Α' Εξάμηνο
ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εργαστήριο – 9η εβδομάδα.
Κοζάνη, 2 Δεκεμβρίου 2008.

Δίνονται παραδείγματα που αποσαφηνίζουν και συμπληρώνουν όσα αναφέρθηκαν στο μάθημα σχετικά με τις δομές δεδομένων.

Παράδειγμα 9-1
Η δήλωση μιας νέας δομής δεδομένων μπορεί να περιλαμβάνει ταυτόχρονα και τη δήλωση μιας μεταβλητής αυτού του νέου τύπου που ορίζει η δομή. Επίσης, μια μεταβλητή που δηλώνεται ως δομή δεδομένων μπορεί να αρχικοποιηθεί συγχρόνως, αν γράψουμε τις τιμές των μελών της σε αγκύλες, όπως κάνουμε με τους πίνακες.
Στο παρακάτω παράδειγμα χρησιμοποιούνται και οι δύο αυτές δυνατότητες συγχρόνως σε μία εντολή. Ορίζεται μία δομή με το όνομα Book και συγχρόνως δηλώνεται μια μεταβλητή αυτού του τύπου με το όνομα my_book. Στα μέλη αυτής αποδίδονται αρχικές τιμές (“Το όνομα του ρόδου” κλπ).

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

int main(int argc, char **argv)
{
    struct Book {
        char *title;
        char *author;
        int pages;
        float price;
    } my_book = {"Το όνομα του ρόδου", "Umberto Eco", 500, 39.5};

    printf("Τίτλος: \"%s\"\nΣυγγραφέας: %s\nΣελίδες: %d\nΤιμή: %8.2f\n",
           my_book.title, my_book.author, my_book.pages, my_book.price);
    return 0;
}
```

Παράδειγμα 9-2
Αν δύο μεταβλητές μ1 και μ2 είναι ακριβώς του ίδιου τύπου struct, μπορούν να εξισωθούν (δηλαδή τα μέλη της μιας να γίνουν ίσα με τα αντίστοιχα της άλλης) αν γράψουμε απλά

μ1 = μ2;

Αυτό είναι το αντίθετο απ' ο,τι ισχύει για τους πίνακες όπου πρέπει να κάνουμε χρήση κάποιας δομής επανάληψης, όπως for, για να “διατρέξουμε” όλα τα στοιχεία των δύο πινάκων που θέλουμε να εξισώσουμε.
Από την άλλη, τίποτε δε μας εμποδίζει να εξισώσουμε ξεχωριστά ένα μέλος μιας μεταβλητής struct με το αντίστοιχο μιας άλλης, αν αυτό είναι που θέλουμε να κάνουμε. Τα παραπάνω απεικονίζονται

στο επόμενο παράδειγμα. Το πρώτο και το δεύτερο στοιχείο του πίνακα my_books τίθενται ίσα με το your_book. Αλλά στο δεύτερο στοιχείο, μόνο το μέλος author είναι ίδιο, οπότε διορθώνουμε τα υπόλοιπα.

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

int main(int argc, char **argv)
{
    struct Book {
        char *title;
        char *author;
        int pages;
        float price;
    } my_books[2], your_book = {"Το όνομα του ρόδου", "Umberto Eco", 500,
39.5};

    my_books[0] = your_book;
    my_books[1] = your_book;
    my_books[1].title = "Μπαουντολίνο";
    my_books[1].pages = 560;
    my_books[1].price = 45.0;

    printf("\nΤα βιβλία μου:\nΤίτλος: \"%s\"\nΣυγγραφέας: %s\nΣελίδες:
%d\nΤιμή: %8.2f\n",my_books[0].title, my_books[0].author,
my_books[0].pages, my_books[0].price);
    printf("\nΤίτλος: \"%s\"\nΣυγγραφέας: %s\nΣελίδες: %d\nΤιμή:
%8.2f\n", my_books[1].title, my_books[1].author, my_books[1].pages,
my_books[1].price);
    printf("\nΤο βιβλίο σου:\nΤίτλος: \"%s\"\nΣυγγραφέας: %s\nΣελίδες:
%d\nΤιμή: %8.2f\n",your_book.title, your_book.author, your_book.pages,
your_book.price);
    return 0;
}
```

Παράδειγμα 9-3

Οι τύποι των μελών μιας δομής μπορούν να είναι και αυτοί επίσης δομές struct. Τότε, μπορούμε αν χρειαστεί να έχουμε πρόσβαση και στα μέλη των μελών μιας δομής. Το επόμενο παράδειγμα ορίζει μια δομή για την αναπαράσταση ενός γεωμετρικού σημείου στο επίπεδο (άρα, με δύο συντεταγμένες) και μετά χρησιμοποιεί αυτή για να ορίσει μια νέα δομή για διανύσματα που περιέχει επίσης δύο μέλη τύπου “σημείο”. Μετά, ορίζεται ένα διάνυσμα και υπολογίζονται οι συνιστώσες και το μήκος του.

