\n",sizeof(Test));
    return 0;
}
```

Έξοδος: `Size = 28`

Το προσδιοριστικό typedef (I)

- Η λέξη κλειδί typedef χρησιμοποιείται για να ορίσουμε ένα συνώνυμο για κάποιον υπαρκτό τύπο δεδομένων

- Π.χ., η δήλωση

```
typedef unsigned int size_t;
```

κάνει το όνομα size_t συνώνυμο του τύπου unsigned int

- Άρα, οι δηλώσεις:

unsigned int i; και size_t i; είναι ισοδύναμες

- Η σύνταξη μοιάζει με τη δήλωση μεταβλητής (π.χ. όπως θα δηλώναμε τη size_t), απλά προηγείται η λέξη typedef

Το προσδιοριστικό typedef (II)

- Το όνομα αποτελεί συνώνυμο του τύπου.
- Π.χ., με τη δήλωση:

```
typedef float arr[100];
```

- το όνομα `arr` γίνεται συνώνυμο για ένα πίνακα 100 πραγματικών τύπου `float`
- Προσοχή, η `typedef` δεν δημιουργεί μεταβλητή (δηλαδή, το `arr` δεν είναι μεταβλητή πίνακας), ούτε ένα νέο τύπο, απλά ένα νέο όνομα, άρα, με την εντολή:

```
arr arr1;
```

- η μεταβλητή `arr1` δηλώνεται σαν πίνακας 100 πραγματικών τύπου `float`
- Αν στο μέλλον χρειαστεί να αλλάξουμε τον τύπο του πίνακα σε `double`, απλά αλλάζουμε το `float` σε `double`

Το προσδιοριστικό typedef (III)

- Παρόμοια, μπορούμε να χρησιμοποιήσουμε την typedef για να δηλώσουμε ένα συνώνυμο για έναν τύπο δομής, π.χ.:

```
typedef struct Mybook
{
        char title[100];
        int year;
        float price;
} Book;
```

- Το Mybook είναι μία ετικέτα και πρέπει να προσθέτουμε τη λέξη struct όταν τη χρησιμοποιούμε

- Αντίθετα, επειδή το typedef όνομα αποτελεί συνώνυμο τύπου δεν επιτρέπεται να προσθέτουμε τη λέξη struct, π.χ. :

```
Book b1, b2;
```

Το προσδιοριστικό typedef (IV)

- Σημειώστε ότι η ετικέτα της δομής μπορεί να έχει το ίδιο όνομα με το συνώνυμο, δηλαδή αντί για **Mybook** να γράψουμε **Book**, η σημασία τους όμως εξακολουθεί να είναι διαφορετική

- Επίσης, θα μπορούσαμε να παραλείψουμε την ετικέτα και να χρησιμοποιούμε το **typedef** όνομα για να δηλώσουμε μεταβλητές, π.χ.:

```
typedef struct
{
        char title[100];
        int year;
        float price;
} Book;
```

Αρχικοποίηση πεδίων δομής (Ι)

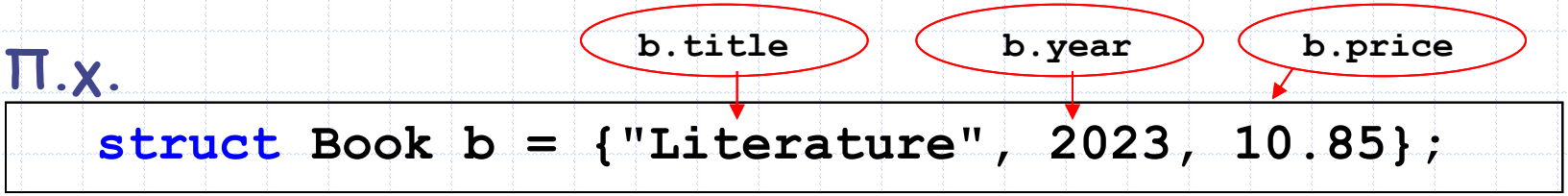
- Ο πιο συνηθισμένος τρόπος για να προσπελάσουμε τα πεδία μίας δομής είναι να γράψουμε το όνομα της δομής να προσθέσουμε τον τελεστή τελεία (.) και μετά το όνομα του πεδίου που μας ενδιαφέρει
- Το επόμενο πρόγραμμα αρχικοποιεί και εμφανίζει τα πεδία της **b** και τα εμφανίζει στην οθόνη

