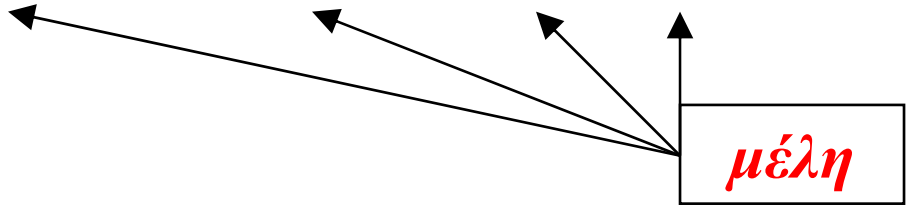
Θεματική ενότητα 7: Δομές – απαριθμητικοί τύποι δεδομένων

Απαριθμητικοί τύποι δεδομένων (enumerated)

```
#include <stdio.h>


                                        // προσδιορισμός τύπου enum
enum week_days {Sun, Mon, Tue, Wed, Thu, Fri, Sat};

main()
{
  week_days day1,day2;   //ορισμός μεταβλητών
  day1=Mon;     //απόδοση τιμής
  day2=Thu;     //απόδοση τιμής
  int diff=day2-day1;  //εκτελεί αριθμητική ακεραίων
  printf( "Days between =%d\n",diff );
  if (day1<day2)      //μπορεί να κάνει συγκρίσεις
    printf( "day1 is followed by day2\n" );
}
```



Αποτέλεσμα=3

- **Εσωτερικά, ο χειρισμός των τύπων δεδομένων *enum* γίνεται σαν να ήταν ακέραιοι (γι' αυτό και μπορούν να εκετελεσθούν αριθμητικές και συγκριτικές πράξεις με αυτούς). Στο πρώτο όνομα δίνεται η τιμή 0 (Sun στο προηγούμενο παράδειγμα), στο επόμενο η τιμή 1 (Mon στο προηγούμενο παράδειγμα) κ.ο.κ.**

- **Αν θέλουμε να ξεκινά η αρίθμηση από άλλον ακέραιο, απλώς ενεργούμε ως εξής:**

```
enum week_days {Sun=30, Mon, Tue, Wed, Thu, Fri, Sat};
```

οπότε η αρίθμηση ξεκινά από το 30.

- **Ωστόσο, αν θέσετε π.χ. day1=5 και δε θέσετε π.χ. day1=Sun ή Mon, ο μεταγλωττιστής θα βγάλει μήνυμα σφάλματος.**

➢Με τον τύπο *enum* μπορούμε να ορίσουμε τις λογικές τιμές της άλγεβρας Boole true και false ως εξής:

```
enum boolean {False, True};
```

οπότε η False =0 και η True=1. Έτσι μπορούμε, στη συνέχεια να ορίσουμε *boolean* μεταβλητές

➢Εναλλακτικά τα παραπάνω επιτελούνται με χρήση της πρότασης *define* του προεπεξεργαστή:

```
#define False 0

#define True 1
```

Οι μεταβλητές απαριθμητικού τύπου παρουσιάζουν το μειονέκτημα ότι δεν μπορούν να χρησιμοποιηθούν άμεσα στις συναρτήσεις εισόδου–εξόδου (*scanf()*, *printf()*). Οι συναρτήσεις αυτές τυπώνουν τον ακέραιο στον οποίο αντιστοιχεί το συμβολικό όνομα.

Στη C απαιτείται να συμπεριληφθεί στον ορισμό μεταβλητής απαριθμητικού τύπου η λέξη *enum*, π.χ. enum week_days day1,day2;. Στη C++ η λέξη *enum* δεν είναι απαραίτητη και η δήλωση γίνεται week_days day1,day2;, όπως ακριβώς ορίζεται ένας βασικός τύπος δεδομένου, π.χ. int var1.

