Περιεχόμενα

Κεφάλαιο 9

Δομές και Ενώσεις Δεδομένων

Στο κεφάλαιο 7 είδαμε ότι, πολλές φορές είναι βολικό να αντιμετωπίζουμε ένα σύνολο δεδομένων του ίδιου τύπου σαν μια ενότητα, που ονομάζουμε πίνακα. Ο πίνακας είναι ειδική περίπτωση μιας γενικότερης μορφής με το όνομα **δομή δεδομένων** (data structure). Η δομή δεδομένων προσφέρει τη δυνατότητα ομαδοποίησης χαρακτηριστικών στοιχείων ενός αντικειμένου για εύκολη αναφορά και επεξεργασία. Οι δομές δεδομένων αποτελούν τις βάσεις κάθε εμπορικής εφαρμογής, επειδή διευκολύνουν το χειρισμό μιας ομάδας μεταβλητών που σχετίζονται μεταξύ τους με βάση λογικούς κανόνες που προσδιορίζει ο άνθρωπος.

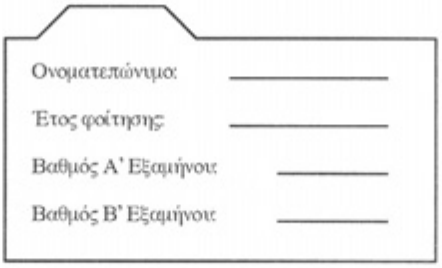
Πέρα από τη βασική έννοια της δομής δεδομένων που αναπτύσουμε στις πρώτες παραγράφους του κεφαλαίου, παρουσιάζουμε επίσης τρόπους δημιουργίας άλλων πιο πολύπλοκων κατασκευών, όπως είναι οι **πίνακες δομών** (arrays of structures), οι **λίστες** (lists), και τα **ιεραρχικά δέντρα** (hierarchical trees), και περιγράφουμε τις εντολές ορισμού νέων τύπων δεδομένων. Τέλος εισάγουμε την έννοια της **ένωσης δεδομένων** (data union) και δείχνουμε τον τρόπο αποθήκευσης διαφορετικών τύπων δεδομένων σε μια μεταβλητή, ανάλογα με τις ανάγκες του προγράμματος.

9.1 Η Έννοια της Δομής Δεδομένων

Δομή δεδομένων (data structure) ονομάζουμε μια συλλογή από μεταβλητές διαφορετικών τύπων που συνδέονται μεταξύ τους με μια λογική σχέση. Λέγοντας λογική σχέση εννοούμε ότι τα δεδομένα που αποτελούν τη δομή καθορίζονται από τον άνθρωπο με βάση κανόνες και συνήθειες της ζωής του. Θεωρήστε για παράδειγμα ότι θέλουμε να αναπαραστήσουμε στον υπολογιστή τις καρτέλλες των σπουδαστών μιας τάξης, που κάθε μία περιλαμβάνει τις ακόλουθες πληροφορίες:

- `Ονοματεπώνυμο σπουδαστή`
- `Έτος φοίτησης`
- `Βαθμός Α' εξαμήνου`
- `Βαθμός Β' εξαμήνου`

Τα στοιχεία αυτά είναι λογικά συνδεδεμένα μεταξύ τους, επειδή κάθε τέτοια τετράδα δεδομένων αναφέρεται σ' ένα συγκεκριμένο σπουδαστή. Λέμε τότε ότι τα τέσσερα αυτά δεδομένα αποτελούν τα **μέλη** (members) της δομής "σπουδαστής" (`student`). Συχνά είναι βολικό να φανταζόμαστε τη δομή δεδομένων σαν μια καρτέλλα που έχει γραμμένα πάνω της τα μέλη της δομής και διαθέτει κάποιο ελεύθερο χώρο για την αναγραφή των αντίστοιχων τιμών τους. Για την περίπτωση της δομής `student` η καρτέλλα παίρνει τη μορφή που δείχνει το σχήμα 9.1.



Σχήμα 9.1. *Η δομή δεδομένων student.*

Αφού καθορίσουμε το πλήθος των μελών της δομής, προχωρούμε στον προσδιορισμό του τύπου και μεγέθους κάθε μέλους. Συνήθως η επιλογή αυτή είναι απλή συνέπεια των δεδομένων που αντιπροσωπεύει κάθε μέλος. Για παράδειγμα, το έτος φοίτησης είναι λογικό να οριστεί ως ακέραιος, αφού οι δυνατές τιμές του είναι μόνο θετικοί ακέραιοι αριθμοί. Παρόμοια μπορούμε να ορίσουμε τον τύπο των άλλων μελών:

Ονομα μέλους	Τύπος δεδομένων
Ονοματεπώνυμο	Πίνακας χαρακτήρων 40 θέσεων
Έτος φοίτησης	Ακέραιος
Βαθμός Α' εξαμήνου	Πραγματικός απλής ακρίβειας
Βαθμός Β' εξαμήνου	Πραγματικός απλής ακρίβειας

Στην περίπτωση αλφαριθμητικών δεδομένων, το μήκος του αλφαριθμητικού καθορίζεται αυθαίρετα με την προυπόθεση ότι εξασφαλίζει το απαραίτητο μέγεθος για την αποθήκευση της μεγαλύτερης δυνατής τιμής. Για παράδειγμα, στη δομή `student` το ονοματεπώνυμο ορίστηκε σαν ένας πίνακας χαρακτήρων 40 θέσεων, με τη λογική ότι τα περισσότερα ελληνικά ονοματεπώνυμα σχηματίζονται με λιγότερους από 40 χαρακτήρες.

