Διάλεξη 5: Απαριθμητές (enums)
Δομές (structures) και Ενώσεις (unions)

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:
Εισαγωγή στις έννοιες:

- Απαριθμητές
- Δομές, φωλιασμένες δομές, τρόποι δήλωσης δομών
- Δομές ως παράμετροι σε συναρτήσεις, δείκτες σε δομές
- Αυτό-αναφορικές δομές, χρήση ενώσεων

Διδάσκων: Παναγιώτης Ανδρέου

Απαριθμητές <enum>

- Συχνά, χρειάζεται να δηλώσουμε και να χρησιμοποιήσουμε κάποιες απλές ακέραιες μεταβλητές για να επιβάλουμε ένα συγκεκριμένο πεδίο τιμών σε κάποιες μεταβλητές
- Παράδειγμα: τα χρώματα κάποιου αυτοκίνητου
  - const int cRED = 0;
  - const int cBLUE = 1;
  - …
  - int auto_colour;
  - auto_colour = cBLUE;
- Ερώτηση: Είναι λάθος η δήλωση auto_colour = -13;
- Απάντηση: **ΟΧΙ** αφού η μεταβλητή auto_colour είναι ακέραιος.

  Λογικό Λάθος: Δεν υπάρχει χρώμα με τον αριθμό -13.
- Οι απαριθμητές μπορούν να εξαλείψουν αυτό το πρόβλημα.

Απαριθμητές <enum> (συν.)

Ορισμός Απαριθμητή <enum>: Ένα σύνολο από σταθερές ακέραιες μεταβλητές. Δηλώνεται με enum { var_1, var_2, …, var_n }

- Παράδειγμα:
  - enum Color { cRED, cBLUE, cYELLOW, cGREEN, cSILVERGREY };
- Η δήλωση της μεταβλητής int auto_colour; Αλλάζει τώρα σε Color auto_colour;
- Ερώτηση: Είναι λάθος η δήλωση auto_colour = -13;
- Απάντηση: **ΝΑΙ** αφού η μεταβλητή auto_colour δεν είναι πλεόν ακέραιος αλλά τύπου Color.
- Πλεονεκτήματα Απαριθμητών:
  - Γρηγορότερη δήλωση σταθερών μεταβλητών/ Πιο αποτελεσματική συντήρηση
  - Περιορισμός του πεδίου τιμών κάποιων μεταβλητών
  - Αυτόματος Έλεγχος

Δομές (structs)

Ορισμός Δομής (struct): Δομή είναι ένα σύνολο από άλλες μεταβλητές απλές (int, char, κτλ.) ή/και σύνθετες (structs, unions, κτ.λ.)

Δηλώνεται με **struct** <όνομα δομής> { δηλώσεις πεδίων } ;

- Παράδειγμα:

```
struct address{
    char street[50];
    int number;
};
```

- Η λέξη struct εισάγει τη δήλωση μιας δομής. Οι μεταβλητές που κατονομάζονται μέσα στη δομή ονομάζονται *μέλη* ή *πεδία*.
- Μια **δήλωση** struct ορίζει ένα τύπο. Για να δηλώσουμε μεταβλητές ή πίνακες τύπου δομής γράφουμε:

```
struct address   x; //μία μεταβλητή τύπου <address>
struct address   friends_addresses[10]; //ένας πίνακες με
                                        μεταβλητές τύπου
                                        <address>
```

Δομές (structs) (συν.)

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

```
struct Person p1 =
        {“Ανδρέας”,“Ανδρέου”, ’M’, 43};
struct Person people[] = {
    {“Χρίστος”, “Ανδρέου”,     ’M’, 43},
    {“Μαρία”,   “Γεωργίου”,    ’F’, 38},
    {“Χρίστος”, “Χαραλάμπους”, ’M’, 14} };
```

Παράδειγμα Δομής Person

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

- Αναφορά σε μέλος μιας δομής γίνεται μέσω της κατασκευής:

  <όνομα δομής>.μέλος

- Εναλλακτική αρχικοποίηση `struct Person p1`

```
struct Person p1;
strcpy(p1.firstName, "Ανδρέας");
strcpy(p1.lastName, "Ανδρέου");
p1.gender = 'M';
p1.age = 43;
```