Η λέξη κλειδί typedef

Με τη λέξη κλειδί *typedef* αποδίδονται νέα ονόματα σε τύπους δεδομένων:

typedef <τύπος> <όνομα>;

•Η δήλωση

typedef float real_number;

καθιστά το όνομα real_number συνώνυμο του *float*. Ο τύπος real_number μπορεί πλέον να χρησιμοποιηθεί όπως ακριβώς χρησιμοποιείται ο τύπος *float*, *με τη διαφορά ότι ο* real_number *θα είναι ενεργός αποκλειστικά μέσα στο πρόγραμμα που δημιουργείται.*

•Με την *typedef* δε δημιουργούνται νέοι τύποι, απλά αλλάζουν οι ετικέτες. Π.χ. η δήλωση

real_number num1, num2;

δηλώνει τις μεταβλητές κινητής υποδιαστολής num1 και num2.

Δομές (structures)

- **Μπορούμε να ομαδοποιήσουμε δεδομένα χρησιμοποιώντας πίνακες.**
- **Ωστόσο, τα στοιχεία των πινάκων πρέπει να είναι όλα του ίδιου τύπου.**
- **Οπότε, πώς δημιουργούμε ΜΕΙΚΤΟΥΣ τύπους δεδομένων;**

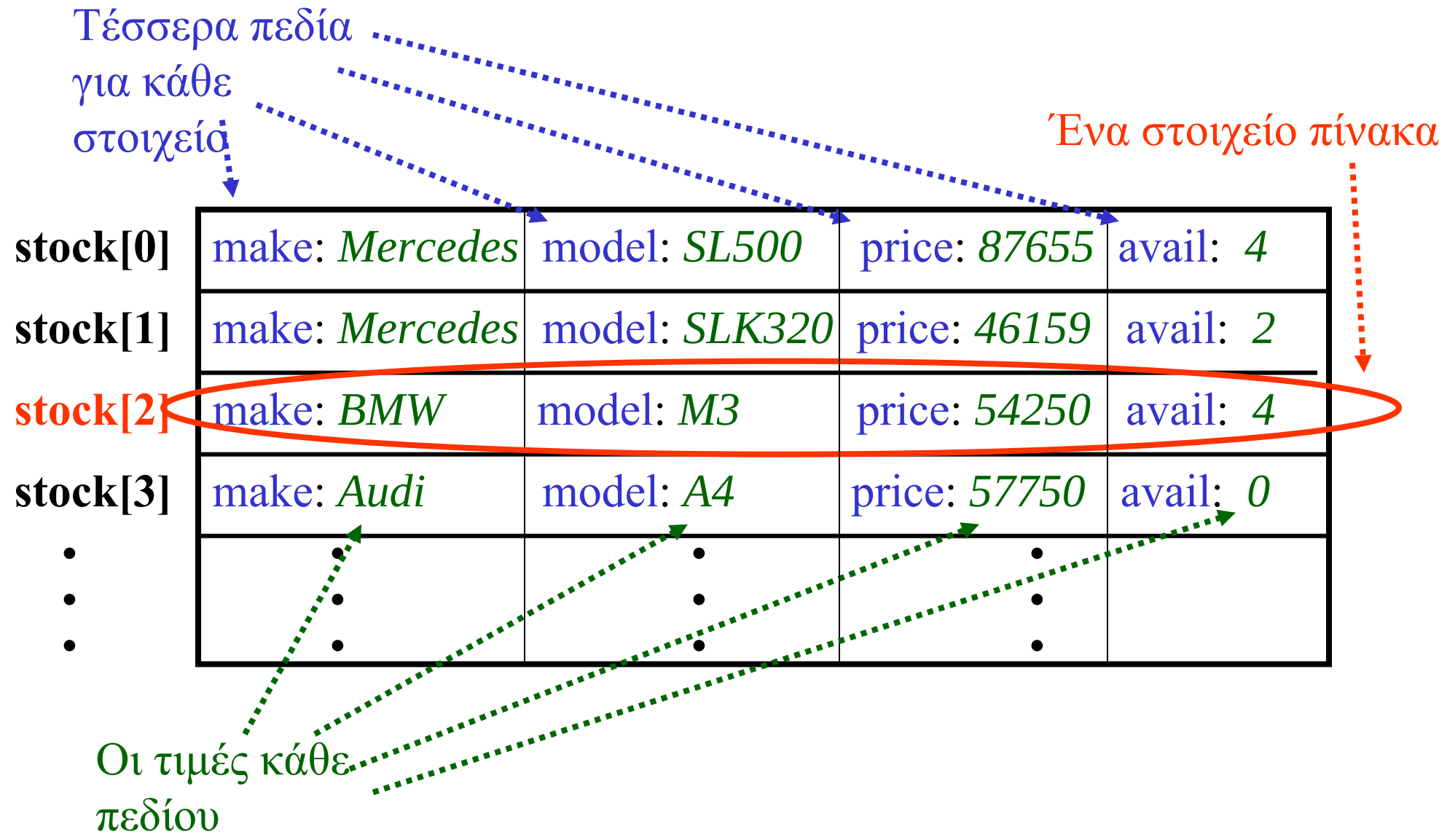
Με τη χρήση των δομών.

