MYY502

MYY502

MYY502

Η γλώσσα C

Νέοι και σύνθετοι τύποι

(typedef , structs, unions)

typedef (I)

- Η C επιτρέπει να δώσουμε ***οποιαδήποτε ονομασία μας αρέσει σε έναν υπάρχων τύπο***.

- Π.χ. δεν μου αρέσει το “int”, “char” κλπ.
  - Μπορώ να τα «βαφτίσω» με δεύτερο όνομα (ψευδώνυμο) με την typedef.

- Παραδείγματα:

```
typedef int Integer;    /* ψευδώνυμο του τύπου int */
typedef char Character; /* ψευδώνυμο του τύπου char */
int     n, m;
Integer x;
Character c;
x = n;
c = 'a';
```

typedef (II)

- Η C επιτρέπει να ορίσουμε ***έναν νέο τύπο, με όποια ονομασία μας αρέσει.***

- Παράδειγμα: θέλω να ορίσω τον τύπο «δείκτη σε χαρακτήρα» και να τον ονομάσω «pchar».
  - Το κάνω πάλι με την **typedef**.

```
typedef char * pchar;        /* Νέος τύπος, “pchar” */
char c, name[20];
pchar s, t;   /* pointer, αλλά χωρίς το αστεράκι!! */
s = &c;
t = name;     /* Δηλαδή, t = &name[0] */
*s = t[1];
```

Πολύπλοκα typedef

- Και πιο πολύπλοκοι τύποι!
- Π.χ. θέλω να ορίσω τον τύπο **intarray**, «πίνακας 10 ακεραίων».

```
typedef int intarray[10];
intarray A = { 0, 1, 2 };
A[5] = 15;
```

- Π.χ. τύπος πίνακα με 10 pointers σε χαρακτήρες

```
typedef char * pchar;       /* Νέος τύπος, “pchar” */
typedef pchar pcarray[10]; /* Πίνακας με 10 pchar */
pcarray names;
names[0] = "Bill";
names[1] = "Demi";
names[2] = names[0];
```

Συνταγή για typedef

- Αν θέλετε να ορίσετε έναν νέο τύπο, αλλά δυσκολεύεστε στο πως να τον εκφράσετε, υπάρχει ο εξής απλός κανόνας:
  1. Σκεφτείτε ένα όνομα (π.χ. `mytype`)
  2. Ορίστε μία *μεταβλητή* `mytype` ακριβώς όπως θα την θέλατε
  3. Τοποθετήστε τη λέξη "`typedef`" μπροστά από τη δήλωση της μεταβλητής.

- Παράδειγμα:
  - Κάνω επεξεργασία εικόνων και για κάθε πίξελ χρησιμοποιώ ένα πίνακα 3 χρωμάτων (RGB – red, green, blue). Θέλω να ορίσω έναν νέο τύπο που να μου δίνει αυτόν τον πίνακα.
  1. Θα τον πω "`color`".
  2. Είναι πίνακας 3 ακεραίων, άρα αν ήταν μεταβλητή θα οριζόταν ως εξής:
     ```
     int color[3];
     ```
  3. Ολοκλήρωση με `typedef`:
     ```
     typedef int color[3];
     ```

- Χρήση:
  ```
  color c = {50, 50, 50}, pixels[100];
  pixels[5][0] = c[0];
  ```

MYY502

MYY502

MYY502

Η γλώσσα C

Δομές (structs)

Εισαγωγή

- Οι δομές στη C επιτρέπουν τον ορισμό **σύνθετων τύπων δεδομένων** πέραν των βασικών τύπων που προσφέρει η γλώσσα
  - Με τον τρόπο αυτό επιτυγχάνουμε καλύτερη οργάνωση των δεδομένων

