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** από μεταβλητές **οποιουδήποτε τύπου**, οι οποίες 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...

Πρότυπο Δομής (Ι)

- Το πρότυπο μίας δομής καθορίζει τον τύπο και το πλήθος των μεταβλητών που περιέχει μία δομή
- Μία δομή μπορεί να περιέχει μεταβλητές οποιουδήποτε τύπου, συμπεριλαμβανομένων δεικτών, πινάκων, ακόμα και άλλων δομών
- Η γενική περίπτωση δήλωσης του προτύπου μίας δομής είναι:

```
struct όνομα_προτύπου
{
        τύπος_πεδίου_1    όνομα_πεδίου_1;
        τύπος_πεδίου_2    όνομα_πεδίου_2;
        ...
        τύπος_πεδίου_ν    όνομα_πεδίου_ν;
};
```

- Παραδείγματα προτύπων δομών:

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

```
struct person
{
        char name[50];
        int age;
        float height;
};
```

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

Πρότυπο 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

- Αν, για παράδειγμα, το πρόγραμμα απαιτεί να υπάρχει ένα πεδίο που να δηλώνει το βάρος του ανθρώπου και ένα άλλο πεδίο που να δηλώνει τη διεύθυνση κατοικίας του, τότε η δομή `person` του προηγούμενου παραδείγματος θα έπρεπε π.χ. να έχει την ακόλουθη μορφή:

```
struct person
{
        char name[50];
        int age;
        float height;
        float weight;
        char address[30];
};
```

Παρατηρήσεις

- Το πρότυπο μίας δομής αποτελεί δήλωση ενός τύπου και όχι δήλωση μεταβλητής, δηλαδή, όταν δηλώνεται το πρότυπο μίας δομής, ο μεταγλωττιστής δεν δεσμεύει μνήμη για την αποθήκευσή του

- Συνήθως, το πρότυπο μίας δομής δηλώνεται με καθολική εμβέλεια πριν από τη συνάρτηση `main()`, ώστε να είναι ορατό σε όλο το πρόγραμμα

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

- Σε αυτή την περίπτωση, οι υπόλοιπες συναρτήσεις του προγράμματος αγνοούν την ύπαρξή του και δεν μπορούν να το χρησιμοποιήσουν για να δηλώσουν δομές τέτοιου τύπου

Δήλωση Δομής

- Μία μεταβλητή, της οποίας ο τύπος είναι δομή μπορεί να δηλωθεί με έναν από τους παρακάτω δύο τρόπους

- **Α' Τρόπος (θεωρώντας ότι έχει δηλωθεί το πρότυπό της)**

  Γενική μορφή δήλωσης:

```
struct όνομα_προτύπου όνομα_δομής_1,..., όνομα_δομής_ν;
```

Π.χ.

```
struct person person_1;
```

```
struct company comp_1, comp_2;
```

- **Β' Τρόπος (δήλωση ταυτόχρονα με τη δήλωση του προτύπου της)**

Π.χ.

```
struct book
{
        char title[100];
        int year;
        float price;
} book_1, book_2;
```

Παρατηρήσεις (Ι)

- Όταν δηλώνεται μία δομή ο μεταγλωττιστής **δεσμεύει μνήμη** για την αποθήκευσή της
- Το μέγεθος της μνήμης που δεσμεύεται είναι **τουλάχιστον ίσο** με το άθροισμα της μνήμης που δεσμεύεται **για καθένα από τα πεδία της**
- Συνεπώς, η έξοδος του προγράμματος αναμένεται να είναι 12...

```
#include <stdio.h>

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

int main()
{
        struct date date_1;
        printf("%d\n", sizeof(date_1));
        return 0;
}
```

...και όντως, είναι 12 !!

Παρατηρήσεις (II)

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

```
#include <stdio.h>

struct date
{
        int day;
        int month;
        int year;
        char test;
};

int main()
{
        struct date date_1;
        printf("%d\n", sizeof(date_1));
        return 0;
}
```

Απαντήσατε 13
αλλά εμφανίστηκε... 16 ???
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 (ΙΙΙ)

- Θυμηθείτε ότι το μέγεθος της μνήμης που δεσμεύεται είναι τουλάχιστον ίσο (και όχι απαραίτητα ακριβώς ίσο) με το άθροισμα της μνήμης που δεσμεύεται για καθένα από τα πεδία της
- Στην πράξη, το μέγεθος της μνήμης που δεσμεύεται για μία δομή εξαρτάται από τον υπολογιστή που εκτελείται το πρόγραμμα και από τους τύπους των πεδίων δεδομένων
- Π.χ., μπορεί να απαιτείται κάθε πεδίο της δομής να αποθηκεύεται σε μία θέση μνήμης που η τιμή της να είναι πολλαπλάσια κάποιου αριθμού (συνήθως του 4) ή πολλαπλάσια του αριθμού των οκτάδων που δεσμεύει ο μεγαλύτερος τύπος δεδομένων των πεδίων της δομής