Σημειώστε ότι για την κλήση των συναρτήσεων στον υπολογισμό του μήκους (pow για την ύψωση σε δύναμη και sqrt για την τετραγωνική ρίζα) χρειάζεται και το αρχείο επικεφαλίδας math.h. Όταν χρησιμοποιείται το math.h, η μεταγλώττιση απαιτεί και την επιλογή -lm στη γραμμή εντολών:

```
% cc -lm paradeigma3.c
```

```
#include <stdio.h>
#include <math.h>

int main(int argc, char **argv)
{
    struct point { float x, y; };
    struct vector {
        struct point A, B;
        float dx, dy, length;
    };

    struct vector new_vector;

    new_vector.A.x = new_vector.A.y = 0.;
    new_vector.B.x = 3.;
    new_vector.B.y = 4.;

    new_vector.dx = new_vector.B.x - new_vector.A.x;
    new_vector.dy = new_vector.B.y - new_vector.A.y;

    new_vector.length = sqrt(pow(new_vector.dx,2) +
pow(new_vector.dy,2));

    printf("Ευθύγραμμο τμήμα AB\nΣυνιστώσες: ΔX = %6.2f, ΔY =
%6.2f\nΜήκος = %6.2f\n", new_vector.dx, new_vector.dy,
new_vector.length);
    return 0;
}
```

Παράδειγμα 9-4

Μπορούμε να ορίσουμε δείκτες σε μεταβλητές struct. Τότε, αντί για τον τελεστή “τελεία” για την πρόσβαση των μελών, χρησιμοποιούμε τον τελεστή “βέλος”, ->, σε συνδυασμό με το δείκτη στη μεταβλητή. Αυτό είναι ιδιαίτερα χρήσιμο όταν περνάμε μια δομή μέσω αναφοράς ή διεύθυνσης (πράγμα προτιμώτερο από το να περνάμε όλα τα bytes μιας δομής με πέρασμα μέσω τιμής).
Στο επόμενο παράδειγμα φαίνεται επίσης ότι οι δομές struct μπορούν (και συνηθίζεται) να δηλώνονται έξω και πριν από το σώμα των συναρτήσεων όπου χρησιμοποιούνται. Συχνά τοποθετούμε τις δηλώσεις αυτές, ενδεχομένως μαζί με πρωτότυπα συναρτήσεων, σε δικά μας αρχεία επικεφαλίδας που μετά θα κάνουμε include βάζοντας το όνομά του σε διπλά εισαγωγικά " " αντί για τις αγκύλες < και >. Εδώ τις έχουμε στο ίδιο αρχείο με το υπόλοιπο πρόγραμμα, αλλά στην αρχή του, πριν από τη main.
Το πρόγραμμα είναι το ίδιο με το προηγούμενο, μόνο που το “σπάσαμε” σε συναρτήσεις, την κύρια, μία για απόδοση συντεταγμένων και μία για τον υπολογισμό των συντεταγμένων και του μήκους του διανύσματος.
Παρατηρείστε ότι χρησιμοποιούμε τον τελεστή βέλος για την πρόσβαση των μελών της δομής vector αλλά τον τελεστή τελεία για τα μέλη της τύπου point. Αυτό είναι φυσικό, αφού αυτά τα μέλη δεν έχουν οριστεί ως δείκτες. Μόνο η δομή vector εμφανίζεται ως δείκτης όταν μια μεταβλητή αυτού του τύπου περνά σε συναρτήσεις μέσω αναφοράς – στο συγκεκριμένο παράδειγμα.

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

struct point { float x, y; };

struct vector {
    struct point A, B;
    float dx, dy, length;
};

void define_vector(struct vector*, float, float, float, float);
void calc_vector(struct vector*);

int main(int argc, char **argv)
{
    struct vector new_vector;

    define_vector(&new_vector, 0., 0., 3., 4.);
    calc_vector(&new_vector);

    printf("Ευθύγραμμο τμήμα AB\nΣυνιστώσες: ΔX = %6.2f, ΔY =
%6.2f\nΜήκος = %6.2f\n", new_vector.dx, new_vector.dy,
new_vector.length);

    return 0;
}

void define_vector(struct vector *vec, float ax, float ay, float bx,
float by)
{
    vec->A.x = ax;
    vec->A.y = ay;
    vec->B.x = bx;
    vec->B.y = by;
    return;
}