- Πολλές φορές θέλουμε να διατηρούμε συλλογή από πληροφορίες για **MIA** οντότητα, π.χ. μία μεταβλητή «εργαζόμενος» θα θέλαμε να εμπεριέχει και το όνομα και το επώνυμο και την ηλικία του, αντί να ορίσουμε 3 ξεχωριστές μεταβλητές.
  - Μια δομή περιλαμβάνει μια συλλογή από ανομοιογενή πεδία (διαφορετικών τύπων) που ομαδοποιούνται με ένα μόνο όνομα που αντιστοιχεί στον νέο τύπο δεδομένων

- Ο ορισμός μιας μεταβλητής του νέου τύπου δεδομένων αντιστοιχεί στον ορισμό μιας συλλογής από μεταβλητές που αντιστοιχούν με τη σειρά τους στα πεδία της δομής που ορίσαμε.

```
struct <ετικέτα δομής> {
    <δηλώσεις πεδίων>
};
```

```
struct person {
    char firstName[20];
    char lastName[20];
    char gender;
    int age;
};
```

Δηλώσεις μεταβλητών

- Προσοχή στη λέξη **`struct:`**
  - Αρχικά χρησιμοποιείται **για να ορίσει την δομή** και την ετικέτα/όνομα της και
  - Στη συνέχεια χρησιμοποιείται **για να ορίσει μεταβλητές** αυτού του τύπου δομής.
- Παράδειγμα:

```
struct person {    /* Ορισμός ονόματος & πεδίων δομής */
        char firstName[20];
        char lastName[20];
        char gender;
        int  age;
      };
struct person x;   /* Ορισμός μεταβλητής τύπου struct person */
```

- Μπορεί κάποιος να κάνει και τα δύο σε ένα (όχι καλό!):

```
struct person {
        ...
    } x;
```

- **Αρχικοποίηση με αγκύλες:**

```
struct person x = {"Stelios", "Balis", 'M', 40};
```

  - Όπως και στους πίνακες, μπορεί να γίνει αρχικοποίηση μόνο μέχρι κάποιο πεδίο. Τα επόμενα πεδία αρχικοποιούνται αυτόματα στο 0.

Πίνακες από δομές

```
struct person {
      char firstName[20];
      char lastName[20];
      char gender;
      int age;
};
```

- Πίνακας από δομές:

```
struct person people[40];
```

- Ορισμός δομής & μαζί δήλωση μεταβλητής:

```
struct person {
   ...
} people[40];
```

- Αρχικοποίηση:

```
struct person people[] = {
                {"Stelios", "M", 'M', 40},
                {"Kostas", "A", 'M', 50}
              };
```

Πρόσβαση στα πεδία μίας δομής

- Χρήση: **<δομή>.<πεδίο>**
  - x.firstName
  - x.age

```
struct person {
      char firstName[20];
      char lastName[20];
      char gender;
      int age;
};
```

- Π.χ.

```
struct person x = {"Stelios", "Balis", 'M', 40};
printf("%c", x.gender);
x.age ++;
```

- Π.χ.

```
struct person people[40];          /* Πίνακας με δομές */
...
printf("%c", people[3].gender);    /* Στοιχείο 3 */
people[4].age ++;                  /* Στοιχείο 4 */
```

Παράδειγμα Ι

```
struct person {
       char firstName[20];
       char lastName[20];
       char gender;
       int age;
   };

int main() {
   struct person x, y;

   strcpy(x.firstName, "rivaldo");
   strcpy(x.lastName, "unknown");
   x.gender = 'M';
   x.age = 37;

   y = x;    /* Προσέξτε αυτό! Αντιγράφονται ΌΛΑ ΤΑ BYTES */
   return 0;
}
```

Παράδειγμα II

```
struct person {
       char firstName[20];
       char lastName[20];
       char gender;
       int age;
   };

int main() {
   struct person x, y;

   scanf("%s", x.firstName);
   scanf("%s", x.lastname);
   scanf("%c", &(x.gender));
   scanf("%d", &(x.age));

   y = x;
   return 0;
}
```

Προσοχή στα πεδία που χρησιμοποιείτε!! (I)

```
struct person {
        char *firstName;        /* Αντί για πίνακες */
        char *lastName;
        char gender;
        int age;
    };

int main() {
   struct person x, y;

   scanf("%s", x.firstName);
   scanf("%s", x.lastname);
   scanf("%c", &(x.gender));
   scanf("%d", &(x.age));

   y = x;
   return 0;
}
```

Είναι όλα OK???

