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

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

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

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

Δομές

- Δομή είναι μια συλλογή από μια ή περισσότερες μεταβλητές, πιθανώς διαφορετικών 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;
     int    age;
};
```

Δομές

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

Ετικέτα (tag) - προαιρετικό

πεδίο

- Εναλλακτικά μπορούμε επίσης να παραλείψουμε την ετικέτα της δομής και να περιγράψουμε απλά τα μέλη της:

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

Δομές

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

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

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

- Αναφορά σε μέλος μιας δομής γίνεται μέσω της κατασκευής:

  `όνομα-δομής.μέλος`

Για παράδειγμα

```
if (x.age > 40)
   printf("%s age: %d \n", x.lastName, x.age);
```

typedef

- Για να αποφεύγετε η συνεχής χρήση του «**struct structname», μπορούμε να χρησιμοποιήσουμε την typedef**
- Η C διαθέτει την typedef για δημιουργία νέων ονομάτων τύπων δεδομένων, π.χ.

  typedef unsigned short int uint16;

  Φτιάχνει τον τύπο 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:

```
struct Person{
   char   firstName[15];
   char   lastName[15];
   char   gender;
   int    age;
};
struct Person p;
p.age = 12;
```



```
typedef struct{
   char   firstName[15];
   char   lastName[15];
   char   gender;
   int    age;
} Person;
Person p;
p.age = 12;
```

Φωλιασμένες Δομές

- Δομές μπορεί να είναι αλληλένθετες, δηλαδή να φωλιάζονται 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 numofchild;
        Person children[5];
} Family;

int main(void){
  Person x, y, z;
  Family fml;

  fml.numofchild = 2;
  strcpy(fml.father.firstName, “Joe”);
  strcpy(fml.children[0].firstName, “Mary”);
  …
}
```

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.

Δείκτες σε Δομές

- 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.

- Προσοχή: 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"

Δείκτες σε Δομές

- 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 &).

```
Person p;
. . .
initPerson(&p);
. . .
void initPerson(Person *p){
  strcpy( p->firstName, “Ανδρέας”);
  strcpy( p->lastName, “Ανδρέου”);
  p->gender = ’M’;
  p->age =  43;
}
```

ή

```
Person *p;
(δέσμευση χώρου για το *p [malloc])
….
initPerson(p);
. . .
void initPerson(Person *p){
  strcpy( p->firstName, “Ανδρέας”);
  strcpy( p->lastName, “Ανδρέου”);
  p->gender = ’M’;
  p->age =  43;
}
```

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;
} constant;
```

  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

- 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

```
#define INT_TYPE   1;
#define REAL_TYPE  2;

struct item {
   int type;
   union {
            int   ival;
            float fval;
           } constant;
}
…
struct item x;

if (x.type == INT_TYPE)
   printf("value of const = %d\n", x.constant.ival);
if (x.type == REAL_TYPE)
   printf("value of const = %f\n", x.constant.fval);
```