```c
#include <stdio.h>
#include <string.h>

struct Book
{
        char title[100];
        int year;
        float price;
};

int main(void)
{
        struct Book b;

        strcpy(b.title, "Literature");/*Αρχικοποίηση του πεδίου title.*/
        b.year = 2023; /* Αρχικοποίηση του πεδίου year. */
        b.price = 10.85; /* Αρχικοποίηση του πεδίου price. */
        printf("%s %d %.2f\n", b.title, b.year,  b.price);
        return 0;
}
```

Αρχικοποίηση πεδίων δομής (ΙΙ)

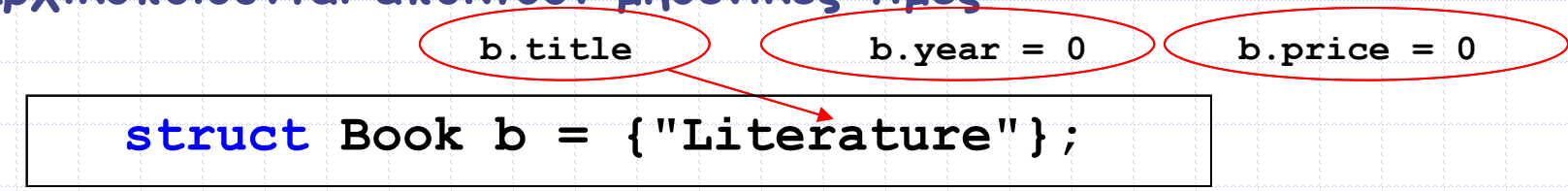
- Εναλλακτικός τρόπος αρχικοποίησης μίας δομής είναι μαζί με τη δήλωσή της

Π.χ.



- Όπως τα στοιχεία ενός πίνακα, έτσι και τα πεδία μίας δομής που δεν αρχικοποιούνται αποκτούν μηδενικές τιμές

Π.χ.



- Παρόμοια, με τη δήλωση:

```
struct Book b = {0} ;
```

οι χαρακτήρες του πεδίου `title` αρχικοποιούνται με τον τερματικό χαρακτήρα (`'\0'`) και τα πεδία `year` και `price` με 0

Αρχικοποίηση πεδίων δομής (III)

- Τέλος, μία μεταβλητή μπορεί να αρχικοποιηθεί και όταν δηλώνεται στη `λίστα_μεταβλητών` π.χ.:

```
struct Book
{
        char title[100];
        int year;
        float price;
} b = {"Literature", 2023, 10.8};
```

Δείκτης σε πεδίο δομής

- Όπως χρησιμοποιούμε έναν δείκτη σε μία μεταβλητή, μπορούμε να τον χρησιμοποιήσουμε και σε ένα πεδίο μίας δομής

Π.χ. δείτε πώς εμφανίζονται τα πεδία της δομής b με χρήση δεικτών

```c
#include <stdio.h>
#include <string.h>

struct Book
{
        char title[100];
        int year;
        float price;
};

int main(void)
{
        char *ptr1;
        int *ptr2;
        float *ptr3;
        struct Book b;

        strcpy(b.title, "Literature");
        b.year = 2023;
        b.price = 10.8;

        ptr1 = b.title;
        ptr2 = &b.year;
        ptr3 = &b.price;

        printf("%s %d %.2f\n", ptr1, *ptr2, *ptr3);
        return 0;
}
```

Αντιγραφή και Σύγκριση δομών

- Για την αντιγραφή μιας δομής σε μία άλλη χρησιμοποιείται ο τελεστής =

Για να εκτελεστεί η αντιγραφή, οι δομές **πρέπει να είναι του ίδιου τύπου**. Π.χ.

```
#include <stdio.h>

struct Student
{
        int code;
        float grd;
};

int main(void)
{
        struct Student s1, s2;

        s1.code = 1234;
        s1.grd = 6.7;
        s2 = s1; /* Αντιγραφή δομής. */
        printf("C:%d G:%.2f\n", s2.code, s2.grd);
        return 0;
}
```

Παρατηρήσεις (I)

- Με την εντολή:

```
s2 = s1;
```

οι τιμές των πεδίων της δομής `s1` αντιγράφονται στα αντίστοιχα πεδία της δομής `s2`

Δηλαδή, η παραπάνω εντολή είναι ισοδύναμη με:

```
s2.code = s1.code;
s2.grd = s1.grd;
```

Αν οι `s1` και `s2` δεν ήταν μεταβλητές του ίδιου τύπου δομής, η εντολή `s2 = s1;` δεν θα μεταγλωττιζόταν, ακόμα κι αν τα δύο διαφορετικά πρότυπα δομής περιείχαν τα ίδια ακριβώς πεδία και σε αριθμό και σε τύπο δεδομένων

Παρατηρήσεις (ΙΙ)

- Σημειώστε, επίσης, ότι αν μία δομή περιέχει πίνακα (π.χ. `name`), αν και, όπως γνωρίζετε, δεν μπορούμε να γράψουμε