Κάθε δομή δεδομένων χρειάζεται επίσης ένα όνομα με το οποίο αναφερόμαστε σ' αυτή, το οποίο ονομάζεται **ετικέτα της δομής** (structure tag). Αντίθετα με το όνομα ενός πίνακα που χρησιμοποιείται για αναφορά των μεταβλητών απο τις οποίες αποτελείται (στοιχεία του πίνακα), η ετικέτα μιας δομής είναι το όνομα που δίνουμε στην περιγραφή της διάταξης των μελών της. Ο σχηματισμός του ονόματος μιας ετικέτας ακολουθεί τους ίδιους βασικούς κανόνες με το όνομα μιας μεταβλητής, αλλά ο υπολογιστής χρησιμοποιεί την ετικέτα δομής σαν νέο τύπο δεδομένων αντί σαν αυτόνομη μονάδα μνήμης που κρατά δεδομένα του προγράμματος.

Η δυνατότητα της γλώσσας C να επιτρέπει τον ορισμό νέων τύπων δεδομένων από τον προγραμματιστή είναι πολύ σημαντική, ιδιαίτερα για την ανάπτυξη μεγάλων εφαρμογών. Αυτό συμβαίνει διότι ο σχεδιασμός πολύπλοκων εφαρμογών επιβάλλει να αντιμετωπίζουμε τα δεδομένα σαν **αντικείμενα** (objects) με ιδιότητες παρόμοιες μ' αυτές των πραγματικών αντικειμένων που αναπαριστούν. Η ευκολία χρήσης δομών ήταν και ο βασικός λόγος που η C αποτέλεσε τη βάση της γλώσσας C++ και του **αντικειμενοστρεφούς προγραμματισμού** (object-oriented programming ή OOP).

9.2 Δήλωση και Ορισμός Δομών

Πριν χρησιμοποιήσουμε μια δομή δεδομένων πρέπει να προσδιορίσουμε την **εσωτερική μορφή** της (format), δηλαδή το πλήθος και τον τύπο των μελών της. Ταυτόχρονα πρέπει να **δηλώσουμε** (declare) το όνομα της στο πρόγραμμα, ώστε να μπορούμε να αναφερόμαστε σ' αυτή. Στη γλώσσα C η δήλωση μιας δομής γίνεται με την εντολή `struct` που έχει το εξής συντακτικό:

```
struct ετικέτα_δομής
{
    ορισμός μέλους;
    ορισμός μέλους;
    ...
};
```

Κάθε *ορισμός μέλους* είναι μια κοινή εντολή ορισμού μιας μεταβλητής (π.χ. `int x;` ή `char name[40];`). Η εντολή `struct` δηλώνει μια νέα δομή δεδομένων με το όνομα της ετικέτας, αλλά δεν εξασφαλίζει χώρο μνήμης για την αποθήκευσή της. Στην πραγματικότητα η εντολή `struct` ορίζει ένα νέο τύπο δεδομένων πέρα από αυτούς που περιέχει εσωτερικά η γλώσσα C (internal types), δηλαδή τους `char`, `int`, `float` και `double`. Ο νέος αυτός τύπος παίρνει το όνομα: `struct` *ετικέτα* και μπορεί να χρησιμοποιηθεί με τον ίδιο ακριβώς τρόπο όπως και κάθε άλλος απλός τύπος της C. Για το λόγο αυτό λέμε ότι η δομή δεδομένων είναι ένας **παραγόμενος τύπος** (derived type) που σχηματίζεται από το συνδυασμό άλλων πιο απλών τύπων.

Σημειώστε ότι η λέξη `struct` είναι απαραίτητη πριν από το όνομα της ετικέτας για να επισημαίνει τη διαφορά από μια κοινή μεταβλητή του προγράμματος. Η εντολή `struct` μπορεί προαιρετικά να χρησιμοποιηθεί και για τον **ορισμό** (definition) μεταβλητών τύπου `struct` *ετικέτα*, ως εξής:

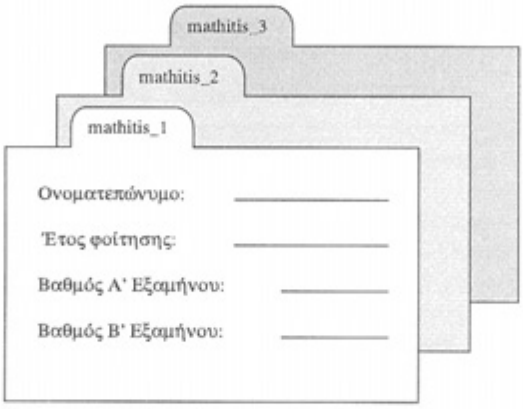
```
struct ετικέτα_δομής
{
    ορισμός μέλους;
    ορισμός μέλους;
    ...
} λίστα_μεταβλητών;
```

Στην περίπτωση αυτή η εντολή `struct` δρά όχι μόνο σαν δήλωση της δομής, αλλά και σαν ορισμός των μεταβλητών της *λίστας*. Για παράδειγμα μπορούμε να έχουμε:

```
struct student
{
    char name[40];
    int year;
    float grade_A;
    float grade_B;
} mathitis_1, mathitis_2, mathitis_3;
```

Η λέξη `student` είναι η ετικέτα (όνομα) της δομής δεδομένων. Με την εντολή αυτή ενημερώνουμε τον υπολογιστή ότι στο πρόγραμμα που ακολουθεί, θα χρησιμοποιηθούν τρεις μεταβλητές με τα ονόματα: `mathitis_1`, `mathitis_2` και `mathitis_3` που έχουν τη **μορφή** (format) της δομής δεδομένων `student`. Λέμε

ότι οι μεταβλητές αυτές είναι **αντίτυπα** (instances) της δομής `student`. Ο υπολογιστής ανταποκρίνεται στην εντολή δήλωσης της δομής εξασφαλίζοντας αρκετό χώρο μνήμης για την αποθήκευση των μεταβλητών που ορίζονται σ' αυτή, όπως δείχνει το σχήμα 9.2.



Σχήμα 9.2. *Ορισμός μεταβλητών (αντίτυπων) τύπου* `struct student`.

