Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου

ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Διάλεξη 8: Δομές, Ενώσεις και Απαριθμητοί Τύποι

(Κεφάλαιο 16, KNK-2ED)

Δημήτρης Ζεϊναλιπούρ

http://www.cs.ucy.ac.cy/courses/EPL232

Περιεχόμενο Διάλεξης 8

- **Δομές (Structures)**
  - Αναπαράσταση στη Μνήμη, Δηλώσεις & Τύποι Δομών, Αρχικοποίηση
  - Εμφωλευμένες Δομές & Πίνακες
  - Δείκτες σε Δομές, Δομές ως ορίσματα συναρτήσεων και τιμές επιστροφής
  - Παραδείγματα: Σύγκριση, sameFamily, addToFamily
  - Δομές, sizeof και Θέματα Ευθυγράμμισης Μνήμης
- **Ενώσεις (Unions)**
  - Αναπαράσταση στη Μνήμη, Δηλώσεις & Αρχικοποίηση
- **Απαριθμητοί Τύποι Δεδομένων (Enumerations)**
  - Αναπαράσταση στη Μνήμη, Δηλώσεις & Αρχικοποίηση

Δομές (Structures)

- Δομή είναι μια συλλογή από μια ή περισσότερες μεταβλητές, πιθανώς διαφ**ορετικών τύπων** που ομαδοποιούνται με ένα όνομα για ευκολότερο χειρισμό.
- Ορισμός δομών γίνεται με τη σύνταξη:

```
struct name
{
  δηλώσεις πεδίων
};
```

- Παράδειγμα:

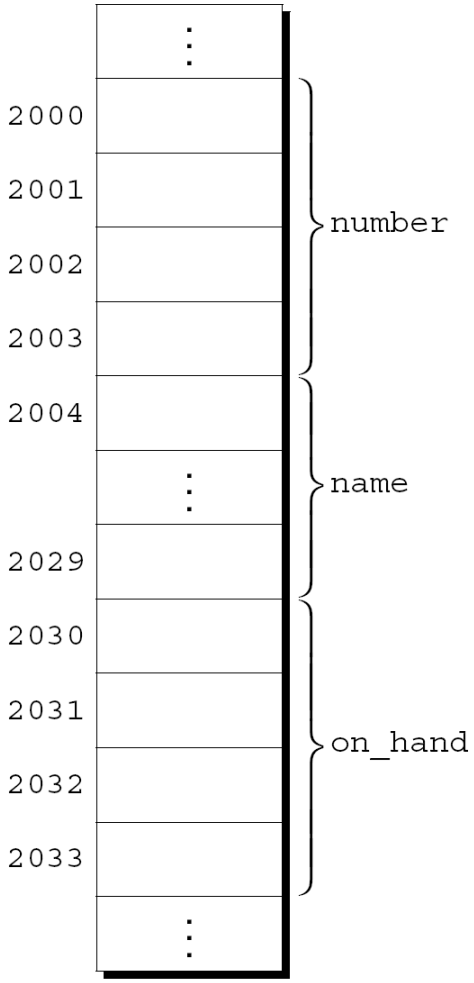
```
struct Person {
    char    firstName[15];
    char    lastName[15];
    char    gender;              /* M  ή  F */
    int     age;
};
```

Δομές (Αναπαράσταση στη Μνήμη)

- Μια δομή αναπαριστάται σε συνεχόμενες διευθύνσεις μνήμης, όπως και οι πίνακες.

```
struct {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
} part1, part2;
```

Το part1, part2 ορίζει 2 μεταβλητές της εν λόγω δομής.