```
s2.name = s1.name;
```

με την αντιγραφή της δομής (π.χ. `s2 = s1`) αντιγράφεται και ο πίνακας της `s1` στην `s2`

- Επίσης, αν η δομή περιέχει πεδίο δείκτη, μετά την αντιγραφή και οι δύο δείκτες θα δείχνουν στο ίδιο σημείο, γεγονός που μπορεί να αποβεί πολύ επικίνδυνο

Παρατηρήσεις (III)

- Εκτός από την ανάθεση **καμία** άλλη ενέργεια δεν επιτρέπεται μεταξύ δομών

- Π.χ. οι τελεστές == και != δεν μπορούν να χρησιμοποιηθούν για τον έλεγχο της ισότητας δύο δομών

  Δηλαδή, **δεν επιτρέπεται** να γράψετε:

```
if(s1 == s2)
```

ή

```
if(s1 != s2)
```

- Αν πρέπει **να γίνει έλεγχος** αν δύο δομές περιέχουν τα ίδια ακριβώς δεδομένα, πρέπει αναγκαστικά να λάβει χώρα σύγκριση **όλων των πεδίων** των δομών **ένα προς ένα** και **ξεχωριστά**

  Δηλαδή:

```
if((s1.code == s2.code) && (s1.grd == s2.grd))
```

Παράδειγμα δομής που περιέχει πίνακες

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```c
#include <stdio.h>
#include <string.h>

struct Student
{
        char name[50];
        float grd[2];
};

int main(void)
{
        struct Student s1, s2;

        strcpy(s1.name, "somebody");
        s1.grd[0] = 8.5;
        s1.grd[1] = 7.5;
        printf("%s %c %c\n", s1.name, s1.name[0], *s1.name);

        s2 = s1;
        printf("%s %.1f %.1f\n", s2.name, s2.grd[0], s2.grd[1]);
        return 0;
}
```

Έξοδος:

```
somebody s s
somebody 8.5 7.5
```

Παράδειγμα δομής που περιέχει δείκτες

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```c
#include <stdio.h>

struct Student
{
        char *name;
        float *avg_grd;
};

int main(void)
{
        float grd = 8.5;
        struct Student s1, s2;

        s1.name = "somebody";
        s1.avg_grd = &grd;
        printf("%s %.2f\n", s1.name+3, *s1.avg_grd);

        s2.name = "else";
        s2 = s1; /* Προσοχή. */
        grd = 3.4;
        printf("%s %.2f\n", s2.name, *s2.avg_grd);
        return 0;
}
```

Έξοδος:

```
ebody 8.50
somebody 3.40
```

Ένθετες δομές

- Μία δομή μπορεί να περιέχει μία ή περισσότερες δομές, οι οποίες ονομάζονται ένθετες δομές (nested structures)
- Μία ένθετη δομή πρέπει να ορίζεται πριν από τον ορισμό της δομής στην οποία περιέχεται, αλλιώς ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους
- Για να προσπελάσουμε τα πεδία μίας δομής που περιέχεται μέσα σε μία άλλη δομή χρησιμοποιούμε δύο φορές τον τελεστή τελεία (.)
- Την πρώτη φορά ανάμεσα στο όνομα της εξωτερικής και της ένθετης δομής και τη δεύτερη φορά ανάμεσα στο όνομα της ένθετης δομής και το όνομα του πεδίου που μας ενδιαφέρει (το οποίο ανήκει στην ένθετη δομή)

Παράδειγμα δομής που περιέχει δομή (1/2)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

struct Date
{
        int day;
        int month;
        int year;
};

struct  Product  /*  Αφού  ο  τύπος  Date  έχει  οριστεί,  μπορεί  να
χρησιμοποιηθεί για να δηλώσουμε ένθετες δομές. */
{
        char name[50];
        double price;
        struct Date s_date;
        struct Date e_date;
};
```

Παράδειγμα δομής που περιέχει δομή (2/2)

```
int main(void)
{
        struct Product prod;

        strcpy(prod.name, "product");
        prod.s_date.day = 1;
        prod.s_date.month = 9;
        prod.s_date.year = 2023;


        prod.e_date.day = 1;
        prod.e_date.month = 9;
        prod.e_date.year = 2025;

        prod.price = 7.5;
        printf("The product's life is %d years\n", prod.e_date.year -
prod.s_date.year);
        return 0;
}
```

Έξοδος:

The product's life is 2 years

Πεδία Δομής με μέγεθος bit (Ι)

- Ένα πεδίο δομής μπορεί να περιέχει πεδία, των οποίων το μέγεθος να δηλώνεται σαν ένας συγκεκριμένος αριθμός από bits
- Ένα τέτοιο πεδίο ονομάζεται **πεδίο bit** και δηλώνεται με τον ακόλουθο τρόπο:

```
τύπος_δεδομένων όνομα_πεδίου_bit : αριθμός_bits;
```

- Δηλαδή, η δήλωση ενός **πεδίου bit** γίνεται με τη χρήση της **άνω-κάτω τελείας** : ανάμεσα στο όνομα του πεδίου και τον αριθμό των bits που θα έχει
- Η προσπέλαση των **πεδίων bit** γίνεται με τους ίδιους τρόπους, όπως και με τα απλά πεδία μίας δομής
- Οι περισσότεροι μεταγλωττιστές εκτός από τους τύπους `int`, `signed int` και `unsigned int` υποστηρίζουν και τους τύπους `char` και `long` ως επιτρεπτούς τύπους δεδομένων ενός **πεδίου bit**

Πεδία Δομής με μέγεθος bit (II)

Παράδειγμα:

```
struct Person
{
        unsigned int sex : 1;
        unsigned int married : 1;
        unsigned int children : 4;
};
```

Μέγεθος bit: 1
Εύρος τιμών: 0 – 1

Μέγεθος bit: 4
Εύρος τιμών: 0 – 15

Πεδία Δομής με μέγεθος bit (III)

- Για την αποθήκευση των τιμών του προηγούμενου παραδείγματος απαιτούνται συνολικά: `1 + 1 + 4 = 6` bits
- Αφού ο τύπος δεδομένων των πεδίων είναι `unsigned int`, τότε ο μεταγλωττιστής δεσμεύει `4` byte μνήμης, και `4×8-6 = 26` bits μένουν αχρησιμοποίητα
- Το κύριο πλεονέκτημα της χρήσης των πεδίων bit είναι η εξοικονόμηση μνήμης
- Π.χ., στο προηγούμενο πρόγραμμα ο μεταγλωττιστής δεσμεύει τέσσερις οκτάδες αντί για δώδεκα που θα δέσμευε αν δεν χρησιμοποιούσαμε bit πεδία
- Επομένως, αφού για τα στοιχεία ενός ανθρώπου εξοικονομούμε οκτώ οκτάδες, αν έπρεπε να αποθηκεύσουμε τα στοιχεία `200.000` ανθρώπων θα εξοικονομούσαμε `1.600.000` οκτάδες

Παρατηρήσεις (I)

Για βέλτιστη εξοικονόμηση μνήμης, οι δηλώσεις των bit πεδίων **να γίνονται όλες μαζί στην αρχή της δομής** και να μην παρεμβάλλονται μεταξύ των υπολοίπων πεδίων

Αν η λειτουργία του προγράμματός σας εξαρτάται από τη σειρά που αποθηκεύονται τα πεδία bits στη μνήμη και επιθυμείτε να είναι φορητό στα διάφορα μηχανήματα, πρέπει να λάβετε υπόψη σας ότι **ο τρόπος δέσμευσης μπορεί να διαφέρει**

Όταν εκχωρείται μία τιμή σε ένα **πεδίο bit**, τότε αυτή η τιμή **δεν θα πρέπει να κωδικοποιείται σε περισσότερα bits από ότι το μέγεθος του πεδίου**, διότι – σε μία τέτοια περίπτωση – είναι πολύ πιθανό να εισάγουμε λογικό λάθος (bug) στον κώδικά μας

Παρατηρήσεις (II)

```c
#include <stdio.h>

struct Person
{
        unsigned char sex : 1;
        unsigned char married : 1;
        unsigned char children : 4;
        char name[30];
};

int main(void)
{
        struct Person person_1;

        person_1.married = 2; /* Λανθασμένη εκχώρηση. */
        printf("Married = %d\n",person_1.married);
        return 0;
}
```

- **Γιατί είναι λανθασμένη η παραπάνω εκχώρηση???**

Η τιμή 2 απαιτεί 2 bits για να κωδικοποιηθεί (10_2) και όχι 1, όπως καθορίζεται στο μέγεθος του πεδίου. Τελικά, στο πεδίο **married** θα αποθηκευτεί το bit που θεωρεί ο μεταγλωττιστής ως «χαμηλότερης σημασίας». Αν π.χ. αποθηκευτεί το δεξιότερο bit, τότε το πρόγραμμα θα εμφανίσει: **Married = 0**