Όταν η δήλωση μιας δομής συνοδεύεται από τον ορισμό αντίτυπων, η ετικέτα της δομής μπορεί να παραληφθεί. Κάτι τέτοιο όμως απαγορεύει τον ορισμό άλλων αντίτυπων αργότερα στο πρόγραμμα, αφού δεν υπάρχει τρόπος αναφοράς του τύπου που ορίζει η δομή. Για το λόγο αυτό, καλό είναι να τοποθετούμε πάντα την ετικέτα στη δήλωση μιας δομής δεδομένων.

Όπως και με τις κοινές μεταβλητές του προγράμματος, ο ορισμός του αντίτυπου μιας δομής μπορεί να συνοδεύεται από την απόδοση αρχικών τιμών. Για να δώσουμε αρχικές τιμές σ' ένα αντίτυπο τοποθετούμε τις αρχικές τιμές των μελών του μέσα σε άγκιστρα, όπως ακριβώς και με τις αρχικές τιμές των πινάκων. Για παράδειγμα μπορούμε να έχουμε:

```
struct book
{
    char title[30];
    char author[30];
    char publisher[30];
    int year;
    int price;
} one_book = { "Πόλεμος και Ειρήνη", "Λέων Τολστόι",
               "Διεθνή βιβλία Ο.Ε.", 1990, 15000};
```

Με την εντολή αυτή δηλώνουμε μια νέα δομή δεδομένων με το όνομα `book`, που αποτελείται από πέντε μέλη: Τον τίτλο του βιβλίου, το όνομα του συγγραφέα και του εκδοτικού οίκου, το έτος έκδοσης, και την τιμή του βιβλίου σε δραχμές. Παράλληλα ορίζουμε ένα αντίτυπο της δομής με το όνομα `one_book` που παίρνει αρχικές τιμές ταυτόχρονα με τον ορισμό του, με τις τιμές που δίνονται μέσα στις αγκύλες. Προσέξτε ότι δεν επιτρέπεται η απ'ευθείας απόδοση αρχικών τιμών σ' ένα μέλος της δομής `book`. Για παράδειγμα είναι λάθος να γράψουμε:

```
char title[30] = "Πόλεμος και Ειρήνη";      /* Λάθος */
```

Η παρατήρηση αυτή είναι απλή συνέπεια της διπλής λειτουργίας της εντολής `struct` (δήλωση και ορισμός) και υπογραμμίζει ότι τα μέλη μιας δομής δεν είναι κανονικές μεταβλητές αλλά απλώς συμβολικά ονόματα που δείχνουν στον υπολογιστή την εσωτερική διάταξη και το μέγεθος κάθε δεδομένου. Μόνο με την τοποθέτηση του ονόματος ενός αντίτυπου (π.χ. `one_book`) ο υπολογιστής εξασφαλίζει κάποιο συγκεκριμένο χώρο στη μνήμη του υπολογιστή για την αποθήκευση των δεδομένων.

9.3 Επεξεργασία Δομών

Η απόδοση αρχικών τιμών σε αντίτυπο μιας δομής ταυτόχρονα με τον ορισμό του αντιτύπου δεν εφαρμόζεται συχνά στην πράξη, γιατί τις περισσότερες φορές δεν γνωρίζουμε εκ των προτέρων τα δεδομένα που θα επεξεργαστεί το πρόγραμμα μας. Επιπλέον, η εργασία μ' ένα μεγάλο σύνολο δομών που καθεμία μπορεί να αποτελείται από αρκετά μέλη, κάνει την απόδοση αρχικών τιμών δύσκολη αν όχι αδύνατη. Ένας καλύτερος τρόπος εργασίας με τις δομές δεδομένων βασίζεται στον τελεστή τελείας (.). Ο τελεστής αυτός επιτρέπει την πρόσβαση σε κάθε μέλος της δομής ξεχωριστά και έχει το εξής συντακτικό:

όνομα_αντίτυπου . όνομα_μέλους

Μπορούμε έτσι να γράψουμε:

```
one_book.price = 15000;    ή    mathitis_1.grade_A = 9.5;
```

Στο πρώτο παράδειγμα αποδίδουμε στο μέλος `price` του αντίτυπου `one_book` την τιμή `15000` ενώ στο δεύτερο αποδίδουμε την τιμή `9.5` στο μέλος `grade_A` του αντίτυπου `mathitis_1`. Οι παλαιότερες εκδόσεις της γλώσσας C επέβαλλαν τον τρόπο αυτό επεξεργασίας για κάθε λειτουργία μεταξύ αντίτυπων δομών. Η ANSI C όμως επιτρέπει την αντιγραφή των τιμών των μελών ενός αντίτυπου σε ένα άλλο, εφόσον δέβαια έχουν την ίδια ακριβώς εσωτερική διάταξη (δηλ. ανήκουν ουσιαστικά στην ίδια δομή). Επομένως η εντολή:

```
mathitis_1 = mathitis_2;
```

είναι ισοδύναμη με το πακέτο των εντολών:

```
mathitis_1.name = mathitis_2.name;
mathitis_1.year = mathitis_2.year;
mathitis_1.grade_A = mathitis_2.grade_A;
mathitis_1.grade_B = mathitis_2.grade_B;
```

Η δυνατότητα αυτή της ANSI C διευκολύνει σχετικά την επεξεργασία των δομών αλλά δεν εξαφανίζει τελείως την ανάγκη πρόσβασης κάθε μέλους του αντίτυπου μιας δομής ξεχωριστά. Στην πραγματικότητα για οποιαδήποτε άλλη επεξεργασία πρέπει να εκτελέσουμε τις απαραίτητες εντολές για κάθε μέλος της δομής.

Επειδή τα αντίτυπα των δομών (π.χ. `one_book`, `mathitis_1`) μεταβιβάζουν δεδομένα μέσω τιμής (και όχι μέσω αναφοράς όπως οι πίνακες), συνηθίζουμε να δηλώνουμε τις δομές του προγράμματος καθολικά (έξω από το σώμα των συναρτήσεων) και να ορίζουμε τοπικά τις μεταβλητές δομών (αντίτυπα) που χρησιμοποιεί κάθε συνάρτηση. Το επόμενο παράδειγμα δείχνει μια εφαρμογή της ιδέας αυτής:

```
 1 #include <stdio.h>
 2
 3 struct student          /* Δήλωση της δομής student (global) */
 4 {
 5     char name[40];      /* Ονοματεπώνυμο σπουδαστή */
 6     int year;           /* Έτος φοίτησης */
 7     float grade_A;      /* Βαθμός Α' εξαμήνου */
 8     float grade_B;      /* Βαθμός Β' εξαμήνου */
 9 };
10 main()
11 {
12   struct student mathitis_1, mathitis_2;  /*Ορισμός αντίτυπων */
13
14   printf("Δώστε το όνομα του μαθητή:");        /* Εισαγωγή στοιχείων */
15   scanf("%s", mathitis_1.name);                /* στο αντίτυπο:      */
16   printf("Δώστε το έτος φοίτησης:");           /* mathitis_1         */
17   scanf("%d", &mathitis_1.year);
18   printf("Δώστε το βαθμό του Α εξάμηνου:");
19   scanf("%f", &mathitis_1.grade_A);
20   printf("Δώστε το βαθμό του Β εξάμηνου:");
21   scanf("%f", &mathitis_1.grade_B);
22
23   mathitis_2 = mathitis_1;    /* Αντιγραφή των μελών */
24
25   printf("Όνομα.............: %s\n", mathitis_2.name);
26   printf("Έτος φοίτησης .....: %d\n", mathitis_2.year);
27   printf("Βαθμός Α εξαμήνου ..: %f\n", mathitis_2.grade_A);
28   printf("Βαθμός Β εξαμήνου...: %f\n", mathitis_2.grade_B);
29 }
```

Το πρόγραμμα εισάγει τις τιμές των μελών του αντίτυπου `mathitis_1` της δομής `student` αλλά τυπώνει τις αντίστοιχες τιμές του αντίτυπου `mathitis_2` της ίδιας δομής. Επειδή όμως μεταξύ της εισόδου (εντολές 14-21) και της εξόδου των στοιχείων (εντολές 25-28) μεσολαβεί η εντολή μεταφοράς των τιμών της μιας δομής στην άλλη (εντολή 23), τα στοιχεία που τυπώνονται είναι ίδια με αυτά που εισήγαγε ο χρήστης.

Προσέξτε ότι η δομή `student` δηλώνεται στην αρχή του προγράμματος (εντολές 3-9), έξω από κάθε συνάρτηση του προγράμματος. Άρα η δήλωση αυτή είναι καθολική και μπορεί να χρησιμοποιηθεί από κάθε συνάρτηση του προγράμματος για τον ορισμό τοπικών μεταβλητών τύπου `student`. Για παράδειγμα, η συνάρτηση `main()` ορίζει δύο τέτοιες μεταβλητές με ονόματα `mathitis_1` και `mathitis_2` στην εντολή 13. Συνεπώς, η δήλωση της δομής σχηματίζει ένα νέο τύπο δεδομένων με το όνομα `struct student` που μπορεί να χρησιμοποιηθεί όπως οποιοσδήποτε άλλος εσωτερικός τύπος δεδομένων (π.χ. `char`, `int`, κτλ.) για τον ορισμό μεταβλητών, συναρτήσεων κ.τλ. Για παράδειγμα, θα μπορούσαμε να ορίσουμε μια συνάρτηση ώς εξής:

```
struct student function(struct student παράμετρος)
```

Η εντολή ορίζει τη συνάρτηση `function()` που δέχεται σαν παράμετρο μια δομή τύπου `struct student` και επιστρέφει μια τιμή του ίδιου τύπου. Η μορφή μπορεί να φαίνεται κάπως πολύπλοκη αλλά εκτελεί την ίδια ακριβώς λειτουργία με μια εντολή της μορφής:

```
τύπος function(τύπος παραμέτρος)
```

με τη μοναδική διαφορά ότι ο τύπος εδώ είναι μια δομή δεδομένων και όχι ένας από τους βασικούς τύπους δεδομένων της C. Τα μέλη των αντίτυπων `mathitis_1` και `mathitis_2` της δομής `student` είναι κοινές μεταβλητές και χρησιμοποιούνται όπως κάθε άλλη μεταβλητή του προγράμματος. Για παράδειγμα, στις εντολές 17, 19 και 21 εισάγουμε τις τιμές τους με τη βοήθεια της συνάρτησης `scanf()`, μεταβιβάζοντας τις διευθύνσεις των θέσεων μνήμης όπου βρίσκονται αποθηκευμένες. Αντίστοιχα στις εντολές 25-28 τυπώνουμε τις τιμές τους χρησιμοποιώντας κατάλληλους προσδιορισμούς (`%s`, `%d` ή `%f`) ανάλογα με τον τύπο τους. Η μοναδική διαφορά επομένως από τις κοινές μεταβλητές, είναι ότι το όνομά τους σχηματίζεται από το συνδυασμό δύο άλλων ονομάτων που συνδέονται με τον τελεστή της τελείας (π.χ. `mathitis_2.name`).

9.4 Δομές Μέσα σε Δομές

Εφόσον μια δομή αντιμετωπίζεται από τον υπολογιστή όπως οποιοσδήποτε άλλος τύπος δεδομένων είναι προφανές ότι, μπορεί να καθορίζει τον τύπο ενός μέλους μιας άλλης δομής. Δημιουργούμε έτσι **ένθετες δομές** (nested structures) που μπορούν να φανούν χρήσιμες σε προγράμματα που εργάζονται με πολλές παρόμοιες δομές. Όταν δύο ή περισσότερες δομές έχουν κάποια κοινά μέλη, τότε τα μέλη αυτά μπορούν να σχηματίσουν μια ξεχωριστή δομή που βρίσκεται φωλιασμένη μέσα στις αρχικές δομές. Αν, για παράδειγμα, στο πρόγραμμα λογιστηρίου μιας επιχείρησης έχουν οριστεί δύο δομές, μία για τις αγορές και μία άλλη για τις πωλήσεις, όπως φαίνεται παρακάτω:

```
struct buy
{
    int receipt;
    char source[40];
    char address[40];
    char city[20];
    char country[20];
    int amount;
};
```

```
struct sale
{
    int invoice;
    char destination[40];
    char address[40];
    char city[20];
    char country[20];
    int amount;
};
```

τότε μπορούμε να ενοποιήσουμε τα κοινά στοιχεία των δύο δομών ορίζοντας μια νέα δομή με το όνομα `common` που περιέχει τα μέλη: `address`, `city`, `country` και `amount`:

```
struct buy
{
 int receipt;
 char source[40];
 struct common b_info;
};
```

```
struct sale
{
 int invoice;
 char destination[40];
 struct common s_info;
};
```

```
struct common
{
 char address[40];
 char city[20];
 char country[20];
 int amount;
};
```

Προσέξτε ότι η δήλωση της δομής `common` πρέπει να βρίσκεται πριν από τις δηλώσεις των δομών `buy` και `sale` ώστε ο υπολογιστής να γνωρίζει τη μορφή της δομής `common` όταν προσπαθήσει να καθορίσει την εσωτερική διάταξη των δύο άλλων δομών που τη χρησιμοποιούν (δηλ. `buy` και `sale`). Με τον ορισμό της δομής `common` οι αρχικές δομές `buy` και `sale` περιέχουν τώρα τρία μέλη η κάθε μία. Η πρόσβαση των στοιχείων της δομής `common` γίνεται με διπλή χρήση του τελεστή τελείας. Αν για παράδειγμα ορίσουμε το αντίτυπο `transaction` της δομής `buy` και το αντίτυπο `info` της δομής `common` όπως δείχνουμε παρακάτω:

```
struct buy transaction;
struct common info;
```

τότε μπορούμε να έχουμε: `transaction.info.amount = 50000;`

όπου θέτουμε το μέλος `amount` του αντίτυπου `info` της δομής `common` ίσο με `50000`.

9.5 Πίνακες Δομών

Οι δομές δεδομένων μπορούν να οργανωθούν σε πίνακες με παρόμοιο τρόπο όπως και οι απλές μεταβλητές του προγράμματος για να απλοποιήσουν την επεξεργασία ενός μεγάλου πλήθους όμοιων στοιχείων. Ο ορισμός ενός **πίνακα δομών** (array of structures) γίνεται με τον προσδιορισμό μέσα σε αγκύλες του πλήθους των δομών που περιέχει. Το συντακτικό είναι το ακόλουθο:

```
struct επικέτα_δομής
{
    ορισμός μέλους;
    ορισμός μέλους;
    ...
} όνομα_πίνακα[πλήθος_στοιχείων];
```

ή

```
struct επικέτα_δομής
{
 ορισμός μέλους;
 ορισμός μέλους;
 ...
};
struct επικέτα όνομα[πλήθος];
```

Οι δύο μορφές είναι ισοδύναμες όπως είδαμε στις προηγούμενες παραγράφους. Με την πρώτη μορφή, η δήλωση της δομής συνοδεύεται από τον ορισμό ενός πίνακα όμοιων δομών, ενώ στη δεύτερη ο ορισμός του πίνακα γίνεται σε ξεχωριστή εντολή και επομένως μπορεί να γίνει τοπικά μέσα στο σώμα μιας συνάρτησης. Ανεξάρτητα από τη μορφή που θα χρησιμοποιηθεί, οι εντολές αυτές ορίζουν έναν πίνακα με συγκεκριμένο *πλήθος* στοιχείων. Για παράδειγμα, ο ορισμός ενός πίνακα δομών `book` 10 στοιχείων γράφεται:

```
struct book
    {
        char title[30];
        char author[30];
        char publisher[30];
        int year;
        int price;
    } array[10];
```

Η εντολή ορίζει τον πίνακα `array[]` που αποτελείται από 10 αντίτυπα της δομής `book` (στοιχεία του πίνακα). Το όνομα κάθε στοιχείου του πίνακα σχηματίζεται από το όνομα του πίνακα και έναν ακέραιο αριθμό μέσα σε αγκύλες. Για παράδειγμα το τρίτο στοιχείο του πίνακα είναι το `array[2]`. Όπως είδαμε, η ANSI C επιτρέπει την αποδοση των τιμών ενός αντίτυπου δομής σ' ένα άλλο, με μια μόνο εντολή απόδοσης τιμής. Επομένως η εντολή:

```
array[3] = array[2];
```

μεταφέρει τις τιμές των μελών του στοιχείου `array[2]` στα αντίστοιχα μέλη του `array[3]`. Φυσικά το όνομα κάθε μέλους ενός στοιχείου μπορεί να σχηματιστεί κατά τα γνωστά με χρήση του τελεστή της τελείας. Για παράδειγμα η παράσταση:

```
array[3].year = 1993;
```

αποδίδει την τιμή `1993` στο μέλος `year` του αντίτυπου `array[3]` της δομής `book`. Προσέξτε ότι οι αγκύλες τοποθετούνται αμέσως μετά το όνομα του πίνακα και όχι στο τέλος του ονόματος του μέλους. Αυτό επιβάλλεται για να διακρίνουμε τους πίνακες δομών από τους πίνακες μέλη, όπως στις επόμενες δύο περιπτώσεις:

```
array[2].title      και      array.title[2]
```

Εδώ το πρώτο δεδόμενο παριστάνει το πρώτο μέλος του τρίτου στοιχείου του πίνακα δομών `array[]`. Το δεύτερο δεδόμενο είναι το τρίτο στοιχείο του πίνακα χαρακτήρων `title` που είναι μέλος του πρώτου στοιχείου του πίνακα `array[]`.

Παρ' ότι η αναφορά στα μέλη των στοιχείων ενός πίνακα δομών μπορεί να φαίνεται πολύπλοκη (ιδιαίτερα στην περίπτωση πολυδιάστατων πινάκων), ακολουθεί τους ίδιους βασικούς κανόνες που περιγράψαμε για τις απλές δομές. Αυτό που πρέπει **πάντα** να έχουμε υπ'όψη μας είναι ότι ο τελεστής τελείας έχει στα αριστερά του το όνομα του αντίτυπου μιας δομής και στα δεξιά του το όνομα ενός μέλους της δομής αυτής. Στην επόμενη παράγραφο παρουσιάζουμε ένα απλό παράδειγμα χρήσης των πινάκων δομών.

9.6 Παράδειγμα 9.1

Έστω ότι εργαζόμαστε σ' ένα video club και θέλουμε να δημιουργήσουμε έναν κατάλογο με τις ταινίες που διαθέτει το κατάστημα. Τα στοιχεία που μας ενδιαφέρουν για κάθε ταινία είναι τα εξής:

Περιγραφή μέλους	Ονομα μέλους δομής	Τύπος
Κωδικός ταινίας	code	int
Τίτλος ταινίας	title[]	char
Ονοματεπώνυμο πελάτη	customer[]	char
Ημερομηνία	date[]	char
Τιμή	price	int
Κατηγορία ταινίας	category	int

Με βάση τα στοιχεία αυτά δημιουργούμε (δηλώνουμε) μια δομή που την ονομάζουμε tape_type και στη συνέχεια ορίζουμε έναν πίνακα τέτοιων δομών. Το πρόγραμμα εισάγει και τυπώνει τα στοιχεία του πίνακα αυτού. Αν και οι λειτουργίες που υποστηρίζονται είναι πολύ βασικές, το πρόγραμμα δίνει μια ιδέα για την πολυπλοκότητα και το μεγέθος που μπορεί να πάρει ένα πρόγραμμα επεξεργασίας δομών δεδομένων.

```
/* Παράδειγμα 9.1 */

#include <stdio.h>
#define NO_TAPES 100        /* Μέγιστος αριθμός ταινιών */

struct tape_type
{
    int  code;              /* Κωδικός ταινίας */
    char title[30];         /* Τίτλος ταινίας */
    char customer[45];      /* Ονοματεπώνυμο πελάτη */
    char date[10];          /* Ημερομηνία */
    int  price;             /* Τιμή χρέωσης */
    int category;           /* Κατηγορία */
};

int main(void);
void input_data(struct tape_type tapes[]);
void print_data(struct tape_type tapes[]);
int get_index(void);

int main(void)
{
    int choice;
    struct tape_type tapes[NO_TAPES]; /* Ορισμός πίνακα δομών */
    do
    {
        printf("\n\n\n\n\n\n");        /* Εμφάνιση μενού επιλογών */
        printf("1. Εισαγωγή\n");
        printf("2. Εκτύπωση\n");
```

```
30          printf("0. Έξοδος\n\n");
31          printf("Επιλογή:");
32          scanf("%d", &choice);
33          switch(choice)
34          {
35              case 0: { break; }
36              case 1: { input_data(tapes); break; }
37              case 2: { print_data(tapes); break; }
38          }
39      } while(choice != 0);
40  }
41  void input_data(struct tape_type tapes[])
42  {
43      int i;
44      while((i = get_index()) != 100)
45      {
46         tapes[i].code = i;
47         printf("Τίτλος ταινίας..:"); gets(tapes[i].title);
48         printf("Όνομα πελάτη .:"); gets(tapes[i].customer);
49         printf("Ημερομηνία....:"); gets(tapes[i].date);
50         printf("Χρέωση (δρχ.)..:"); scanf("%d", &tapes[i].price);
51         printf("Κατηγορία ..:"); scanf("%d", &tapes[i].category);
52      }
53      return;
54  }
55  void print_data(struct tape_type tapes[])
56  {
57      int i;
58      while((i = get_index()) != 100)
59      {
60          printf("Κωδικός ταινίας..:%d\n", tapes[i].code);
61          printf("Τίτλος ταινίας....:%s\n", tapes[i].title);
62          printf("Όνομα πελάτη ...:%s\n", tapes[i].customer);
63          printf("Ημερομηνία......:%s\n", tapes[i].date);
64          printf("Χρέωση (δρχ.) ....:%d\n", tapes[i].price);
65          printf("Κατηγορία.......:%d\n", tapes[i].category);
66      }
67      return;
68  }
69  int get_index()
70  {
71      int index;
72      do
73      {
74          printf("\nΔώστε τον κωδικό της ταινίας (0-99):\n");
```

```
75          printf("(Για έξοδο δώστε 100)\nΕπιλογή:");
76          scanf("%d", &index);
77          if((index < 0) || (index > 100))
78          {
79              printf("\nΛάθος κωδικός\n");
80          }
81      }while((index < 0) || (index > 100));
82      getchar();
83      return(index);
84  }
```

Το πρόγραμμα εμφανίζει στην οθόνη ένα μενού επιλογών (εντολές 27-31) και εισάγει την επιλογή του χρήστη στην εντολή 32. Ανάλογα με την τιμή της μεταβλητής `choice` καλείται η συνάρτηση `input_data()` ή η συνάρτηση `print_data()`. Οι δύο αυτές συναρτήσεις δέχονται σαν μοναδικό όρισμα το όνομα του πίνακα `tapes[]` που κάθε στοιχείο του είναι μια δομή τύπου `tape_type`.

Τόσο στην εισαγωγή όσο και στην εκτύπωση των στοιχείων, η πρόσβαση κάθε στοιχείου του πίνακα `tapes[]` γίνεται καθορίζοντας τον κωδικό της αντίστοιχης ταινίας. Δηλαδή το μέλος `tapes[i].code` λειτουργεί σαν ευρετήριο του πίνακα που προσδιορίζει το στοιχείο που επιθυμεί να επεξεργαστεί ο χρήστης. Λέμε τότε ότι ο κωδικός ταινίας είναι το **κλειδί** (key) αναφοράς στα στοιχεία του πίνακα `tapes[]`.

Η διακοπή εισαγωγής ή εκτύπωσης στοιχείων του πίνακα `tapes[]` γίνεται με εισαγωγή του ειδικού αριθμού 100 στην εντολή 76. Η τιμή αυτή είναι έξω από το επιτρεπτό πεδίο τιμών του κωδικού ταινίας (άρα και των στοιχείων του πίνακα) και επομένως μπορεί να χρησιμοποιηθεί για τον έλεγχο των διαδικασιών εισαγωγής και εκτύπωσης. Φυσικά, οποιαδήποτε άλλη τιμή έξω από την περιοχή [0 - 99] μπορεί να χρησιμοποιηθεί για το σκοπό αυτό.

9.7 Δείκτες Δομών

Οι **δείκτες σε δομές δεδομένων** (pointers to structures) ορίζονται με την ίδια ευκολία όπως και οι δείκτες απλούστερων τύπων (π.χ. `int`). Είδαμε δηλαδή ότι αν ορίσουμε:

```
int x;      /* Ακέραια μεταβλητή */
int *px;    /* Δείκτης ακέραιου δεδομένου */
```

και θέσουμε:

```
px = &x;
```

τότε ο δείκτης `px` "δείχνει" τη διεύθυνση που βρίσκεται αποθηκευμένη η μεταβλητή `x`. Παρόμοια, αν δηλώσουμε τη δομή δεδομένων `magazine_type` με την εντολή:

```
struct magazine_type
{
    char title[20];
    char publisher[40];
    int month;
    int year;
```

```
    int price;
}
```

και ορίσουμε τα `magazine` και `mag_ptr` με τις εντολές:

```
struct magazine_type magazine;
struct magazine_type *mag_ptr;
```

είναι προφανές ότι μπορούμε να γράψουμε:

```
mag_ptr = &magazine;
```

οπότε ο δείκτης `mag_ptr` "δείχνει" τη θέση μνήμης που βρίσκεται αποθηκευμένο το αντίτυπο `magazine` της δομής `magazine_type`. Δηλαδή ο `mag_ptr` είναι ένας **δείκτης δομής** (structure pointer) και μπορεί να χρησιμοποιηθεί για την επεξεργασία των μελών του αντίτυπου `magazine`. Η πρόσβαση των μελών αυτών μπορεί να γίνει με χρήση του τελεστή τελείας ως εξής:

`(*mag_ptr).`*μέλος*

Συνήθως όμως για το σκοπό αυτό χρησιμοποιούμε ένα άλλο τελεστή που ονομάζεται **τελεστής βέλους** (arrow operator). Η αναφορά τότε ενός μέλους της δομής γράφεται:

`mag_ptr->`*μέλος*

Όπως και με όλους τους άλλους δείκτες, έτσι και οι δείκτες δομών δε "δείχνουν" πουθενά συγκεκριμένα πριν λάβουν ως αρχική τιμή τη διεύθυνση του αντίτυπου μιας δομής, όπως δείχνει το επόμενο παράδειγμα:

```
1  /* Παράδειγμα χρήσης δεικτών σε δομές */
2
3  #include <stdio.h>
4
5  struct magazine_type
6  {
7      char title[20];              /* Τίτλος περιοδικού */
8      char publisher[40];          /* Εκδότης */
9      int month;                   /* Μήνας έκδοσης */
10     int year;                    /* Έτος έκδοσης */
11     int price;                   /* Τιμή τεύχους */
12 };
13 int main(void);
14 void input_data(struct magazine_type *ptr);
15 void print_data(struct magazine_type magazine);
16
17 int main(void)
18 {
19   struct magazine_type magazine;     /* Ορισμός αντίτυπου δομής */
20   struct magazine_type *mag_ptr;     /* Ορισμός δείκτη δομής */
21
```

```
22    mag_ptr = &magazine;                    /* Αρχική τιμή δείκτη δομής */
23    input_data(mag_ptr);                    /* Κλήση μέσω αναφοράς */
24    print_data(magazine);                   /* Κλήση μέσω τιμής */
25  }
26  void input_data(struct magazine_type *ptr)
27  {
28    printf("Τίτλος περιοδικού..:"); scanf("%s", ptr->title);
29    printf("Όνομα εκδότη ....:"); scanf("%s", ptr->publisher);
30    printf("Μήνας έκδοσης....:"); scanf("%d", &ptr->month);
31    printf("Έτος έκδοσης .....:"); scanf("%d", &ptr->year);
32    printf("Τιμή τεύχους......:"); scanf("%d", &ptr->price);
33    return;
34  }
35  void print_data(struct magazine_type magazine)
36  {
37    printf("\n");
38    printf("Τίτλος περιοδικού..: %s\n", magazine.title);
39    printf("Όνομα εκδότη ....: %s\n", magazine.publisher);
40    printf("Μήνας έκδοσης....: %d\n", magazine.month);
41    printf("Έτος έκδοσης .....: %d\n", magazine.year);
42    printf("Τιμή τεύχους......: %d\n", magazine.price);
43    return;
44  }
```

Προσέξτε ότι η δήλωση της δομής `magazine_type` (εντολές 5-12) προηγείται των προτύπων των συναρτήσεων (εντολές 13-15), επειδή ο υπολογιστής πρέπει να γνωρίζει όλους τους τύπους δεδομένων που παίρνουν μέρος στη δήλωση ενός προτύπου συνάρτησης. Αν η δήλωση της δομής τοποθετηθεί μετά από τη δήλωση των πρότυπων των συναρτήσεων, ο μεταφραστής αδυνατεί να μεταγλωττίσει σωστά το πρόγραμμα και σταματά εμφανίζοντας μήνυμα σφάλματος.

