ΠΕΡΙΕΧΟΜΕΝΑ

Δομές

Πέρα από τους πίνακες, που αποτελούν ομαδοποιήσεις ομοειδών μεταβλητών, η C παρέχει στον προγραμματιστή τη δυνατότητα ομαδοποίησης μεταβλητών διαφορετικών βασικών τύπων, οι οποίες ονομάζονται **δομές (structures)**.

Κάθε δομή έχει όνομα και χώρο αποθήκευσης στη μνήμη. Έχει, δηλαδή, ιδιότητες μεταβλητής. Πράγματι, οι δομές είναι μεταβλητές ο τύπος των οποίων δημιουργείται από τον προγραμματιστή, σε αντίθεση με τους βασικούς τύπους μεταβλητών – int, float, char, κ.λ.π. – που εμπεριέχονται στη γλώσσα και αποτελούν θεμελιώδη δομικά της στοιχεία. Η ευχέρεια δημιουργίας μεταβλητών προσαρμοσμένων στις ανάγκες του προς επίλυση προβλήματος συντελεί στην ευελιξία και συνεκτικότητα των προγραμμάτων.

Ενώ, λοιπόν, για τις μεταβλητές των βασικών τύπων αρκούν δηλώσεις όπως, για παράδειγμα, οι:

```
char ch;
float fn;
```

ώστε οι ch και fn να αποτελέσουν μεταβλητές ενός προγράμματος, για την περίπτωση των δομών ο προγραμματιστής πρέπει να δώσει στον compiler πρόσθετη πληροφορία για τον τρόπο συγκρότησής τους. Η πρόσθετη αυτή πληροφορία είναι **η δήλωση της μορφής (shape) της δομής.**

Ο γενικός τύπος δήλωσης μορφής δομής είναι ο ακόλουθος:

```
struct όνομα της μορφής της δομής
{
  μέλος;
  μέλος;
  . . . . .
  μέλος;
};
```

Στην εντολή δήλωσης κάθε μορφής δομής προηγείται **πάντοτε η δεσμευμένη λέξη struct**. Ακολουθεί το, *προαιρετικό*, όνομα που ο προγραμματιστής δίνει στη συγκεκριμένη **μορφή** δομής και το σώμα με τα μέλη της δομής, που είναι μεταβλητές των βασικών τύπων της γλώσσας ή μπορεί να είναι και δομές άλλου τύπου. **Η λατινική άνω τελεία (;) τοποθετείται πάντοτε μετά το τελευταίο άγκιστρο } της δήλωσης της μορφής δομής**, όπως συμβαίνει και με τις δηλώσεις μεταβλητών που ανήκουν στους βασικούς τύπους της γλώσσας.

Ας δούμε τα πράγματα με ένα παράδειγμα.

```
sruct book
{
  char publisher[25];
  char title[45];
  char author[25];
  int No_Of_Pages;
  float price;
};
```

Με την εντολή αυτή εισάγουμε ένα νέο τύπο δεδομένων, **έναν νέο τύπο δομής**, στον οποίο έχουμε δώσει το όνομα **book**. Έχουμε δηλαδή ουσιαστικά δημιουργήσει ένα **καλούπι** από το οποίο μπορούμε να παράγουμε δομές (μεταβλητές) που δεν ανήκουν στους βασικούς τύπους της C, αλλά στον τύπο book. (Ας υπενθυμίσουμε εδώ ότι " τύπος " είναι η παλαιότερη ελληνική λέξη για το καλούπι). Η μορφή book έχει πέντε μέλη, τρεις πίνακες χαρακτήρων καταλλήλων διαστάσεων, μια μεταβλητή int και μια μεταβλητή float. Η ως άνω δήλωση μορφής δεν δεσμεύει μνήμη, αλλά αποτελεί ενημέρωση του compiler ότι ο προγραμματιστής, εκμεταλλευόμενος της δυνατότητες της γλώσσας, προτίθεται να δημιουργήσει νέου τύπου αντικείμενα που θα χρησιμοποιηθούν στο υπό σύνταξη πρόγραμμα.

