Δ.Π.Θ Δομημένος Προγραμματισμός

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ & ΔΙΟΙΚΗΣΗΣ

Δομές και Ενώσεις

Dr. Αθανάσιος Μπαλαφούτης
Εργαστηριακό Διδακτικό Προσωπικό
Τομέας Συστημάτων Παραγωγής
Εργαστήριο Ρομποτικής και Αυτοματισμών
abalafou@pme.duth.gr
Γραφείο 304, τηλ.: 25410 – 79892

Ορισμός προβλήματος

Έστω ότι θέλουμε να διαχειριστούμε μια αποθήκη υλικών ενός εργοστασίου.

Για κάθε προϊόν της αποθήκης, θέλουμε αναλυτική καταγραφή των χαρακτηριστικών του.

Υλικά αποθήκης - Ρουλεμάν

Κωδικός Προϊόντος	A13-BG5
Περιγραφή	Ρουλεμάν
Υλικό	Ανοξείδωτο ατσάλι
Μέγιστο φορτίο	500 kg
Ταχύτητα περιστροφής	10.000 rpm
Βάρος	0.75 kg

Υλικά αποθήκης - Τροχαλία

Κωδικός Προϊόντος	T212-WW5
Περιγραφή	Τροχαλία
Υλικό	Χυτοσίδηρος
Μέγιστο φορτίο	600 kg
Ταχύτητα περιστροφής	6.500 rpm
Βάρος	2.3 kg

Υλικά αποθήκης - Άξονας

Κωδικός Προϊόντος	Z2Q2FF15
Περιγραφή	Άξονας
Υλικό	Χάλυβας
Μέγιστο φορτίο	1.500 kg
Ταχύτητα περιστροφής	8.000 rpm
Βάρος	5.8 kg

Υλικά αποθήκης - Μεταβλητές

Κωδικός Προϊόντος	A13-BG5
Περιγραφή	Ρουλεμάν
Υλικό	Ανοξείδωτο ατσάλι
Μέγιστο φορτίο	500 kg
Ταχύτητα περιστροφής	10.000 rpm
Βάρος	0.75 kg

```
char *id1 = "A13-BG5";
char *descr1 = "Ρουλεμάν";
char *material1 = "Ανοξείδωτο ατσάλι";
int max_load1 = 500;
int speed1 = 10000;
float weight1 = 0.75;
```

Υλικά αποθήκης - Μεταβλητές

Κωδικός Προϊόντος	A13-BG5
Περιγραφή	Ρουλεμάν
Υλικό	Ανοξείδωτο ατσάλι
Μέγιστο φορτίο	500 kg
Ταχύτητα περιστροφής	10.000 rpm
Βάρος	0.75 kg

```
char *id1 = "A13-BG5";
char *descr1 = "Ρουλεμάν";
char *material1 = "Ανοξείδωτο ατσάλι";
int max_load1 = 500;
int speed1 = 10000;
float weight1 = 0.75;
```

Κωδικός Προϊόντος	T212-WW5
Περιγραφή	Τροχαλία
Υλικό	Χυτοσίδηρος
Μέγιστο φορτίο	600 kg
Ταχύτητα περιστροφής	6.500 rpm
Βάρος	2.3 kg

```
char *id2 = "T212-WW5";
char *descr2 = "Τροχαλία";
char *material2 = "Χυτοσίδηρος";
int max_load2 = 600;
int speed2 = 6500;
float weight2 = 2.3;
```

Υλικά αποθήκης - Μεταβλητές

Για κάθε προϊόν θα χρειαστώ 6 μεταβλητές. Πως όμως θα διαχειριστώ χιλιάδες προϊόντα;

```
char *id1 = "A13-BG5";
char *descr1 = "Ρουλεμάν";
char *material1 = "Ανοξείδωτο ατσάλι";
int max_load1 = 500;
int speed1 = 10000;
float weight1 = 0.75;
```

```
char *id2 = "T212-WW5";
char *descr2 = "Τροχαλία";
char *material2 = "Χυτοσίδηρος";
int max_load2 = 600;
int speed2 = 6500;
float weight2 = 2.3;
```

Η χρήση πινάκων για την αποθήκευση των δεδομένων, θα ήταν καλή επιλογή;

Υλικά αποθήκης - Μεταβλητές

Για κάθε προϊόν θα χρειαστώ 6 μεταβλητές. Πως όμως θα διαχειριστώ χιλιάδες προϊόντα;

```
char *id1 = "A13-BG5";
char *descr1 = "Ρουλεμάν";
char *material1 = "Ανοξείδωτο ατσάλι";
int max_load1 = 500;
int speed1 = 10000;
float weight1 = 0.75;
```

```
char *id2 = "T212-WW5";
char *descr2 = "Τροχαλία";
char *material2 = "Χυτοσίδηρος";
int max_load2 = 600;
int speed2 = 6500;
float weight2 = 2.3;
```

Η χρήση πινάκων για την αποθήκευση των δεδομένων, θα ήταν καλή επιλογή;

Σε ένα πίνακα όλα τα δεδομένα πρέπει να ανήκουν στον ίδιο τύπο!