Το πρόγραμμα παρουσιάζει τους δύο τρόπους μεταβίβασης μιας δομής σε μια συνάρτηση. Στη συνάρτηση `input_data()` περνάμε το δείκτη της δομής `magazine_type` (εντολή 23) ενώ στη συνάρτηση `print_data()` περνάμε το όνομα του αντίτυπου `magazine` της δομής αυτής (εντολή 24). Σημειώστε ότι η κλήση της συνάρτησης `input_data()` θα μπορούσε να γραφεί και ώς εξής:

```
input_data(&magazine);
```

Αν και η ANSI C επιτρέπει μια δομή να μεταβιβαστεί σαν παράμετρος σε μια συνάρτηση (εντολή 24), η δυνατότητα αυτή δε χρησιμοποιείται συχνά επειδή οι δομές καταλαμβάνουν συνήθως μεγάλη έκταση μνήμης και γι' αυτό απαιτείται ανάλογα μεγάλο ποσό **πληροφοριών υποστήριξης** (overhead information) κατά τη μεταβίβασή τους σε άλλες συναρτήσεις. Έτσι, οι περισσότεροι προγραμματιστές προτιμούν να μεταβιβάζουν το δείκτη μιας δομής αντί την ίδια τη δομή, ώστε να επιταχύνουν τη διαδικασία κλήσης και επιστροφής από συνάρτηση. Η τακτική αυτή, εξασφαλίζει επίσης συμβατότητα με παλαιότερους μεταγλωττιστές της γλώσσας που δεν επιτρέπουν μεταβίβαση μιας δομής μέσω τιμής.

Γενικά λοιπόν είναι προτιμότερο να μεταβιβάζουμε μεγάλες δομές δεδομένων μέσω αναφοράς για να αποφεύγουμε την υπερφόρτωση του προγράμματος με ανούσιες διεργασίες "τακτοποίησης" των αδυναμιών του προγράμματος. Παρόμοια λογική ισχύει και για την επιστρεφόμενη τιμή. Ενώ δηλαδή η ANSI C επιτρέπει μια συνάρτηση να επιστρέφει μια τιμή τύπου δομής, συνήθως επιλέγουμε η τιμή μιας τέτοιας συνάρτησης να είναι ο δείκτης της δομής.

Θα κλείσουμε την παράγραφο με μια ενδιαφέρουσα παρατήρηση που αφορά την οργάνωση ενός συνόλου δεικτών σε πίνακες από δείκτες δομών. Όπως θα περίμενε κανείς, η γλώσσα C επιτρέπει τέτοιες κατασκευές που ορίζονται και χρησιμοποιούνται όπως και οι κανονικοί πίνακες δεικτών που περιγράψαμε στην παράγραφο 7.12. Ένας ορισμός πίνακα δεικτών σε δομές είναι ο ακόλουθος:

```
struct magazine_type *mag_ptr[5];
```

Στην εντολή αυτή ορίζουμε έναν πίνακα πέντε στοιχείων που το καθένα είναι δείκτης σ' ένα αντίτυπο της δομής `magazine_type`. Με την ίδια λογική μπορούμε να ορίσουμε ένα δείκτη στον πίνακα αυτό ή όπως λέμε ένα **δείκτη σε δείκτη δομής** (pointer to pointer to structure). Γράφουμε:

```
struct magazine_type **mag_pp;
```

Προφανώς οι κατασκευές αυτές μπορούν να φανούν χρήσιμες σε πολλές εξελιγμένες εφαρμογές, αλλά για το επίπεδο των γνώσεων που μας ενδιαφέρει εδώ θα περιοριστούμε στο να υπογραμμίσουμε την ευελιξία που προσφέρει η γλώσσα C στον ορισμό των πολύπλοκων αυτών δομών δεδομένων. Στην επόμενη παράγραφο περιγράφουμε μερικές βασικές εφαρμογές των δεικτών σε δομές δεδομένων.

9.8 Αυτοαναφερόμενες Δομές

Μια από τις πιο ενδιαφέρουσες ιδιότητες των δομών είναι η δυνατότητα ορισμού μελών που αποτελούν δείκτες άλλων δομών. Όταν οι δομές δεδομένων περιέχουν τέτοια μέλη ονομάζονται **δυναμικές** (dynamic structures) επειδή απαιτούν **δυναμική κατανομή** (dynamic allocation) της μνήμης του υπολογιστή ανάλογα με τις ανάγκες του προγράμματος. Η κατασκευή τους είναι σχετικά απλή: δεσμεύουμε ένα μέλος της δομής ώστε να είναι ο δείκτης του εαυτού της και λέμε τότε ότι δημιουργούμε μια **αυτοαναφερόμενη δομή** (self-referential structure). Για παράδειγμα μπορούμε να γράψουμε:

```
struct element
{
    int value;
    struct element *link;
};
```

Η δήλωση αυτή προσδιορίζει τη μορφή της δομής `element` η οποία αποτελείται από δύο μέλη: έναν ακέραιο αριθμό με το όνομα `value` και ένα δείκτη σ' ένα άλλο αντίτυπο της δομής `element`. Η κατασκευή που περιγράφει η δήλωση αυτή μπορεί να παρασταθεί με τον κρίκο μιας αλυσίδας, όπως δείχνει το σχήμα 9.3.