Προσοχή στα πεδία που χρησιμοποιείτε!! (I)

```
struct person {
        char *firstName;        /* Αντί για πίνακες */
        char *lastName;
        char gender;
        int age;
    };

int main() {
   struct person x, y;

   x.firstname = (char *) malloc(20); /* Should check for NULL... */
   x.lastname  = (char *) malloc(20); /* Should check for NULL... */
   scanf("%s", x.firstName);
   scanf("%s", x.lastname);
   scanf("%c", &(x.gender));
   scanf("%d", &(x.age));

   y = x;
   return 0;
}
```

QUIZ

Τι γίνεται στο τέλος με το y???

typedef για ευκολία

- Μπορούμε να χρησιμοποιήσουμε την typedef ώστε κάνουμε τις δηλώσεις μας απλούστερες, π.χ.

```
struct person_s {
  char firstName[20];
  char lastName[20];
  char gender;
  int age;
  };
typedef struct person_s person_t;

int main() {
  person_t p;
  p.age = 12;
  return 0;
}
```

Στυλ προγραμματισμού:

Το struct και το όνομα του τύπου από το typedef πρέπει να συνδέονται με συστηματικό τρόπο (ή ακόμα και συνώνυμο).

Συνήθης τακτική / σύμβαση:

Struct: person ή person_s

Typedef: Person ή person_t

Συνδυασμός struct με typedef

Ορισμός struct και τύπου μαζί:

```
typedef struct person_s {
     char firstName[20];
     char lastName[20];
     char gender;
     int age;
  } person_t;

int main() {
     person_t p;
     p.age = 12;
      return 0;
}
```

Κι άλλη (κακή) εκδοχή: *μη-ονοματισμένο struct*

```
typedef struct {
     char firstName[20];
     char lastName[20];
     char gender;
     int age;
 } person_t;

int main() {
     person_t p;
     p.age = 12;
     return 0;
}
```

Εμφωλευμένες δομές

- Οι δομές μπορεί να είναι *ένθετες*, δηλαδή να φωλιάζονται η μια μέσα στην άλλη, δημιουργώντας πιο πολύπλοκες δομές:

```
struct family {
    struct person father;
    struct person mother;
    int    numofchildren;
    struct person children[5];
  };
```

Εμφωλευμένες δομές – παράδειγμα

```
struct family {
   struct person  father;
   struct person  mother;
   int            numofchildren;
   struct person  children[5];
 };

int main() {
   struct person x, y, z;
   struct family fml;

   fml.numofchildren = 2;
   strcpy(fml.father.firstName, "Joe");
   strcpy(fml.children[0].firstName, "Marry");
   return 0;
}
```

Πράξεις/λειτουργίες σε δομές

- Επιτρεπτές πράξεις/λειτουργίες σε μια δομή είναι:
  - η αντιγραφή της
  - η απόδοση τιμής σ' αυτήν ως σύνολο
  - η εξαγωγή της διεύθυνσής της με &
  - η προσπέλαση των μελών της

- Επιτρέπονται
  - <struct> = <struct> (**αντιγράφονται τα πάντα**)
  - &<struct> (δείκτης στο χώρο που αποθηκεύεται το struct)
  - <struct>.πεδίο

- Ο δομές **δεν μπορούν να συγκριθούν**, δηλ. δεν επιτρέπεται:
  - <struct> == <struct>

