ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Δομές, Ενώσεις και Απαριθμητοί Τύποι (Κεφάλαιο 16, KNK-2ED)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

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

Dr. Andreas Aristidou a.aristidou@ieee.org

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 **διαφορετικών τύπων** που ομαδοποιούνται με ένα όνομα για ευκολότερο χειρισμό.

- Οι ιδιότητες μιας ***δομής*** είναι διαφορετικές από αυτές ενός πίνακα.
  - Τα στοιχεία μιας δομής (***μέλη***) δεν απαιτείται να έχουν τον ίδιο τύπο.
  - Τα μέλη μιας δομής έχουν ονόματα. Για να επιλέξουμε ένα συγκεκριμένο μέλος, απλά δηλώνουμε το όνομά του και όχι τη θέση του.
- Ιδανική για την αποθήκευση μιας συλλογής από συσχετισμένα στοιχεία δεδομένων.

Ορισμός δομών γίνεται με τη σύνταξη:

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

Παράδειγμα:

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

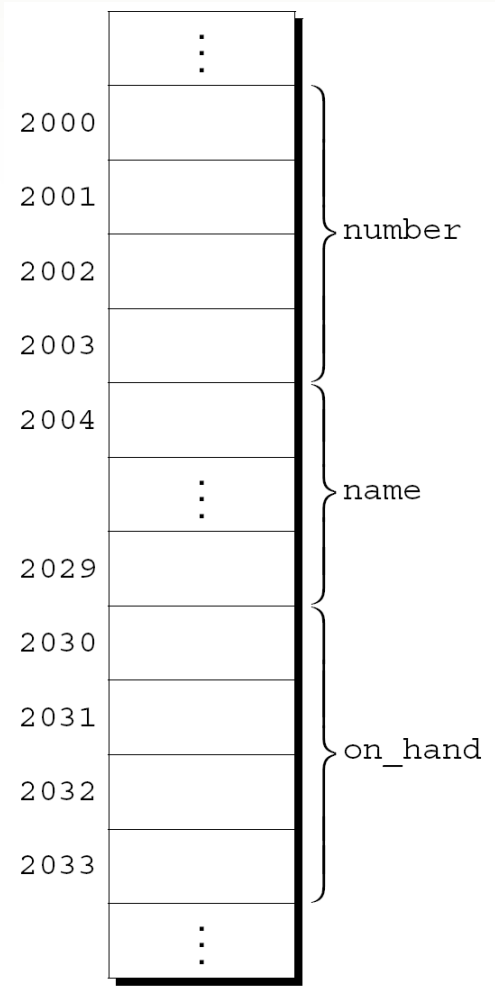
Dr. Andreas Aristidou a.aristidou@ieee.org

Δομές (Αναπαράσταση στη Μνήμη)

- Μια δομή αναπαρίσταται 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];
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Δομές (Δήλωση & Τύποι Δομών)

- Κάθε δομή αντιπροσωπεύει ένα νέο σκοπό.
- Οποιαδήποτε ονόματα δηλώνονται για αυτό το σκοπό δεν συγκρούονται με άλλα ονόματα στο ίδιο πρόγραμμα.

Για παράδειγμα, οι ακόλουθες δηλώσεις μπορούν να εμφανιστούν στο ίδιο πρόγραμμα:

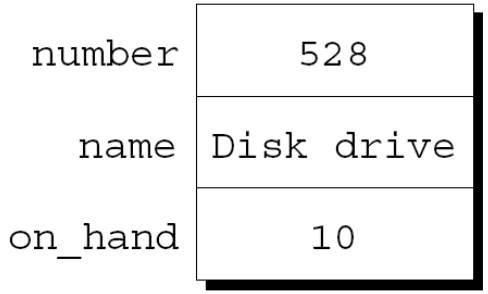
```
struct {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} part1, part2;

struct {
  char name[NAME_LEN+1];
  int number;
  char sex;
} employee1, employee2;
```

Δομές (Αρχικοποίηση)

- **Αρχικοποίηση** μιας **μεταβλητής** ή **πίνακα τύπου δομής** γίνεται αποδίδοντας τιμές στα μέλη της.
- Μια δήλωση δομής μπορεί να περιλαμβάνει και αρχικοποίηση:

```
struct {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} part1 = {528, "Disk drive", 10},
  part2 = {914, "Printer cable", 5};
```

- Το `part1` μετά την αρχικοποίηση:



Παράδειγμα

Δομές (Αρχικοποίηση)

```
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}};
```

- Η πρόσβαση σε ένα τμήμα του πίνακα γίνεται με τη χρήση subscripting:
  `print_part(inventory[i]);`
- Πρόσβαση σε ένα μέλος μέσα σε μια δομή `part` απαιτεί συνδυασμό subscripting και την επιλογή του μέλους:
  `inventory[i].number = 883;`
- Η πρόσβαση σε έναν μόνο χαρακτήρα :
  `inventory[i].name[0] = '\0';`

Δομές (Αρχικοποίηση)