Μετά τη δήλωση μορφής είμαστε σε θέση να αρχίσουμε την παραγωγή **αντιτύπων (instants)**, δηλαδή **των δομών** που θα βγουν από το συγκεκριμένο καλούπι. Μια τέτοια παραγωγή πραγματοποιείται με τη **δήλωση δομών**. Έτσι, μεταβλητές (δομές), τύπου book, δηλώνονται με εντολές όπως οι:

```
struct book book_1;
struct book book_2, book_3;
```

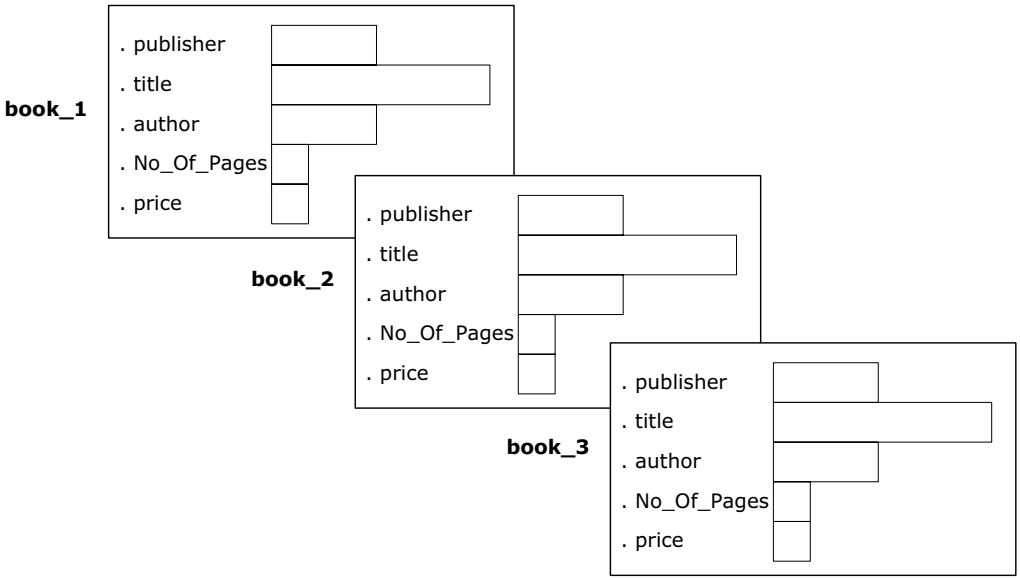
με τις οποίες γίνεται αυτομάτως και ο ορισμός των μελών της κάθε μιας δομής, δεσμεύεται δηλαδή μνήμη για τα μέλη αυτά. Η δήλωση γίνεται πάντοτε με χρήση της δεσμευμένης λέξης struct η οποία ακολουθείται από το όνομα που δόθηκε στη συγκεκριμένη μορφή (στο συγκεκριμένο καλούπι) δομής, εδώ book, και καταλήγει με τα ονόματα μιας η περισσότερων δομών του υπ' όψη τύπου, π.χ. book_2, book_3.

Η προσπέλαση των μελών των δομών γίνεται με χρήση του τελεστή **τελεία (.)**, ο οποίος παρέχει τη δυνατότητα πρόσβασης και απόδοσης τιμής σε κάθε ένα μέλος μιας δομής ξεχωριστά

Έτσι, ο γενικός τύπος απόδοσης τιμής σε μέλη δομής είναι:

```
όνομαΔομής.όνομαΜέλους = τιμή συμβατού τύπου;
```

Με τις παραπάνω δύο δηλώσεις δομών δημιουργούνται οι book_1, book_2, book_3 και τους παραχωρείται χώρος στη μνήμη όπως φαίνεται στο σχήμα.