Πράξεις σε δομές

- Μεταβλητές τύπου δομής μπορούν να μεταβιβαστούν ως ορίσματα σε συναρτήσεις όπως επίσης και να επιστραφούν ως αποτελέσματα συναρτήσεων

```
struct person inc_age(struct person x) {
  x.age += 1;
  return x;
}

int main() {
   struct person x1, x2;
   x1.age = 45;
   x2 = inc_age(x1);  /* Πέρασμα με τιμή (copy) */
   return 0;
}
```

Δομές και δείκτες

- Παράδειγμα δείκτη σε δομή

  struct person p;

  struct person *pp;

- Νόμιμες εκφράσεις:

  pp = &p;

  printf("%s", (*pp).firstName);

- Οι εκφράσεις τύπου **(*pp).firstName** απαιτούν πάντα παρενθέσεις

- Εναλλακτικά: **pp->firstName**, δηλαδή

  (*pp). ≡ pp->

Παράδειγμα - Μεταβίβαση με τιμή

```
#include <stdio.h>
struct person {
    char firstName[20];
    char lastName[20];
    char gender;
    int age;
  };

void initPerson(struct person p) {
    strcpy(p.firstName, "xxxxx");
    strcpy(p.lastName, "yyyyy");
    p.gender = 'M'; p.age = 40;
}

int main() {
    struct person q;
    q.age = 0;
    strcpy(q.firstName, "");
    strcpy(q.lastName, "");
    initPerson(q);          /* Με τιμή */
    printf("%s %s %d\n", q.firstName, q.lastName, q.age);
    return 0;
}
```

Παράδειγμα - Μεταβίβαση με αναφορά

```
#include <stdio.h>
struct person {
    char firstName[20];
    char lastName[20];
    char gender;
    int age;
  };

void initPerson(struct person *p) {
    strcpy(p->firstName, "xxxxx");
    strcpy(p->lastName, "yyyyy");
    p->gender = 'M'; p->age = 40;
}

int main() {
    struct person q;
    q.age = 0;
    strcpy(q.firstName, "");
    strcpy(q.lastName, "");
    initPerson(&q);         /* Με αναφορά */
    printf("%s %s %d\n", q.firstName, q.lastName, q.age);
    return 0;
}
```

Για μεγάλες δομές, η μεταβίβαση με δείκτη είναι γενικά αποτελεσματικότερη, επίσης χρειάζεται όταν χρειάζεται να κάνουμε **πέρασμα με αναφορά.**

Παράδειγμα 1

```
#include <stdio.h>
struct person {
     char name[20];
     int age;
  };

struct person inc_age(struct person x) {
   x.age += 1;
   return x;
}

int main() {
   struct person a = {"Me", 10};
   struct person c;

   c = inc_age(a);
   printf("C:%d A:%d\n", c.age, a.age);
   return 0;
}
```

```
$./a.out
C:11 A:10
```

Παράδειγμα 2 – δομές και δείκτες

```
#include <stdio.h>
#include <string.h>
struct person {
        char name[20];
        int age;
    };

struct person inc_age(struct person x) {
   x.age += 1;
   return x;
}

void inc_age_ptr(struct person *x) {
   x->age += 1;
}

int main() {
   struct person a = {"XXX", 10}, b = {"YYY", 20}, c;

   c = inc_age(a);
   inc_age_ptr(&b);
   printf("C:%d A:%d B:%d \n", c.age, a.age, b.age);
   return 0;
}
```

```
$./a.out
C:11 A:10 B:21
```

Παράδειγμα 3 (1/2)

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

struct name{
    char *fname;
    char *lname;
    int letters;     /* Πλήθος γραμμάτων επώνυμου + μικρού ονόματος */
  };

void getinfo(struct name *pst) {
   char temp[81];

   printf("First name ? ");
   fgets(temp, 80, stdin);                              /* Διάβασμα */
   pst->fname = (char *) malloc(strlen(temp) + 1);    /* Δέσμευση μνήμης*/
   if (pst->fname == NULL) exit(1);
   strcpy(pst->fname, temp);                            /* Αντιγραφή */

   printf("Last name ? ");
   fgets(temp, 80, stdin);
   pst->lname = (char *) malloc(strlen(temp) + 1));
   if (pst->fname == NULL) exit(1);
   strcpy(pst->lname, temp);
}
```

Παράδειγμα 3 (2/2)