- Οι αρχικοποιήσεις σε δομές έχουν τους ίδιους κανόνες με εκείνους για των πινάκων.
- Μια αρχικοποίηση μπορεί να έχει λιγότερα μέλη από αυτά της δομής.
  - Όλα τα "εναπομείναντα" μέλη παίρνουν 0 ως αρχική τιμή.
- **Καθορισμένη αρχικοποίηση - Designated Initializers (C99)**
  - Η αρχικοποίηση του `part1` από το προηγούμενο παράδειγμα:

    ```
    {528, "Disk drive", 10}
    ```

  - Κάθε τιμή μπορεί να χαρακτηριστεί από το όνομα του μέλους της:

    ```
    {.number = 528, .name = "Disk drive", .on_hand = 10}
    ```

  - Οι τιμές δεν χρειάζεται να τοποθετούνται με την ίδια σειρά που αναφέρονται στη δομή.

    ```
    {.on_hand = 10, .number = 528, "Disk drive"}
    ```

    Εδώ ο μεταγλωττιστής υποθέτει πως το `"Disk drive"` αρχικοποιεί το μέλος που ακολουθεί το `number` στη δομή.
  - Όλα τα μέλη που το πρόγραμμα προετοιμασίας αποτυγχάνει να αρχικοποιήσει ορίζονται μηδέν.

Dr. Andreas Aristidou a.aristidou@ieee.org

Λειτουργίες σε δομές

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

π.χ. 1,
```
printf("Part number: %d\n", part1.number);
printf("Part name: %s\n", part1.name);
printf("Quantity on hand: %d\n", part1.on_hand);
```

π.χ. 2,
```
scanf("%d", &x.age);
if (x.age > 40)
printf("Age: %d \n", x.age);
```

π.χ. 3,
```
part1.number = 258;
      /* changes part1's part number */
part1.on_hand++;
      /* increments part1's quantity on hand */
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Λειτουργίες σε δομές

π.χ. 4, `scanf("%d", &part1.on_hand);`

Ο τελεστής `.` προηγείται του τελεστή `&`, οπότε το `&` υπολογίζει τη διεύθυνση της `part1.on_hand`.

π.χ. 5, Η άλλη κύρια λειτουργία δομής είναι η εκχώρηση:

```
part2 = part1;
```

Το αποτέλεσμα αυτής της δήλωσης είναι η αντιγραφή του `part1.number` στο `part2.number`, το `part1.name` στο `part2.name`, κτλ.

Ο τελεστής = μπορεί να χρησιμοποιηθεί μόνο με δομές ***συμβατών*** τύπων (π.χ., δυο δομές που δηλώθηκαν μαζί).

Οι τελεστές, `==` και `!=` δεν μπορούν να χρησιμοποιηθούν σε δομές.

Dr. Andreas Aristidou a.aristidou@ieee.org

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];
```

structure tag

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 (Δήλωση & Τύποι Δομών)

- Όλες οι δομές που έχουν δηλωθεί ότι έχουν τύπο `struct part` είναι συμβατές μεταξύ τους:

```
struct part part1 = {528, "Disk drive", 10};
struct part part2;

part2 = part1;
  /* legal; both parts have the same type */
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Δομές (Δήλωση & Τύποι Δομών)

- Ως εναλλακτική λύση για την κήρυξη μιας ετικέτας δομής, μπορούμε να χρησιμοποιήσουμε το `typedef` για να ορίσετε ένα τύπο.
- Ένας ορισμός ενός τύπου με το όνομα `Part`:

```
typedef struct {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} Part;
```

- `Part` μπορεί να χρησιμοποιηθεί με τον ίδιο τρόπο όπως οι ενσωματωμένοι τύποι:

```
Part part1, part2;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Structures and Functions

Δομές και Συναρτήσεις

- 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 ολόκληρο μέσα στη συνάρτηση)

(αργότερα θα δούμε πως περνιούνται δομές δια διεύθυνσης)

Dr. Andreas Aristidou a.aristidou@ieee.org

Δομές και Συναρτήσεις

- Μια συνάρτηση με ένα όρισμα δομής:

```
void print_part(struct part p)
{
  printf("Part number: %d\n", p.number);
  printf("Part name: %s\n", p.name);
  printf("Quantity on hand: %d\n", p.on_hand);
}
```

- Ένα κάλεσμα της `print_part`:

```
print_part(part1);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Structures and Functions

Δομές και Συναρτήσεις

- Μια συνάρτηση που επιστρέφει μια δομή `part`:

```
struct part build_part(int number,
                       const char *name,
                       int on_hand)
{
  struct part p;

  p.number = number;
  strcpy(p.name, name);
  p.on_hand = on_hand;
  return p;
}
```

- Μια κλήση της `build_part`:

```
part1 = build_part(528, "Disk drive", 10);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Άσκηση Κατανόησης

- 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);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Άσκηση Κατανόησης

```
#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;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Nested Structures
Εμφωλευμένες Δομές

Δομές μπορεί να είναι **αλληλένδετες**, δηλαδή να **φωλιάζονται (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");
  …
```