void calc_vector(struct vector* vec)
{
    vec->dx = vec->B.x - vec->A.x;
    vec->dy = vec->B.y - vec->A.y;
    vec->length = sqrt(pow(vec->dx,2) + pow(vec->dy,2));
    return;
}
```

Παράδειγμα 7-5

Όχι μόνο μπορούμε να ορίσουμε νέους, παράγωγους, τύπους με τη βοήθεια των δηλώσεων struct, αλλά και νέα ονόματα χρησιμοποιώντας το προσδιοριστικό typedef ως εξής:

typedef τύπος νέο_όνομα;
typedef struct όνομα1 όνομα2;

Μπορούμε να ορίσουμε νέα ονόματα ακόμη και για τους υπάρχοντες τύπους, π.χ.

typedef float real;

και μετά να χρησιμοποιούμε το νέο όνομα (εδώ: real) αντί για το παλιό (εδώ: float), αν και αυτό δεν επιβάλλεται ούτε συνηθίζεται.

Επίσης, μπορούμε να συνδυάσουμε έναν προσδιορισμό typedef με μια δήλωση struct, ως εξής:

typedef struct [όνομα1] { .... } όνομα2;

όπου βάλαμε σε άγκιστρα [και] το όνομα1 της struct για να δείξουμε ότι δεν είναι καν υποχρεωτικό.

Τα παραπάνω εφαρμόζονται τροποποιώντας το πρόγραμμα του προηγούμενου παραδείγματος ως εξής:

```
#include <stdio.h>
#include <math.h>

typedef struct { float x, y; } simio;
typedef struct {
    simio A, B;
    float dx, dy, length;
} dianysma;

void define_vector(dianysma*, float, float, float, float);
void calc_vector(dianysma*);

int main(int argc, char **argv)
{
    dianysma new_vector;

    define_vector(&new_vector, 0., 0., 3., 4.);
    calc_vector(&new_vector);


    printf("Ευθύγραμμο τμήμα AB\nΣυνιστώσες: ΔX = %6.2f, ΔY =
%6.2f\nΜήκος = %6.2f\n", new_vector.dx, new_vector.dy,
new_vector.length);

    return 0;
}

void define_vector(dianysma *vec, float ax, float ay, float bx, float by)
{
    vec->A.x = ax;
    vec->A.y = ay;
    vec->B.x = bx;
```

```
    vec->B.y = by;
    return;
}

void calc_vector(dianysma *vec)
{
    vec->dx = vec->B.x - vec->A.x;
    vec->dy = vec->B.y - vec->A.y;
    vec->length = sqrt(pow(vec->dx,2) + pow(vec->dy,2));
    return;
}
```

Παράδειγμα 7-6

Το πιο ενδιαφέρον με τις δηλώσεις struct είναι ότι μπορούμε να ορίσουμε παράγωγους τύπους που αναφέρονται, μέσω ενός μέλους-δείκτη, στον εαυτό τους. Τότε, μπορούμε στη συνέχεια να κατασκευάσουμε πιο πολύπλοκες δομές δεδομένων, αποτελούμενες από μεταβλητές που η μία "δείχνει" στην άλλη. Τέτοιες δομές είναι οι συνδεδεμένες λίστες, οι ουρές, οι στοίβες και τα δέντρα και μπορούν να είναι πολύ πιο ευέλικτες σε ο,τι αφορά την προσθήκη και αφαίρεση στοιχείων από τους πίνακες.

Αρχίζουμε την παρουσίαση αυτού του θέματος με ένα στοιχειώδες παράδειγμα ορισμού αυτοαναφορικής δομής. Βέβαια, αυτό το πρόγραμμα δεν κάνει ακόμη κάτι αξιόλογο, αλλά θα χρησιμεύσει ως σημείο εκκίνησης για να εισάγουμε σταδιακά πιο πολύπλοκες λειτουργίες.

```
#include <stdio.h>

struct POINT {
    float x, y;
    struct POINT *next;
};
typedef struct POINT simio;
void define_point(simio*, float, float);

int main(int argc, char **argv)
{
    simio point;

    define_point(&point, 0., 0.);

    return 0;
}

void define_point(simio *p, float x, float y)
{
    p->x = x;
    p->y = y;
    p->next = NULL;
    return;
```