```
/* Συνέχεια */


void computeLen(struct name *pst) {
   pst->letters = strlen(pst->fname) + strlen(pst->lname);
}

void cleanup(struct name *pst) {
   free(pst->fname);
   free(pst->lname);
}

int main() {
   struct name x;

   getinfo(&x);
   computeLen(&x);
   printf(" %s %s %d\n", x.fname, x.lname, x.letters);
   cleanup(&x);
   return 0;
}
```

Παράδειγμα – πίνακες και δομές (1/2)

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

typedef struct student {    /* Θα φυλάμε βαθμολογίες μαθητών */
     int AM;
     int grade;
   } Student;

void calculate_stats(Student class[], int studNo);
void print_student_details(Student class[], int studNo);

int main(int argc, char *argv[]) {
   Student *class;
   int i, n;

   n = atoi(argv[1]);       /* Εκτέλεση του προγράμματος με ορίσματα */
   class = (Student *) malloc(n*sizeof(Student));
   for (i = 0; i < n; i++)  /* Διάβασμα n φοιτητών-βαθμών */
      scanf("%d%d", &class[i].AM, &class[i].grade);

   print_student_details(class, n);
   calculate_stats(class, n);
   return 0;
}
```

Παράδειγμα – πίνακες και δομές (2/2)

```
void calculate_stats(Student class[], int studNo) {
   int i;
   int max = 0;

   for (i = 0; i < studNo; i++) {
      if (class[i].grade > max)
        max = class[i].grade;
   }
   printf("MAX GRADE = %d\n", max);
}

void print_student_details(Student class[], int studNo) {
   int i;

   printf("AA \t AM \t GRADE\n");

   for (i = 0; i < studNo; i++) {
      printf("%d \t %d \t %d\n", i, class[i].AM, class[i].grade);
   }
}
```

Σύνθετο παράδειγμα

```
struct Name     { char first[20]; char last[20]; };
struct An8rwpos { struct Name gname; float income; }

int main() {
   struct An8rwpos people[2] = { {{"A", "B"}, 123.4}, {{"C", "D"}, 345.67} };
   struct An8rwpos *p, me;

   p = &people[0];         /* Δείχνει στο 0 στοιχείο του people */
   me = *(p+1);            /* = p[1] (το πρώτο στοιχείο του p)  */
   printf("%s %s %f \n\n", /* A B 123.400000 */
          p->gname.first, p->gname.last, p->income);
   printf("%s %s %f \n", people[1].gname.first,
         people[1].gname.last,  people[1].income);
   printf("%s %s %f \n",
         me.gname.first, me.gname.last, me.income);
   printf("%s %s %f \n\n", (*(p+1)).gname.first,
          (*(p+1)).gname.last, (*(p+1)).income);

   strcpy(p[1].gname.first, "X");
   strcpy(p[1].gname.last,  "Y");
   p[1].income = 0.0;

   printf("%s %s %f \n", people[1].gname.first,
         people[1].gname.last, people[1].income);
   printf("%s %s %f \n\n",
         me.gname.first, me.gname.last, me.income);
}
```

```
$./a.out
A    B    123.400000

C    D    345.670000
C    D    345.670000
C    D    345.670000

X    Y    0.000000
C    D    345.670000
```

Ισοδύναμες εκφράσεις στο προηγούμενο παράδειγμα

```
/* struct An8rwpos people[2] = { {{"A", "B"}, 123.4},
                                 {{"C", "D"}, 345.67} };
   struct An8rwpos *p, me;
   p = &people[0];
   me = *(p+1);
*/
```