Παρατηρήσεις (III)

- Ο κύριος περιορισμός των πεδίων bit είναι ότι ένας δείκτης δεν μπορεί να δείξει στη διεύθυνση ενός πεδίου bit
- Δηλαδή, στο παρακάτω πρόγραμμα ο μεταγλωττιστής θα εμφάνιζε μήνυμα λάθους

```
#include <stdio.h>

struct Person
{
	unsigned char sex : 1;
	unsigned char married : 1;
	unsigned char children : 4;
	char name[30];
};

int main(void)
{
	struct Person person_1;

	unsigned char* ptr;

	ptr = &person_1.married; /* Μη επιτρεπτή ενέργεια. */

	return 0;
}
```

Δείκτης σε Δομή (Ι)

- Ένας δείκτης σε μία δομή χρησιμοποιείται με τον ίδιο τρόπο όπως κάθε άλλος δείκτης

  Π.χ. με τη εντολή:

```
struct Student *stud_ptr;
```

  η μεταβλητή stud_ptr δηλώνεται ως δείκτης προς κάποια δομή τύπου Student

- Υπενθυμίζεται ότι ένας δείκτης, πριν χρησιμοποιηθεί, πρέπει να έχει σαν τιμή τη διεύθυνση κάποιας μεταβλητής, ή ισοδύναμα να «δείχνει» σε κάποια υπαρκτή μεταβλητή

```
struct Student *stud_ptr;
struct Student stud;
stud_ptr = &stud;
```

- Με την τελευταία εντολή ο δείκτης stud_ptr δείχνει στη διεύθυνση της μνήμης που έχει αποθηκευτεί η δομή stud και συγκεκριμένα στη διεύθυνση μνήμης του πρώτου πεδίου της δομής stud

Δείκτης σε Δομή (ΙΙ)

- Χρησιμοποιώντας τον τελεστή * πριν από το όνομα του δείκτη, αποκτούμε πρόσβαση στο περιεχόμενο της μνήμης που δείχνει ο δείκτης, άρα μπορούμε να προσπελάσουμε τα πεδία μίας δομής

- Π.χ.

```c
#include <stdio.h>
#include <string.h>

struct Student
{
        char name[50];
        float grd;
};

int main(void)
{
        struct Student *ptr, stud;

        ptr = &stud;
        strcpy((*ptr).name, "somebody");
        (*ptr).grd = 6.7;

        printf("N: %s G: = %.2f\n", stud.name, stud.grd);
        return 0;
}
```

Έξοδος: `N: somebody G: 6.70`

Δείκτης σε Δομή (III)

- Ένας εναλλακτικός τρόπος για να αποκτήσουμε πρόσβαση στα πεδία μίας δομής με χρήση δείκτη είναι χρησιμοποιώντας τον τελεστή -> αντί του τελεστή τελεία (.)
- Θα δείτε ότι ο συγκεκριμένος τελεστής (->) συνηθίζεται να χρησιμοποιείται στις δυναμικές δομές δεδομένων
- Δηλαδή, για τη δομή:

```
struct Student
{
        char name[50];
        float grade;
};
```

Αν:

```
struct Student *ptr;
```

οι παρακάτω εκφράσεις, είναι ισοδύναμες:

`ptr->name`	ισοδύναμη με	`(*ptr).name`
`ptr->grade`	ισοδύναμη με	`(*ptr).grade`

Δείκτης σε Δομή (IV)

- Π.χ. το προηγούμενο παράδειγμα θα μπορούσε να γραφεί με χρήση του τελεστή -> αντί του τελεστή τελεία (.) ως εξής:

```c
#include <stdio.h>
#include <string.h>

struct Student
{
        char name[50];
        float grd;
};

int main(void)
{
        struct Student *ptr, stud;

        ptr = &stud;
        /* Πρόσβαση στα πεδία της δομής με χρήση του δείκτη ptr και του
τελεστη -> */
        strcpy(ptr->name, "somebody");
        ptr->grd = 6.7;

        printf("N: %s G: %.2f\n", ptr->name, ptr->grd);
        return 0;
}
```

Έξοδος: `N: somebody G: 6.70`

Παρατηρήσεις

- Για να αποκτήσουμε πρόσβαση στα πεδία μίας δομής με χρήση δείκτη προτιμάται η χρήση του τελεστή ->

- Για έναν δείκτη σε δομή ισχύουν οι ίδιοι κανόνες αριθμητικής που ισχύουν για τους απλούς δείκτες

  Π.χ. αν **αυξηθεί η τιμή του δείκτη κατά ένα**, τότε **η τιμή του δείκτη θα αυξηθεί κατά το μέγεθος της δομής** στην οποία δείχνει

- Η συνηθέστερη χρήση δεικτών σε δομές είναι κατά **τη δημιουργία δυναμικών δομών δεδομένων**, όπως **λίστες**, **στοίβες** και **ουρές**

Πίνακας Δομών

- Ένας **πίνακας δομών** είναι ένας πίνακας, του οποίου **κάθε στοιχείο είναι μία δομή**

Π.χ. αν έχουμε δηλώσει τη δομή:

```
struct Student
{
        char name[50];
        int code;
        float grd;
};
```

τότε η δήλωση:

```
struct Student stud[100];
```

σημαίνει ότι η μεταβλητή `stud` είναι ένας πίνακας 100 στοιχείων, και το κάθε στοιχείο του πίνακα είναι μία δομή τύπου `Student`

Αρχικοποίηση Πίνακα Δομών (Ι)

- **Α' Τρόπος (κατά τη δήλωση του πίνακα δομών)**

Π.χ. για τη δομή:

```
struct Student
{
        char name[50];
        int code;
        float grd;
};
```

με την παρακάτω αρχικοποίηση:

```
struct Student stud[] = {{"nick stergiou",555,7.3},
                         {"john nikas",556,5.8},
                         {"peter karras",557,6.7}};
```

η τιμή του πεδίου `stud[0].name` γίνεται `"nick stergiou"`
η τιμή του πεδίου `stud[1].code` γίνεται `556`
η τιμή του πεδίου `stud[2].grd` γίνεται `6.7`

Αρχικοποίηση Πίνακα Δομών (ΙΙ)

- Β' Τρόπος (απευθείας ενσωμάτωση στη δήλωση του τύπου της δομής)

Π.χ. :

```
struct Student
{
        char name[50];
        int code;
        float grd;
} stud[] = {{"nick stergiou",555,7.3},
            {"john nikas",556,5.8},
            {"peter karras",557,6.7}};
```

- Και με τους 2 τρόπους αρχικοποίησης, τα εσωτερικά άγκιστρα μπορούν να παραλειφθούν (προτείνεται όμως να χρησιμοποιούνται, ώστε ο κώδικας να είναι περισσότερο ευανάγνωστος, δηλ. να ξεχωρίζουν μεταξύ τους οι δομές)

Παρατηρήσεις (Ι)

- Οι **πίνακες δομών** χρησιμοποιούνται **πολύ συχνά** σε προγράμματα που απαιτείται **αποθήκευση πληροφορίας** που αντιστοιχεί σε **διαφορετικές καταχωρήσεις** (π.χ. ένα πρόγραμμα για την καταχώρηση των στοιχείων των υπαλλήλων μίας εταιρείας, των στοιχείων των φοιτητών μίας σχολής, της πληροφορίας των προϊόντων μίας αποθήκης, κτλ...)

- Δηλαδή, οι πίνακες δομών μπορούν να χρησιμοποιηθούν σαν μία **βάση δεδομένων**

- Όταν γίνεται η δήλωση του πίνακα δομών μπορούμε να θέσουμε την τιμή 0 σε όλα τα πεδία κάθε δομής

Π.χ. αν γράψουμε:

```
struct Student stud[100] = {0}
```

τότε όλα τα πεδία κάθε δομής αποκτούν την τιμή 0

Παρατηρήσεις (II)

- Για τους πίνακες δομών ισχύουν όλοι οι κανόνες που ισχύουν για τους απλούς πίνακες

  Π.χ. για την προηγούμενη δομή `stud` (τύπου `Student`) αφού το όνομα ενός πίνακα είναι δείκτης στη διεύθυνση του 1ου στοιχείου του, τότε:
  - το `*stud` είναι ισοδύναμο με `stud[0]`
  - το `*(stud + 1)` είναι ισοδύναμο με `stud[1]`
  - το `*(stud + 2)` είναι ισοδύναμο με `stud[2]` κ.ο.κ.

- Άρα, αν θέλουμε να προσπελάσουμε το πεδίο `grd` του τρίτου υπαλλήλου, οι εκφράσεις `stud[2].grd` και `(*(stud + 2)).grd` είναι ισοδύναμες (οι παρενθέσεις στη δεύτερη περίπτωση είναι απαραίτητες λόγω των προτεραιοτήτων)

- Προφανώς, ο χειρισμός ενός πίνακα δομών με χρήση της θέσης του κάθε στοιχείου στον πίνακα, είναι πιο απλός και ευανάγνωστος από τον αντίστοιχο χειρισμό με δείκτη

Συνάρτηση με παράμετρο Δομή

- Μία δομή μπορεί να μεταβιβαστεί σε μία συνάρτηση όπως οποιαδήποτε άλλη μεταβλητή, δηλαδή **είτε η ίδια η δομή είτε η διεύθυνση μνήμης της δομής**, ενώ επίσης, μία συνάρτηση μπορεί να επιστρέφει δομή
- Υπενθυμίζεται ότι, όταν μεταβιβάζεται **η τιμή** μιας παραμέτρου σε συνάρτηση, τότε στη συνάρτηση διοχετεύονται **αντίγραφα** των παραμέτρων του προγράμματος που την καλεί, επομένως, οποιαδήποτε αλλαγή γίνει στις τιμές των πεδίων της δομής μέσα στη συνάρτηση **δεν επηρεάζει** τις αντίστοιχες τιμές των πεδίων της δομής που διοχετεύθηκε στη συνάρτηση, γιατί οι τυχόν αλλαγές γίνονται σε αντίγραφό της
- **Αντίθετα**, σε περίπτωση που μεταβιβάζεται **η διεύθυνση** της παραμέτρου σε συνάρτηση, στη συνάρτηση διοχετεύονται **οι διευθύνσεις μνήμης** των παραμέτρων του προγράμματος που την καλεί και όχι αντίγραφά τους, όπως προηγουμένως, επομένως, αφού η συνάρτηση έχει πρόσβαση στη διεύθυνση της δομής του προγράμματος που την κάλεσε, τότε **μπορεί να μεταβάλλει** τις τιμές των πεδίων της

Διοχέτευση τιμής της δομής σε συνάρτηση (Ι)

- Π.χ. αν θεωρήσουμε ότι έχει δηλωθεί η παρακάτω δομή:

```
struct Student
{
        char name[50];
        int code;
        float grd;
};
```

- Και έχει δηλωθεί κι η παρακάτω συνάρτηση:

```
void funct(struct Student stud_1);
```

Όρισμα συνάρτησης:

δομή τύπου Student

- Αφού επίσης δηλωθεί και αρχικοποιηθεί μία μεταβλητή-δομή (π.χ. stud) τύπου Student:

```
struct Student stud;
strcpy(stud.name, "somebody");
stud.code = 555;
stud.grd = 7;
```

Διοχέτευση τιμής της δομής σε συνάρτηση (ΙΙ)

- Κατά τη διοχέτευση της τιμής μιας δομής σε συνάρτηση (π.χ. από το κύριο πρόγραμμα, δηλ. μέσα από τη συνάρτηση `main()`):

```
funct(stud);
```

δημιουργείται προσωρινά στη μνήμη η μεταβλητή `stud_1`, η οποία αποτελεί **αντίγραφο** της μεταβλητής `stud`

- Αντίγραφο σημαίνει ότι οι αρχικές τιμές των πεδίων της δομής `stud_1` θα γίνουν ίσες με τις αντίστοιχες τιμές των πεδίων της δομής `stud`
- Η μεταβλητή `stud_1` δεν έχει καμία σχέση με τη μεταβλητή `stud` του κυρίως προγράμματος, αφού βρίσκονται σε διαφορετικές θέσεις μνήμης
- Επομένως, οποιαδήποτε αλλαγή γίνει στις τιμές των πεδίων της δομής `stud_1` **δεν επηρεάζει** τις αντίστοιχες τιμές των πεδίων της δομής `stud`

Διοχέτευση διεύθυνσης της δομής σε συνάρτηση (Ι)

- Π.χ. αν θεωρήσουμε ότι έχει δηλωθεί η παρακάτω δομή:

```
struct Student
{
        char name[50];
        int code;
        float grd;
};
```

- Και έχει δηλωθεί κι η παρακάτω συνάρτηση:

```
void funct(struct Student *stud_ptr);
```

Όρισμα συνάρτησης: δείκτης σε δομή τύπου Student

- Αφού επίσης δηλωθεί και αρχικοποιηθεί μία μεταβλητή-δομή (π.χ. stud) τύπου Student:

```
struct Student stud;
strcpy(stud.name, "somebody");
stud.code = 555;
stud.grd = 7;
```

Διοχέτευση διεύθυνσης της δομής σε συνάρτηση (ΙΙ)

- Κατά τη διοχέτευση της διεύθυνσης μιας δομής σε συνάρτηση (π.χ. από το κύριο πρόγραμμα, δηλ. μέσα από τη συνάρτηση `main()`):

```
funct(&stud);
```

- στη συνάρτηση διοχετεύονται οι διευθύνσεις μνήμης των παραμέτρων του προγράμματος που την καλεί και όχι αντίγραφά τους, όπως προηγουμένως