Ορισμός:

Μία δομή είναι μία συλλογή μεταβλητών, η οποία αποθηκεύεται και παρουσιάζεται ως μία λογική οντότητα.

- Τα μέλη μίας δομής μπορούν να ανήκουν στους βασικούς τύπους int, char, float, double, μπορούν να είναι πίνακες ή ακόμη κι άλλες δομές.
- Οι δομές μάς επιτρέπουν να δημιουργήσουμε τους δικούς μας τύπους.
- Για να ορισθούν μεταβλητές τύπου δομής πρέπει πρώτα να ορισθεί η δομή.

Παράδειγμα: Επιχείρηση πώλησης αυτοκινήτων



stock[0]	make: *Mercedes*	model: *SL500*	price: *87655*	avail: *4*
stock[1]	make: *Mercedes*	model: *SLK320*	price: *46159*	avail: *2*
stock[2]	make: *BMW*	model: *M3*	price: *54250*	avail: *4*
stock[3]	make: *Audi*	model: *A4*	price: *57750*	avail: *0*
•	•	•	•	
•	•	•	•	
•	•	•	•	

Κάθε πεδίο έχει τιμή διαφορετικού τύπου:

make: απαριθμητικός τύπος δεδομένου

model: πίνακας χαρακτήρων

price: αριθμός κινητής υποδιαστολής (float)

avail: ακέραιος (integer)

Ορισμός νέου τύπου

```
enum {Mercedes, BMW, Audi} carmakeT;


struct stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};
```

struct:

- Ομάδα μεταβλητών.
- Επιτρέπει την ομαδοποίηση στοιχείων διαφορετικού τύπου.
- Η *typedef* χρησιμοποιείται για τη δημιουργία νέου τύπου δεδομένων.

```
enum {Mercedes, BMW, Audi} carmakeT;

struct  stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};
```

Ορισμός νέου τύπου δεδομένων.

```
enum {Mercedes, BMW, Audi} carmakeT;

struct stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};
```

Είναι μία δομή

με όνομα stockT. Το όνομα μπορεί να γραφεί και δεξιά της struct.

T για *τύπος (type)*

```
enum {Mercedes, BMW, Audi} carmakeT;

struct stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};
```

Προσοχή στο ;

Η δομή είναι συλλογή τέτοιων αριθμών

- όλοι πρέπει να έχουν τύπο και όνομα, όπως κάθε άλλη μεταβλητή,
- αλλά αυτές είναι *υπομεταβλητές* μέσα στον τύπο ‘stockT’

```
enum { Mercedes, BMW, Audi } carmakeT;

struct stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};

void main () {
        stockT mycar, inventory[40];
        ...
}
```

Η stockT ενεργεί ως τύπος δεδομένων, ακριβώς όπως οι char, int, float, κ.λ.π.