Παραδείγματα απόδοσης τιμών σε μέλη των ως άνω δομών είναι τα ακόλουθα:

```
book_3.No_Of_Pages = 384;
book_1.price = 25.00;
scanf("%s", book_1.publisher);
```

Οι τιμές αυτές είναι δυνατόν να αλλάξουν κατά τη ροή του προγράμματος, αφού τα μέλη των δομών είναι μεταβλητές. Για παράδειγμα οι εντολές:

```
book_3.No_Of_Pages = 2384;
book_1.price = 225.00;
scanf("%s", book_1.publisher);
```

αλλάζουν τις τιμές των ως άνω μεταβλητών, εφόσον στη νέα κλήση της scanf() πληκτρολογείται διαφορετικό όνομα για τον publisher.

Είναι ευνόητο ότι όταν αναφερόμαστε στα μέλη μιας δομής πρέπει απαραιτήτως να καθορίζουμε τη δομή στην οποία αυτά ανήκουν. Έξω από τη δομή τα μέλη της δεν έχουν υπόσταση. Η εντολή :

```
No_Of_Pages = 404;
```

δεν είναι αναγνωρίσιμη από τον compiler.

Όπως συμβαίνει και με τους συμβατικούς τύπους μεταβλητών οι δομές μπορούν να ομαδοποιηθούν σε **πίνακες δομών**. Με την εντολή:

```
struct book Library[1200];
```

δηλώνεται ο πίνακας Library με 1200 στοιχεία καθένα από τα οποία είναι δομή τύπου book.

Ας δούμε μια πιο ολοκληρωμένη παρουσίαση των πραγμάτων με τη βοήθεια του προγράμματος:

```
#include <stdio.h>

struct book
{
  char publisher[25];
  char title[45];
  char author[25];
  int No_Of_Pages;
  float price;
};

main()
{
  int i;
  struct book bookself[3];
  for(i=0;i<3;i++)
  {
    printf("Book No %d\n", i+1);

    printf("Type the name of the publisher\n");
    scanf("%s", bookself[i].publisher);

    printf("Type the title\n");
    scanf("%s", bookself[i].title);

    printf("Type the name of the author\n");
    scanf("%s", bookself[i].author);

    printf("Type the number of pages\n");
   scanf("%d", &bookself[i].No_Of_Pages);

    printf("Type the price\n");
    scanf("%f", &bookself[i].price);
  }

  return 0;
}
```

Εδώ δηλώνεται ως **καθολική** (και άρα διαθέσιμη για χρήση και από, τυχόν υπάρχουσες, άλλες συναρτήσεις του προγράμματος) η μορφή της δομής book. Μέσα στη main() δηλώνεται ο πίνακας bookself (= ράφι) τριών δομών τύπου book. Στο βρόχο for καλείται ο χρήστης του προγράμματος να εισαγάγει τις τιμές που θα αποδοθούν στα μέλη κάθε δομής του bookself[3]. Προσέξτε τον τρόπο με τον οποίο ονομάζονται με χρήση του τελεστή τελεία τα μέλη των δομών – στοιχείων του bookself[3].

Σημειώστε ότι εντολές όπως η:

```
bookself[1].title[45] = "Gone with the wind";
```

δεν γίνονται αποδεκτές από τον compiler, διότι, όπως έχουμε τονίσει στο κεφάλαιο για τους πίνακες, αρχικοποίηση ενός πίνακα είναι δυνατή μόνο κατά τη δήλωσή του. Εδώ με την εντολή:

```
struct book bookself[3];
```

δηλώθηκαν οι τέσσερις δομές του bookself[3], χωρίς όμως να αρχικοποιηθούν και οι πίνακες - μέλη που περιέχονται σε αυτές. Κατά συνέπεια, απόδοση τιμής στα εν λόγω μέλη, σε άλλο σημείο του προγράμματος, μπορεί να πραγματοποιηθεί μόνο με χρήση συναρτήσεων όπως η scanf() ή η gets().

Τρόποι Δήλωσης Δομών

Η C παρέχει την δυνατότητα δήλωσης των δομών κατά διαφορετικούς τρόπους. Σε όλες όμως τις περιπτώσεις **οι δηλώσεις των δομών πρέπει να γίνονται πριν από κάθε εντολή της συνάρτησης που τις περιέχει**, όπως άλλωστε ισχύει και για τις δηλώσεις μεταβλητών των βασικών τύπων.