Δομές - 1η εκδοχή σύνταξης

Μια **Δομή (structure)** είναι ένας νέος τύπος δεδομένων που ορίζει ο προγραμματιστής, με σκοπό να συνδέσει δεδομένα διαφορετικού τύπου, σε μια ενιαία δομή.

```
struct{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
} part1, part2;
```

Δομές - 1η εκδοχή σύνταξης

Μια **Δομή (structure)** είναι ένας νέος τύπος δεδομένων που ορίζει ο προγραμματιστής, με σκοπό να συνδέσει δεδομένα διαφορετικού τύπου, σε μια ενιαία δομή.

```
struct{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
} part1, part2;
```

```
int main(){

   part1.id = "A13-BG5";
   part2.id = "T212-WW5";
   printf("%s \n", part1.id);
   printf("%s \n", part2.id);
   return 0;
}
```

Δομές - 2η εκδοχή σύνταξης

Μια **Δομή (structure)** είναι ένας νέος τύπος δεδομένων που ορίζει ο προγραμματιστής, με σκοπό να συνδέσει δεδομένα διαφορετικού τύπου, σε μια ενιαία δομή.

```
struct part{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
};
```

```
int main(){
   struct part p1;
   struct part p2;

   p1.id = "A13-BG5";
   p2.id = "T212-WW5";
   printf("%s \n", p1.id);
   printf("%s \n", p2.id);
   return 0;
}
```

Η εντολή typedef

typedef: Επιτρέπει στον προγραμματιστή να ορίσει έναν δικό του τύπο δεδομένων

Σύνταξη εντολής: **typedef υπάρχων_τύπος_δεδομένων νέος_τύπος_δεδομένων**

Παράδειγμα:

```c
#include <stdio.h>
typedef int INTEGER;

int main(){

   INTEGER var = 100;
   printf("%d \n", var);
   return 0;
}
```

Η εντολή typedef

Σύνταξη εντολής: **typedef υπάρχων_τύπος_δεδομένων νέος_τύπος_δεδομένων**

```
struct part{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
};

int main(){
   struct part p1;
   ...
}
```

```
typedef struct part{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
}part;

int main(){
   part p1;
   ...
}
```

Αρχικοποίηση Δομής - 1ος τρόπος


```c
#include <stdio.h>
typedef struct part{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
}part;

int main(){
   part p1 = {"A13-BG5", "Ρουλεμάν", "Ανοξείδωτο ατσάλι", 500, 10000, 0.75};
   printf("%s \n", p1.id);
   printf("%.2f\n", p1.weight);
   return 0;
}
```

Αρχικοποίηση Δομής - 2ος τρόπος


```c
#include <stdio.h>
typedef struct part{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
}part;
int main(){
      part p1 = {.descr = "Ρουλεμάν", .weight = 0.75, .speed = 10000, .id =
                "A13-BG5", .material = "Ανοξείδωτο ατσάλι", .max_load =  500};
   printf("%s \n", p1.id);
   printf("%.2f\n", p1.weight);
   return 0;
}
```

Αρχικοποίηση Δομής - 3ος τρόπος


```c
#include <stdio.h>

typedef struct part{
   char *id;
   char *descr;
   char *material;
   int max_load;
   int speed;
   float weight;
}part;
```

```c
int main(){
   part p1;
   p1.id = "A13-BG5";
   p1.descr = "Ρουλεμάν";
   p1.material = "Ανοξείδωτο ατσάλι";
   p1.max_load =  500;
   p1.speed = 10000;
   p1.weight = 0.75;

   printf("%s \n", p1.id);
   printf("%.2f\n", p1.weight);
   return 0;
}
```

Πίνακας με Δομές


```c
#include <stdio.h>

typedef struct part {
    char id[20];
    char descr[100];
    char material[50];
    int max_load;
    int speed;
    float weight;
} part;
```

```c
int main() {
    part p1[5];

    for (int i = 0; i < 5; i++) {
        printf("Για το υλικό %d δώσε το id: ", i + 1);
        scanf("%19s", p1[i].id);
        ...
    }

    printf("%s\n", p1[3].id);
    ...
    printf("%.2f\n", p1[3].weight);
    return 0;
}
```

Πίνακας με Δομές - malloc


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

typedef struct part {
    char *id;
    char *descr;
    char *material;
    int max_load;
    int speed;
    float weight;
} part;
```

```c
int main() {
    part p1[5];

    for (int i = 0; i < 5; i++) {
        p1[i].id = (char*)malloc(20 * sizeof(char));
        printf("Για το υλικό %d δώσε το id: ", i + 1);
        scanf("%19s", p1[i].id);
        ...
    }

    printf("%s\n", p1[3].id);
    ...
    printf("%.2f\n", p1[3].weight);
    return 0;
}
```

Δομές και Δείκτες


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

typedef struct part {
    char *id;
    char *descr;
    char *material;
    int max_load;
    int speed;
    float weight;
} part;
```

```c
int main(){
   part p1;
   p1.id = "A13-BG5";
   p1.descr = "Ρουλεμάν";
   p1.material = "Ανοξείδωτο ατσάλι";
   p1.max_load =  500;
   p1.speed = 10000;
   p1.weight = 0.75;

   part *ptr = &p1;
   printf("%s %s\n", (*ptr).id, ptr->id);
   printf("%.2f %.2f\n", (*ptr).weight, ptr->weight);
   return 0;
}
```

Δομές και ευθυγράμμιση μνήμης

Τι θα εκτυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

typedef struct part{
  char a;
  char b;
  int c;
}part;

int main(){
   part p1;
   printf("%lu\n", sizeof(p1));

   return 0;
}
```

Δομές και ευθυγράμμιση μνήμης

Τι θα εκτυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

typedef struct part{
  char a;
  char b;
  int c;
}part;

int main(){
   part p1;
   printf("%lu\n", sizeof(p1));

   return 0;
}
```