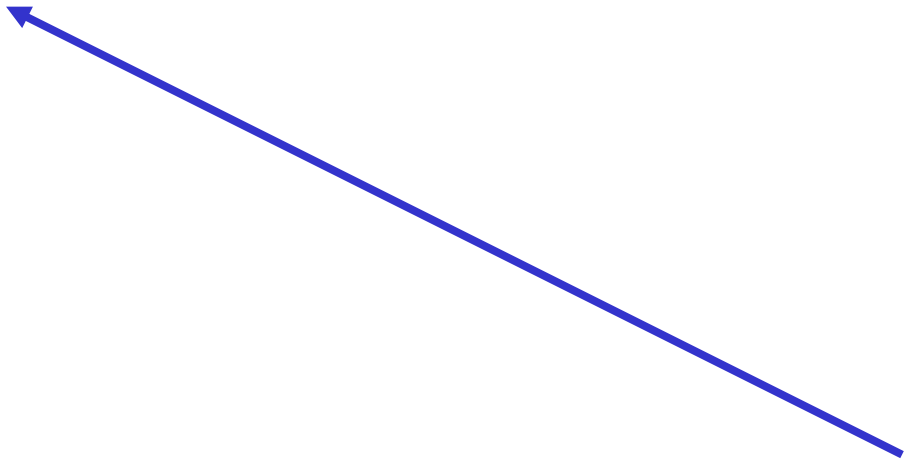
Δηλώσαμε μία μεταβλητή τύπου stockT, ονόματι mycar, και έναν πίνακα στοιχείων τύπου stockT, ονόματι inventory).

Δομές, απόδοση αρχικών τιμών

```
enum {Mercedes, BMW, Audi} carmakeT;

struct  stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};

void main () {
        stockT mycar, inventory[40];
        mycar.make = Mercedes;
        mycar.price = 87655;
        ...
}
```

Χρησιμοποιείται ο τελεστής **dot** για να προσπελασθούν τα μέλη (‘members’ ή ‘fields’) της δομής.

```
void main () {
        int     counter;
        stockT mycar;
}
```



Ο τύπος δεδομένων είναι ΔΙΑΦΟΡΕΤΙΚΟΣ από τη μεταβλητή!

Μπορεί να δοθεί τιμή στη μεταβλητή mycar.

Δεν έχει νόημα να δοθεί τιμή στη stockT.

```
main () {
        int     counter;
        stockT mycar;
        stockT.model = Mercedes;
}
```

ΣΦΑΛΜΑ! stockT είναι τύπος δεδομένων.
Μη συγχέετε τον *τύπο δεδομένων* (data type) που ορίσατε με τη δομή και τις *μεταβλητές* (variables) τέτοιου τύπου.

```
enum {Mercedes, BMW, Audi} carmakeT;

struct stockT {
        carmakeT make;
        char model[5];
        float price;
        int avail;
};

main () {
        stockT inventory[40];
        inventory[1].make = Mercedes;
        inventory[1].avail = 2;
        ...
}
```

Μπορείτε να δηλώσετε πίνακα τύπου δομής. Χρησιμοποιήστε τον τελεστή dot για να προσπελάσετε κάθε πεδίο.

Η αρχικοποίηση δομής που περιλαμβάνει αλφαριθμητικό μέλος παρουσιάζεται ακολούθως:

```
struct addressT {
 char name[20]; char city[15]; char street[12];
 int number; int zip_code;
};
struct addressT addr1={“Demis”, “Serres”, “Rodou”,23,62124};
```

Λειτουργίες δομών

Οι λειτουργίες που μπορούν να εκτελεσθούν σε μία δομή είναι:

- **Ανάθεση μεταβλητών του ίδιου τύπου. Εάν address1 και address2 είναι δύο μεταβλητές τύπου δομής addrT, με την ανάθεση**

address1=address2

τα μέλη τής address1 αποκτούν τις τιμές των αντίστοιχων μελών τής address2.