Δομές (Δήλωση & Τύποι Δομών)

- Η λέξη **struct** εισάγει τη δήλωση μιας δομής. Μπορεί να ακολουθείται **(προαιρετικά)** από κάποιο **όνομα**, που αποκαλείται ***ετικέτα δομής*** και "ονοματίζει" το είδος της δομής. Π.χ., struct **Person** { … }
- Οι μεταβλητές που κατονομάζονται μέσα στη δομή ονομάζονται ***μέλη*** ή ***πεδία***., *π.χ.*, `int age;`
- Μια δήλωση struct ορίζει ένα τύπο. Για να δηλώσουμε μεταβλητές ή πίνακες τύπου δομής γράφουμε:

```
struct Person x;
struct Person people[40];
```

- Εναλλακτικά μπορούμε επίσης να παραλείψουμε το όνομα της δομής αλλά να περιγράψουμε τα μέλη της:

```
struct {
        char   firstName[15];
        char   lastName[15];
        char   gender;
        int    age;
} x, people[40];
```

Δομές (Δήλωση & Τύποι Δομών)

```
struct {
    char    firstName[15];
    char    lastName[15];
    char    gender;
    int     age;
} x, people[40];
```

Ίδιο με

```
struct person {
    char    firstName[15];
    char    lastName[15];
    char    gender;
    int     age;
};
struct person x, people[40];
```

και

```
typedef struct person PERSON;
PERSON x, people[40];
```

Συνιστώμενος τρόπος Δήλωσης Δομής και Ορισμού Δομής για το ΕΠΛ232

```
typedef struct {
    char    firstName[15];
    char    lastName[15];
    char    gender;
    int     age;
} PERSON;

PERSON x, people[40];
```

Δομές (Αρχικοποίηση)

- **Αρχικοποίηση** μιας **μεταβλητής** ή **πίνακα τύπου δομής** γίνεται αποδίδοντας τιμές στα μέλη ως εξής:

```
PERSON x = {"Ανδρέας", "Ανδρέου", 'M', 43};

PERSON people[] =
        {"Ανδρέας", "Ανδρέου",     'M', 43,
         "Μαρία",   "Γεωργίου",    'F', 38,
         "Χρίστος", "Χαραλάμπους", 'M', 14
         };
```

ή (καλύτερα)

```
PERSON people[] = {{"Ανδρέας", "Ανδρέου", 'M',
  43}, {"Μαρία", "Γεωργίου", 'F', 38},
  { "Χρίστος", "Χαραλάμπους", 'M', 14}};
```

- **Αναφορά σε μέλος** μιας δομής γίνεται μέσω της κατασκευής:

όνομα_δομής.μέλος

π.χ.,

```
scanf("%d", &x.age);
if (x.age > 40)
printf("Age: %d \n", x.age);
```

Εμφωλευμένες Δομές (Nested Structures)

Δομές μπορεί να είναι **αλληλένδετες**, δηλαδή να **φωλιάζοντας (εμφωλεύονται)** η μια μέσα στην άλλη, δημιουργώντας πιο πολύπλοκες δομές:

```
struct Person {
        char firstName[15];
        int age;
};
struct family {
        struct Person father;
        struct Person mother;
        int numofchild;
        struct Person children[5];
};

int main(void){
  struct Person x, y, z;
  struct family fml;
  fml.numofchild = 2;
  strcpy(fml.father.firstName, "Joe");
  strcpy(fml.children[0].firstName, "Mary");
  …
```

ή

(καλύτερα)

```
typedef struct {
        char firstName[15];
        int age;
} PERSON;
typedef struct {
        PERSON father;
        PERSON mother;
        int numofchildren;
        PERSON children[5];
} FAMILY;

int main(void){
  PERSON x, y, z;
  FAMILY fml;
  fml.numofchildren = 2;
  strcpy(fml.father.firstName, "Joe");
  strcpy(fml.children[0].firstName, "Mary");
  …
```

Εμφωλευμένες Δομές (Nested Structures)

```
typedef struct {
   char firstName[15];
   char lastName[15];
   char gender;
   int age;
} PERSON;

typedef struct {
   char title[30];
   int salary;
   PERSON employee;
} POSITION;
```

```
int main(void) {
   POSITION programmer, secretaries[10];
   strcpy(programmer.title, "Software Engineer");
   programmer.salary = 2500; /*per month */
   strcpy(programmer.employee.firstName, "Joe");
   strcpy(programmer.employee.lastName, "Hacker");
   ....
   strcpy(secretaries[0].employee.firstName, "Jane");
}
```