8

Δομές και ευθυγράμμιση μνήμης

Οι Δομές αποθηκεύονται σε συνεχόμενες θέσεις μνήμης

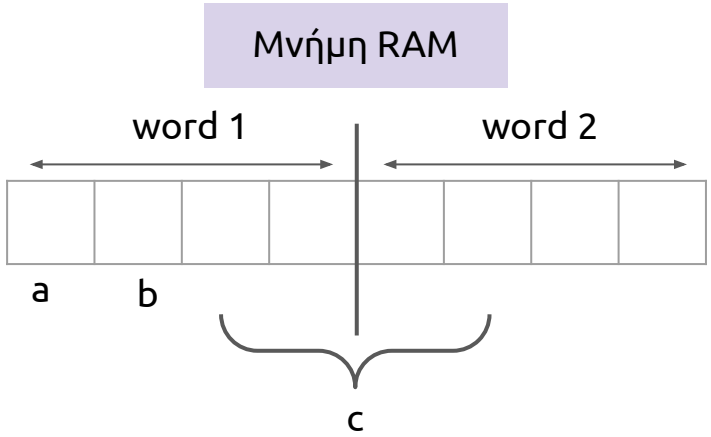
```c
#include <stdio.h>

typedef struct part{
  char a;
  char b;
  int c;
}part;
```

Ο επεξεργαστής 32 bit (CPU), σε κάθε κύκλο, δεν διαβάζει από την μνήμη 1 byte,
αλλά, μια **λέξη (word)** που αντιστοιχεί σε **4 bytes.**

Δομές και ευθυγράμμιση μνήμης

Οι Δομές αποθηκεύονται σε συνεχόμενες θέσεις μνήμης

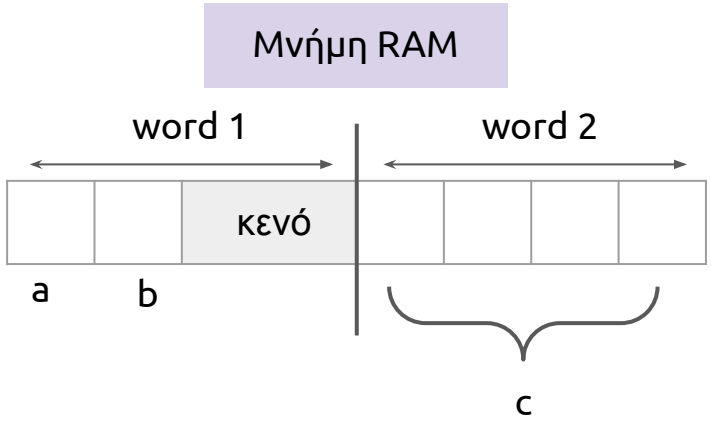
```c
#include <stdio.h>

typedef struct part{
  char a;
  char b;
  int c;
}part;
```



Δομές και ευθυγράμμιση μνήμης

Οι Δομές αποθηκεύονται σε συνεχόμενες θέσεις μνήμης

```c
#include <stdio.h>

typedef struct part{
  char a;
  char b;
  int c;
}part;
```



Συνολικά bytes στη μνήμη: **8**

Δομές και ευθυγράμμιση μνήμης

Τι θα συμβεί αν αλλάξω τη σειρά που δηλώνω τις μεταβλητές μέσα στη Δομή;

```c
#include <stdio.h>

typedef struct part{
  char a;
  int c;
  char b;
}part;

int main(){
   part p1;
   printf("%lu\n", sizeof(p1));

   return 0;
}
```

???

Δομές και ευθυγράμμιση μνήμης

Τι θα συμβεί αν αλλάξω τη σειρά που δηλώνω τις μεταβλητές μέσα στη Δομή;

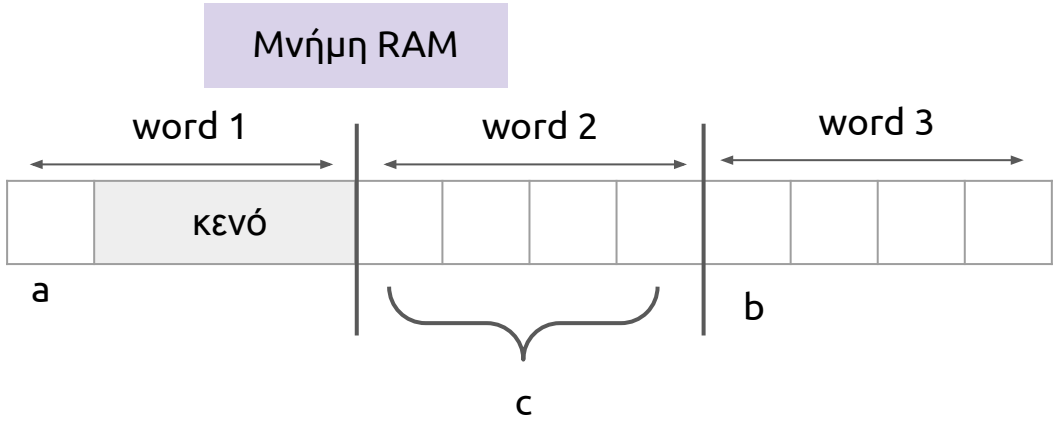
```c
#include <stdio.h>

typedef struct part{
  char a;
  int c;
  char b;
}part;

int main(){
   part p1;
   printf("%lu\n", sizeof(p1));

   return 0;
}
```