Στον πρώτο τρόπο δήλωσης αναφερθήκαμε ήδη παραπάνω, αλλά για την πληρότητα της ανάπτυξης θα τον παρουσιάσουμε και πάλι εδώ.

1. *Ξεχωριστή δήλωση μορφής και δήλωση αντιτύπων.*

   ```
   struct Player {char Name[30]; int Age; };
   struct Player player1, player2, BasketBallTeam[5];
   ```

   Στην πρώτη από τις ανωτέρω δύο εντολές δηλώνεται η μορφή της δομής Player, ενώ στη δεύτερη αφ' ενός οι δομές player1, player2, που είναι τύπου Player, και αφ' ετέρου ο πίνακας δομών BasketBallTeam[5] που έχει πέντε στοιχεία κάθε ένα από τα οποία είναι μια δομή τύπου Player.

2. *Ταυτόχρονη δήλωση μορφής, μέ ή χωρίς όνομα, και αντιτύπων*

   ```
   struct {char capital; float population;} Spain, Portugal;
   struct Identity {char* Name; int Age; float Salary;}
      Actor, Musician;
   ```

   Στην πρώτη από τις δύο περιπτώσεις, στην ίδια εντολή έχουμε δήλωση της μορφής και δήλωση δύο δομών των Spain και Portugal που ανήκουν στη μορφή αυτή που δηλώθηκε μεν αλλά δεν της δόθηκε κάποιο όνομα. Με μια τέτοια όμως δήλωση, δεν μπορούμε να δηλώσουμε άλλες δομές πέρα από τις δύο που δηλώθηκαν, και αυτό διότι δεν έχουμε όνομα της μορφής (του σχήματος, του καλουπιού) στο οποίο πρέπει να αναφερθούμε για να γίνει δυνατή η δήλωση και άλλων αντιτύπων. Η έλλειψη αυτή καλύπτεται με μια δήλωση τόσο της μορφής με το όνομά της όσο και των αντιτύπων όπως γίνεται στη δεύτερη περίπτωση.

3. *Δήλωση με χρήση της δεσμευμένης λέξης* **typedef**

   Εάν έχει γίνει η δήλωση:

   ```
   struct Player {char Name[30]; int Age; };
   ```

   τότε με την εντολή:

   ```
   typedef struct Player BestPlayer;
   ```

   η λέξη BestPlayer ορίζεται ως συνώνυμο των δύο λέξεων struct Player. Αυτό έχει ως συνέπεια να μπορούμε να δηλώνουμε δομές τύπου Player ως εξής:

   ```
   BestPlayer player3, player4;
   ```

   Η δήλωση αυτή είναι ισοδύναμη με την:

   ```
   struct Player player3, player4;
   ```

   την οποία επίσης μπορούμε να εξακολουθούμε να τη χρησιμοποιούμε στο ίδιο πρόγραμμα.

   Χρήση της typedef μπορεί να γίνει και στην ακόλουθη περίπτωση:

   ```
   typedef {char Name[30]; int Age; } GoodPlayer;
   ```

   Εδώ αν και δεν έχουμε δώσει όνομα στη μορφή της δομής μπορούμε να δηλώσουμε:

   ```
   GoodPlayer player5, FootBallTeam[11];
   ```

   Ο ορισμός συνωνύμων με την ως άνω χρήση της typedef μας απαλλάσσει από την ονοματοδοσία των μορφών των δομών.

   Γενικώς χρησιμοποιούμε την typedef, όταν θέλουμε να εισαγάγουμε ένα συνώνυμο ενός από τους υπάρχοντες ή από τους παραγόμενους τύπους της γλώσσας. Έτσι με την εντολή:

   ```
   typedef float decimal;
   ```

   Εισάγεται η λέξη decimal ως συνώνυμο της λέξης float και μπορεί να χρησιμοποιηθεί αντί αυτής, όπως γίνεται με την εντολή:

   ```
   decimal x, y, x;
   ```

   όπου τα x, y, x είναι στην πραγματικότητα μεταβλητές τύπου float.