Δομές και Συναρτήσεις (Structures and Functions)

- Επιτρεπτές πράξεις σε μια δομή είναι:
  - η **αντιγραφή** της, η **απόδοση τιμής** σ' αυτήν σαν σύνολο, η **εξαγωγή της διεύθυνσής** της, και η προσπέλαση των μελών της.
  - Μεταβλητές τύπου δομής μπορούν να **περαστούν** ως **ορίσματα** σε συναρτήσεις όπως επίσης και να **επιστραφούν ως αποτελέσματα συναρτήσεων**
- Παράδειγμα:

```
PERSON inc_age (PERSON x){
    x.age += 1;
    return x;
 }
  …
PERSON x1, x2;
x2 = inc_age(x1);
```

Πέρασμα δια τιμής!

(το x αντιγράφεται μέσα στη συνάρτηση)

(αργότερα θα δούμε πως περνιούνται δομές δια διεύθυνσης)

Άσκηση 1

- Δομές *ΔΕΝ* μπορούν να συγκριθούν:
  π.χ. η έκφραση x1 == x2 ΔΕΝ είναι έγκυρη.

- **Άσκηση 1:** Να γράψετε συνάρτηση η οποία παίρνει δύο παραμέτρους τύπου struct PERSON και τις συγκρίνει επιστρέφοντας TRUE εάν είναι τα ίδια άτομα.

```
/* Εργασία 1 */
#define NAMESIZE 15
typedef struct {
    char firstName[NAMESIZE];
    char lastName[NAMESIZE];
    char gender;
    int age;
} PERSON;
```

```
// function prototype
int samePerson(PERSON, PERSON);
```

Άσκηση 1

```
/* Εργασία 1 */
#define NAMESIZE 15
typedef struct {
    char firstName[NAMESIZE];
    char lastName[NAMESIZE];
    char gender;
    int age;
} PERSON;
int samePerson(PERSON, PERSON); // function prototype

int samePerson(PERSON x , PERSON y) {
    return( (strcmp(x.firstName, y.firstName) == 0) &&
            (strcmp(x.lastName, y.lastName) == 0) &&
            (x.gender == y.gender) &&
            (x.age == y.age));
}

int main() {
    PERSON per1 = {"Marios", "Michael", 'M', 24};
    PERSON per2 = {"Antonis", "Charalambous", 'M', 22};
    if (samePerson(per1,per1)) printf("Same Persons");
    else printf("Different Persons");
    return 0;
}
```

Δείκτες σε Δομές (Pointer to Structures)

- Μπορούμε επίσης να χρησιμοποιήσουμε δείκτες σε δομές.
- **Δείκτης σε Δομή – παράδειγμα δήλωσης:**

```
struct Person p, *pp;  ή PERSON p, *pp;
```

η μεταβλητή **pp** είναι δείκτης προς μια δομή τύπου Person .

- Έτσι μπορούμε να γράψουμε

```
pp = &p;
printf("%s", (*pp).firstName);
```

όπου ***pp** είναι η δομή που δείχνεται από τον δείκτη, ενώ **(*pp).firstName** είναι το πρώτο πεδίο της δομής.

- Προσοχή: η προτεραιότητα του τελεστή μέλους δομής, '.', είναι *μεγαλύτερη* από αυτή του τελεστή έμμεσης αναφοράς, '*'.
- Επομένως οι παρενθέσεις στο **(*pp).firstName** είναι απαραίτητες.