12

Δομές και ευθυγράμμιση μνήμης

Τι θα συμβεί αν αλλάξω τη σειρά που δηλώνω τις μεταβλητές μέσα στη Δομή;

```c
#include <stdio.h>

typedef struct part{
  char a;
  int c;
  char b;
}part;
```



Συνολικά bytes στη μνήμη: **12**

Παράδειγμα

Τι θα εκτυπώσει το παρακάτω πρόγραμμα:

```c
#include <stdio.h>

struct point{
  int x, y, z;
};

int main(){
   struct point p1 = {.y = 0, .z = 1, .x = 2};
   printf("%d %d %d", p1.x, p1.y, p1.z);

   return 0;
}
```

Παράδειγμα

Τι θα εκτυπώσει το παρακάτω πρόγραμμα:

```c
#include <stdio.h>

struct point{
  int x, y, z;
};

int main(){
   struct point p1 = {.y = 0, .z = 1, .x = 2};
   printf("%d %d %d", p1.x, p1.y, p1.z);

   return 0;
}
```

2 0 1

Ενώσεις (Unions)

Οι **Ενώσεις (Unions)** είναι και αυτός ένας τύπος δεδομένων που ορίζει ο προγραμματιστής.

Σε αντίθεση με τις δομές, τα μέλη (οι μεταβλητές) μιας ένωσης καταλαμβάνουν την ίδια διεύθυνση μνήμης

```c
#include <stdio.h>

union MyUnion {
    int a;     // 4 bytes
    char b;    // 1 byte
};
```

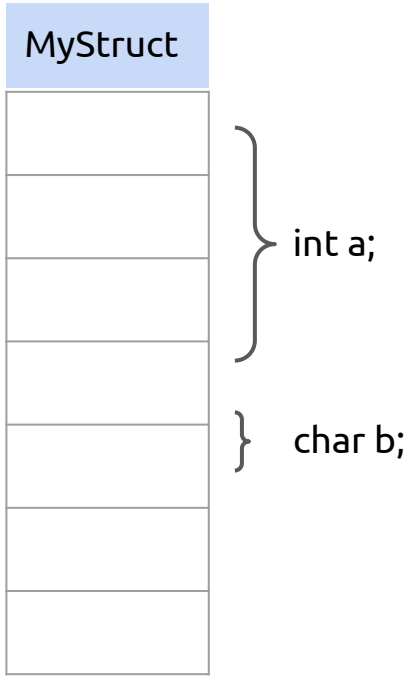
```c
int main() {
    union MyUnion u;

    printf("%p\n", &u.a);
    printf("%p\n", &u.b);

    return 0;
}
```

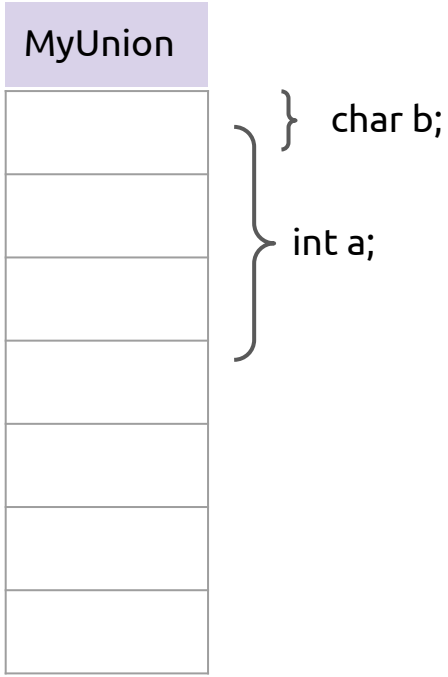
→ 0x7ffddfd4f01c

Ενώσεις (Unions)

Αναπαράσταση στη μνήμη RAM:

```
struct MyStruct {
   int a;
   char b;
};

union MyUnion {
   int a;
   char b;
};
```

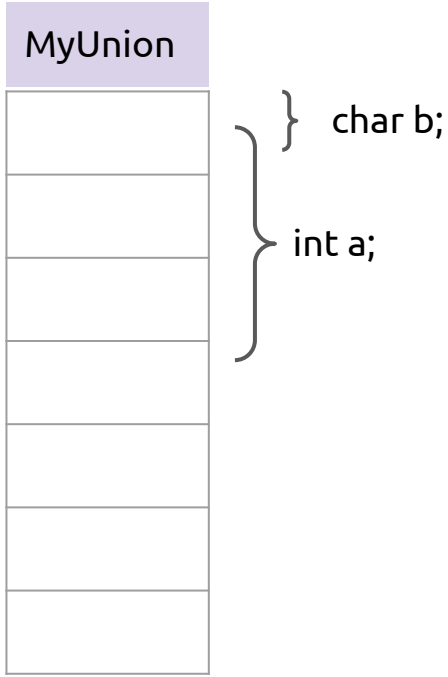




Ενώσεις (Unions)

Αφού τα μέλη μιας ένωσης μοιράζονται τον ίδιο χώρο στη μνήμη, η καταχώρηση μιας τιμής για το ένα μέλος θα επηρεάσει και το άλλο.

```
union MyUnion {
   int a;
   char b;
};
```



Ενώσεις (Unions)

Αφού τα μέλη μιας ένωσης μοιράζονται τον ίδιο χώρο στη μνήμη, η καταχώρηση μιας τιμής για το ένα μέλος θα επηρεάσει και το άλλο.