- **Η διεύθυνση μίας μεταβλητής τύπου δομής μπορεί να ανατεθεί σε ένα δείκτη (pointer).**

- **Χρήση του τελεστή *sizeof*. Η εντολή**

sizeof(struct address1)

επιστρέφει τον αριθμό των bytes που απαιτούνται για την αποθήκευση στη μνήμη μίας μεταβλητής τύπου δομής addrT.

Παράδειγμα: Ένθεση δομών (struct within struct)

```
struct personT {
  char name[16];
  char address[12];
  char tel[10];
  struct dateT birthdate;
  struct dateT hiredate;
};
```

Μέσα στη δομή person υπάρχουν οι δομές **birthdate** *και* **hiredate**

Οι δομές **birthdate** *και* **hiredate** *έχουν την ακόλουθη μορφή:*

```
struct dateT {
  int day;
  int month;
  int year;
  char mon_name[4];
};
```

Η αναφορά στο έτος γέννησης γίνεται με την έκφραση:

bemp.birthdate.year

Παράδειγμα: Πίνακες δομών

Πίνακας δομής: **struct addressT addr[10];**

Η απόδοση αρχικής τιμής στη δομή ακολουθεί το γενικό κανόνα αρχικοποίησης. Έτσι, για την αρχικοποίηση των τριών πρώτων στοιχείων του πίνακα addr έχουμε την ακόλουθη δήλωση αρχικοποίησης:

```
struct addressT addr[10]={
        {“Demis”, “Serres”, “Rodou”, 23, 2321049118},
        {“Nikos”, “Athens”, “Kalymnou”, 459, 2105496783},
        {“Kiki”, “Herakleion”, “Kritis”, 21, 2810953013},
};
```

Για την αναφορά στα μέλη των στοιχείων του πίνακα ακολουθείται η προφανής σύνταξη. Έτσι, η έκφραση

addr[0].name ⟶ **όνομα της πρώτης διεύθυνσης**

addr[1].city ⟶ **πόλη της δεύτερης διεύθυνσης**

Στη C απαιτείται να συμπεριληφθεί στον ορισμό δομής η δεσμευμένη λέξη struct, π.χ. struct addressT addr[10]. Στη C++ η δεσμευμένη λέξη δεν είναι απαραίτητη, π.χ. addressT addr[10], όπως ακριβώς ορίζεται ένας ακέραιος, π.χ. int var1.

Προσοχή: **Αν ορίζαμε μία δομή χωρίς τον τίτλο του προσδιοριστή της, δηλαδή αντί να ορίσουμε**

```
struct rectangleT {
  …
} rect1;
```

ορίζαμε

```
struct {
  …
} rect1;
```

Απουσιάζει ο προσδιοριστής rectangleT

Θα προέκυπτε η μεταβλητή rect1 του συγκεκριμένου τύπου αλλά ο τύπος δε θα μπορούσε να ξαναχρησιμοποιηθεί γιατί δε θα είχε όνομα. Σε περίπτωση εκ νέου χρησιμοποίησης δομής της μορφής τής rect1, θα έπρεπε να θέσουμε εκ νέου:

```
struct {
  …
} rect2;
```