Dr. Andreas Aristidou a.aristidou@ieee.org

Nested Structures
Εμφωλευμένες Δομές

```
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");
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Pointer to Structures
Δείκτες σε Δομές

- 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.

Dr. Andreas Aristidou a.aristidou@ieee.org

Pointer to Structures

Δείκτες σε Δομές & Συναρτήσεις

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

-> arrow operator

Παράδειγμα

```
#include<stdio.h>

struct dog
{
    char name[10];
    char breed[10];
    int age;
    char color[10];
};

int main()
{
    struct dog my_dog = {"tyke", "Bulldog", 5, "white"};
    struct dog *ptr_dog;
    ptr_dog = &my_dog;

    printf("Dog's name: %s\n", ptr_dog->name);
    printf("Dog's breed: %s\n", ptr_dog->breed);
    printf("Dog's age: %d\n", ptr_dog->age);
    printf("Dog's color: %s\n", ptr_dog->color);

    // changing the name of dog from tyke to jack
    strcpy(ptr_dog->name, "jack");

    // increasing age of dog by 1 year
    ptr_dog->age++;

    printf("Dog's new name is: %s\n", ptr_dog->name);
    printf("Dog's age is: %d\n", ptr_dog->age);

    // signal to operating system program ran fine
    return 0;
}
```

Expected Output:

```
Dog's name: tyke
Dog's breed: Bulldog
Dog's age: 5
Dog's color: white

After changes

Dog's new name is: jack
Dog's age is: 6
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Άσκηση Κατανόησης 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
```

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

```
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;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Άσκηση Κατανόησης 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;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

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 ☹**

Dr. Andreas Aristidou a.aristidou@ieee.org

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)

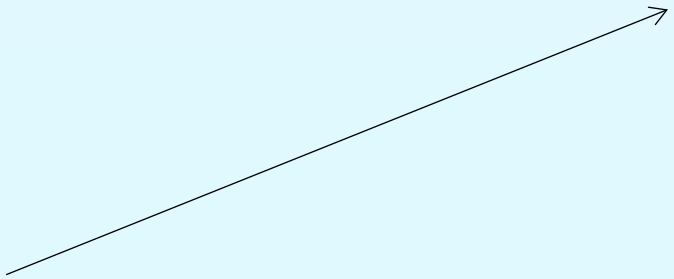
Dr. Andreas Aristidou a.aristidou@ieee.org

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

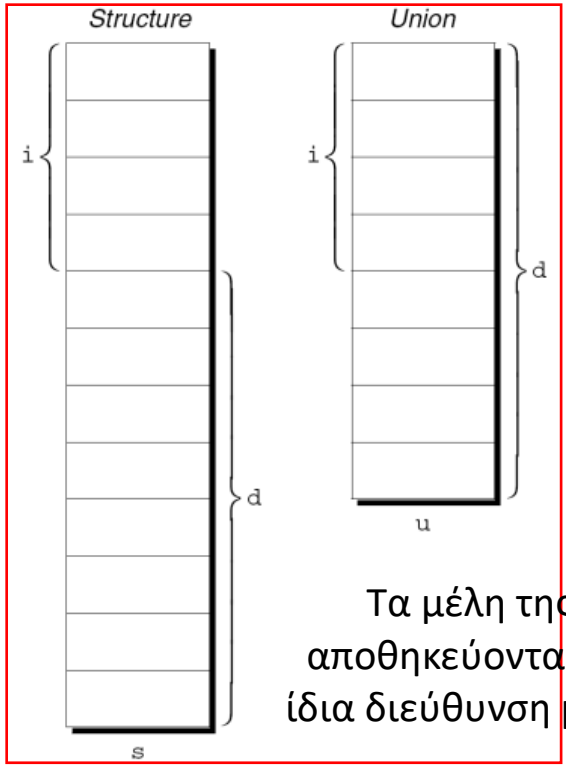
Dr. Andreas Aristidou a.aristidou@ieee.org

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;
```



Τα μέλη της `u` αποθηκεύονται στην ίδια διεύθυνση μνήμης

- 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.

Hint: *Union types were specifically not included in Java, due to the security and type-safety issues they caused in C, and because Java has safer and more expressive ways to code instance-level polymorphism. Scala does provide something approximating union types, with case classes.*

Εφαρμογές: Για εξοικονόμηση μνήμης αλλά κυρίως για δημιουργία ανάμεικτων δομών δεδομένων

Dr. Andreas Aristidou a.aristidou@ieee.org

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.

Dr. Andreas Aristidou a.aristidou@ieee.org

Απαριθμητοί Τύποι (Παράδειγμα)

```
enum dept
{RESEARCH = 20,
 PRODUCTION = 10,
 SALES = 25};
```

```
#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*.

Dr. Andreas Aristidou a.aristidou@ieee.org

Απαριθμητοί Τύποι (Παράδειγμα)

- Οι Απαριθμητοί Τύποι είναι ιδανικοί για τον προσδιορισμό του τελευταίου μέλους μιας Ένωσης που πήρε τιμή.
- Στη δομή `Number`, μπορούμε να κάνουμε το μέλος `kind` σαν απαριθμητό τύπο αντί για `int`:

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

Dr. Andreas Aristidou a.aristidou@ieee.org