Παράδειγμα:

```c
#include <stdio.h>

union MyUnion {
   int a;
   char b;
};
```

```c
int main() {
   union MyUnion var;

   var.a = 65;

   printf("a = %d\n", var.a);
   printf("b = %c\n", var.b);

   return 0;
}
```

```
a = 65
b = A
```

Μέγεθος Ενώσης

Το μέγεθος μιας Ένωσης εξαρτάται από το μέγεθος του μεγαλύτερου μέλους της δομής.

Παράδειγμα:

```c
#include <stdio.h>

union MyUnion {
   int a;
   char b;
};

int main() {
   union MyUnion var;
   printf("%ld\n", sizeof(var));
   return 0;
}
```

4

Παράδειγμα - Υλικά Αποθήκης

Σε μια βιομηχανία, μπορεί να θέλω να καταγράψω υλικά με διαφορετικά χαρακτηριστικά

Παράδειγμα:

ID	A13-BG5
Χρώμα	Μαύρο
Βάρος	0.75 kg
Τιμή	15,17€

ID	D56-XT4
Ταχύτητα περιστροφής	6.500 rpm
Τάση Λειτουργίας	380 V
Τιμή	15.000€

Παράδειγμα - Υλικά Αποθήκης

Θα μπορούσα να χρησιμοποιήσω μια δομή για να περιγράψω την αποθήκη μου

Παράδειγμα:

ID	A13-BG5
Χρώμα	Μαύρο
Βάρος	0.75 kg
Τιμή	15,17€

ID	D56-XT4
Ταχύτητα περιστροφής	6.500 rpm
Τάση Λειτουργίας	380 V
Τιμή	15.000€

```
typedef struct materials{
   char *id;
   float price;
   char *color;
   float weight;
   int speed;
   int voltage;
}materials;
```

Παράδειγμα - Υλικά Αποθήκης

Αν εισάγω το υλικό “A13-BG5” μέσα στη δομή:

```c
typedef struct materials{
   char *id;
   float price;
   char *color;
   float weight;
   int speed;
   int voltage;
}materials;
```

```c
int main() {
   materials m1;

   m1.id = "A13-BG5";
   m1.color = "Μαύρο";
   m1.weight = 0.75;
   m1.price = 15.17;

   return 0;
}
```

Παράδειγμα - Υλικά Αποθήκης

Αν εισάγω το υλικό “A13-BG5” μέσα στη δομή:

```c
typedef struct materials{
   char *id;
   float price;
   char *color;
   float weight;
   int speed;
   int voltage;
}materials;
```

Δεν χρησιμοποιούνται.
Σπατάλη μνήμης!

```c
int main() {
   materials m1;

   m1.id = "A13-BG5";
   m1.color = "Μαύρο";
   m1.weight = 0.75;
   m1.price = 15.17;

   return 0;
}
```

Παράδειγμα - Υλικά Αποθήκης

Καλύτερη προσέγγιση:

```
typedef struct materials{
    char *id;
    float price;
    union {
        struct{
            char *color;
            float weight;
        } part;
        struct{
            int speed;
            int voltage;
        } engine;
    } item;
}materials;
```

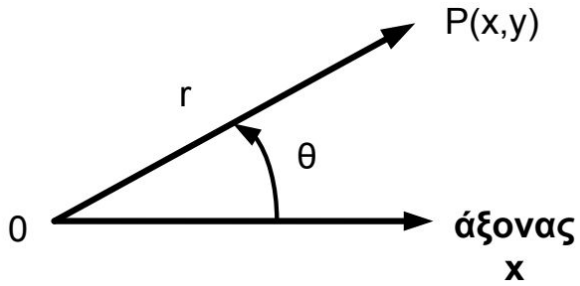
```
int main() {
  materials m1, e1;

  m1.id = "A13-BG5";
  m1.item.part.color = "Μαύρο";
  m1.item.part.weight = 0.75;
  m1.price = 15.17;

  e1.id = "D56-XT4";
  e1.item.engine.speed = 6500;
  e1.item.engine.voltage = 380;
  e1.price = 15000;

  return 0;
}
```

Άσκηση 2.7

Να γραφεί μια συνάρτηση με όνομα `convert_to_polar` που να δέχεται ως είσοδο τις καρτεσιανές συντεταγμένες ενός σημείου `x`, `y` και να επιστρέφει τις πολικές συντεταγμένες του `r`, `θ` . Οι πολικές συντεταγμένες υπολογίζονται σύμφωνα με τους τύπους :

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right)$$



Η συνάρτηση θα καλείται από τη main().