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()
{
    printf("Size = %d\n",sizeof(test));
    return 0;
}
```

Έξοδος: `Size = 24`

Έξοδος: `Size = 28`

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

Δήλωση Δομής με χρήση του προσδιοριστικού typedef

- Ένα πρότυπο δομής μπορεί να δηλωθεί εναλλακτικά με χρήση του προσδιοριστικού **typedef**
- Σε αυτή την περίπτωση το όνομα του προτύπου εισάγεται μετά το δεξί άγκιστρο, π.χ.

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

- Αν το πρότυπο μίας δομής έχει δηλωθεί με χρήση της εντολής **typedef**, τότε **δεν προσθέτουμε τη λέξη struct** πριν από τη δήλωση μίας δομής
- Δηλ. αν είχαμε δηλώσει την παραπάνω δομή (**book**), θα μπορούσαμε να δηλώσουμε τις μεταβλητές **book_1** και **book_2** ως εξής:

```
book book_1, book_2;
```

Παρατηρήσεις

- Συνήθως, το προσδιοριστικό `typedef` χρησιμοποιείται για τη δημιουργία συνωνύμων με βασικούς τύπους δεδομένων

- Π.χ. με τη δήλωση:

```
typedef unsigned int size_t;
```

δημιουργείται ένας νέος τύπος δεδομένων που ονομάζεται `size_t` και είναι συνώνυμο του τύπου `unsigned int`

Δηλ. οι δηλώσεις:

```
unsigned int i;
```

και

```
size_t i;
```

είναι ισοδύναμες

- Π.χ. με τη δήλωση:

```
typedef int arr[100];
```

δημιουργείται ένας νέος τύπος δεδομένων που ονομάζεται `arr` και είναι ένας πίνακας 100 ακεραίων

Δηλ. με τη δήλωση:

```
arr arr1;
```

η μεταβλητή `arr1` είναι και αυτή ένας πίνακας 100 ακεραίων

Αρχικοποίηση πεδίων δομής (Ι)

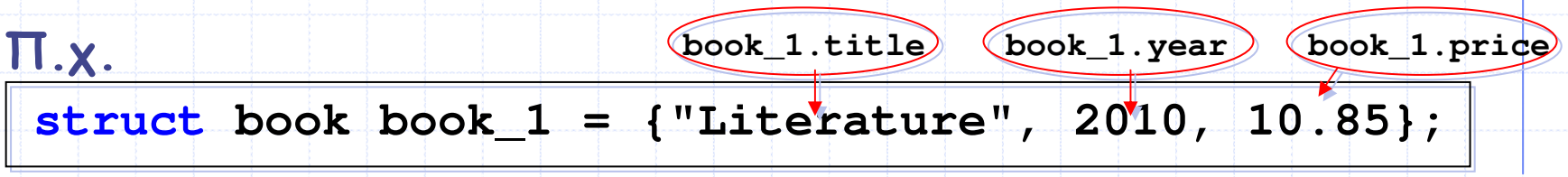
- Ο πιο συνηθισμένος τρόπος για να προσπελάσουμε τα πεδία μίας δομής είναι να γράψουμε το όνομα της δομής να προσθέσουμε τον τελεστή τελεία (.) και μετά το όνομα του πεδίου που μας ενδιαφέρει
- Το επόμενο πρόγραμμα δηλώνει το πρότυπο μίας δομής με όνομα **book**, αρχικοποιεί τα πεδία της δομής **book_1** και τα εμφανίζει στην οθόνη

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

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

int main()
{
	struct book book_1;

	strcpy(book_1.title, "Literature");
	book_1.year = 2010;
	book_1.price = 10.85;
	printf("%s %d %.2f\n", book_1.title, book_1.year,  book_1.price);
	return 0;
}
```

Αρχικοποίηση πεδίων δομής (ΙΙ)

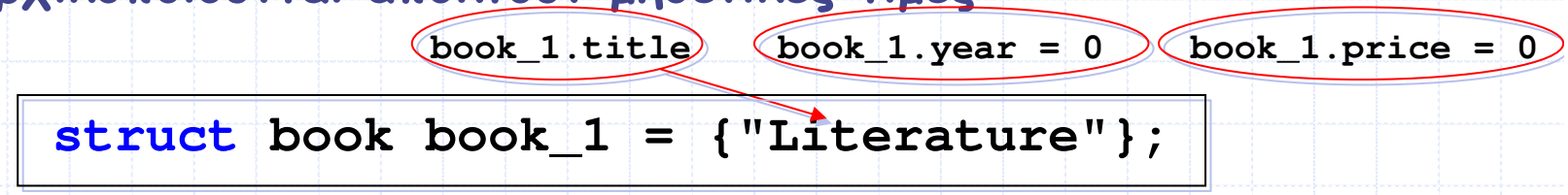
- Εναλλακτικός τρόπος αρχικοποίησης μίας δομής είναι μαζί με τη δήλωσή της (αφού – εννοείται – έχει δηλωθεί το πρότυπό της)

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 book_1 = {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, η δήλωση μίας δομής μαζί με τη δήλωση του προτύπου της μπορεί να επεκταθεί για την απόδοση αρχικών τιμών στα πεδία της

Π.χ.

```
struct book
{
        char title[100];
        int year;
        float price;
} book_1 = {"Literature", 2010, 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 **book_1** με χρήση δεικτών

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

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

int main()
{
        char *ptr1;
        int *ptr2;
        float *ptr3;
        struct book book_1;

        strcpy(book_1.title, "Literature");
        book_1.year = 2010;
        book_1.price = 10.8;

        ptr1 = book_1.title;
        ptr2 = &book_1.year;
        ptr3 = &book_1.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()
{
        struct student stud1, stud2;

        stud1.code = 1234;
        stud1.grd = 6.7;
        stud2 = stud1; /* Αντιγραφή δομών. */
        printf("Code: %d Grade: %.2f\n", stud2.code, stud2.grd);
        return 0;
}
```

Παρατηρήσεις (I)

- Με την εντολή:

```
stud2 = stud1;
```

οι τιμές των πεδίων της δομής stud1 αντιγράφονται στα αντίστοιχα πεδία της δομής stud2

Δηλαδή, η παραπάνω εντολή είναι ισοδύναμη με:

```
stud2.code = stud1.code;
stud2.grd = stud1.grd;
```

Αν οι stud1 και stud2 δεν ήταν μεταβλητές του ίδιου προτύπου δομής, η εντολή stud2 = stud1; 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)

- 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(stud1 == stud2)
```

ή

```
if(stud1 != stud2)
```

- 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((stud1.code == stud2.code) && (stud1.grd == stud2.grd))
```

Παράδειγμα δομής που περιέχει πίνακες

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

struct student
{
	int age;
	char name[50];
	float grades[10];
};

int main()
{
	struct student stud_1;

	strcpy(stud_1.name,"somebody");	/* Αντιγραφή του αλφαριθμητικού "somebody" στο πεδίο name. */
	stud_1.grades[0] = 8.5;
	stud_1.grades[1] = 7.5;

	printf("Values:  %s  %c  %c\n",stud_1.name,stud_1.name[0], *stud_1.name);
	return 0;
}
```

Έξοδος:

```
Values: somebody s s
```

Παράδειγμα δομής που περιέχει δείκτες

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

struct student
{
	char* name;
	float* avg_grade;
};

int main()
{
	float grade;
	struct student stud_1;

	grade = 8.5;

	stud_1.name = "somebody";
	stud_1.avg_grade = &grade;

	printf("%s %.2f\n",stud_1.name + 3,*stud_1.avg_grade);
	return 0;
}
```

Έξοδος:

```
ebody 8.50
```

Δομή που περιέχει δομή

- Μία δομή μπορεί να περιέχει μία ή περισσότερες δομές, οι οποίες ονομάζονται **ένθετες δομές (nested structures)**
- Το πρότυπο μιας **ένθετη δομής** πρέπει **να δηλώνεται πριν** από τη δήλωση 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 δηλωθεί, μπορεί να
χρησιμοποιηθεί σαν τύπος ένθετης δομής. */
{
	char name[50];
	double price;
	struct date s_date;
	struct date e_date;
};
```

Παράδειγμα δομής που περιέχει δομή (2/2)

```
int main()
{
        struct product prod_1;
        strcpy(prod_1.name, "product");
        prod_1.s_date.day = 1;
        prod_1.s_date.month = 9;
        prod_1.s_date.year = 2012;

        prod_1.e_date.day = 1;
        prod_1.e_date.month = 9;
        prod_1.e_date.year = 2015;