```
}
```

Παράδειγμα 7-7

Για να χρησιμεύσουν οι αυτοαναφορικές δομές σε κάτι πρακτικό θα πρέπει να είμαστε σε θέση να εισάγουμε σε αυτές νέα στοιχεία κατά βούληση, φτιάχνοντας "αλυσίδες" στοιχείων, αλλά και να αφαιρούμε από αυτές όποια στοιχεία δε χρειαζόμαστε. Για να το πετύχουμε αυτό, χρησιμοποιούμε συναρτήσεις που επιτρέπουν τη δυναμική διαχείριση μνήμης, με τις οποίες δεσμεύουμε περιοχές μνήμης για κάθε νέο στοιχείο ή να τις απελευθερώνουμε όταν δε χρειάζονται. Οι συναρτήσεις αυτές ορίζονται στο αρχείο επικεφαλίδας stdlib.h και είναι οι εξής:

```
void *malloc (size_t size);
void *calloc (size_t elements, size_t size);
void *free(void *ptr);
```

Η malloc δεσμεύει συγκεκριμένο αριθμό από bytes (ίσο με size) και επιστρέφει έναν δείκτη void* στην αρχή της δεσμευμένης μνήμης. Δε δίνει αρχικές τιμές στα bytes που δεσμεύει. Η calloc είναι παρόμοια και δεσμεύει μνήμη αρκετή για elements στοιχεία μεγέθους size το καθένα. Δίνει μηδενική αρχική τιμή στα bytes που δεσμεύει. Η free αποδεσμεύει τη μνήμη που δεσμεύτηκε με malloc ή calloc και αποδόθηκε στο δείκτη ptr. Όταν ένα πρόγραμμα τελειώνει την εκτέλεσή του, η δεσμευμένη μνήμη επιστρέφεται στο σύστημα.

> Αξίζει να σημειωθεί ότι οι παραπάνω συναρτήσεις και ιδιαίτερα η calloc, μπορούν να χρησιμοποιηθούν και για την κατασκευή πινάκων όταν το μέγεθός τους δεν είναι γνωστό εκ των προτέρων αλλά εξαρτάται από την εξέλιξη των υπολογισμών, π.χ.
>
> ```
> int n;
> float *pinakas;
> ...
> pinakas = calloc(n, sizeof(float));
> ```
>
> Έτσι, αποφεύγουμε να ορίσουμε πίνακες με πολύ μεγάλη διάσταση για να αποφύγουμε υπέρβαση των ορίων τους και δε σπαταλάμε μνήμη που μπορεί να μη χρειάζεται στις περισσότερες περιπτώσεις.

Το επόμενο παράδειγμα κατασκευάζει μια λίστα από τρία σημεία και τη διατρέχει με μια επαναληπτική δομή for για να τυπώσει τα περιεχόμενά της. Παρατηρείστε τα εξής:
α) τη δέσμευση μνήμης με τη βοήθεια της malloc, που αποδίδεται στο δείκτη point, αλλά και στο first (οι δείκτες πρέπει να έχουν την τιμή μιας συγκεκριμένης διεύθυνσης εκτός NULL, για να χρησιμοποιηθούν, είτε με ανάθεση της διεύθυνσης μιας άλλης μεταβλητής είτε με malloc ή calloc),
β) πώς το μέλος next χρησιμοποιείται για να οριστεί ένα νέο μέλος της λίστας με την εντολή point = point->next (το point έχει οριστεί ως δείκτης!),
γ) πώς ο δείκτης point χρησιμοποιείται στη δομή for (έχουμε προνοήσει να αποθηκεύσουμε το πρώτο στοιχείο της λίστας στο first, καθώς και να θέσουμε ίσο με NULL το μέλος next του τελευταίου στοιχείου της λίστας ώστε να υπάρχουν συνθήκες έναρξης και τερματισμού),
δ) τη δέσμευση μνήμης που γίνεται στη συνάρτηση define_point με τη βοήθεια της malloc. Η μνήμη ανατίθεται στο μέλος next που έτσι, είναι έτοιμο να δεχτεί τα νέα δεδομένα.

```
#include <stdio.h>

struct POINT {
    float x, y;
    struct POINT *next;
};
typedef struct POINT simio;
void define_point(simio*, float, float);
```

```
int main(int argc, char **argv)
{
    simio *point, *first;

    first = point = (simio *) malloc(sizeof(simio));
    define_point(point, 0., 0.);
    point = point->next;
    define_point(point, 1., 0.);
    point = point->next;
    define_point(point, 1., 1.);
    point->next = NULL;

    for(point = first;  point != NULL; point = point->next)
        printf("Συντεταγμένες σημείου %d: x = %8.2f, y = %8.2f\n", ++n,
point->x, point->y);
    return 0;
}

void define_point(simio *p, float x, float y)
{
    p->x = x;
    p->y = y;
    p->next = (simio *) malloc(sizeof(simio));
    return;
}
```