Άσκηση 2.7 - covvert_to_polar


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

// Συνάρτηση μετατροπής από καρτεσιανές σε πολικές συντεταγμένες
void convert_to_polar(double x, double y, double *r, double *theta) {
    *r = sqrt(x * x + y * y);       // Υπολογισμός της ακτίνας r
    *theta = atan2(y, x);           // Υπολογισμός της γωνίας θ σε ακτίνια
}
```

$$r = \sqrt{x^2 + y^2} \qquad \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

Άσκηση 2.7 - main


```c
int main() {
    double x, y, r, theta;

    printf("Εισάγετε τις καρτεσιανές συντεταγμένες (x, y): ");
    scanf("%lf %lf", &x, &y);

    convert_to_polar(x, y, &r, &theta);

    printf("Πολικές συντεταγμένες: r = %.2f, θ = %.2f μοίρες\n", r, theta * (180.0 / M_PI));

    return 0;
}
```

Άσκηση 2.7 - Λύση με χρήση Δομών


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

typedef struct cartesian {
   double x;
   double y;
}cartesian;

typedef struct polar {
   double r;
   double theta;
}polar;
```

Άσκηση 2.7 - convert_to_polar


```
polar convert_to_polar(cartesian point) {
   polar polar_point;
   polar_point.r = sqrt(point.x * point.x + point.y * point.y);
   polar_point.theta = atan2(point.y, point.x);

   return polar_point;
}
```

Άσκηση 2.7 - main


```
int main() {
    cartesian c_point;
    polar p_point;

    printf("Εισάγετε τις καρτεσιανές συντεταγμένες (x, y): ");
    scanf("%lf %lf", &c_point.x, &c_point.y);

    p_point =  convert_to_polar(c_point);

    printf("Πολικές συντεταγμένες: r = %.2f, θ = %.2f μοίρες\n", p_point.r, p_point.theta *
           (180.0 / M_PI));

    return 0;
}
```

Άσκηση 8.1

Το Σφαιρικό Σύστημα Συντεταγμένων καθορίζει τη θέση ενός σημείου του τρισδιάστατου επίπεδου χώρου με τη βοήθεια δύο γωνιών και μιας απόστασης από την αρχή *O*. Ο μετασχηματισμός από τις Σφαιρικές Συντεταγμένες (r, θ, φ) στις Καρτεσιανές Συντεταγμένες (x, y, z) πραγματοποιείται μέσω των τύπων:

$$x = r \sin\theta \cos\varphi, \quad y = r \sin\theta \sin\varphi, \quad z = r \cos\theta$$

Να ορίσετε μια δομή με όνομα `m_coords` για τη διαχείριση υλικών σημείων μάζας `m` ενός στερεού σώματος. Τα μέλη της δομής είναι η μάζα και οι τρεις σφαιρικές συντεταγμένες (r, θ, φ) όπως ορίστηκαν προηγουμένως. Όλες οι τιμές είναι τύπου `double`.

Άσκηση 8.1

Το πρόγραμμα θα πρέπει να περιλαμβάνει συναρτήσεις για:

1. την εισαγωγή των δεδομένων `n` υλικών σημείων (`n` = γνωστό) σε έναν πίνακα δομών τύπου `m_coords` (συνάρτηση `data_entry`). Να χρησιμοποιηθούν τυχαίοι αριθμοί τύπου double με όρια `50.0` για τη μάζα, `10.0` για την απόσταση και `2π` (σε ακτίνια) για τις γωνίες.
2. την εμφάνιση των στοιχείων του πίνακα δομών σε σειρές (1 σειρά για κάθε υλικό σημείο) (συνάρτηση `data_display`).
3. την εύρεση των καρτεσιανών συντεταγμένων του κέντρου μάζας του στερεού σώματος από όλα τα υλικά σημεία του πίνακα δομών, σύμφωνα με τους παρακάτω τύπους (μία συνάρτηση με όνομα `find_mass_center`):

$$x_{KM} = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i}, \quad y_{KM} = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i}, \quad z_{KM} = \frac{\sum_{i=1}^{n} m_i z_i}{\sum_{i=1}^{n} m_i}$$

Οι τρεις συναρτήσεις θα καλούνται διαδοχικά από τη `main()`. Μετά την κλήση της συνάρτησης `find_mass_center` το πρόγραμμα θα πρέπει να εμφανίζει τις τιμές για τα x_{KM}, y_{KM}, z_{KM}.

Άσκηση 8.1 - data_entry


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

#define n 5

typedef struct m_coords {
   double mass;
   double r;
   double theta;
   double phi;
} m_coords;
```

```c
void data_entry(m_coords coords[]) {
   srand(time(0));

   for (int i = 0; i < n; i++) {
       // Τυχαία μάζα από 1 έως 50
       coords[i].mass = (rand() % 50 + 1) * 1.0;
       // Τυχαία απόσταση από 0.1 έως 10
       coords[i].r = (rand() % 100 + 1) * 0.1;
       // Τυχαία γωνία θ (σε ακτίνια, από 0 έως 2π)
       coords[i].theta = (rand() % 6283) * 1.0 / 1000;
       // Τυχαία γωνία φ (σε ακτίνια, από 0 έως 2π)
       coords[i].phi = (rand() % 6283) * 1.0 / 1000;
   }
}
```

Άσκηση 8.1 - data_display


```
void data_display(m_coords coords[]) {
   printf("Μάζα (kg), Απόσταση (m), Θ (rad), Φ (rad):\n");
   for (int i = 0; i < n; i++) {
       printf("%3.2f, %3.2f, %3.2f, %3.2f\n", coords[i].mass, coords[i].r,
              coords[i].theta, coords[i].phi);
   }
}
```

Άσκηση 8.1 - find_mass_center

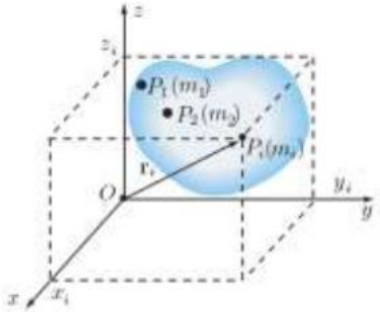