Δημιουργία Τύπων με typedef

- Για να αποφεύγετε η συνεχής χρήση του «**struct structname»**, **μπορούμε να χρησιμοποιήσουμε την `typedef`** η οποία δημιουργεί/καθορίζει ένα καινούριο τύπο.
- Η C διαθέτει την typedef για δημιουργία νέων ονομάτων τύπων δεδομένων, π.χ., `typedef unsigned short int uint16;`
- Έτσι αποφεύγουμε να γράφουμε κάθε φορά «`unsigned short int`» και δηλώνουμε απλά `uint16 X,Y;` Παρόμοια, `typedef char *String;`
- Τέλος, μπορεί να χρησιμοποιηθεί για να δώσει ονόματα σε δομές:

Παράδειγμα με struct

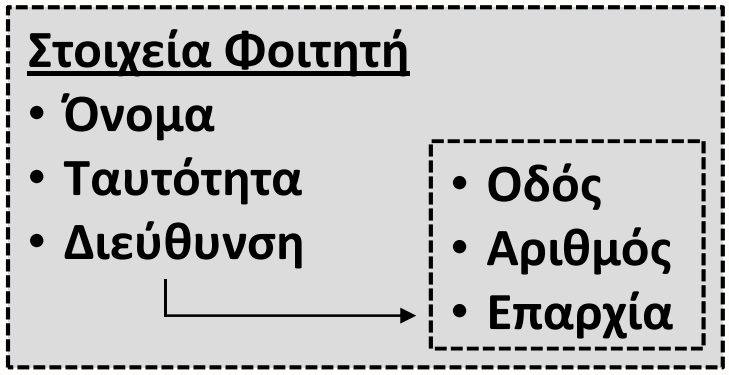
```
struct Person {
     char firstName[15];
     char lastName[15];
     char gender;
     int  age;
};
struct Person p1;
p1.age=43;
```

Παράδειγμα με typedef

```
typedef struct {
     char firstName[15];
     char lastName[15];
     char gender;
     int  age;
} Person;
Person p1;
p1.age=43;
```

Φωλιασμένες Δομές

- Δομές μπορεί να είναι αλληλένδετες, δηλαδή να φωλιάζονται η μια μέσα στην άλλη, δημιουργώντας πιο πολύπλοκες δομές:
- Παράδειγμα:



Δομής Φοιτητή με struct

```
struct address{
    char street[50];
 int number;
};
struct student{
 char name[30];
 int id;
 struct address saddress;
};
```

Δομής Φοιτητή με typedef

```
typedef struct {
  char street[50];
  int number;
} address;

typedef struct {
  char name[30];
  int id;
  address saddress;
} student;

student s1;
s1.id=123456;

strcpy(s1.name, "Ανδρέας");

s1.saddress.number = 14;
```

Δομές και Συναρτήσεις

- Επιτρεπτές πράξεις σε μια δομή είναι:
  - η **αντιγραφή** της, η **ανάθεση** τιμής σ' αυτήν (σαν σύνολο), η **εξαγωγή** της διεύθυνσής της, και η **προσπέλαση των στοιχείων** 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 αντιγράφετε μέσα στην συνάρτηση)

Δομές και Συναρτήσεις (συν.)

- Δομές *δεν* μπορούν να συγκριθούν:

  π.χ. η έκφραση x1 == x2 δεν είναι έγκυρη.

- Αν θέλουμε να συγκρίνουμε δυο δομές πρέπει η σύγκριση να γίνει βάση των πεδίων αυτών των δομών

  π.χ. x1.age == x2.age

- Εξάσκηση (στο σπίτι): Να γράψετε συνάρτηση η οποία παίρνει δύο παραμέτρους τύπου **Person** και επιστρέφει την ηλικία του μεγαλύτερου ατόμου.

Δομές και Δείκτες

- Μπορούμε επίσης να χρησιμοποιήσουμε **δείκτες σε δομές**.
- Για παράδειγμα η δήλωση

```
Person *pp, p;
```

δηλώνει ότι η μεταβλητή pp είναι δείκτης προς μια δομή τύπου Person . Έτσι μπορούμε να γράψουμε

```
pp = &p;
printf("%d", (*pp).age);
```

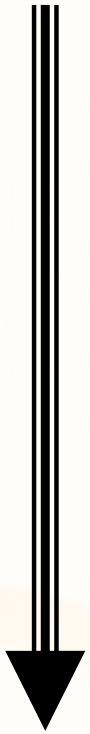
όπου **`*pp`** είναι η δομή που δείχνεται από τον δείκτη, ενώ **`(*pp).firstName`** είναι το πρώτο πεδίο της δομής.

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

  Δηλαδή το "*a.age=5" θα δώσει "compile error"

Προτεραιότητες (επανάληψη)

υψηλότερη



χαμηλότερη

```
()
. -> (struct)
!
* / %
+ –
< <= >= >
== !=
&&
||
=
```

Δομές και Δείκτες (συν.)

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

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

- Αν μια μεγάλη δομή πρόκειται να μεταβιβαστεί για επεξεργασία σε μια συνάρτηση γενικά είναι αποτελεσματικότερη **η μεταβίβαση ενός δείκτη προς τη δομή και όχι η αντιγραφή της**. Αυτό μπορεί να γίνει όπως και σε απλούστερες δομές δεδομένων (δηλαδή με το &).

```
void initPerson(Person *p){
   strcpy( p->firstName, "Ανδρέας");
   strcpy( p->lastName, "Ανδρέου");
   p->gender = 'M';
   p->age =  43;
}
```

```
Person p;
initPerson( &p );
```

```
Person *p;
//(δέσμευση χώρου για το *p [malloc])
initPerson( p );
```

Ενώσεις <unions>

- Ο τύπος union χρησιμοποιείται στην C για τη δήλωση μεταβλητών που μπορούν να αποθηκεύσουν σε διαφορετικές στιγμές δεδομένα διαφορετικών τύπων και μεγεθών *στην ίδια περιοχή μνήμης*.
- Θεωρήστε την περίπτωση που θέλουμε μια σταθερά σε ένα πρόγραμμα να έχει ως τιμή είτε μια ακέραια τιμή (int) είτε μια πραγματική τιμή (float). Θα μπορούσαμε να γράψουμε:

```
struct {
        int   ival;
        float fval;
} var;
```

και να τοποθετούμε τη σταθερά μας σε ένα από τα δύο πεδία της δομής σύμφωνα με την τιμή της.

- Μειονέκτημα αυτής της λύσης είναι ότι σπαταλά άσκοπα μνήμη: δεσμεύει μνήμη για δύο πεδία ενώ θα χρησιμοποιηθεί μόνο ένα.

Ενώσεις <unions> (συν.)

- Χρησιμοποιώντας ενώσεις η δήλωση της μεταβλητής μας γίνεται ως εξής:

```
union {
        int    ival;
        double fval;
} A,B;
```

- Η μνήμη που δεσμεύεται σε αυτή την περίπτωση είναι η μέγιστη που απαιτείται για αποθήκευση ακριβώς μιας μεταβλητής από οποιοδήποτε από τους τύπους των πεδίων (στην πιο πάνω περίπτωση η μνήμη που απαιτείται για αποθήκευση είναι 8 bytes).
- Η τιμή που ανακτάται κάθε φορά είναι η αυτή που αποθηκεύτηκε πιο πρόσφατα κατά την εκτέλεση. Είναι ευθύνη του προγραμματιστή να παρακολουθεί και να γνωρίζει ποιος τύπος είναι αποθηκευμένος σε μια ένωση.

Ενώσεις <unions> (συν.)

- Παράδειγμα: Κάθε άτομο (Person), εκτός από τα πεδία που μελετήσαμε μέχρι τώρα, μπορεί να έχει καταχωρημένη ταυτότητα ή διαβατήριο.

```
typedef struct {
     char firstName[15];
     char lastName[15];
     char gender;
     int  age;
     union {
        int id;
        char passport[15];
     } identification;
} Person;

Person p;
//Αρχικοποίηση firstname,
lastname, gender, age
```

Ερώτηση: Τι θα τυπώσει η ακόλουθη δήλωση σε κάθε βήμα;

```
printf("%d %s\n",
p1.identification.id,
p1.identification.passport);
```

1. Μετά την αρχικοποίηση.
2. p1.identification.id=15;
3. strcpy(p1.identification .passport, "E123456");