Δείκτες σε Δομές & Συναρτήσεις (Pointer to Structures)

- Δείκτες για δομές χρησιμοποιούνται τόσο συχνά που παρέχεται ο πιο κάτω εναλλακτικός συμβολισμός ως συντομογραφία:

  `(*p).μέλος_δομής` = `p->μέλος_δομής`

- **Πέρασμα Παραμέτρων Δια Αναφοράς:** Σε περίπτωση που θέλουμε μία συνάρτηση να **αλλάξει** τα **περιεχόμενα** μίας δομής, περνάμε ως παράμετρο στη συνάρτηση το **δείκτη της δομής** αυτής
  - όπως ακριβώς εργαζόμασταν και σε άλλες δομές δεδομένων, π.χ., :

```
PERSON p;
. . .
initPerson(&p);                          /* Κλήση συνάρτησης */
. . .
void initPerson(PERSON *p){
    strcpy( p->firstName, "Ανδρέας");
    strcpy( p->lastName, "Ανδρέου");
    p->gender = 'M';
    p->age =  43;
}
```

Άσκηση 2

- **Άσκηση 2:** Να γράψετε συνάρτηση η οποία παίρνει ως δεδομένο εισόδου μια **οικογένεια** (τύπου struct **Family**) και ένα **άτομο** (τύπου struct **Person**), και αν το άτομο **δεν ανήκει** ήδη στα παιδιά της οικογένειας τότε **το προσθέτει**.

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

#define NAMESIZE 15
#define CHILDSIZE 5
```

```
typedef struct {
    char firstName[NAMESIZE];
    char lastName[NAMESIZE];
    char gender;
    int age;
} PERSON;

typedef struct {
        PERSON father;
        PERSON mother;
        int numofchildren;
        PERSON children[CHILDSIZE];
} FAMILY;
```

```
int addToFamily(FAMILY *, PERSON);
```

Άσκηση 2

```
int main() {
    FAMILY fml;
    PERSON per1 = {"Marios", "Michael", 'M', 24};
    fml.numofchildren = 0;
    addToFamily(&fml,per1);
}
```

```
int addToFamily(FAMILY *fml, PERSON per) {
    if (fml == NULL) return EXIT_FAILURE;
    if (fml->numofchildren == CHILDSIZE) {
        printf("Children Array is full!"); return EXIT_FAILURE;
    }

    // find if x belongs to family
    for (int i=0; i<fml->numofchildren; i++) {
        if ((strcmp(fml->children[i].firstName, per.firstName) == 0) &&
            (strcmp(fml->children[i].lastName, per.lastName) == 0) &&
            (fml->children[i].gender == per.gender) &&
            (fml->children[i].age == per.age) )
         { printf("Already Family Member!"); return EXIT_FAILURE; }
    }

    strcpy(fml->children[fml->numofchildren].firstName, per.firstName);
    strcpy(fml->children[fml->numofchildren].lastName, per.lastName);
    fml->children[fml->numofchildren].gender = per.gender;
    fml->children[fml->numofchildren].age = per.age;
    (fml->numofchildren)++;
    return EXIT_SUCCESS;
}
```

Δομές, sizeof και Θέματα Ευθυγράμμισης Μνήμης

Το Πρόβλημα

```
typedef struct {
    int ID;                 // 4B
    char state[3];          // 3B
    int salary;             // 4B
} EMPLOYEE;
```

`sizeof(EMPLOYEE)`=12 αντί 11 σε μία **4-byte aligned** μνήμη!

- Αυτό συμβαίνει επειδή τα δεδομένα είναι ευθυγραμμισμένα σε 4B και το state έχει 1 byte padding
- **Γρηγορότερη Πρόσβαση στα δεδομένα ☺ (προτιμητέα μέθοδος)** αλλά **Σπαταλείται Μνήμη ☹**

Δομές, sizeof και Θέματα Ευθυγράμμισης Μνήμης

- **Ευθυγράμμιση Μνήμης (Memory Alignment):**
  - Ο **μεταγλωττιστής** προσθέτει **padding (κενά bytes)** σε δομές για να βρίσκονται σε διευθύνσεις πολλαπλάσιες μιας βασικής μονάδας, π.χ., 2b,4b,8b
  - Το **padding** δεν είναι ορατό σε μεταβλητές, ούτε και μας ενοχλεί αλλά **καταλαμβάνει χώρο**!
  - Ο λόγος ύπαρξης του είναι ότι επιτρέπει στο Λ.Σ. να **ανακτά γρηγορότερα δεδομένα** από την μνήμη (π.χ., 4 bytes ανά κύκλο αντί 1 byte ανά κύκλο)
  - Ωστόσο απαιτεί **χώρο** που μπορεί να **ΜΗΝ** υπάρχει
    - π.χ., σε περίπτωσης όπου η μνήμη είναι πολύ **περιορισμένη** (π.χ., σε μικροεπεξεργαστές) ή εάν θέλουμε να **χωρέσουμε πολλά δεδομένα στη μνήμη**, (π.χ., ένα μεγάλο ευρετήριο)

Δομές, sizeof και Θέματα Ευθυγράμμισης Μνήμης

Παράδειγμα

Όρισμα του GCC μόνο

```
typedef struct {
    int ID;                 // 4B
    char state[3];          // 3B
    int salery;             // 4B
} __attribute__ ((__packed__)) EMPLOYEE;
```

`sizeof(EMPLOYEE)` επιστρέφει **11** σε μια 4-byte aligned μνήμη!

- Αυτό συμβαίνει επειδή τα δεδομένα γίνονται pack (συμπτύσσονται) χωρίς επιπρόσθετο padding
- **Αργότερη Πρόσβαση ☹** αλλά **Δεν Σπαταλείται Μνήμη ☺ (χρησιμοποιείται όταν μας ενδιαφέρει ο χώρος!)**

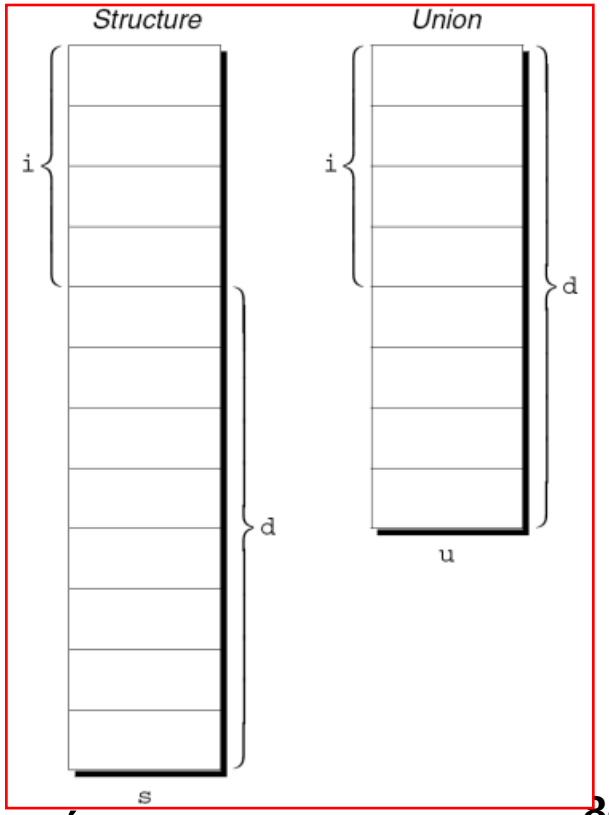
Περισσότερα: http://gcc.gnu.org/onlinedocs/gcc/Type-Attributes.html

Ενώσεις (Unions)

- Μια **ένωση (union),** αποτελείται από ένα ή περισσότερα πεδία, με ενδεχομένως διαφορετικό τύπο, αλλά καταλαμβάνει **χώρο ίσο μόνο με το μεγαλύτερο πεδίο.**

```
struct {
   int i;
   double d;
} s;
```

```
union {
   int i;
   double d;
} u;
```



- Η βασική **διαφορά** με τη **δομή**, είναι ότι ο μεταγλωττιστής δεσμεύει **αρκετό χώρο** μόνο για το **μεγαλύτερο από τα πεδία**.
- Συνεπώς μια ένωση αποθηκεύει ανά πάσα στιγμή **ΜΟΝΟ** ένα από τα **πεδία**.
  - Ο προγραμματιστής (μέσω επιπλέον μεταβλητής) πρέπει να θυμάται ποιο.
  - Εάν αλλάξει το `u.i` χάνεται το `u.d` και αντίστροφα

Ενώσεις (Παράδειγμα)