```
void find_mass_center(m_coords coords[], double *xKM, double *yKM, double *zKM) {
   double total_mass = 0.0;
   double weighted_x = 0.0;
   double weighted_y = 0.0;
   double weighted_z = 0.0;
   for (int i = 0; i < n; i++) {

       double x = coords[i].r * sin(coords[i].theta) * cos(coords[i].phi);
       double y = coords[i].r * sin(coords[i].theta) * sin(coords[i].phi);
       double z = coords[i].r * cos(coords[i].theta);

       total_mass += coords[i].mass;
       weighted_x += coords[i].mass * x;
       weighted_y += coords[i].mass * y;
       weighted_z += coords[i].mass * z;
   }
   *xKM = weighted_x / total_mass;
   *yKM = weighted_y / total_mass;
   *zKM = weighted_z / total_mass;
}
```

Άσκηση 8.1 - main


```c
int main() {
    m_coords coords[n];
    double xKM, yKM, zKM;

    data_entry(coords);
    data_display(coords);
    find_mass_center(coords, &xKM, &yKM, &zKM);

    printf("Κέντρο Μάζας:\n");
    printf("xKM: %.2f, yKM: %.2f, zKM: %.2f\n", xKM, yKM, zKM);

    return 0;
}
```

Άσκηση 8.2

Να ορίσετε μία δομή με όνομα `mass_point` που θα χρησιμοποιηθεί για τη διαχείριση υλικών σημείων ενός στερεού σώματος. Τα μέλη της δομής είναι η μάζα του υλικού σημείου και οι τρεις συντεταγμένες του στο χώρο, όπως φαίνεται στο σχήμα (απεικονίζονται τα υλικά σημεία P_1, P_2, ..., P_i με μάζες αντίστοιχα m_1, m_2, ..., m_i). Όλες οι τιμές είναι τύπου double.



$$I_{xOy} = \sum_i m_i z_i^2 \quad , I_{yOz} = \sum_i m_i x_i^2 \quad , I_{zOx} = \sum_i m_i y_i^2$$

Άσκηση 8.2

Το πρόγραμμα θα πρέπει να περιλαμβάνει συναρτήσεις για:

1. την εισαγωγή των δεδομένων `n` υλικών σημείων (`n` = γνωστό) σε έναν πίνακα δομών τύπου `mass_point` (συνάρτηση `data_entry`). Να χρησιμοποιηθούν τυχαίοι αριθμοί με όρια `100.0` για τη μάζα και `30.0` για κάθε μία από τις συντεταγμένες.
2. την εμφάνιση των στοιχείων του πίνακα δομών σε σειρές (1 σειρά για κάθε υλικό σημείο) (συνάρτηση `data_display`).
3. την εύρεση των ροπών αδρανείας I_{xOy}, I_{yOz}, I_{zOx} (σε μία συνάρτηση με όνομα `find_inertia`) λαμβάνοντας υπόψη όλα τα στοιχεία του πίνακα δομών.

Οι τρεις συναρτήσεις θα καλούνται διαδοχικά από τη `main()`. Μετά την κλήση της συνάρτησης `find_inertia` το πρόγραμμα θα πρέπει να εμφανίζει τις τιμές για τα I_{xOy}, I_{yOz}, I_{zOx}.

Άσκηση 8.2 - Δομή


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

#define N 5   // Πλήθος υλικών σημείων
#define MASS_LIMIT 100.0
#define COORD_LIMIT 30.0

// Δομή για το υλικό σημείο
typedef struct mass_point {
   double mass;
   double x;
   double y;
   double z;
}mass_point;
```

Άσκηση 8.2 - data_entry


```
void data_entry(mass_point points[], int n) {

   for (int i = 0; i < n; i++) {
       points[i].mass = ((double)rand() / RAND_MAX) * MASS_LIMIT;
       points[i].x = ((double)rand() / RAND_MAX) * COORD_LIMIT;
       points[i].y = ((double)rand() / RAND_MAX) * COORD_LIMIT;
       points[i].z = ((double)rand() / RAND_MAX) * COORD_LIMIT;
   }

}
```

Άσκηση 8.2 - data_display


```
void data_display(mass_point points[], int n) {
   printf("Στοιχεία υλικών σημείων:\n");

   for (int i = 0; i < n; i++) {
       printf("%5.2f %7.2f %7.2f %7.2f\n",
              points[i].mass,
              points[i].x,
              points[i].y,
              points[i].z);
   }

}
```

Άσκηση 8.2 - find_inertia


```
void find_inertia(mass_point points[], int n, double* IxOy, double* IyOz, double* IzOx)
{
    *IxOy = 0.0;
    *IyOz = 0.0;
    *IzOx = 0.0;

    for (int i = 0; i < n; i++) {
        *IxOy += points[i].mass * pow(points[i].z, 2); // z^2
        *IyOz += points[i].mass * pow(points[i].x, 2); // x^2
        *IzOx += points[i].mass * pow(points[i].y, 2); // y^2
    }
}
```

Άσκηση 8.2 - main


```
int main() {
    mass_point points[N];
    double IxOy, IyOz, IzOx;

    srand(time(0));

    data_entry(points, N);
    data_display(points, N);
    find_inertia(points, N, &IxOy, &IyOz, &IzOx);

    printf("Ροπές αδράνειας:\n");
    printf("IxOy = %.2f\n", IxOy);
    printf("IyOz = %.2f\n", IyOz);
    printf("IzOx = %.2f\n", IzOx);
    return 0;
}
```

Άσκηση 8.3

Το κέντρο βάρους (ή κεντροειδές) ενός στοιχειώδους επίπεδου σχήματος *i*, έχει συντεταγμένες x_i, y_i και εμβαδόν A_i.

Για τον υπολογισμό του κεντροειδούς μιας διατομής (δηλαδή των $\bar{x}$, $\bar{y}$) απαιτείται ο υπολογισμός των $\bar{x}$, $\bar{y}$ από όλα τα στοιχειώδη επίπεδα σχήματα που συνθέτουν τη διατομή, σύμφωνα με τους παρακάτω τύπους:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i A_i}{\sum_{i=1}^{n} A_i} \quad , \quad \bar{y} = \frac{\sum_{i=1}^{n} y_i A_i}{\sum_{i=1}^{n} A_i}$$

Άσκηση 8.3

Να οριστεί μια δομή (struct) με όνομα `centroid` για τη διαχείριση στοιχειωδών επίπεδων σχημάτων (τύπου `centroid`) που ανήκουν όλα στην ίδια διατομή. Τα μέλη της δομής είναι οι συντεταγμένες και το εμβαδόν, σύμφωνα με τα παραπάνω. Όλες οι τιμές είναι τύπου `double`.
Το πρόγραμμα θα πρέπει να περιλαμβάνει συναρτήσεις για:

1. την εισαγωγή των δεδομένων `n` στοιχειωδών επίπεδων σχημάτων (`n` = γνωστό) σε έναν πίνακα δομών τύπου `centroid` (συνάρτηση `data_entry`). Να χρησιμοποιηθούν τυχαίοι αριθμοί με όρια `50` για τις συντεταγμένες και `2000` για το εμβαδόν.
2. την εμφάνιση των στοιχείων του πίνακα δομών σε σειρές (1 σειρά για κάθε στοιχειώδες επίπεδο σχήμα) (συνάρτηση `data_display`).
3. την εύρεση των $\bar{x}$, $\bar{y}$ της διατομής (μία συνάρτηση με όνομα `find_centroid`), λαμβάνοντας υπόψη όλα τα στοιχεία του πίνακα δομών.

Οι τρεις συναρτήσεις θα καλούνται διαδοχικά από τη `main()`. Μετά την κλήση της συνάρτησης `find_centroid` το πρόγραμμα θα πρέπει να εμφανίζει τις τιμές για τα $\bar{x}$, $\bar{y}$.

Άσκηση 8.3 - Δομή


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

#define N 5   // Πλήθος στοιχειωδών σχημάτων
#define COORD_LIMIT 50.0
#define AREA_LIMIT 2000.0

typedef struct centroid {
   double x;
   double y;
   double area;
}centroid;
```

Άσκηση 8.3 - data_entry


```
void data_entry(centroid shapes[], int n) {

   for (int i = 0; i < n; i++) {
       shapes[i].x = ((double)rand() / RAND_MAX) * COORD_LIMIT;
       shapes[i].y = ((double)rand() / RAND_MAX) * COORD_LIMIT;
       shapes[i].area = ((double)rand() / RAND_MAX) * AREA_LIMIT;
   }

}
```

Άσκηση 8.3 - data_display


```
void data_display(centroid shapes[], int n) {

   printf("Στοιχεία στοιχειωδών επίπεδων σχημάτων:\n");
   for (int i = 0; i < n; i++) {
       printf("%7.2f %7.2f %7.2f \n",
              shapes[i].x,
              shapes[i].y,
              shapes[i].area);
   }
}
```

Άσκηση 8.3 - find_inertia


```
void find_inertia(mass_point points[], int n, double* IxOy, double* IyOz,
                  double* IzOx) {
   *IxOy = 0.0;
   *IyOz = 0.0;
   *IzOx = 0.0;

   for (int i = 0; i < n; i++) {
       *IxOy += points[i].mass * pow(points[i].z, 2); // z^2
       *IyOz += points[i].mass * pow(points[i].x, 2); // x^2
       *IzOx += points[i].mass * pow(points[i].y, 2); // y^2
   }
}
```

Άσκηση 8.3 - find_centroid


```
void find_centroid(centroid shapes[], int n, double* x_bar, double* y_bar) {

   double sum_area = 0.0;
   double sum_xA = 0.0;
   double sum_yA = 0.0;
   for (int i = 0; i < n; i++) {
       sum_area += shapes[i].area;
       sum_xA += shapes[i].x * shapes[i].area;
       sum_yA += shapes[i].y * shapes[i].area;
   }
   if (sum_area != 0) {
       *x_bar = sum_xA / sum_area;
       *y_bar = sum_yA / sum_area;
   } else {
       *x_bar = 0;
       *y_bar = 0;
   }
}
```

Άσκηση 8.3 - main


```c
int main() {
    centroid shapes[N];
    double x_bar, y_bar;

    srand(time(0));

    data_entry(shapes, N);
    data_display(shapes, N);
    find_centroid(shapes, N, &x_bar, &y_bar);

    printf("Κέντρο βάρους διατομής:\n");
    printf("x̄ = %.2f\n", x_bar);
    printf("ȳ = %.2f\n", y_bar);

    return 0;
}
```