   Αποτελεί συνήθη προγραμματιστική τακτική το να περιλαμβάνουμε τις δηλώσεις μορφής των δομών που χρησιμοποιούμε συχνά σε ένα ξεχωριστό αρχείο, το οποίο μπορούμε να το εισάγουμε στα προγράμματά μας με μια οδηγία #include του προεπεξεργαστή, κάθε φορά που το χρειαζόμαστε. Έτσι σε περίπτωση που είναι επιθυμητή η αλλαγή της δομής δεν έχουμε παρά να αλλάξουμε τον κώδικα σε ένα και μοναδικό αρχείο. Είναι προφανές ότι την ίδια αντιμετώπιση θα πρέπει να έχουν και οι συναρτήσεις που συντάσσονται από τον προγραμματιστή και χρησιμοποιούνται συχνά στα προγράμματα με την αρχική τους μορφή ή με μικρές παραλλαγές.

Ιδιότητες των Δομών

- Τα μέλη μιας δομής μπορούν να αρχικοποιηθούν με τη δήλωση της. Π.χ.

```
struct {char model[15]; int Year;} car1={"Fiat", 1960},
                                  car2={"Ferrari", 1984};
```

όπου η δήλωση της μορφής ακολουθείται από τη δήλωση των αντιτύπων car1 και car2 με αρχικοποίηση των μελών τους. Αν κάποια από τα μέλη δεν αρχικοποιηθουν τους αποδίδεται μηδενική τιμή.

Επίσης

```
struct Italian_car {char model[15]; int Year;} Ferrari=
                                    {"Testa Rossa", 1990};
```

όπου έχουμε ταυτόχρονη δήλωση και ονοματοδοσία της μορφής της δομής και παραγωγή του αντιτύπου Ferrari με αρχικοποίηση των μελών του.

- Οι δομές μπορούν να δηλωθούν ως extern, static, auto, register, όπως και οι βασικές μεταβλητές.
- Ο χειρισμός των μελών των δομών γίνεται όπως και όλων των κοινών μεταβλητών

```
Player1. age=23;
```

Ή και

```
Player1.age = player2.age;
```

- Μπορούμε να δηλώσουμε δείκτες σε δομές συγκεκριμένης μορφής, και να αποδώσουμε εν συνεχεία σε αυτούς ως τιμές τις διευθύνσεις αντιτύπων που παράγονται από την εν λόγω μορφή.

```
struct Player {char Name[30]; int Age}, player6, player7, *pstr;
pstr = &player6;
pstr = &player7;
```

- Μία ολόκληρη δομή είναι δυνατόν να αποδοθεί ως τιμή σε μία άλλη που έχει παραχθεί από το ίδιο καλούπι.

```
player6 = player7;
```

- Είναι επιτρεπτός ο ορισμός συναρτήσεων που περιέχουν στη λίστα των παραμέτρων τους δομές. Για παράδειγμα στο πρόγραμμα:

```
#include <stdio.h>
struct Player {char Name[30]; int Age;};
void ShowPlayer(struct Player Cards_Player)
{
 printf("%s %d\n", Cards_Player.Name, Cards_Player.Age);
}
```

```
main()
{
  struct Player player1={"Jordan", 48}, player2;
  ShowPlayer(player1);
  printf("Input the player's name\n");
  scanf("%s", player1.Name);
  printf("Input the player's age\n");
  scanf("%d", &player1.Age);
  player2 = player1;
  ShowPlayer(player2);
  return 0;
}
```

η συνάρτηση ShowPlayer() έχει ως παράμετρο μια δομή τύπου Player, της οποίας με χρήση της printf() εμφανίζει τα μέλη. Μέσα στη main() αρχικοποιείται η player1 και player2, αντίτυπα της μορφής Player. Στη πρώτη κλήση της ShowPlayer() εμφανίζονται τα μέλη της player1. Ακολούθως τα μέλη της player1 λαμβάνουν νέες τιμές, η δε player1, με τις νέες τιμές των μελών της, αποδίδεται ολόκληρη ως τιμή στη δομή player2. Στη δεύτερη κλήση της ShowPlayer() της διαβιβάζεται ως όρισμα η player2, της οποίας και εμφανίζει τα μέλη. Σημειώστε ότι η διαβίβαση τιμής στην ShowPlayer() γίνεται με αντιγραφή της player2 στην παράμετρο Cards_Player, που είναι και αυτή δομή τύπου Player. Το αποτέλεσμα της εκτέλεσης είναι το:

```
Jordan 48
Input the player's name
Johnson
Input the player's age
50
Johnson 50
```

- Μπορούμε να συντάξουμε συναρτήσεις που να έχουν επιστρεφόμενη τιμή μια ολόκληρη δομή. Ας το δούμε αυτό με μια τροποποίηση του παραπάνω προγράμματος:

```
#include <stdio.h>
struct Player {char Name[30]; int Age;};
void ShowPlayer(struct Player Cards_Player)
{
 printf("%s %d\n", Cards_Player.Name, Cards_Player.Age);
}

struct Player returnStruct()
{
  struct Player assistant;
  printf("Input the name\n");
  scanf("%s", assistant.Name);
  printf("Input the age\n");
```

```
  scanf("%d", &assistant.Age);
  return assistant;
}

main()
{
  struct Player player1={"Jordan", 48}, player2;
  ShowPlayer(player1);
  player2 = returnStruct("Jones", 54);
  ShowPlayer(player2);
  return 0;
}
```

Εν προκειμένω ορίζεται η συνάρτηση returnStruct() που επιστρέφει μια δομή τύπου Player. Η επιστρεφόμενη δομή αποδίδεται ως τιμή στην player2, τα μέλη της οποίας εμφανίζονται με χρήση της ShowPlayer(). Το αποτέλεσμα είναι:

```
Jordan 48
Input the name
Jones
Input the age
54
Jones 54
```

- Απαγορεύεται η σύγκριση δομών μεταξύ τους, ακόμη και αν προέρχονται από την ίδια μορφή. Εντολές όπως οι:

  ```
  while (player1<=player2) {. . . }
  ```

  ή

  ```
  while (player1==player2) {. . . }
  ```

  είναι μη αποδεκτές από τον compiler.

- **Τα ονόματα των δομών δεν είναι διευθύνσεις της μνήμης.**

Οι ως άνω ιδιότητες αναδεικνύουν τη διαφορά μεταξύ των δομών και των πινάκων, και τους ρόλους που επιφυλάσσει η C για καθένα από τα δύο αυτά είδη ομαδοποιημένων δεδομένων.

Τελεστής βέλος, προσπέλαση μελών μέσω δεικτών

Έχουμε ήδη αναφερθεί στον τελεστή τελεία (.) που χρησιμοποιούμε προκειμένου να αποκτήσουμε πρόσβαση στα μέλη μιας δομής.

Ας δούμε τώρα πώς μπορούμε να προσπελάσουμε τα μέλη μιας δομής μέσω δεικτών,

θεωρώντας τις εντολές:

```
struct Player {char Name[30]; int Age}, player6, player7, *pstr;
pstr = &player6;
```

Στη πρώτη από αυτές δηλώνονται οι δομές player6, player7, αλλά και ο δείκτης pstr σε δομή τύπου Player. Στη δεύτερη αποδίδεται στον pstr ως τιμή η διεύθυνση της δομής player6, δηλαδή ο pstr είναι η διεύθυνση της δομής, επομένως η *pstr είναι η ίδια η δομή. Συνεπώς με χρήση του τελεστή τελεία (.) θα μπορούσαμε να προσπελάσουμε το μέλος Age της player6 ως:

```
*pstr.Age = 26;
```

με βάση, όμως, τους κανόνες προτεραιότητας / σχετιστικότητας ο τελεστής τελεία, που βρίσκεται στη γραμμή 1 του αντίστοιχου πίνακα, έχει υψηλότερη προτεραιότητα από τον τελεστή *, που βρίσκεται στη δεύτερη γραμμή. Άρα η παραπάνω εντολή είναι ισοδύναμη με την:

```
*(pstr.Age) = 26;
```

αλλά η παράσταση αυτή δεν έχει έννοια εν προκειμένω.