- Επομένως, αφού η συνάρτηση έχει πρόσβαση στη διεύθυνση της δομής του προγράμματος που την κάλεσε, τότε μπορεί να μεταβάλλει τις τιμές των πεδίων της

Παρατηρήσεις

- Για καλύτερη απόδοση, να μεταβιβάζετε σε μία συνάρτηση **τη διεύθυνση της δομής**, ακόμα και αν η συνάρτηση δεν πρόκειται να αλλάξει τις τιμές των πεδίων της
- Ο κύριος λόγος είναι γιατί **αποφεύγεται η διαδικασία δημιουργίας αντιγράφου της δομής στη στοίβα**, άρα **το πρόγραμμα εκτελείται πιο γρήγορα**
- Σε περίπτωση που η συνάρτηση δεν πρέπει να αλλάξει τις τιμές των πεδίων της δομής, τότε ο προγραμματιστής, για να είναι σίγουρος ότι κάτι τέτοιο δεν θα συμβεί στο μέλλον, μπορεί να προσθέσει στη δήλωση της συνάρτησης τη λέξη `const` πριν από τη λέξη `struct`, δηλαδή να δηλώσει τον δείκτη `stud_ptr` ως `const`
- Π.χ.

```
void funct(const struct Student *stud_ptr);
```

Ενώσεις (Unions)

- Μία ένωση (union) στη C μοιάζει με μία δομή (structure), με τη σημαντική όμως διαφορά, ότι μόνο ένα πεδίο της ένωσης μπορεί να προσπελαύνεται κάθε φορά

- Αυτό συμβαίνει, γιατί τα πεδία μίας ένωσης δεν καταλαμβάνουν ξεχωριστό χώρο στη μνήμη, αλλά έναν κοινό χώρο μνήμης

Δήλωση Ένωσης

- Το πρότυπο μίας ένωσης δηλώνεται με τον ίδιο τρόπο όπως δηλώνεται και το πρότυπο μίας δομής με τη διαφορά ότι αντί της λέξης `struct` χρησιμοποιείται η λέξη `union`

- Όπως και στην περίπτωση των δομών, έτσι και στις ενώσεις, το πρότυπο μίας ένωσης δεν αποτελεί μεταβλητή, δηλαδή, όταν δηλώνεται το πρότυπο μίας ένωσης, δεν δεσμεύεται μνήμη για την αποθήκευσή του

- Προφανώς, αντίστοιχα με τις δομές, όταν δηλώνεται μία ένωση, ο μεταγλωττιστής δεσμεύει μνήμη για την αποθήκευσή της

- Το μέγεθος της μνήμης που δεσμεύεται είναι ίσο με τη μνήμη που δεσμεύεται για το μεγαλύτερο πεδίο της ένωσης

- Δηλαδή, τα πεδία μιας ένωσης αποθηκεύονται σε έναν κοινό χώρο μνήμης και όχι σε ξεχωριστή μνήμη το καθένα (όπως συμβαίνει στις δομές)

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```c
#include <stdio.h>

union Sample
{
        char ch;
        int i;
        double d;
};

int main(void)
{
        union Sample s;
        printf("Size: %u\n", sizeof(s));
        return 0;
}
```

Έξοδος: `Size: 8`

Πρόσβαση πεδίων Ένωσης

- Τα πεδία μίας ένωσης μπορούν να προσπελαστούν με τους ίδιους ακριβώς τρόπους που προσπελαύνονται και τα πεδία μίας δομής

- Ωστόσο, επειδή όλα τα πεδία αποθηκεύονται σε κοινή μνήμη, μόνο το τελευταίο πεδίο στο οποίο εκχωρήθηκε μία τιμή μπορεί να χρησιμοποιηθεί (όλα τα υπόλοιπα πεδία χάνουν τις προηγούμενες τιμές που πιθανόν τους είχαν ανατεθεί και αποκτούν νέες τυχαίες τιμές)

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```c
#include <stdio.h>

union Sample
{
        char ch;
        int i;
        double d;
};

int main(void)
{
        union Sample s;

        s.ch = 'a';
        printf("%c %d %f\n", s.ch, s.i, s.d);

        s.i = 64;
        printf("%c %d %f\n", s.ch, s.i, s.d);

        s.d = 12.48;
        printf("%c %d %f\n", s.ch, s.i, s.d);
        return 0;
}
```

Έξοδος:

```
a τυχαία τιμή τυχαία τιμή
τυχαία τιμή 64 τυχαία τιμή
τυχαία τιμή τυχαία τιμή 12.480000
```