        prod_1.price = 7.5;
        printf("The     product's     life     is     %d     years\n",
prod_1.e_date.year - prod_1.s_date.year);
        return 0;
}
```

Έξοδος:

The product's life is 3 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 τον τύπο `int` υποστηρίζουν και τους τύπους `char`, `short` και `long` ως επιτρεπτούς τύπους δεδομένων ενός **πεδίου bit**

Πεδία Δομής με μέγεθος bit (ΙΙ)

Παράδειγμα:

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

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 char, τότε ο μεταγλωττιστής δεσμεύει 1 byte μνήμης, από το οποίο τα 2 bits δεν χρησιμοποιούνται
- Αν ο τύπος δεδομένων των πεδίων είχε δηλωθεί σαν unsigned int αντί για unsigned char, τότε ο μεταγλωττιστής θα δέσμευε 4 bytes μνήμης και δεν θα χρησιμοποιούσε: (4 * 8) – 6 = 26 bits
- Το μεγάλο πλεονέκτημα της χρήσης των πεδίων bit είναι η εξοικονόμηση μνήμης
- Στο προηγούμενο παράδειγμα ο μεταγλωττιστής δεσμεύει 1 byte μνήμης αντί για 3 bytes που θα δέσμευε αν δεν χρησιμοποιούσαμε πεδία bit (δηλ. για κάθε αποθήκευση των στοιχείων μίας εγγραφής εξοικονομούμε 2 bytes)
- Άρα, αν έπρεπε να αποθηκεύσουμε σε ένα αρχείο 200.000 εγγραφές, θα εξοικονομούσαμε 400.000 bytes μνήμης, κ.ο.κ.

Παρατηρήσεις (Ι)

- Για **βέλτιστη εξοικονόμηση** μνήμης **οι δηλώσεις των πεδίων bit** συνίσταται **να γίνονται όλες μαζί** και να μην παρεμβάλλονται δηλώσεις απλών πεδίων μεταξύ των
- Κατά τη δήλωση μίας δομής, προτείνεται τα **πεδία bit** να δηλώνονται στην αρχή της δομής
- 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 (ΙΙ)

```c
#include <stdio.h>

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

int main()
{
        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()
{
        struct person person_1;

        unsigned char* ptr;

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

        return 0;
}
```

Παρατηρήσεις (IV)

- Η τιμή ενός πεδίου bit μπορεί να είναι θετική ή αρνητική, ανάλογα με το αν ο μεταγλωττιστής διαχειρίζεται το bit υψηλότερης σημασίας **σαν πρόσημο**
- Για να αποφύγετε την περίπτωση του προσήμου, να δηλώνετε τα πεδία bit με τον τύπο `unsigned`
- Αν το μέγεθος ενός πεδίου bit είναι ένα bit, πρέπει υποχρεωτικά να δηλωθεί ως `unsigned`, γιατί **ένα bit δεν μπορεί να έχει πρόσημο**

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.

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

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

int main()
{
        struct student *stud_ptr;
        struct student stud;

        stud_ptr = &stud;

        strcpy((*stud_ptr).name, "somebody");
        (*stud_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 – και τότε θα κλάψετε - 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 *stud_ptr;
```

οι παρακάτω εκφράσεις, είναι ισοδύναμες:

`stud_ptr->name`	ισοδύναμη με	`(*stud_ptr).name`
`stud_ptr->grade`	ισοδύναμη με	`(*stud_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:

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

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

int main()
{
        struct student *stud_ptr;
        struct student stud;

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

        stud_ptr = &stud;
        printf("N: %s G: %.2f\n", stud_ptr->name, stud_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 προτύπου 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 (ΙΙ)

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

  Π.χ. για την προηγούμενη δομή `stud` (με πρότυπο `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 διοχετευθεί σαν παράμετρος σε μία συνάρτηση είτε με κλήση **μέσω τιμής** είτε με κλήση **μέσω αναφοράς**
- Υπενθυμίζεται ότι, όταν γίνεται κλήση συνάρτησης **μέσω τιμής**, 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 κλήσης **μέσω αναφοράς**, 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

Κλήση μέσω τιμής (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;
```

Κλήση μέσω τιμής (ΙΙ)

- Κατά την κλήση της συνάρτησης **μέσω τιμής** (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`

Κλήση μέσω αναφοράς (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;
```

Κλήση μέσω αναφοράς (ΙΙ)

- Κατά την κλήση της συνάρτησης **μέσω αναφοράς** (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 τη μεταβίβαση μίας δομής σε μία συνάρτηση προτείνεται να γίνεται χρήση κλήσης **μέσω αναφοράς**, ακόμα και αν δεν απαιτείται αλλαγή στις 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 ???

```
#include <stdio.h>

union sample
{
        char ch;
        int i;
        double d;
};
int main()
{
        printf("Size: %d\n",sizeof(sample));
        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 ???

```
#include <stdio.h>

union sample
{
        char ch;
        int i;
        double d;
};
int main()
{
        union sample sample_1;

        sample_1.ch = 'a';
        printf("%c %d %f\n",sample_1.ch,sample_1.i,sample_1.d);

        sample_1.i = 64;
        printf("%c %d %f\n",sample_1.ch,sample_1.i,sample_1.d);

        sample_1.d = 12.48;
        printf("%c %d %f\n",sample_1.ch,sample_1.i,sample_1.d);
        return 0;
}
```

Έξοδος:

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