- Υπάρχουν δύο σύνολα ισοδύναμων εκφράσεων
- 1ο σύνολο:
  - `printf("%s\n", p->gname.first);`
  - `printf("%s\n", people[0].gname.first);`
  - `printf("%s\n", (*p).gname.first);`
- 2ο σύνολο:
  - `printf("%s\n", me.gname.first);`
  - `printf("%s\n", people[1].gname.first);`
  - `printf("%s\n", (*(p+1)).gname.first);`

MYY502
MYY502
MYY502

Η γλώσσα C

Αναφορές σε δομές

Αναφορές (προς ίδιου τύπου δομή)

- Δεν επιτρέπεται:

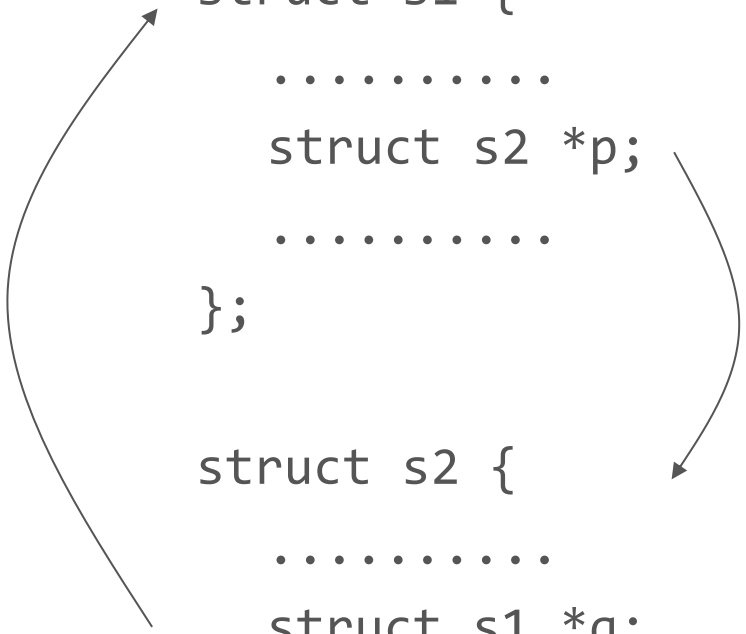
```
struct Employee {
 char name[20];
 int age;
 struct Employee manager;
};
```

- Επιτρέπεται:

```
struct Employee {
 char name[20];
 int age;
 struct Employee *manager;
};
```

Αναφορές

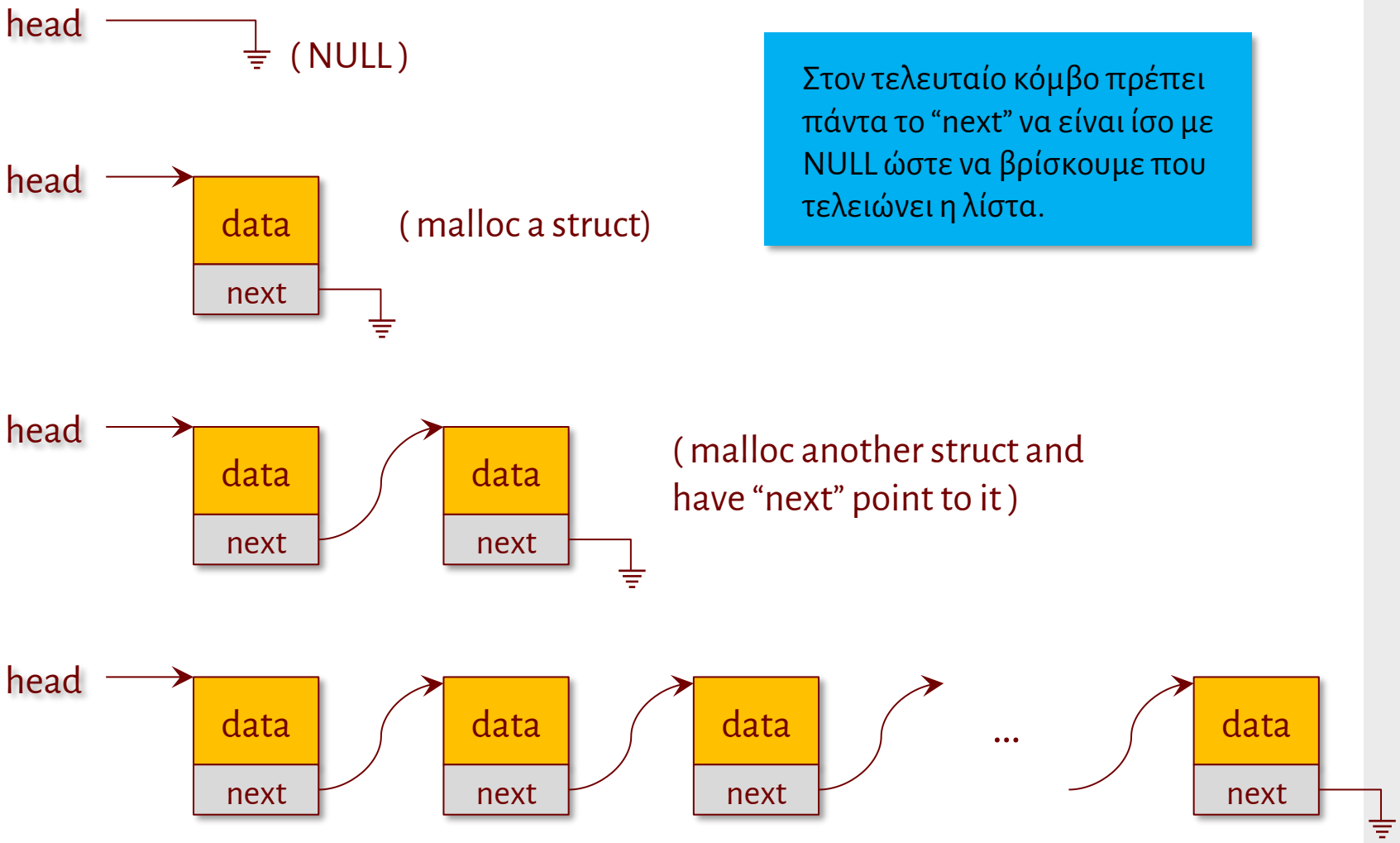
- Επίσης επιτρέπεται (κυκλική αναφορά):



```
struct s1 {
   ..........
   struct s2 *p;
   ..........
};

struct s2 {
   ..........
   struct s1 *q;
   ..........
};
```

Παράδειγμα – αναφορές / δυναμική λίστα (1/3)



Στον τελευταίο κόμβο πρέπει πάντα το “next” να είναι ίσο με NULL ώστε να βρίσκουμε που τελειώνει η λίστα.

Παράδειγμα – αναφορές / δυναμική λίστα (2/3)

```c
#include <stdio.h>
#include <stdlib.h>

struct listnode {            /* Define our simple struct */
  int value;                 /* We store our data (a number) here */
  struct listnode *next;     /* Pointer to the next node in the list */
};

typedef struct listnode listnode_t;    /* for simplicity & clarity */

/* Calculate the sum of all numbers in the list */
void find_sum(listnode_t *node)
{
  int sum = 0;

  while (node != NULL) {     /* Traverse the list till its end */
    sum += node->value;      /* Value of current node */
    node = node->next;       /* Point to the next node */
  }
  printf("sum of numbers = %d\n", sum);
}
```

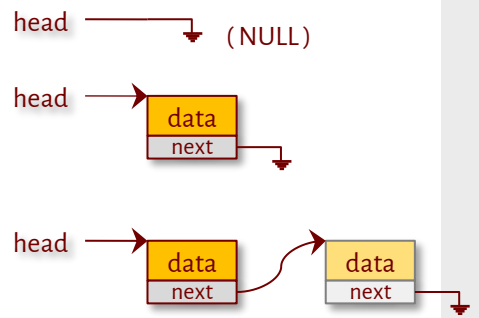
Παράδειγμα – αναφορές / δυναμική λίστα (3/3)

```
int main()
{
  listnode_t *node, *head = NULL;   /* head: pointer to start of list (empty now) */
  int         n;

  printf("How many numbers? ");
  scanf("%d", &n);                  /* This should be positive… */
  for (; n > 0; n--) {
    node = (listnode_t *) malloc(sizeof(listnode_t));  /* malloc 1 node (struct) */
    if (node == NULL) return (1);

    scanf("%d", &node->value);     /* Read & store number */

    if (head == NULL) {   /* Empty list */
      node->next = NULL;    /* No more nodes after this one */
      head = node;          /* The only node in the list */
    }
    else {                /* Non-empty list; insert new node */
      node->next = head;    /* Inserted in the beginning */
      head = node;          /* Head now points to the new node */
    }
  }
  find_sum(head);             /* Call by passing the starting node of the list */
  return (0);
}
```



MYY502
MYY502
MYY502

Η γλώσσα C

Ενώσεις (unions)

Ενώσεις (Unions)

- Οι ενώσεις είναι μια ειδική περίπτωση «δομών» οι οποίες επιτρέπουν να κάνουμε οικονομία στη μνήμη.
  - Σε αντίθεση με τα structs, **δεν αποθηκεύονται όλα τα πεδία αλλά μόνο ένα από αυτά!!**
- Για παράδειγμα, ένα πρόγραμμα δέχεται ως είσοδο έναν int ή έναν double (ανάλογα με το τι επιλέγει ο χρήστης) – όχι και τα δύο μαζί. Πώς το αποθηκεύουμε αυτό;

```
int main() {                        /* Απλή λύση: 2 μεταβλητές */
   int ival;
   float fval;
   char choice;
   scanf("%c", &choice);
   if (c == 'i') scanf("%d", &ival);
   if (c == 'f') scanf("%f", &fval);
   ...
}
```

Ενώσεις

- Μια ένωση αποτελείται από μια συλλογή πεδίων εκ των οποίων το πρόγραμμά μας μπορεί να επιλέξει ένα από όλα κάθε φορά.
- **Τα στοιχεία/πεδία μιας ένωσης μοιράζονται τον ίδιο χώρο μνήμης!!**
- Χειρισμός σαν να είναι structs.

```
typedef union trick {
   int   ival;
   float fval;
} trick;

int main() {
   trick value;      /* sizeof(value): 4 bytes, not 8 bytes! */
   char choice;
   scanf("%c", &choice);
   if (c == 'i') scanf("%d", &value.ival);
   if (c == 'f') scanf("%f", &value.fval);
   ...
}
```

Παράδειγμα – Ενώσεις

```
#define INT_TYPE  1
#define REAL_TYPE 2
struct item {
   int type;
   union {
       int   ival;
       float fval;
   } value;
};

int main() {
   struct item x;

   x.type = INT_TYPE;
   x.value.ival = 4;
   printf("%d %f\n", x.value.ival, x.value.fval);
   x.type = REAL_TYPE;
   x.value.fval = 28000.5;
   printf("%d %f\n", x.value.ival, x.value.fval);
   return 0;
}
```

```
$./a.out
4 0.000000
1188741376 28000.500000
```

Παράδειγμα – Ενώσεις

```
#define INT_TYPE  1
#define REAL_TYPE 2
struct item {
   int type;
   union {
       int ival;
       float fval;
   } value;
};

void print_item(struct item x) {
   if (x.type == INT_TYPE)
       printf("value = %d\n", x.value.ival);
   if (x.type == REAL_TYPE)
       printf("value = %f\n", x.value.fval);
}

int main() {
   struct item x = {INT_TYPE, { 4 }};
   print_item(x);
   return 0;
}
```

```
$./a.out
value = 4
```