```
#define INT_TYPE   1
#define REAL_TYPE  2

struct item {
    int type;
    union {
        int    ival;
        double dval;
    } info;
}
…
struct item x;
…
if (x.type == INT_TYPE)
   printf("value of info = %d\n”, x.info.ival);
if (x.type == REAL_TYPE)
   printf("value of info = %lf\n", x.info.dval);
```

Σημαία (flag) που χρησιμοποιείται για να θυμάται το πρόγραμμα τι έχει αποθηκευτεί μέσα στο union.

Ενώσεις (Unions)

- **Εφαρμογές:** Για εξοικονόμηση μνήμης αλλά κυρίως για δημιουργία ανάμεικτων δομών δεδομένων
  - Παράδειγμα ενός Page το οποίο κατά την εκτέλεση μπορεί να συμπεριφέρεται ως RootPage, DirectoryPage, IndexPage ή DataPage (ελεγχόμενο από πεδίο typ)

```
typedef struct {
    short typ;
    short crc;
    short pwc;
    char siz;
    int ppa;
    union {
        RootP rootP;
        DirP dirP;
        DataP dataP;
        IdxP idxP;
    };
} __attribute__((packed))
Page ;
```

```
typedef struct {
    DataRec records[DREC];
} __attribute__((packed)) DataP ;
```

```
typedef struct {
    long ts;
    int val1;
} __attribute__((packed)) DataRec;
```

```
typedef struct {
    long lastTS;
    IdxRec records[IREC];
} __attribute__((packed)) IdxP ;
```

Απαριθμητοί Τύποι (Enumerations)

- **Απαριθμητός Τύπος (Enumeration):** ορίζει **μεταβλητές** των οποίων η τιμή **μπορεί** να ορίζεται, μεταξύ άλλων, από ένα προσδιορισμένο σύνολο σταθερών
- Π.χ., είδος κάρτας που επέλεξε ο χρήστης

```
typedef enum {CLUBS, DIAMONDS, HEARTS, SPADES} CARD;
CARD s = CLUBS;       /* s είναι τώρα 0 (CLUBS) */
CARD s = 10;          /* # OK, αλλά επικίνδυνο */
```

- Παρατηρήσεις
  - Όμοια σύνταξη με `struct` και `union`
  - Το `s` είναι στην ουσία μεταβλητή ακέραιου τύπου και τα `CLUBS`, `DIAMONDS`, `HEARTS`, και `SPADES` είναι συμβολικά ονόματα για τις ακέραιες τιμές 0, 1, 2, και 3.

Απαριθμητοί Τύποι (Παράδειγμα)

```
#include <stdio.h>

typedef enum {
     CLUBS,
     DIAMONDS,
     HEARTS,
     SPADES
} CARD;

int main(void) {
     CARD s;
     s = CLUBS;   /* s = 0 (CLUBS)   */
     printf("%d\n", s);
     s++;         /* s = 1 (DIAMONDS)*/
     printf("%d\n", s);
     s = SPADES; /* s = 3 (SPADES)*/
     printf("%d\n", s);
     return 0;
}
```

Ορισμός "σταθερών" ενός απαριθμητού τύπου

Πλεονεκτήματα:

a)Δε χρειάζεται να δώσουμε 4 ξεχωριστά define

b)Δηλώνουμε ρητά (στον προγραμματιστή) ότι το CARD θα πρέπει να παίρνει μόνο αυτές τις 4 τιμές (0-3), εάν και δεν αυτό δεν ελέγχεται από τον μεταγλωττιστή.

Σύνθεση Enum με struct & union

```
typedef struct {
   enum {IKIND, DKIND} kind;
   union {
    int i;
    double d;
   } u;
} Number;
```