Αν γράψουμε όμως:

```
(*pstr).Age = 26;
```

τότε αφού η (*pstr) είναι, όπως είπαμε, η δομή, τότε το (*pstr).Age είναι πράγματι το μέλος της που επιθυμούμε να προσπελάσουμε.

Η C παρέχει έναν κομψότερο τρόπο για την προσπέλαση των μελών των δομών μέσω δεικτών. Για το σκοπό αυτό έχει εισαχθεί ο **τελεστής βέλος (->)** που σχηματίζεται από το σημείο του "πλην" (-) ακολουθούμενο από το σημείο του "μεγαλύτερο από" (>), και **αντικαθιστά τον τελεστή τελεία, όταν πρόκειται για προσπέλαση μελών με χρήση δεικτών**. Έτσι η παραπάνω εντολή:

```
(*pstr).Age = 26;
```

μπορεί να γραφεί ισοδύναμα ως:

```
pstr->Age = 26;
```

Κατ' αναλογία μπορούμε να γράψουμε:

```
scanf("%s", pstr->Name);
```

για προσπέλαση του πρώτου μέλους της δομής player6. Καθώς επίσης και

```
pstr = &player7;
pstr->Age = 26;
```

οπότε τώρα η

```
pstr->Age = 26;
```

αποδίδει τιμή στο δεύτερο μέλος της player7.

Στην περίπτωση που έχουμε πίνακες δομών, έστω για παράδειγμα τον Library[1200], που, μαζί με τον q δείκτη σε δομή τύπου book, δηλώνεται στις παρακάτω εντολές:

```
struct book
{
  char publisher[25];
  char title[45];
  char author[25];
  int No_Of_Pages;
  float price;
};
```

```
struct book Library[1200], *q;
```

τότε το όνομα του πίνακα, το Library, είναι η και διεύθυνση του καθώς και η διεύθυνση του πρώτου του στοιχείου - δομής, όπως συμβαίνει με όλους τους πίνακες στη C.

Επομένως το 900^{ο} στοιχείο – δομή του εν λόγω πίνακα μπορεί να προσπελασθεί είτε ως Library[899], είτε μέσω της διεύθυνσής του που είναι Library+899. Στην πρώτη περίπτωση το μέλος author του εν λόγω 900^{ου} στοιχείου – δομής μπορεί να προσπελασθεί ως

```
Library[899].author,
```

ενώ στη δεύτερη ως

```
(Library+899)->author.
```

Οι παρενθέσεις εδώ είναι απαραίτητες, διότι χωρίς αυτές η υψηλότερη προτεραιότητα του τελεστή βέλους (->), σε σχέση με εκείνη του συμβόλου της πρόσθεσης (+), θα μας έδινε παράσταση ισοδύναμη με την Library+(899->author) η οποία στερείται λογικής έννοιας.

Έτσι με μια εντολή όπως η:

```
scanf("%s", (Library+899)->author);
```

έχουμε τη δυνατότητα να αρχικοποιήσουμε το μέλος της εν λόγω δομής ή να αλλάξουμε το περιεχόμενό του, που θα μπορούσε να εμφανισθεί στην οθόνη με μια printf(), όπως η:

```
printf("%s\n", (Library+899)->author);
```

Είναι προφανές ότι το Library+899 ως σταθερά διεύθυνσης μπορεί να αποδοθεί σε μεταβλητή κατάλληλου τύπου. Μια τέτοια είναι ο ανωτέρω δηλωθείς δείκτης q. Επομένως είναι αποδεκτές εντολές όπως οι:

```
q = Library; /*Το όνομα κάθε πίνακα είναι σταθερά διεύθυνσης*/
q = &Library[i]; /*για 0<= i <=1199*/
q = Library+i;  /*για 0<= i <=1199*/
```

καθώς και οι:

```
scanf("%s", (q+99)->author);
printf("%s\n", (q+57)->author);
```

κ.ο.κ.