Παράδειγμα: Ένθεση δομών και πίνακες δομών

```
#include <stdio.h>

struct addressT {
  char name[40];
  char street[15];
  int number;
  int zip_code;
  char city[15];
};

struct addressT addr1={"John Doe","Telou Agra",10,62124,"Serres"};
```

```
struct addressT addr[10]={
        {"John Doe","Telou Agra",10,62124,"Serres"},
        {"Mitos Doe","Dilou",26,62124,"Serres"},
};

struct dayT {
  int date; int month; int year;
};

struct personT {
  struct addressT addr; struct dayT birthday;
};

struct personT p={
        {"Mitsos Doe","Dilou",26,61124,"Serres"},
        {28,1,79},
};
```

```
void main()
{
  printf( "struct address\n");
  printf( "%s\n%s\n%d\n%d\n%s\n",addr1.name,addr1.street,
        addr1.number,addr1.zip_code,addr1.city );

  printf( "struct person\n");
  printf( "%s\n%s\n%d\n%d\n%s\n",p.addr.name,p.addr.street,
             p.addr.number,p.addr.zip_code,p.addr.city );
  printf( "%d-%d-%d\n",p.birthday.date,p.birthday.month,
            p.birthday.year );

  printf( "Pinakas\n" );
  printf( "%s\n%s\n",addr[0].name,addr[1].name );
  printf( "%c\n",addr[1].name[0] );
} // τέλος της main
```

Σχολιασμός πηγαίου κώδικα:

```
struct addressT {
  char name[40];
  char street[15];
  int number;
  int zip_code;
  char city[15];
};
```

Δήλωση του τύπου δεδομένων δομής addressT. Η νέα δομή περιλαμβάνει τα μέλη name, street, number, zip και city.

```
struct addressT addr1={"John Doe","Telou Agra",10,62124,"Serres"};
```

Δηλώνεται η μεταβλητή τύπου δομής addressT με όνομα addr1 και αποδίδονται αρχικές τιμές στα μέλη της.

```
struct addressT addr[10]={
        {"John Doe","Telou Agra",10,62124,"Serres"},
        {"Mitos Doe","Dilou",26,62124,"Serres"},
};
```

Δηλώνεται ένας πίνακας με όνομα addr, ο οποίος έχει 10 στοιχεία τύπου δομής addressT, και αρχικοποιούνται τα δύο πρώτα στοιχεία του πίνακα.

```
struct dayT {
  int date;
  int month;
  int year;
};
```

Δηλώνεται ο τύπος δεδομένων δομής dayT με μέλη date, month, year.

```
struct personT {
  struct addressT addr;
  struct dayT birthday;
};
```

Δηλώνεται ο τύπος δεδομένων personT με μέλη addr και birthday, τα οποία είναι και αυτά δομές τύπων addressT και dayT, αντίστοιχα.

```
struct personT p={
        {"Mitsos Doe","Dilou",26,62124,"Serres"},
        {28,1,79},
};
```

Δηλώνεται η μεταβλητή p ως τύπου δομής personT και αποδίδονται αρχικές τιμές στα μέλη της. Θα πρέπει να προσεχθεί πως οι αρχικοποιήσεις κάθε μέλους της δομής περικλείονται σε άγκιστρα.

```
printf( "%d-%d-%d\n",p.birthday.date,p.birthday.month,p.birthday.year );
```

Προσέξτε την αναφορά στα μέλη ένθετων δομών. Χρησιμοποιείται ο τελεστής dot (.) χωρίς περιορισμό στο βάθος έκθεσης.

```
printf( "%s\n%s\n",addr[0].name,addr[1].name );
```

Κλήση της *printf* για εκτύπωση του πρώτου και δεύτερου στοιχείου του πίνακα addr.

```
printf( "%c\n",addr[1].name[0] );
```

Κλήση της *printf* για εκτύπωση του πρώτου χαρακτήρα του μέλους name του δεύτερου στοιχείου του πίνακα addr.

Παράδειγμα:

1) Να δημιουργηθεί ο πίνακας directory[40], ο οποίος θα αντιστοιχεί σε προσωπική ατζέντα. Κάθε στοιχείο του directory θα αποτελεί μεταβλητή τύπου δομής personT, η οποία θα έχει μέλη:

i) Δομή idT με: *α)* το ονοματεπώνυμο και *β)* δομή addressT με τη διεύθυνση του καταγεγραμμένου.

ii) Δομή teleT με τα τηλέφωνα (σταθερό, κινητό) και fax του καταγεγραμμένου.

iii) Δομή emT με το προσωπικό email και αυτό της εργασίας του καταγεγραμμένου.

2) Να γραφεί η main, μέσα στην οποία θα γίνεται ανάγνωση και εκτύπωση δύο στοιχείων του directory.

Λύση:

Σύμφωνα με την υπόθεση ο τύπος personT περιλαμβάνει ως μέλη μεταβλητές που είναι αποκλειστικά τύπου δομής. Επιπρόσθετα ο τύπος idT περιλαμβάνει ένα μέλος που είναι τύπου δομής (addressT). Κατά συνέπεια, η δήλωση των τύπων δεδομένων θα πρέπει να γίνει με την ακόλουθη σειρά:

```
struct addressT {
        . . .
};

struct idT {          Ή    struct teleT { Ή    struct emT {
  . . .                       . . .                 . . .
};                          };                    };

struct personT {
  . . .
};
```

Εφόσον δεν προσδιορίζονται επακριβώς τα περιεχόμενα του τύπου addressT, αυτά επιλέγονται κατά το δοκούν:

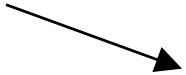
```
struct addressT {
  char street_name[40];
  int street_number;
  char city[40];
  int zip_code;
};
```

Είναι προτιμητέο οι τηλεφωνικοί αριθμοί να δηλώνονται ως αλφαριθμητικά γιατί αποτελούν 10-ψήφιους ή 14-ψήφιους ακέραιους. Κατά συνέπεια ο τύπος teleT μπορεί να έχει τη μορφή:

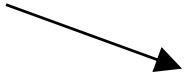
```
struct teleT {
    char wr_no[15];    // σταθερό τηλέφωνο
    char cell_no[15];  // κινητό τηλέφωνο
    char fax_no[15];
} ;
```

Συνολικά ο κώδικας είναι ο ακόλουθος:

```
#include <stdio.h>

struct addressT
{
  char street_name[40];
  int street_number;
  char city[40];
  int zip_code;
};

struct idT
{
  char name[20];
  char surname[40];
  struct addressT addr;
};
```



```
struct teleT
{
  char wr_no[15];
  char cell_no[15];
  char fax_no[15];
} ;

struct emT
{
  char em_work[40];
  char em_home[40];
} ;

struct personT
{
  struct idT ident;
  struct teleT tel;
  struct emT email;
};
```



```
main()  {
  struct personT directory[40];
  int i;
  for (i=0;i<=1;i++)  {
    printf( "\nRecord %d:",i+1 );
    printf( "\n\tName: ");          scanf( "%s",directory[i].ident.name );
    printf( "\n\tSurname: ");      scanf( "%s",directory[i].ident.surname );
    printf( "\n\tStreet name: ");
    scanf( "%s",directory[i].ident.addr.street_name );
    printf( "\n\tStreet number: ");
    scanf( "%d",&directory[i].ident.addr.street_number );
    printf( "\n\tCity: ");             scanf( "%s",directory[i].ident.addr.city );
    printf( "\n\tZip code: ");       scanf( "%d",&directory[i].ident.addr.zip_code );
    printf( "\n\tTelephone: ");    scanf( "%s",directory[i].tel.wr_no );
    printf( "\n\tCell telephone: ");          scanf( "%s",directory[i].tel.cell_no );
    printf( "\n\tFax: ");              scanf( "%s",directory[i].tel.fax_no );
    printf( "\n\tE-mail (work): ");      scanf( "%s",directory[i].email.em_work );
    printf( "\n\tE-mail (home): ");     scanf( "%s",directory[i].email.em_home );
  }  // τέλος της for
}  // τέλος της main
```

Αποτελέσματα: