ΤΕΙ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ
(ΓΡΕΒΕΝΑ)

Ασκήσεις εργαστηρίου *C*

Εργαστήριο: Προγραμματισμός Η/Υ

Δρ. Δημήτριος Συνδουκάς
Επίκ. Καθηγητής

ΓΡΕΒΕΝΑ 2012-2015

Άσκηση 1. Ένα απλό πρόγραμμα.

Το παρακάτω πρόγραμμα δίνει την τιμή **20** σε μια μεταβλητή **ακέραιου τύπου** με το όνομα: **age** και στη συνέχεια τυπώνει την τιμή της age στην οθόνη με το αντίστοιχο μήνυμα.

```
#include <stdio.h>
/* Program No 1 */

void main() {
  int age;

  age = 20;
  printf("My age is %d years", age);
}
```

Σχόλιο

• Εάν τρέξουμε το πρόγραμμα στον υπολογιστή μας θα δούμε στην οθόνη το παρακάτω μήνυμα:

```
My age is 20 years
```

Άσκηση 2. Πρόγραμμα μετατροπής ποδιών σε μέτρα.

Να γραφεί ένα πρόγραμμα που:
α) Να τυπώνει στη οθόνη το μήνυμα: DOSE TO MHKOS SE PODIA:
β) Να διαβάζει τον αριθμό των ποδιών (που θα είναι **δεκαδικός**) από το πληκτρολόγιο,
γ) Να κάνει τη μετατροπή από πόδια σε μέτρα και
δ) Να τυπώνει το μήνυμα:
TA *αριθμός ποδιών* PODIA ISOUNTAI ME *αριθμός μέτρων* METRA.
→ Δίνεται ο τύπος μετατροπής: **μέτρα = πόδια x 0.3048**

```
#include <stdio.h>

void main() {
  float podia, metra;

  printf("DOSE TO MHKOS SE PODIA:");
  scanf("%f",&podia);
  metra = podia*0.3048;
  printf("TA %f PODIA ISOYNTAI ME %f METRA",podia,metra);
}
```

Σχόλια: Προσέξτε ότι επιλέγω να χρησιμοποιήσω μεταβλητές τύπου **float** στο πρόγραμμά μου.
• Όταν τρέξω το πρόγραμμα, θα δω τα παρακάτω στην οθόνη. (Ότι είναι σημειωμένο με έντονα γράμματα, πληκτρολογείται από τον χρήστη).

```
DOSE TO MHKOS SE PODIA:3150.5
TA 3150.500000 PODIA ISOYNTAI ME 960.272400 METRA
```

Άσκηση 3. Να ξαναγραφεί το πρόγραμμα 2, όμως η σταθερά **0.3048** να ονομαστεί **PAR** με χρήση της **define**.

```
#include <stdio.h>
#define PAR 0.3048

void main() {
  float podia, metra;

  printf("DOSE TO MHKOS SE PODIA:");
  scanf("%f",&podia);
  metra = podia*PAR;
  printf("TA %f PODIA ISOYNTAI ME %f METRA",podia,metra);
}
```

Σχόλια: Οι αλλαγές που έχουν γίνει στο πρόγραμμα 2 σημειώνονται με έντονα γράμματα. Εάν τρέξουμε το πρόγραμμα, στην οθόνη μας θα δούμε ότι και στο πρόγραμμα 2.

Άσκηση 4. Πρόγραμμα υπολογισμού του **πηλίκου** και του **υπόλοιπου** της διαίρεσης δύο ακεραίων αριθμών.

Να γραφεί ένα πρόγραμμα που να τυπώνει στην οθόνη το μήνυμα: DOSE TON DIAIRETEO KAI TON DIAIRETH:, να διαβάζει από το πληκτρολόγιο τους δύο αυτούς αριθμούς και να υπολογίζει και να τυπώνει το πηλίκο και το υπόλοιπο της διαίρεσής τους με τα αντίστοιχα μηνύματα.

```
#include <stdio.h>

void main() {
  int d1, d2, pil, ypol;

  printf("DOSE TON DIAIRETEO KAI TON DIAIRETH:");
  scanf("%d %d",&d1,&d2);
  pil = d1/d2;
  ypol = d1%d2;
  printf("TO PHLIKO EINAI= %d \n", pil);
  printf("TO YPOLOIPO EINAI= %d", ypol);
}
```

Σχόλια: Εδώ επιλέγω να χρησιμοποιήσω μεταβλητές τύπου **int**, γιατί ζητάω το ακέραιο πηλίκο. Προσέξτε ότι ο τελεστής **%** επιτρέπεται να χρησιμοποιηθεί μόνο σε ακέραιους αριθμούς για να δώσει το υπόλοιπο της διαίρεσής τους.
Ο συνδυασμός των χαρακτήρων **\n** στην 2η `printf()` αναγκάζει τον υπολογιστή να αλλάξει γραμμή, οπότε το επόμενο μήνυμα (της 3ης `printf()`) θα τυπωθεί στην επόμενη γραμμή της οθόνης.
• Εάν τρέξουμε το πρόγραμμα, θα δούμε τα παρακάτω στην οθόνη μας. (Με έντονα γράμματα φαίνεται ότι πληκτρολόγησε ο χρήστης).

```
DOSE TON DIAIRETEO KAI TON DIAIRETH:19 4
TO PHLIKO EINAI= 4
TO YPOLOIPO EINAI= 3
```

Άσκηση 5. Χρήση απλής **if – else**. Πρόγραμμα διαίρεσης δύο δεκαδικών αριθμών (*κινητής υποδιαστολής απλής ακρίβειας*), με έλεγχο για αδύνατη διαίρεση.

Να γραφεί ένα πρόγραμμα που:

- να τυπώνει στην οθόνη το μήνυμα: `DOSE TON DIAIRETEO KAI TON DIAIRETH:`
- να διαβάζει τους δύο αριθμούς από το πληκτρολόγιο
- **να ελέγχει** εάν ο διαιρέτης είναι ίσος με το μηδέν
  - **οπότε** και να τυπώνει το μήνυμα ότι η διαίρεση είναι αδύνατη
  - **αλλιώς** να τυπώνει το αποτέλεσμα της διαίρεσης με κατάλληλο μήνυμα και 4 δεκαδικά ψηφία.

```
#include <stdio.h>

void main() {
  float d1, d2, ap;

  printf("DOSE TON DIAIRETEO KAI TON DIAIRETH:");
  scanf("%f %f", &d1, &d2);

  if(d2==0)
    printf("H DIAIRESH EINAI ADYNATH!\n");
  else {
    ap = d1/d2;
    printf("TO PHLIKO ISOYTAI ME: %.4f\n", ap);
  }
}
```

Σχόλια:

- Πότε χρησιμοποιούμε άγκιστρα σε μια εντολή `if`; Όταν σε μια εντολή `if`, `else if` ή `else` αντιστοιχεί μια μόνο εντολή δεν είναι υποχρεωτικό να την κλείσουμε σε άγκιστρα. Όταν όμως αντιστοιχούν περισσότερες από μια εντολές πρέπει υποχρεωτικά να κλειστούν σε άγκιστρα. Άρα, στο παραπάνω πρόγραμμα: εάν ισχύει η συνθήκη του `if` θα εκτελεστεί **μια μόνο** εντολή (η `printf` που τυπώνει το μήνυμα λάθους) άρα δεν χρειάζεται να κλειστεί αυτή η εντολή σε άγκιστρα. Εάν όμως δεν ισχύει η συνθήκη, θα εκτελεστούν οι εντολές που αντιστοιχούν στο `else` και επειδή αυτές είναι δύο **πρέπει να κλειστούν σε άγκιστρα.**

- Εάν τρέξουμε το πρόγραμμα και δώσουμε διαιρέτη διάφορο το μηδενός, θα δούμε κάτι παρόμοιο στην οθόνη μας:

```
DOSE TON DIAIRETEO KAI TON DIAIRETH:25.578 4.24
TO PHLIKO ISOYTAI ME: 6.0325
```

Ο χρήστης πληκτρολογεί τους αριθμούς.

- Εάν όμως δώσουμε διαιρέτη ίσο με μηδέν, θα πάρουμε το ακόλουθο μήνυμα λάθους:

```
DOSE TON DIAIRETEO KAI TON DIAIRETH:95.58 0
H DIAIRESH EINAI ADYNATH!
```

Ο χρήστης πληκτρολογεί τους αριθμούς.

- Μπορώ να μην χρησιμοποιήσω την μεταβλητή `ap` για να αποθηκεύσω το αποτέλεσμα της διαίρεσης και ακολούθως να το τυπώσω, αλλά να τυπώσω κατευθείαν το πηλίκο `d1/d2`. Μ' αυτό τον τρόπο **α)** χρησιμοποιώ λιγότερες μεταβλητές (αφού δεν χρειάζομαι πια την `ap`) και **β)** Η εντολή `ap=d1/d2` παραλείπεται, και αφού στην `else` αντιστοιχεί μια μόνο εντολή, παραλείπω και τα άγκιστρα. Το πρόγραμμα γίνεται σ' αυτήν την περίπτωση έτσι:

```
#include <stdio.h>
void main() {
 float d1, d2;

 printf("DOSE TON DIAIRETEO KAI TON DIAIRETH:");
 scanf("%f %f", &d1, &d2);

 if(d2==0)
   printf("H DIAIRESH EINAI ADYNATH!\n");
 else
   printf("TO PHLIKO ISOYTAI ME: %.4f\n", d1/d2);
}
```

Άσκηση 6. Η εντολή **if – else if – else**. Λύση της πρωτοβάθμιας εξίσωσης: **ax+b=0**

Να γραφεί ένα πρόγραμμα που να λύνει την ανωτέρω πρωτοβάθμια εξίσωση χρησιμοποιώντας την εντολή **if**. Ο υπολογιστής να τυπώνει μήνυμα στην οθόνη όπου να ζητά τους δύο γνωστούς, δηλαδή τα `a` και `b` και (αφού κάνει τις απαραίτητες πράξεις) να τυπώνει τη λύση, δηλαδή την τιμή του `x`, με 5 δεκαδικά ψηφία. Δίνονται οι περιπτώσεις από τη θεωρία των μαθηματικών:

α) Αν `a=0` και `b=0` τότε έχουμε *άπειρες λύσεις*

β) Αλλιώς αν μόνο `a=0` τότε *η λύση είναι αδύνατη*

γ) Αλλιώς, η λύση είναι `x=-b/a`

Οι ακολουθίες διαφυγής, όπως το **\n**, μπορούν να μπουν οπουδήποτε ανάμεσα στα εισαγωγικά της `printf`.

```
#include <stdio.h>
void main() {
  float a, b, x;

  printf("LYSH EKSISOSHS: ax+b=0\nDOSE TA a KAI b:");
  scanf("%f %f", &a, &b);

  if((a==0)&&(b==0))
    printf("YPARXOYN APEIRES LYSEIS");
  else if(a==0)
    printf("H LYSH EINAI ADYNATH");
  else {
    x=-b/a;
    printf("H LYSH EINAI: %.5f", x);
  }
}
```

Εδώ τα άγκιστρα δεν είναι απαραίτητα …

… όμως, προσοχή!! Τα άγκιστρα εδώ είναι απαραίτητα!

- Σχόλια: Εάν τρέξουμε το πρόγραμμα θα δούμε τα εξής:

```
LYSH EKSISOSHS: ax+b=0
DOSE TA a KAI b:3.56 -6.89
H LYSH EINAI: 1.93539
```

Ο χρήστης πληκτρολογεί τους αριθμούς.

Άσκηση 7. Η εντολή **switch**. Πρόγραμμα με ***μενού επιλογής*** για πρόσθεση ή αφαίρεση δύο ακέραιων αριθμών.

- Να εμφανίζεται στην οθόνη ένα αριθμημένο μενού απ' το οποίο να μπορεί να επιλέξει ο χρήστης:
  1. PROSTHESH 2 AKERAION ARITHMON
  2. AFAIRESH 2 AKERAION ARITHMON
  TI EPILEGEIS:
- Ο υπολογιστής να διαβάζει την επιλογή του χρήστη (δηλαδή έναν αριθμό που θα αντιστοιχεί σε κάποια επιλογή του μενού – στην περίπτωσή μας το 1 ή το 2) και στη συνέχεια,
- χρησιμοποιώντας την εντολή **switch**, να ενεργεί ως εξής για κάθε περίπτωση (την 1 και την 2) χωριστά:
  - να ***ζητά*** τους δύο ακεραίους που πρόκειται να προσθέσει ή αφαιρέσει,
  - τους οποίους αφού ***διαβάσει*** από το πληκτρολόγιο,
  - να χρησιμοποιεί για ***να κάνει την πράξη*** (πρόσθεση ή αφαίρεση) και
  - να ***τυπώνει*** το αποτέλεσμα, με κατάλληλο μήνυμα.
- Εφόσον ο χρήστης δεν δώσει σωστή επιλογή (πρόσθεσης ή αφαίρεσης), δηλαδή ο αριθμός που θα δώσει δεν είναι 1 ή 2, να τυπώνεται στην οθόνη το μήνυμα: LATHOS EPILOGH.

```
#include <stdio.h>
void main() {
 int a, b, ep, athroisma, diafora;

 printf("1. PROSTHESH 2 AKERAION ARITHMON\n");
 printf("2. AFAIRESH 2 AKERAION ARITHMON\n");
 printf("TI EPILEGEIS:");
 scanf("%d", &ep);

 switch(ep) {
   case 1:
     printf("DOSE PROSTHETEO KAI PROSTHETH:");
     scanf("%d %d", &a, &b);
     athroisma = a + b;
     printf("TO ATHROISMA EINAI: %d", athroisma);
     break;
   case 2:
     printf("DOSE AFAIRETEO KAI AFAIRETH:");
     scanf("%d %d", &a, &b);
     diafora = a - b;
     printf("H DIAFORA EINAI: %d", diafora);
     break;
   default:
```

```
    printf("LATHOS EPILOGH");
 }
}
```

Εάν τρέξετε το πρόγραμμα, θα δείτε στην οθόνη του υπολογιστή σας:

```
1. PROSTHESH 2 AKERAION ARITHMON
2. AFAIRESH 2 AKERAION ARITHMON
TI EPILEGEIS:1
DOSE PROSTHETEO KAI PROSTHETH:5 27
TO ATHROISMA EINAI: 32
```

Άσκηση 8. Να κάνετε τις απαραίτητες μετατροπές στο προηγούμενο πρόγραμμα ώστε ο υπολογιστής να κάνει πρόσθεση αν ο χρήστης πατήσει το πλήκτρο **P** και αφαίρεση αν πατήσει το πλήκτρο **A**, αντί των αριθμών 1 και 2.

Μετατροπή 1. Πρώτα πρέπει αλλάξω τα μηνύματα στο μενού επιλογής, ώστε ο χρήστης να ξέρει ποιο πλήκτρο αντιστοιχεί σε ποια πράξη.

Μετατροπή 2. Η επιλογή του χρήστη, δηλαδή το P ή το A ή γενικότερα οποιοδήποτε γράμμα πληκτρολογήσει, θα αποθηκευτεί στη μεταβλητή `ep`. Η `ep` πρέπει να είναι κατάλληλου τύπου και αφού πρόκειται να δεχτεί χαρακτήρα (και όχι αριθμό) θα πρέπει να δηλωθεί σαν τύπου χαρακτήρα δηλαδή `char`.

Μετατροπή 3. Στην αντίστοιχη `scanf()` πρέπει να υπάρχει η κατάλληλη εντολή μορφοποίησης. Αφού η μεταβλητή `ep` είναι τύπου χαρακτήρα, η εντολή μορφοποίησης είναι η `%c`.

Μετατροπή 4. Το πλήκτρο που πάτησε ο χρήστης θα ελεγχθεί αν είναι ίσο με τον χαρακτήρα P ή με τον χαρακτήρα A στα αντίστοιχα `case` της `switch`. Επειδή τα P και A είναι χαρακτήρες, πρέπει να περιβληθούν από απλά εισαγωγικά.

```
#include <stdio.h>
void main() {
 int a, b;
 char ep;

 printf("P. PROSTHESH 2 AKERAION ARITHMON\n");
 printf("A. AFAIRESH 2 AKERAION ARITHMON\n");
 printf("TI EPILEGEIS:");
 scanf("%c", &ep);

 switch(ep) {
   case 'P':
     printf("DOSE PROSTHETEO KAI PROSTHETH:");
     scanf("%d %d", &a, &b);
     printf("TO ATHROISMA EINAI: %d", a+b);
     break;
   case 'A':
     printf("DOSE AFAIRETEO KAI AFAIRETH:");
     scanf("%d %d", &a, &b);
     printf("H DIAFORA EINAI: %d", a-b);
     break;
   default:
```

```
        printf("LATHOS EPILOGH");
    }
}
```

• Εάν τρέξετε το πρόγραμμα, θα δείτε στην οθόνη του υπολογιστή σας:

```
P. PROSTHESH 2 AKERAION ARITHMON
A. AFAIRESH 2 AKERAION ARITHMON
TI EPILEGEIS:P
DOSE PROSTHETEO KAI PROSTHETH:5 27
TO ATHROISMA EINAI: 32
```

Ο χρήστης πληκτρολογεί όσα σημειώνονται με έντονα γράμματα.

Άσκηση 9. Η εντολή επανάληψης **for**. Εκτύπωση στην οθόνη μιας σειράς αριθμών, από τον μικρότερο -> μεγαλύτερο.

Να γραφεί ένα πρόγραμμα που να τυπώνει στην οθόνη τους αριθμούς από το 1 μέχρι και το 10, τον ένα κάτω από τον άλλο, με βήμα 1. Να χρησιμοποιήσετε την εντολή επανάληψης for.

```
#include <stdio.h>

void main() {
 int i;

 for(i=1; i<=10; i++)
   printf("%d\n", i);
}
```

```
1
2
3
4
5
6
7
8
9
10
```

• Εάν τρέξετε το πρόγραμμα, θα δείτε στην οθόνη σας:

Άσκηση 10. Η εντολή επανάληψης **for**. Εκτύπωση στην οθόνη μιας σειράς αριθμών, από τον μεγαλύτερο -> μικρότερο.

Να γραφεί ένα πρόγραμμα που να τυπώνει στην οθόνη τους αριθμούς από το 10 μέχρι και το 1, τον ένα κάτω από τον άλλο, με βήμα 1. Να χρησιμοποιήσετε την εντολή επανάληψης for.

```
#include <stdio.h>

void main() {
 int i;

 for(i=10; i>=1; i--)
   printf("%d\n", i);
}
```

```
10
9
8
7
6
5
4
3
2
1
```

• Εάν τρέξετε το πρόγραμμα, θα δείτε στην οθόνη σας:

Άσκηση 11. Η εντολή επανάληψης **for**. Εκτύπωση στην οθόνη μιας σειράς αριθμών και του διπλάσιού τους.

Να γραφεί ένα πρόγραμμα που να τυπώνει στην οθόνη τους αριθμούς από το 1 μέχρι και το 10, τον ένα κάτω από τον άλλο, με βήμα 1, και δίπλα σε κάθε αριθμό να τυπώνεται το διπλάσιό του.
Ο κάθε αριθμός να τυπώνεται με εύρος 3 θέσεις. Ανάμεσα στους δύο αριθμούς να υπάρχει ένα κενό.
Να χρησιμοποιήσετε την εντολή επανάληψης for.

```
  1   2
  2   4
  3   6
  4   8
  5  10
  6  12
```

```
#include <stdio.h>

void main() {
 int i, diplo;

 for(i=1; i<=10; i++) {
  diplo = 2*i;
  printf("%3d %3d\n", i, diplo);
 }
}
```

• Εάν τρέξετε το πρόγραμμα, θα δείτε στην οθόνη σας:

Άσκηση 12. Η εντολή επανάληψης **for** σε συνδυασμό με την **if**. Πως μπορούμε να επαναλάβουμε μερικές εντολές όσες φορές θέλουμε.

Να γραφεί ένα πρόγραμμα που:

- να ζητά και να διαβάζει έναν ακέραιο αριθμό,
- κατόπιν **να ελέγχει**
  - **αν** ο αριθμός αυτός είναι μεγαλύτερος του μηδενός, οπότε και να τυπώνει ανάλογο μήνυμα,
  - **αλλιώς αν** είναι μικρότερος του μηδενός, να τυπώνει ανάλογο μήνυμα,
  - **αλλιώς** να τυπώνει μήνυμα ότι ο αριθμός αυτός ισούται με μηδέν.
- Οι εντολές αυτές να επαναλαμβάνονται 4 φορές, χρησιμοποιώντας την εντολή **for**.

```
#include <stdio.h>
void main() {
  int a, i;

  for(i=1; i<=4; i++) {
    printf("DOSE ENAN AKERAIO: ");
    scanf("%d", &a);
    if(a>0)
      printf("O ARITHMOS EINAI MEGALYTEROS TOY 0");
    else if(a<0)
      printf("O ARITHMOS EINAI MIKROTEROS TOY 0"));
    else
      printf("O ARITHMOS EINAI ISOS ME 0");
  }
}
```

Άσκηση 13. Να τροποποιήσετε το προηγούμενο πρόγραμμα, ώστε όταν ο αριθμός που δίνει ο χρήστης είναι ίσος με μηδέν, εκτός από το μήνυμα που τυπώνει το πρόγραμμα, να **σταματούν** αμέσως οι **επαναλήψεις**.

```
#include <stdio.h>
void main() {
  int a, i;

  for(i=1; i<=4; i++) {
    printf("DOSE ENAN AKERAIO: ");
```

```
    scanf("%d", &a);
    if(a>0)
      printf("O ARITHMOS EINAI MEGALYTEROS TOY 0");
    else if(a<0)
      printf("O ARITHMOS EINAI MIKROTEROS TOY 0"));
    else {
      printf("O ARITHMOS EINAI ISOS ME 0");
      break;
    }
  }
}
```

Προσέξτε ότι το else απέκτησε άγκιστρα! (γιατί;)

Η εντολή `break` σταματά αμέσως μια επαναληπτική εντολή, όπως η `for`.

Άσκηση 14. Η εντολή: **do - while.** Επανάληψη ενός προγράμματος, όσες φορές θέλουμε.

1. Αρχικά, να γράψετε ένα πρόγραμμα που να διαιρεί 2 δεκαδικούς (απλής ακρίβειας) αριθμούς τους οποίους θα ζητά από τον χρήστη (π.χ. `DOSE DIAIRETAIO KAI DIAIRETH`) και να τυπώνει το αποτέλεσμα με κατάλληλο μήνυμα και: στοιχισμένο δεξιά, με ελάχιστο εύρος 10 χαρακτήρων, 3 δεκαδικά ψηφία και πάντα με πρόσημο). Τρέξτε το πρόγραμμα.

2. Το παραπάνω πρόγραμμα πρόκειται να επαναλαμβάνεται όσο θέλουμε, γι' αυτό θα προσθέσετε μερικές εντολές.

1) Για να επιτύχετε επανάληψη, να χρησιμοποιήσετε την εντολή **do-while**. (Αποφασίστε εσείς ποιο κομμάτι του προγράμματός σας θα περικλείει).
   Τι θα ελέγχει η συνθήκη της do-while; Φυσικά, αν θέλετε να γίνει επανάληψη ή όχι. Αυτό συνεπάγεται ότι ο χρήστης πρέπει να ερωτηθεί. Άρα,
2) πρώτα κάντε τον υπολογιστή να ρωτάει (αφού τυπώσει το αποτέλεσμα) αν ο χρήστης θέλει να επαναλάβει τον υπολογισμό.
3) Κατόπιν βάλτε τον υπολογιστή να διαβάσει την απάντηση του χρήστη.
4) Η συνθήκη της do-while πρέπει να ελέγχει εάν ο χρήστης πληκτρολόγησε το '**y**' και *Enter* οπότε ο υπολογιστής επαναλαμβάνει τη διαδικασία, ενώ αν έχει πατήσει οποιοδήποτε άλλο πλήκτρο, δεν γίνεται άλλη επανάληψη. Να τρέξετε το πρόγραμμα.

3. Επίσης, να προσθέσετε κατάλληλο έλεγχο για τον διαιρέτη ώστε αν αυτός δοθεί από τον χρήστη μηδενικός να τυπώνεται μήνυμα λάθους και να ξεκινάει μια καινούρια επανάληψη. Να τρέξετε το πρόγραμμα.
→ Τι θα κάνατε ώστε να τερματίζει το πρόγραμμα εάν δοθεί μηδενικός διαιρέτης αντί να ξεκινάει μια καινούρια επανάληψη;

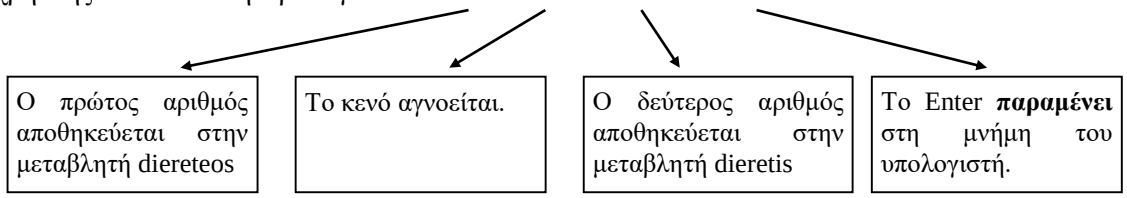
```
#include <stdio.h>
void main(){
 char apantisi = 'y';
 float diereteos, dieretis, pilik;

 do{
   printf("DIAIRESH 2 DEKADIKON ARITHMON\n");
   printf("DOSE TON DIARETAIO KAI TON DIAIRETH:");
   scanf("%f %f", &diereteos, &dieretis);

   if(dieretis == 0) {
     printf("O DIAIRETHS DEN PREPEI NA EINAI MHDENIKOS!\n");
     continue;
   }

   pilik = diereteos/dieretis;
```

```
    printf("TO PILIKO EINAI %+10.3f\n", pilik);

    fflush(stdin);
    printf("Na epanalabo? (y/n)");
    scanf("%c",&apantisi);
  }while(apantisi == 'y');
}
```

Καθαρισμός της μνήμης του υπολογιστή από ότι περίσσεψε από την προηγούμενη `scanf`, με την εντολή `fflush(stdin)`.

Σχόλια:

- Ποια είναι η χρησιμότητα της εντολής **`fflush(stdin);`** Όταν πληκτρολογήσω τα νούμερα που αντιστοιχούν στον διαιρετέο και τον διαιρέτη και πατήσω το πλήκτρο Enter, ο υπολογιστής θα τα διαβάσει με την `scanf` που ακολουθεί. Ωστόσο, το Enter (το οποίο είναι ένας χαρακτήρας) δεν θα διαβαστεί και θα παραμείνει στη μνήμη του υπολογιστή. Δείτε γιατί:
  Ο χρήστης έστω ότι πληκτρολογεί: **`2.5 κενό 3 Enter`**



  και ο υπολογιστής διαβάζει: `scanf("%f %f", &diereteos, &dieretis);`

  Τώρα, όταν έρθει η ώρα του επόμενου `scanf`, όπου ο υπολογιστής ζητάει και διαβάζει την απάντηση του χρήστη, που είναι ένας χαρακτήρας (και όχι ένας αριθμός), δηλαδή:

```
printf("Na epanalabo? (y/n)");
scanf("%c",&apantisi);
```

  επειδή το *Enter* που περίσσεψε από την προηγούμενη `scanf` είναι χαρακτήρας, θα διαβαστεί αυτόματα και θα αποθηκευτεί στη μεταβλητή `apantisi`. Άρα δεν θα διαβαστεί η επιλογή του χρήστη, δηλαδή το **y** ή το **n**! Έτσι η μεταβλητή **apantisi** θα περιέχει τον χαρακτήρα *Enter*. Επιπλέον, επειδή ο χαρακτήρας *Enter* είναι διάφορος του 'y', το πρόγραμμα πάντα θα τερματίζει μετά τον πρώτο κύκλο (αφού η συνθήκη της do-while δεν θα ισχύει)!

  Για να «καθαρίσουμε» τη μνήμη του υπολογιστή από ότι τυχόν περίσσεψε από κάποια προηγούμενη `scanf` χρησιμοποιούμε την εντολή **`fflush(stdin)`**. Αυτός ο «καθαρισμός» έχει περισσότερη χρησιμότητα όταν πρόκειται να διαβάσουμε χαρακτήρα, όπως στην περίπτωσή μας. Επειδή η προηγούμενη `scanf` (αν υπάρχει) αφήνει πάντα το Enter στη μνήμη, εμείς πρέπει να το «εξαφανίσουμε». Αλλιώς θα διαβαστεί αυτόματα αυτό το Enter και όχι ο χαρακτήρας που δίνει ο χρήστης! Στην περίπτωση που διαβάζουμε αριθμούς (όπως στα προηγούμενα προγράμματα) δεν αντιμετωπίζουμε αυτό το πρόβλημα, γι' αυτό και δεν χρησιμοποιήσαμε την εντολή fflush(stdin).

- Εάν ο διαιρέτης είναι μηδενικός, η διαίρεση (αν γίνει) θα δώσει λάθος αποτέλεσμα. Άρα σ' αυτή την περίπτωση δεν αφήνουμε να γίνει διαίρεση αλλά τυπώνεται μήνυμα λάθους και με την **continue** ξεκινάμε έναν νέο κύκλο επανάληψης της do-while (αφού ελεγχθεί η συνθήκη). Εάν θέλαμε να τερματίσει το πρόγραμμα θα βάζαμε αντί της continue την **break**, η οποία μας βγάζει από τον κύκλο επανάληψης do-while, και αφού μετά την do-while δεν υπάρχουν άλλες εντολές, βγαίνουμε και απ' το πρόγραμμα.

- Μετά την εμφάνιση του αποτελέσματος στην οθόνη, ο υπολογιστής ρωτάει τον χρήστη εάν θέλει να επαναλάβει τους υπολογισμούς. Εάν αυτός πατήσει το πλήκτρο **y**, θα επαναλάβει, εάν πατήσει οποιοδήποτε άλλο πλήκτρο, θα τερματίσει. Προσέξτε ότι εάν πατήσετε το κεφαλαίο **Y** και πάλι θα τερματίσει γιατί τα κεφαλαία είναι διαφορετικά απ' τα πεζά στη C.

→ Τι πρέπει να κάνετε ώστε να επαναλαμβάνει είτε πατηθεί το πεζό είτε το κεφαλαίο 'y'; Η συνθήκη πρέπει να γίνει: **`while((apantisi == 'y') || (apantisi == 'Y'))`**

- Εάν τρέξουμε το πρόγραμμα, θα δούμε στην οθόνη του υπολογιστή μας:

```
DIAIRESH 2 DEKADIKON ARITHMON
DOSE TON DIARETAIO KAI TON DIAIRETH:2.5 0
O DIAIRETHS DEN PREPEI NA EINAI MHDENIKOS!
DIAIRESH 2 DEKADIKON ARITHMON
DOSE TON DIARETAIO KAI TON DIAIRETH:2.5 3
TO APOTELESMA EINAI      +0.833
Na epanalabo? (y/n)y
DIAIRESH 2 DEKADIKON ARITHMON
DOSE TON DIARETAIO KAI TON DIAIRETH:5.8 2
TO APOTELESMA EINAI      +2.900
Na epanalabo? (y/n)n
```

Εάν δώσουμε μηδενικό διαιρέτη, ο υπολογιστής «διαμαρτύρεται», και αρχίζει νέο κύκλο επανάληψης.

Νέος κύκλος επανάληψης

Νέος κύκλος επανάληψης

Εάν πληκτρολογήσω 'y' οι υπολογισμοί θα επαναληφθούν, ενώ ... με οποιοδήποτε άλλο χαρακτήρα το πρόγραμμα θα σταματήσει.

Άσκηση 15. Υπολογισμός του μέσου όρου αριθμών - Χρήση **πινάκων αριθμών**. ***απλή***

Να γραφεί ένα πρόγραμμα που:

α) Να ζητά και να διαβάζει έναν-έναν *7* ***δεκαδικούς*** (απλής ακρίβειας) ***αριθμούς***, τους οποίους να αποθηκεύει σε έναν **πίνακα**.

β) Να υπολογίζει ***το άθροισμα*** των αριθμών αυτών (με επαναληπτικό τρόπο).

γ) Να υπολογίζει ***τον μέσο όρο*** των αριθμών αυτών.

δ) Να τυπώνει στην οθόνη το άθροισμα και τον μέσο όρο, σε ξεχωριστές γραμμές, με κατάλληλα μηνύματα. Όλα τα νούμερα (άθροισμα, μέσος όρος) να τυπώνονται καταλαμβάνοντας 9 θέσεις και 3 δεκαδικά ψηφία, στοιχισμένα δεξιά.

```
#include <stdio.h>

void main(){
 float arithm[7], MesOros, athroisma;
 int i;

 printf("YPOLOGISMOS MESOY OROY ARITHMON\n");


 for(i=0; i<=6; i++) {
   printf("DOSE TON EPOMENO ARITHMO: ");
   scanf("%f", &arithm[i]);
 }


 athroisma=0.0;
 for(i=0; i<=6; i++)
   athroisma = athroisma + arithm[i];
```

Ο υπολογιστής **επαναληπτικά**: **Ζητά και διαβάζει τον κάθε αριθμό** απ' το πληκτρολόγιο. Απαιτούνται άγκιστρα, γιατί επαναλαμβάνονται 2 εντολές.

Υπολογισμός του συνολικού αθροίσματος των αριθμών **με επαναληπτικό τρόπο**, … Επαναλαμβάνεται 1 εντολή, άρα δεν απαιτούνται άγκιστρα.

```
 MesOros = athroisma/7.0;
```

… και στο τέλος υπολογισμός του μέσου όρου.

```
 printf("ATHROISMA: %9.3f\n", athroisma);
 printf("MESOS OROS: %9.3f", MesOros);
}
```

Εκτύπωση των αποτελεσμάτων, με αλλαγή γραμμής στο πρώτο μήνυμα.

Σχόλια:

Ο πίνακας 7 θέσεων
float **arithm[7]**;

arithm [0]	7.85	1ος αριθμός
arithm [1]	5.5	2ος αριθμός
arithm [2]	9.2	3ος αριθμός
arithm [3]	10.0	4ος αριθμός
arithm [4]	8.55	5ος αριθμός
arithm [5]	4.9	6ος αριθμός
arithm [6]	5.5	7ος αριθμός

Εκείνο που πρέπει να προσέχετε με τους **πίνακες** είναι ότι η αρίθμηση των στοιχείων τους αρχίζει από το μηδέν. Άρα για έναν πίνακα μεγέθους 7 θέσεων η αρίθμησή του αρχίζει από το 0 και φτάνει ως και το 6 και όχι ως το 7!

Έτσι αν θέλω να γεμίσω τις 7 θέσεις του πίνακα με τιμές, χρησιμοποιώντας για παράδειγμα μια **for**, θα πρέπει να ξεκινήσω από το 0 και να φτάσω ως και το 6. Δηλαδή (έστω ότι οι μεταβλητή `i` είναι ακέραιου τύπου):

```
for(i=0; i<=6; i++)
  scanf("%f", &arithm[i]);
```

- Ο τύπος ενός πίνακα εξαρτάται από το τι είδους αριθμούς θα αποθηκεύσουμε σ' αυτόν. Αν αποθηκεύσουμε δεκαδικούς απλής ακρίβειας αριθμούς, τότε ο πίνακας θα πρέπει να δηλωθεί σαν τύπου `float`.
- Ο πίνακας δηλώνεται με το κανονικό του μέγεθος. Για παράδειγμα, αν ο πίνακας ονομάζεται `arithm` έχει 7 θέσεις, τότε ο πίνακας θα δηλωθεί σαν:
  `float arithm[7];`
- Όμως η πρώτη θέση οποιουδήποτε πίνακα στη γλώσσα C είναι η 0 οπότε η τελευταία θέση θα είναι όσο το μέγεθος του πίνακα πλην 1.
- Τόσο για την ανάγνωση αριθμών από το πληκτρολόγιο, όσο και για τον υπολογισμό του αθροίσματός τους χρησιμοποιώ μια **for.** Το άθροισμα υπολογίζεται σε μια μεταβλητή με το όνομα `athroisma`. Η αρχική τιμή της μεταβλητής `athroisma` τίθεται ίση με 0 γιατί αυτό είναι το ουδέτερο στοιχείο στην πρόσθεση.
- **Ο μετρητής** μιας **for** μπορεί να είναι οποιοδήποτε τύπου (ακόμη και δεκαδικός), όμως όταν χρησιμοποιούμε τον μετρητή αυτόν με πίνακα τότε ο μετρητής πρέπει να είναι κάποιου **ακέραιου τύπου** (απαγορεύεται να είναι δεκαδικού τύπου).
- Μπορείτε να δηλώσετε το μέγεθος του πίνακα σαν σταθερά, π.χ.
  `#define N 7`

Έτσι, στο πρόγραμμα μπορείτε να βάλετε όπου υπάρχει το 7 το N και όπου υπάρχει το 6 το N-1. Τι κερδίζετε κάνοντας αυτό; Αν θελήσετε να κάνετε τους υπολογισμούς για περισσότερους ή λιγότερους από 7 αριθμούς, δεν έχετε παρά να αλλάξετε μόνο την τιμή 7 στην #define και αυτομάτως θα αλλάξει η τιμή του N παντού μέσα στο πρόγραμμα.

- Τι βλέπουμε στην οθόνη όταν τρέξουμε το πρόγραμμα:

```
YPOLOGISMOS MESOY OROY ARITHMON
DOSE TON EPOMENO ARITHMO: 7.85
DOSE TON EPOMENO ARITHMO: 5.5
DOSE TON EPOMENO ARITHMO: 9.2
DOSE TON EPOMENO ARITHMO: 10.0
DOSE TON EPOMENO ARITHMO: 8.55
DOSE TON EPOMENO ARITHMO: 4.9
DOSE TON EPOMENO ARITHMO: 5.5
ATHROISMA:    51.500
MESOS OROS:     7.357
```

Τους αριθμούς με έντονα γράμματα, τους πληκτρολογεί ο χρήστης.

Άσκηση 16. Χειρισμός *ακολουθίας χαρακτήρων (string).* – **Πίνακες χαρακτήρων**.

α) Γράψτε ένα πρόγραμμα, το οποίο να ζητά από τον χρήστη να δώσει το όνομά του, το οποίο ακολούθως **θα διαβάζει** ο υπολογιστής και θα τοποθετεί σε κατάλληλη μεταβλητή. (Αποφασίστε εσείς σε τι είδους μεταβλητή). Στη συνέχεια θα ζητάει και το επώνυμό του, το οποίο θα τοποθετεί σε άλλη μεταβλητή.

✓ Υποθέστε ότι το όνομα θα έχει *μέγιστο μήκος 20 χαρακτήρες* ενώ το επώνυμο θα έχει *μέγιστο μήκος 30 χαρακτήρες* και η ανάγνωσή τους από το πληκτρολόγιο να γίνει με την συνάρτηση **gets()**.

β) Το όνομα και το επώνυμο **να συγκριθούν** και να τυπωθεί ανάλογο μήνυμα αν είναι ίδια ή όχι.

γ) Το όνομα και το επώνυμο **να συνενωθούν** μέσα σε μια καινούρια τρίτη μεταβλητή, και το συνενωμένο ονοματεπώνυμο **να τυπωθεί** στην οθόνη με χρήση της **printf()**.

δ) Να υπολογιστεί το μήκος του ονοματεπώνυμου (δηλαδή πόσους χαρακτήρες περιέχει) και να τυπωθεί ο αριθμός αυτός στην οθόνη, με μήνυμα.

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

void main() {
 char str1[21], str2[31], str3[51];
 int a, mikos;


 printf("DOSE TO ONOMA SOU:");
 gets(str1);
 printf("DOSE TO EPONYMO SOU:");
 gets(str2);


 a = strcmp(str1, str2);
 if(a==0)
   printf("TO ONOMA KAI TO EPONYMO EINAI IDIA\n");
 else
   printf("TO ONOMA KAI TO EPONYMO DEN EINAI IDIA\n");
```

Ο υπολογιστής **ζητά** και **διαβάζει** το όνομα και ακολούθως το επώνυμο.

Ο υπολογιστής **συγκρίνει** τα περιεχόμενα των πινάκων `str1` και `str2`, και ανάλογα με το αποτέλεσμα **τυπώνει μήνυμα**.

```
 strcpy(str3, str1);
 strcat(str3, str2);

 printf("%s\n", str3);


 mikos = strlen(str3);
 printf("TO ONOM/MO SOU EXEI %d GRAMMATA", mikos);
}
```

Για τη **συνένωση** των περιεχόμενων των `str1` και `str2` στον `str3`, απαιτούνται 2 στάδια.

Το περιεχόμενο του `str3` **τυπώνεται** εύκολα.

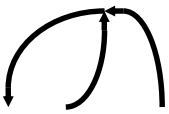
Για τον **υπολογισμό του μήκους** του ονομ/μου (δηλ. του αριθμού των χαρακτήρων που περιέχει) χρησιμοποιούμε τη συνάρτηση **strlen**.

Παρατηρήσεις:

1. Το όνομα, το επώνυμο και γενικότερα οι λέξεις και οι προτάσεις είναι *ακολουθίες χαρακτήρων* (strings) και για την αποθήκευσή τους χρησιμοποιούμε **πίνακες χαρακτήρων**. Αφού το όνομα δεν θα υπερβεί σε μήκος τους 20 χαρακτήρες ο πίνακας που θα το αποθηκεύσουμε αρκεί να έχει μήκος 21 χαρακτήρων - ***η επιπλέον θέση*** χρειάζεται για τον χαρακτήρα NULL. Παρόμοια, το επώνυμο δεν θα υπερβαίνει τους 30 χαρακτήρες, άρα αρκεί ένας πίνακας 31 θέσεων για την αποθήκευσή του. Άρα ο πίνακας που θα τοποθετηθεί η συνενωμένη πρόταση αρκεί να έχει μέγεθος 20+30+1(για το NULL)=51 χαρακτήρων και όχι 52 ή 53!
2. Για τη συνένωση των περιεχομένων των πινάκων `str1` και `str2` στον `str3` απαιτούνται δύο στάδια:

   1ο στάδιο. Με την `strcpy(str3, str1)` **αντιγράφεται** το περιεχόμενο του `str1` στον πίνακα `str3`.

   2ο στάδιο. Με την `strcat(str3, str2)` **συνενώνονται** τα περιεχόμενα των `str3` και `str2` (χωρίς τα NULL) και το αποτέλεσμα τοποθετείται πάλι στον `str3`. Στο τέλος τοποθετείται αυτόματα ένα NULL.

3. Η εκτύπωση του περιεχόμενου του πίνακα `str3` γίνεται πολύ εύκολα, με μια **printf** μόνο. Επειδή θέλω να αλλάξω γραμμή μετά την εκτύπωση, βάζω το `\n` μετά την εντολή μορφοποίησης `%s`.

   Θα μπορούσα να χρησιμοποιήσω την **puts** για την εκτύπωση. Με την **`puts`** ο υπολογιστής θα αλλάξει γραμμή αυτόματα, μετά την εκτύπωση.

4. Για τον υπολογισμό του μήκους του ονοματεπώνυμου (δηλαδή για να βρούμε πόσους χαρακτήρες περιέχει) χρησιμοποιούμε τη συνάρτηση **strlen()**. Το ονοματεπώνυμο βρίσκεται συνενωμένο στον πίνακα `str3`, άρα η εντολή `strlen(str3)` θα μας δώσει το μήκος του ονοματεπώνυμου, το οποίο αποθηκεύουμε στη μεταβλητή `mikos`.

Δεν πρέπει να ξεχάσουμε ότι η χρήση των συναρτήσεων `strcpy`, `strcat`, `strcmp` και `strlen` απαιτεί τη συμπερίληψη του αρχείου **string.h** με μια `#include`.

- Και μια επιπλέον σημείωση για την ανάγνωση των λέξεων (επώνυμου και ονόματος) από το πληκτρολόγιο:

Εδώ χρησιμοποιήσαμε την συνάρτηση **`gets`** για την ανάγνωση από το πληκτρολόγιο του επώνυμου και ονόματος γιατί το ζητούσε η εκφώνηση. Θα μπορούσαμε να χρησιμοποιήσουμε την συνάρτηση **`scanf`** αντί της **`gets`**, με την προϋπόθεση ότι το επώνυμο και το όνομα **δεν περιέχουν κενά**. Δηλαδή, αν το όνομα ήταν: `Anna Maria`, τότε με την `scanf` θα είχαμε πρόβλημα, αφού η `scanf` θα διάβαζε μέχρι το πρώτο κενό, δηλαδή μέχρι και την λέξη `Anna`. Η συνάρτηση `gets` αντιθέτως, διαβάζει όλη την ακολουθία χαρακτήρων (μαζί και τα κενά) μέχρι το *Enter*.

Άσκηση 17. Δείκτες. Οι τελεστές **&** και *****.

Στο πρόγραμμα που θα γράψετε:

α) Να **δηλώσετε** 2 **μεταβλητές**, μια ακέραιου τύπου και μια δεκαδικού απλής ακρίβειας.

β) Να **δηλώσετε** 2 **δείκτες** κατάλληλους να «δείξουν» στις 2 μεταβλητές που δηλώσατε πριν.

γ) Να δώστε τις εντολές ώστε **οι δείκτες** αυτοί **να «δείξουν» στις μεταβλητές** (δηλ. να αποθηκεύσουν, να «δανειστούν», τις διευθύνσεις των μεταβλητών).

δ) Να δώσετε τις εντολές ώστε **να διαβαστούν** από το πληκτρολόγιο δύο αριθμοί (ένας ακέραιος και ένας δεκαδικός) και **να τοποθετηθούν στις θέσεις μνήμης όπου δείχνουν οι δείκτες**. (Εξυπακούεται ότι θα τυπώσετε πριν μήνυμα που να προτρέπει τον χρήστη να δώσει τα νούμερα).

ε) Να τυπωθούν στην οθόνη η διεύθυνση που περιέχει ο πρώτος δείκτης καθώς και το δεδομένο στο οποίο δείχνει ως εξής:

`STH DIEYTYNSH:` ***διεύθυνση*** `BRISKETAI TO:` ***δεδομένο***

και στην επόμενη γραμμή να τυπωθεί το ίδιο για τον δεύτερο δείκτη και το δεδομένο που δείχνει.

```
#include <stdio.h>
void main() {
 int a, *t;
 float b, *x;


 t = &a;
 x = &b;

 printf("DOSE ENAN AKERAIO KAI ENAN DEKADIKO:");
 scanf("%d %f", t, x);

 printf("STH DIEYTYNSH: %p BRISKETAI TO: %d\n", t, *t);
 printf("STH DIEYTYNSH: %p BRISKETAI TO: %f",   x, *x);
}
```

Βήματα (α) και (β). Όταν δηλώνουμε έναν δείκτη (όπως οι `t` και `x`), βάζουμε ένα αστεράκι ***** μπροστά από το όνομά του.

Βήμα (γ). Όταν ο τελεστής **&** εφαρμόζεται σε μια μεταβλητή (όπως οι `a` και `b`), δίνει τη διεύθυνση της μεταβλητής στη μνήμη. Η διεύθυνση αυτή αποθηκεύεται σε δείκτη.

Βήμα (δ).

Βήμα (ε). Ένα αστεράκι μπροστά από το όνομα ενός δείκτη ισοδυναμεί με: «το δεδομένο που δείχνει ο δείκτης».

Σχόλια:

α) Οι δείκτες μπορούν να έχουν τους ίδιους τύπους με τις κανονικές μεταβλητές, δηλαδή τύπου χαρακτήρα, ακέραιου, float κλπ. Όμως όταν δηλώνουμε έναν δείκτη πρέπει πριν το όνομά του να βάλουμε έναν αστερίσκο, π.χ.:

```
int *t;
float *x;
```

Τι καθορίζει **τι τύπου πρέπει να είναι ένας δείκτης**; Ο δείκτης πρέπει να είναι ίδιου τύπου με το δεδομένο που πρόκειται να «δείξει». Άρα στο πρόγραμμά μας ο δείκτης `t` θα δείξει σε έναν ακέραιο, ενώ ο δείκτης `x` θα δείξει σε έναν δεκαδικό απλής ακρίβειας.
Προσοχή: Στο σημείο αυτό έχουμε απλώς δημιουργήσει τους δείκτες. Δεν έχουμε τοποθετήσει κάποια συγκεκριμένη διεύθυνση στους δείκτες. Έτσι, οι δείκτες αυτοί μπορεί να δείχνουν σε κάποια θέση μνήμης που περιέχει χρήσιμα δεδομένα τα οποία δεν πρέπει να πειράξουμε. Άρα, θα πρέπει πρώτα να βάλουμε κάποια «ασφαλή» διεύθυνση στους δείκτες και έπειτα να τους χρησιμοποιήσουμε για να αποθηκεύσουμε δεδομένα εκεί όπου δείχνουν. Και αυτό το κάνουμε (σε αυτό το πρόγραμμα) όπως περιγράφεται στο βήμα (β) παρακάτω.

β) «Δανειζόμαστε» τη διεύθυνση της μεταβλητής `a` και την τοποθετούμε στον δείκτη `t`. Παρόμοια, «δανειζόμαστε» τη διεύθυνση της μεταβλητής `b` και την τοποθετούμε στον δείκτη `x`. Και πως βρίσκουμε ποια είναι η διεύθυνση της μεταβλητής `a` και της μεταβλητής `b`;
Αυτό γίνεται με τον τελεστή **&**. Όταν αυτός ο τελεστής μπαίνει μπροστά από μια μεταβλητή, μας δίνει τη διεύθυνση της μεταβλητής αυτής στη μνήμη (RAM) του υπολογιστή.

```
t = &a;
x = &b;
```

Στο σημείο αυτό μπορεί να σκεφτείτε: Ο δείκτης `t`, για παράδειγμα, δείχνει στην θέση μνήμης της μεταβλητής `a`. Μα τότε, αποθηκεύοντας κάποια τιμή εκεί όπου δείχνει ο δείκτης `t`, δεν θα καταστρέψουμε την τιμή της μεταβλητής `a`; Η απάντηση είναι ναι. Ωστόσο, στο πρόγραμμα αυτό δεν αποθηκεύσαμε ούτε σκοπεύουμε να αποθηκεύσουμε κάποια τιμή στη μεταβλητή `a` και ούτε στην `b`. Δημιουργήσαμε αυτές τις μεταβλητές αποκλειστικά για να «δανειστούμε» τις διευθύνσεις τους. Άρα, δεν τίθεται θέμα καταστροφής κάποιου χρήσιμου δεδομένου σ' αυτήν την περίπτωση.
Ωστόσο, αυτή η στρατηγική προγραμματισμού, δηλαδή να «δανειζόμαστε» διευθύνσεις μεταβλητών για να τις τοποθετήσουμε σε δείκτες, φυσικά δεν είναι σωστή. Όπως θα δούμε σε επόμενο πρόγραμμα, θα χρησιμοποιούμε την συνάρτηση `malloc()` για να πάρουμε «ασφαλείς» διευθύνσεις τις οποίες θα τοποθετούμε σε δείκτες.

γ) Διαβάζουμε τους αριθμούς που πληκτρολογεί ο χρήστης με την `scanf()` και τους αποθηκεύουμε στις θέσεις μνήμης όπου δείχνουν οι δείκτες `t` και `x`. Όπως φαίνεται, σ' αυτή την περίπτωση δεν χρειάζεται ο τελεστής & στην `scanf`. Όπως αναφέρθηκε στο βήμα (β), ο τελεστής & δίνει τη διεύθυνση μιας μεταβλητής. Οι δείκτες είναι διευθύνσεις από μόνοι τους γι' αυτό και δεν χρειάζεται το &.

δ) Η εμφάνιση στην οθόνη **της διεύθυνσης** που περιέχει ένας δείκτης γίνεται με την εντολή μορφοποίησης: **%p**, *ανεξάρτητα του τι τύπου είναι ο δείκτης*.

Όταν όμως αποθηκεύετε **δεδομένα** με την `scanf` ή τυπώνετε **δεδομένα** με την `printf` χρησιμοποιώντας δείκτες, θα το κάνετε χρησιμοποιώντας εντολές μορφοποίησης ανάλογα με τι τύπου είναι οι δείκτες.

ε) Προκειμένου να αναφερθούμε ***στο δεδομένο που «δείχνει» ένας δείκτης*** πρέπει να βάλουμε τον αστερίσκο πριν το όνομά του. Άρα, ενώ το `t` περιέχει μια διεύθυνση, το `*t` ισοδυναμεί με: «το δεδομένο όπου δείχνει ο `t`», δηλαδή έναν ακέραιο αριθμό.

Εάν τρέξουμε το πρόγραμμα θα δούμε στην οθόνη τα παρακάτω. Οι διευθύνσεις ωστόσο, δεν είναι απαραίτητο να είναι οι ίδιες στον υπολογιστή σας. Οι τιμές των διευθύνσεων ίσως σας φανούν παράξενες γιατί είναι τυπωμένες στο δεκαεξαδικό αριθμητικό σύστημα.

Ο χρήστης πληκτρολογεί τα νούμερα με έντονα γράμματα.

```
DOSE ENAN AKERAIO KAI ENAN DEKADIKO:4 5.9
STH DIEYTYNSH: 0064FE00 BRISKETAI TO: 4
STH DIEYTYNSH: 0064FDFC BRISKETAI TO: 5.900000
```

Άσκηση 18. **Δείκτες (pointers)** – Δυναμική δέσμευση μνήμης.

Να γράψετε ένα πρόγραμμα:

- Στο οποίο να δηλώσετε έναν δείκτη κατάλληλο να «δείξει» σε δεδομένα τύπου δεκαδικού απλής ακρίβειας
- Στη συνέχεια το πρόγραμμα να δεσμεύει μνήμη αρκετή για την αποθήκευση ενός δεκαδικού απλής ακρίβειας αριθμό, χρησιμοποιώντας τον παραπάνω δείκτη
- Έπειτα, να ζητά από τον χρήστη και να διαβάζει έναν δεκαδικό αριθμό και να τον αποθηκεύει στη δεσμευμένη μνήμη
- Τέλος να εμφανίζει στην οθόνη τη διεύθυνση στην οποία «δείχνει» αυτός ο δείκτης, καθώς και τον δεκαδικό αριθμό που έχει αποθηκευθεί σ' αυτή τη μνήμη
- Η δεσμευμένη μνήμη να αποδεσμεύεται όταν δεν σας χρειάζεται άλλο.

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

void main() {
 float *p;

 p = (float *)malloc(sizeof(float));

 printf("DOSE ENAN DEKADIKO ARITHMO: ");
 scanf("%f", p);

 printf("STH DIEYTHYNSH: %p\n", p);
 printf("BRISKETAI O ARITHMOS: %f", *p);

 free(p);
}
```

Άσκηση 19. **Δείκτες (pointers)** – Δυναμική δέσμευση μνήμης.

Να γράψετε ένα πρόγραμμα:

- Στο οποίο να δηλώσετε **έναν** δείκτη κατάλληλο να «δείξει» σε δεδομένα τύπου δεκαδικού απλής ακρίβειας
- Στη συνέχεια το πρόγραμμα να δεσμεύει μνήμη αρκετή για την αποθήκευση **τριών** δεκαδικών απλής ακρίβειας αριθμών, χρησιμοποιώντας τον παραπάνω δείκτη
- Έπειτα, να ζητά από τον χρήστη και να διαβάζει δύο δεκαδικούς αριθμούς και να τους αποθηκεύει στις δύο πρώτες θέσεις της δεσμευμένης μνήμης
- Τέλος να προσθέτει τους δύο αριθμούς και να αποθηκεύει το άθροισμα στην τρίτη από τις δεσμευμένες θέσεις μνήμης. Επιπλέον να εμφανίζει στην οθόνη το άθροισμα με ανάλογο μήνυμα
- Η δεσμευμένη μνήμη να αποδεσμεύεται όταν δεν σας χρειάζεται άλλο.

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

void main() {
 float *p;
```

```
 p = (float *)malloc(3 * sizeof(float));

 printf("DOSE 2 DEKADIKOYS ARITHMOYS: ");
 scanf("%f %f", p, (p+1));

 *(p+2) = *p + *(p+1);

 printf("TO ATHROISMA EINAI: %f", *(p+2));
 free(p);
}
```

Άσκηση 20. **Δείκτες (pointers)** – Δυναμική δέσμευση μνήμης.

Να γράψετε ένα πρόγραμμα:

- Στο οποίο να δηλώσετε **έναν** δείκτη κατάλληλο να «δείξει» σε δεδομένα τύπου δεκαδικού απλής ακρίβειας
- Στη συνέχεια το πρόγραμμα να ρωτήσει τον χρήστη **πόσους** αριθμούς θέλει να προσθέσει και να δεσμεύει μνήμη αρκετή για την αποθήκευσή τους, χρησιμοποιώντας τον παραπάνω δείκτη
- Έπειτα, να ζητά ***επαναληπτικά*** από τον χρήστη και να διαβάζει τους δεκαδικούς αριθμούς και να τους αποθηκεύει σε διαδοχικές θέσεις της δεσμευμένης μνήμης
- Τέλος να υπολογίζει το άθροισμά τους σε μια χωριστή μεταβλητή. Επιπλέον να το εμφανίζει στην οθόνη με ανάλογο μήνυμα
- Η δεσμευμένη μνήμη να αποδεσμεύεται όταν δεν σας χρειάζεται άλλο.

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

void main() {
 float *p, sum;
 int plithos, i;

 printf("POSOYS ARITHMOYS THA DOSEIS? ");
 scanf("%d", &plithos);
 p = (float *)malloc(plithos * sizeof(float));

 for(i=0; i<plithos; i++) {
   printf("DOSE TON EPOMENO ARITHMO: ");
   scanf("%f", (p+i));
 }

 sum = 0;
 for(i=0; i<plithos; i++)
   sum = sum + *(p+i);

 printf("TO ATHROISMA EINAI: %f", sum);
 free(p);
}
```

```
if(p == NULL)  {
  printf("DEN YPARXEI ARKETH MNHMH!") ;
  exit(1);
}
```

Σχόλια:

α) Σε προηγούμενο πρόγραμμα, όπου γινόταν υπολογισμός του μέσου όρου ενός συνόλου βαθμών, με χρήση πίνακα έπρεπε να δηλώσουμε το μέγεθος του πίνακα. Αυτό ήταν δεσμευτικό γιατί σήμαινε ότι όταν τρέχαμε το πρόγραμμα δεν μπορούσαμε να πληκτρολογήσουμε περισσότερους αριθμούς από όσους χωρούσε ο πίνακας. Ωστόσο, επειδή **εδώ δεσμεύουμε όση μνήμη θέλουμε τη στιγμή που τρέχει το πρόγραμμα**, δεν τίθεται περιορισμός στο άνω όριο, δηλαδή πόσους δεκαδικούς αριθμούς μπορεί να δώσει ο χρήστης. **Εφόσον ο υπολογιστής έχει διαθέσιμη μνήμη, θα την παρέχει, όση κι αν ζητήσουμε.**

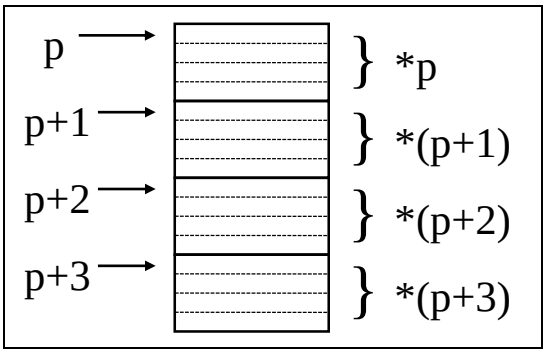
→ Βέβαια, αν θέλαμε να ήμασταν πιο σωστοί, θα έπρεπε να ελέγξουμε αν το πλήθος των αριθμών που θέλει να πληκτρολογήσει ο χρήστης είναι παράλογο (π.χ. δεν μπορώ να πω ότι θα πληκτρολογήσω 0 ή -5 δεκαδικούς!)

β) **Η συνάρτηση malloc αναλαμβάνει να δεσμεύσει την απαιτούμενη μνήμη για την αποθήκευση των δεκαδικών αριθμών.** Η χρήση του είναι ως εξής:

δείκτης = **(τύπος_δεδομένων *) malloc(** *πλήθος δεδομένων***);**

Μέσα στις παρενθέσεις αναφέρουμε το πλήθος των δεδομένων που θα αποθηκεύσουμε στην μνήμη μας και η συνάρτηση malloc τοποθετεί μέσα στον δείκτη τη διεύθυνση του πρώτου byte της μνήμης που δεσμεύτηκε. **Ο δείκτης λοιπόν θα δείχνει στην αρχή του κομματιού μνήμης που δεσμεύτηκε.** (Κοίτα σχήμα παρακάτω).

γ) Από κει και πέρα μπορούμε να χρησιμοποιήσουμε τις δεσμευμένες θέσεις μνήμης μέσω του δείκτη `p`. Το μόνο που πρέπει να προσέξουμε είναι να μην υπερβούμε τα όρια της μνήμης που δεσμεύσαμε. Θα αποθηκεύσουμε `(plithos)` δεκαδικών, άρα το `i` θα μεταβάλλεται από το 0 ως και το `(plithos-1)`.



δ) Όταν δεν χρειαζόμαστε ένα κομμάτι μνήμης που δεσμεύσαμε, μπορούμε να το αποδεσμεύσουμε με τη συνάρτηση free. Μετά από τις παρενθέσεις της `free` πρέπει να γράψουμε το όνομα του δείκτη που χρησιμοποιήσαμε στην malloc, άρα στην περίπτωσή μας `free(p)`.

Άσκηση 21. Πρόγραμμα υπολογισμού του τετραγώνου και της τετραγωνικής ρίζας αριθμού με χρήση των συναρτήσεων **pow** και **sqrt** που μας παρέχει έτοιμες η C.

Να γράψετε ένα πρόγραμμα που:
α) Να **ζητά** έναν αριθμό από τον χρήστη του υπολογιστή,
β) Να **διαβάζει** τον αριθμό που θα πληκτρολογήσει ο χρήστης,
γ) Να **υπολογίζει** το ***τετράγωνο*** και την ***τετραγωνική ρίζα*** του αριθμού αυτού χρησιμοποιώντας έτοιμες συναρτήσεις της C και
δ) Να **τυπώνει** τον αριθμό και το τετράγωνό του με:
ελάχιστο εύρος 10 χαρακτήρων, 3 *δεκαδικά ψηφία*, *στοιχισμένοι δεξιά* και *πάντα με πρόσημο*,
ενώ στην δεύτερη γραμμή να τυπωθούν ο αριθμός και η τετραγωνική ρίζα του με:
ελάχιστο εύρος 8 χαρακτήρων, 2 *δεκαδικά ψηφία*, *στοιχισμένοι αριστερά*.

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

void main() {
 double x, tetr, riza;

 printf("DOSE ENAN ARITHMO:");
 scanf("%lf", &x);

 tetr = pow(x, 2.0);
 riza = sqrt(x);

 printf("TO TETRAGONO TOY %+10.3lf EINAI= %+10.3lf \n", x, tetr);
 printf("H TETRAGONIKH RIZA TOY %-8.2lf EINAI= %-8.2lf", x, riza);
}
```

Η συμπερίληψη του αρχείου `math.h` είναι απαραίτητη, εφ' όσον πρόκειται να χρησιμοποιήσω τις συναρτήσεις `pow` και `sqrt`.

Εδώ γίνεται **η κλήση** των συναρτήσεων. Εφόσον θέλουμε το τετράγωνο του αριθμού, ο εκθέτης θα είναι πάντα το 2.

Σχόλια:
• Για τον υπολογισμό του τετραγώνου και της τετραγωνικής ρίζας αριθμού η C μας δίνει δύο έτοιμες συναρτήσεις, την `pow` και την `sqrt`. Για να χρησιμοποιήσουμε αυτές τις μαθηματικές συναρτήσεις σε κάποιο πρόγραμμά μας, πρέπει να συμπεριλάβουμε (με την εντολή `#include`) το αρχείο επικεφαλίδας **math.h**, όπως ακριβώς για να χρησιμοποιήσουμε τις συναρτήσεις `printf` και `scanf` πρέπει να συμπεριλάβουμε το αρχείο επικεφαλίδας `stdio.h`

• Ποιες έτοιμες συναρτήσεις μας παρέχει η C και ποιο αρχείο επικεφαλίδας πρέπει να συμπεριλάβουμε σε κάθε περίπτωση, μπορούμε να το βρούμε σε όλα τα βιβλία και εγχειρίδια της C.
• Οι εντολές `#include` μπαίνουν η μία κάτω απ' την άλλη, όπως φαίνεται στο πρόγραμμα.

• Οι συνάρτηση `pow` υψώνει έναν αριθμό (τη βάση) σε κάποια δύναμη (τον εκθέτη) και η `sqrt` υπολογίζει την τετραγωνική ρίζα κάποιου αριθμού. Αυτές ορίζονται ως εξής:

pow(*βάση* , *εκθέτης* **)** και **sqrt(** *αριθμός* **)**

• Όλα τα ορίσματα των δύο αυτών συναρτήσεων πρέπει να είναι αριθμοί τύπου **double** (δηλ. κινητής υποδιαστολής διπλής ακρίβειας). Επίσης τύπου **double** θα είναι και τα αποτελέσματα που θα επιστρέψουν οι δύο συναρτήσεις. (Αυτές οι πληροφορίες είναι επίσης διαθέσιμες στα βιβλία της C). Έτσι, τόσο η μεταβλητή `x` (η οποία θα περιέχει τον αριθμό που θα πληκτρολογήσει ο χρήστης) όσο και οι μεταβλητές `tetr` και `riza` (στις οποίες θα αποθηκευτούν τα αποτελέσματα) πρέπει να είναι τύπου `double`.

Άσκηση 22. Χρήση **συναρτήσεων** που δημιουργούμε εμείς και των **πρωτοτύπων** τους.

Να γράψετε ένα πρόγραμμα που:
α) Να ζητά 2 ακέραιους αριθμούς από τον χρήστη και ακολούθως να τους διαβάζει.
β) Να "στέλνει" τους αριθμούς αυτούς σε αντίστοιχες **συναρτήσεις** (επιλέξτε εσείς τα ονόματά τους) οι οποίες θα υπολογίζουν και θα επιστρέφουν:
(1) η πρώτη συνάρτηση, *το πηλίκο* των αριθμών,
(2) η δεύτερη συνάρτηση, *το υπόλοιπο* της διαίρεσής τους και
(3) η τρίτη, *τον μεγαλύτερο* από τους δύο αριθμούς που έδωσε ο χρήστης.
→ Για κάθε συνάρτηση να φτιάξετε και το **πρωτότυπό** της.
γ) Τα αποτελέσματα κάθε συνάρτησης να τυπωθούν στην οθόνη με ανάλογα μηνύματα.

```
#include <stdio.h>

int piliko(int x, int y);
int ypoloipo(int k, int m);
int max(int k1, int k2);

void main(){
 int a, b, z1, z2, z3;

 printf("DOSE 2 AKERAIOYS ARITHMOUS:");
 scanf("%d %d", &a, &b);

 z1 = piliko(a,b);
 z2 = ypoloipo(a,b);
 z3 = max(a,b);

 printf("TO PILIKO EINAI: %d\n", z1);
 printf("TO YPOLOIPO EINAI: %d\n", z2);
 printf("O MEGALYTEROS ARITMOS EINAI TO: %d", z3);
}
```

Τα πρωτότυπα των συναρτήσεων που χρησιμοποιούμε στο πρόγραμμά μας. Προσέξτε ότι χρειάζονται τα ελληνικά ερωτηματικά στο τέλος.

Ο υπολογιστής ζητά από τον χρήστη και ακολούθως διαβάζει και τοποθετεί στις μεταβλητές a και b δύο ακέραιους αριθμούς.

Ο υπολογιστής **«καλεί» τις συναρτήσεις** στέλνοντάς τους τις τιμές που περιέχουν οι a και b. Οι συναρτήσεις «επιστρέφουν» στις μεταβλητές z1, z2 και z3 τα αποτελέσματα.

Τα αποτελέσματα τυπώνονται στην οθόνη και **εδώ τελειώνει το κυρίως πρόγραμμα**.

Κυρίως Πρόγραμμα

```
int  piliko(int x, int y) {
  int z;
  z = x/y;
  return(z);
}
```

Οι συναρτήσεις που χρησιμοποιούμε στο πρόγραμμά μας. Τα ονόματα των συναρτήσεων τα επιλέγουμε μόνοι μας.

1η Συν-άρτηση

3η Συν-άρτηση

```
int  ypoloipo(int k, int m) {
  int c;
  c = k%m;
  return(c);
}

int  max(int k1, int k2) {
 if(k1 > k2) return k1;
 else return k2;
}
```

→ Εάν τρέξουμε το πρόγραμμα θα δούμε στην οθόνη τα εξής:

```
DOSE 2 AKERAIOYS ARITHMOUS:25 7
TO PILIKO EINAI: 3
TO YPOLOIPO EINAI: 4
O MEGALYTEROS ARITMOS EINAI TO: 25
```

Τα νούμερα με έντονα γράμματα τα πληκτρολογεί ο χρήστης.

Άσκηση 23. Χρήση **συναρτήσεων** και των **πρωτοτύπων** τους.

Να **γράψετε** δύο **συναρτήσεις**:

(1) η πρώτη συνάρτηση, να υπολογίζει *την περίμετρο της βάσης* ενός κυλίνδρου, όταν έχει σαν "είσοδό της" την ακτίνα.
Δίνεται ο τύπος: **Περίμετρος = 2*π*ακτίνα** (όπου το π είναι 3.1415)

(2) η δεύτερη συνάρτηση, να υπολογίζει *τον όγκο* ενός κυλίνδρου, όταν έχει σαν "εισόδους της" την ακτίνα και το ύψος.
Δίνεται ο τύπος: **Όγκος = π*ακτίνα^2*ύψος** (όπου το π είναι 3.1415)

→ Για κάθε συνάρτηση να φτιάξετε και το **πρωτότυπό** της.

Να γράψετε ένα πρόγραμμα που:

α) Να ζητά από τον χρήστη την **ακτίνα** της βάσης και τον **ύψος** ενός κυλίνδρου και ακολούθως να τα διαβάζει.
β) Να **καλεί** τις **συναρτήσεις** που φτιάξατε, για να υπολογίσει την περίμετρο και τον όγκο.
γ) Τα αποτελέσματα κάθε συνάρτησης, δηλαδή η περίμετρος και ο όγκος, **να τυπωθούν** στην οθόνη με ανάλογα μηνύματα.

```
#include <stdio.h>


float  perimetros(float a);
float  ogos(float a, float y);
```

Τα πρωτότυπα των συναρτήσεων που χρησιμοποιούμε στο πρόγραμμά μας. Προσέξτε ότι χρειάζονται τα ελληνικά ερωτηματικά στο τέλος.

Κυρίως Πρόγραμμα

```
void main(){
 float aktina, ypsos, per, og;

 printf("DOSE AKTINA KAI YPSOS KYLINDROU:");
 scanf("%f %f", &aktina, &ypsos);

 per = perimetros(aktina);
 og = ogos(aktina, ypsos);


 printf("H PERIMETROS EINAI: %f\n", per);
 printf("O OGOS EINAI: %f\n", og);
}
```

Εδώ **«καλούνται» οι συναρτήσεις**, στις οποίες στέλνονται οι τιμές που περιέχουν η aktina και το ypsos. Οι συναρτήσεις «επιστρέφουν» στις μεταβλητές per και og τα αποτελέσματα.

Εδώ τελειώνει το κυρίως πρόγραμμα.

1η Συν-άρτηση

```
float  perimetros(float a) {
  float p;
  p = 2*3.1415*a;
  return(p);
}
```

Οι συναρτήσεις που χρησιμοποιούμε στο πρόγραμμά μας. Τα ονόματα των συναρτήσεων τα επιλέγουμε μόνοι μας.

2η Συν-άρτηση

```
float  ogos(float a, float y) {
  float o;
  o = 3.1415*a*a*y;
  return(o);
}
```

Άσκηση 24. Ανάγνωση δεδομένων από και **εγγραφή** σε **αρχείο**.

Το πρόγραμμα που θα δημιουργήσετε, θα διαβάζει τα δεδομένα του από ένα αρχείο και θα αποθηκεύει τα αποτελέσματά του σε ένα άλλο αρχείο. Δεν πρόκειται να δώσετε δεδομένα από το πληκτρολόγιο, ούτε να δείτε αποτελέσματα στην οθόνη.

Προκαταρκτικά

Αρχικά, να δημιουργήσετε ένα αρχείο (με τον επεξεργαστή κειμένου της ίδιας της C),και να το αποθηκεύσετε στον κατάλογο `C:\temp` με όνομα: `input.txt`. (Εάν δεν υπάρχει ο κατάλογος να τον δημιουργήσετε). Τα περιεχόμενα του `input.txt` να είναι όπως φαίνεται δίπλα:
Δηλαδή, έστω ότι το αρχείο είναι ένα απλό αρχείο κειμένου που περιέχει αριθμούς.

```
8.5
5.0
10.0
9.5
7.2
6.5
```

Να γράψετε ένα πρόγραμμα που:

α) Να τυπώνει ένα μήνυμα στην οθόνη που να ειδοποιεί τον χρήστη ότι η εκτέλεση του προγράμματος ξεκίνησε.

β) **Να ανοίγει** το αρχείο κειμένου `input.txt` **για ανάγνωση** δεδομένων και ένα νέο αρχείο κειμένου με όνομα: `output.txt` **για εγγραφή** των αποτελεσμάτων (πάλι στον `C:\temp`).
✓ **Να ελέγχει** εάν τα αρχεία ανοίχτηκαν επιτυχώς.

γ) Στη συνέχεια, μέσω μιας επαναληπτικής εντολής **να επαναλαμβάνει** τα βήματα (i) – (iii) μέχρι να επεξεργαστεί όλα τα περιεχόμενα του αρχείου `input.txt`:
(i) **Να «διαβάζει»** από το αρχείο `input.txt` έναν αριθμό.
(ii) Να υπολογίζει το τετράγωνο του αριθμού αυτού.

(iii) **Να «γράφει»** στο αρχείο `output.txt` τον αριθμό και το τετράγωνό του, σε μια γραμμή χωρισμένα με ένα κενό, με ελάχιστο εύρος 8 θέσεων και 2 δεκαδικά ψηφία.

δ) **Να κλείνει** και τα δύο αρχεία που άνοιξε στο βήμα (β).

ε) Να τυπώνει ένα μήνυμα στην οθόνη που να ειδοποιεί τον χρήστη ότι η εκτέλεση του προγράμματος ολοκληρώθηκε.

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

void main(){
 FILE *fp1, *fp2;
 float x, y;

 printf("H EKTELESH TOY PROGRAMMATOS ARXISE\n");

 fp1 = fopen("C:\\temp\\input.txt", "rt");

 if(fp1 == NULL){
   printf("cannot open input.txt");
   exit(1);
 }

 fp2 = fopen("C:\\temp\\output.txt", "wt");

 if(fp2 == NULL){
   printf("cannot open output.txt");
   exit(1);
 }

 while(fscanf(fp1,"%f", &x) != EOF) {

  y = x*x;

  fprintf(fp2,"%8.2f %8.2f\n", x, y);
 }

 fclose(fp1);
 fclose(fp2);

 printf("H EKTELESH TOY PROGRAMMATOS TELEIOSE\n");
}
```

Οι δείκτες τύπου **FILE** θα χρησιμοποιηθούν για να «δείξουν» στα αρχεία που θα ανοίξουμε.

Άνοιγμα του αρχείου input.txt για ανάγνωση μόνο.

Έλεγχος αν το αρχείο input.txt ανοίχτηκε πράγματι.

Άνοιγμα του αρχείου output.txt για εγγραφή μόνο.

Έλεγχος αν το αρχείο output.txt ανοίχτηκε πράγματι.

Διαβάζουμε επαναληπτικά έναν-έναν τους αριθμούς από το αρχείο input.txt, και για όσο διάστημα δεν έχουμε συναντήσει το τέλος του αρχείου, δηλαδή το **EOF**.

Αποθηκεύουμε μέσα στο αρχείο output.txt τον αριθμό και το τετράγωνό του, χωρισμένα με ένα κενό, και κατόπιν αλλάζουμε γραμμή.

Κλείνουμε τα δύο αρχεία.

✓ Για να διαβάσουμε ή να γράψουμε σε ένα αρχείο πρέπει πρώτα να το ανοίξουμε. Αυτό γίνεται καλώντας τη συνάρτηση `fopen()` ως εξής:

άνοιγμα

δείκτης = **fopen(**"*όνομα αρχείου*", "*τρόπος*"**);**

- Στο "***όνομα αρχείου***" δίνουμε το όνομα του αρχείου που ανοίγουμε, μαζί με το μονοπάτι.
- Στην θέση του "***τρόπος***" δηλώνουμε για ποιο λόγο ανοίγουμε το αρχείο. Αν το αρχείο είναι κειμένου (text) και ανοίγεται για ανάγνωση (read) βάζουμε: **"rt"**, ενώ αν είναι πάλι κειμένου (text) αλλά θέλουμε να γράψουμε (write), βάζουμε: **"wt"**. Στην δεύτερη περίπτωση, αν το αρχείο υπάρχει ήδη, διαγράφονται τα προηγούμενα περιεχόμενά του.

- Η συνάρτηση επιστρέφει μια διεύθυνση σε έναν δείκτη (γι' αυτό υπάρχει στην εντολή το =), ο οποίος δείχνει στο αρχείο και όχι στη μνήμη του υπολογιστή όπως είναι η συνήθης περίπτωση με τους δείκτες. Γι' αυτό **ο δείκτης αυτός πρέπει να δηλωθεί σαν τύπου FILE**, δηλαδή:
  `FILE *fp1, *fp2;`

- Εάν ο υπολογιστής δεν μπορέσει να ανοίξει το αρχείο, στον δείκτη αποθηκεύεται η τιμή NULL, οπότε και πρέπει να τυπωθεί ανάλογο μήνυμα λάθους και το πρόγραμμα τερματίζει.
- Πολλά αρχεία μπορεί να είναι ταυτόχρονα ανοιχτά για ανάγνωση ή εγγραφή- στην περίπτωσή μας είναι 2.

✓ Η ανάγνωση δεδομένων από ένα αρχείο και η εγγραφή δεδομένων σε ένα αρχείο γίνεται με τις συναρτήσεις `fscanf()` και `fprintf()`. Η μορφή των συναρτήσεων αυτών είναι ακριβώς η ίδια με των `scanf()` και `printf()`, με το πρόσθετο στοιχείο ότι πρέπει να λέμε στον υπολογιστή σε ποιο αρχείο θέλουμε να διαβάσουμε ή να γράψουμε. Αυτό γίνεται βάζοντας και τον δείκτη του αρχείου μέσα στις παρενθέσεις των συναρτήσεων:

ανάγνωση
εγγραφή

fscanf(*ΔείκτηςΑρχείου, "εντολές μορφοποίησης", &μεταβλητή, &μεταβλητή, ...***);**
fprintf(*ΔείκτηςΑρχείου, "μήνυμα + εντολές μορφοποίησης", μεταβλητή, μεταβλητή, ...***);**

✓ Εάν η `fscanf()` δεν μπορεί να διαβάσει άλλα δεδομένα από το αρχείο, άρα φτάσαμε στο τέλος του, επιστρέφει την τιμή **EOF**. Έτσι, βάζουμε την `fscanf()` μέσα στη συνθήκη της `while` η οποία θα επαναλαμβάνεται όσο η `fscanf()` δεν είναι ίση με EOF, δηλαδή:

```
while(fscanf(........)   !=   EOF) {
  εντολές;.... }
```

για επαναληπτική ανάγνωση

κλείσιμο

✓ Ένα αρχείο που δεν το χρειαζόμαστε άλλο το κλείνουμε καλώντας τη συνάρτηση **fclose()** και βάζοντας μέσα στις παρενθέσεις τον δείκτη του αρχείου.

→ Εάν τρέξουμε το πρόγραμμα και κοιτάξουμε τα περιεχόμενα του αρχείου `output.txt`, το οποίο δημιουργείται από το πρόγραμμά μας μέσα στον κατάλογο `C:\temp`, θα δούμε τα εξής:

```
  8.50    72.25
  5.00    25.00
 10.00   100.00
  9.50    90.25
  7.20    51.84
  6.50    42.25
```

Άσκηση 25. Χρησιμοποίηση δομών (structures).

- Να δημιουργήσετε έναν **ορισμό δομής** (structure definition), ο οποίος να περιέχει μεταβλητές για την αποθήκευση των **συντεταγμένων (x, y) του κέντρου** ενός κύκλου, της **ακτίνας** του (όλα ακέραιοι) και του **εμβαδού** του (δεκαδικός).

Στη συνέχεια, να γράψετε ένα πρόγραμμα, στο οποίο:

- να δηλώσετε **μια μεταβλητή δομής** του παραπάνω τύπου, η οποία θα αντιπροσωπεύει έναν συγκεκριμένο κύκλο και
- να προτρέψετε τον χρήστη να δώσει τις **συντεταγμένες**, και την **ακτίνα** του κύκλου, τα οποία θα διαβάσει ο υπολογιστής και θα αποθηκεύσει στις ανάλογες μεταβλητές της δομής.
- το πρόγραμμά σας κατόπιν να υπολογίζει το εμβαδόν του κύκλου και
- να τυπώνει στην οθόνη τις συντεταγμένες και την ακτίνα του κύκλου με ανάλογα μηνύματα και στην επόμενη γραμμή
- να τυπώνει στην οθόνη το εμβαδόν με ανάλογο μήνυμα και 2 δεκαδικά ψηφία.

```
#include <stdio.h>

struct kyklos {
   int x;
   int y;
   int aktina ;
   float embadon;
};
```

Πρώτα δημιουργούμε έναν **ορισμό δομής**, όπου θα τοποθετηθούν οι συντεταγμένες του κέντρου, η ακτίνα και το εμβαδόν του κύκλου.

Κατόπιν, δηλώνουμε μια **μεταβλητή δομής** με όνομα `k1`, η οποία έχει τη μορφή της `kyklos`.

```
void main() {

struct kyklos k1;

printf("DOSE TIS SYNTETAGMENES x y TOY KENTROY: ");
scanf("%d %d", &k1.x, &k1.y);

printf("DOSE THN AKTINA: ");
scanf("%d", &k1.aktina);

k1.embadon = 3.14 * k1.aktina * k1.aktina;

printf("O KYKLOS ME KENTRO: %d %d EXEI AKTINA: %d\n", k1.x, k1.y,
                                                      k1.aktina);

printf("TO EMBADON TOY KYKLOY EINAI: %.2f", k1.embadon);
}
```

Με τις 2 `scanf` ο υπολογιστής διαβάζει και αποθηκεύει τις συντεταγμένες του κέντρου και την ακτίνα στις αντίστοιχες **μεταβλητές της δομής**.

Υπολογίζουμε το εμβαδόν.

Τυπώνουμε το εμβαδόν, με κατάλληλο μήνυμα.

➡ Εάν τρέξουμε το πρόγραμμα, θα δούμε τα εξής:

```
DOSE TIS SYNTETAGMENES x y TOY KENTROY: 50 60
DOSE THN AKTINA: 20
O KYKLOS ME KENTRO: 50 60 EXEI AKTINA: 20
TO EMBADON TOY KYKLOY EINAI: 1256.00
```

Ο χρήστης πληκτρολογεί αυτά που φαίνονται με έντονα γράμματα.

Σχόλια:

Μια δομή στη γλώσσα C είναι μια συλλογή μεταβλητών που ομαδοποιούνται κάτω από το ίδιο όνομα. Έτσι, στο παραπάνω πρόγραμμα, οι μεταβλητές `x`, `y`, `aktina`, `embadon` ομαδοποιούνται κάτω από το όνομα: `kyklos`.

Ομαδοποιώντας (πακετάροντας) μια συλλογή μεταβλητών κάτω από ένα όνομα, κρατάμε μαζί πληροφορίες που σχετίζονται μεταξύ τους. Είναι φανερό ότι οι συντεταγμένες `x`, `y`, η ακτίνα και το εμβαδόν αφορούν στο ίδιο πράγμα: ***έναν κύκλο***, άρα έχουν σχέση μεταξύ τους, γι' αυτό και τις ομαδοποιούμε μέσα σε μια δομή που ονομάσαμε `kyklos`.

Προσέξτε κάτι σημαντικό: ο `kyklos` είναι στην πραγματικότητα ένας **ορισμός δομής** (structure definition), δηλαδή ένα καλούπι. Είναι ένα καλούπι από το οποίο μπορούμε να «κατασκευάσουμε» κύκλους, που ο καθένας θα έχει τις δικές του συντεταγμένες κέντρου, ακτίνα και εμβαδόν.
Γνωρίζετε ότι η C έχει μερικούς βασικούς τύπους δεδομένων, όπως ο `char`, `int`, `float`, `double`, οι οποίοι είναι ήδη κατασκευασμένοι. Στην πραγματικότητα, ο `kyklos` είναι ένας νέος τύπος δεδομένων τον οποίο κατασκευάσατε εσείς (σαν προγραμματιστής). Άρα, όπως ακριβώς ο `int` είναι ένα «καλούπι» από το οποίο μπορούμε να δημιουργήσουμε μεταβλητές, π.χ.

```
int a, b, c;
```

έτσι και από τον ορισμό δομής `kyklos` μπορούμε να δημιουργήσουμε μεταβλητές (δηλαδή μεταβλητές τύπου δομής), π.χ.:

```
struct kyklos k1, m, e;
```

Τα `k1`, `m`, `e` είναι μεταβλητές τύπου `kyklos` ή (με άλλα λόγια) είναι **μεταβλητές δομής** τύπου `kyklos`.

Παρατηρήστε μια μικρή λεπτομέρεια κατά τη δημιουργία μεταβλητών που είναι δομές: κανονικά, αφού ο `kyklos` είναι ένας νέος τύπος, θα έπρεπε να μπορώ να δηλώσω μεταβλητές που να προέρχονται από αυτόν τον νέο τύπο ως εξής:

✘
```
kyklos k1, m, e;
```

δηλαδή όπως ακριβώς έκανα λίγο παραπάνω όταν δήλωσα τις μεταβλητές `a`, `b`, `c` που ήταν τύπου `int`. Όμως αυτό δεν είναι σωστό! Πρέπει να χρησιμοποιήσω και την λέξη `struct` και όταν δημιουργώ μια φόρμα δομής, αλλά και όταν δημιουργώ δομές από αυτή τη φόρμα.

Ωραία λοιπόν, ένας ορισμός δομής χρησιμεύει στο να διατηρούμε «μαζί» στοιχεία που σχετίζονται μεταξύ τους. Από αυτόν τον ορισμό μπορούμε να δημιουργήσουμε όσες δομές θέλουμε για το πρόγραμμά μας. Στο παραπάνω πρόγραμμα δημιουργήσαμε έναν κύκλο που ονομάσαμε `k1`. Ας υποθέσουμε (για να φανεί ένα πιθανό πρόβλημα) ότι έχουμε δημιουργήσει 3 κύκλους που ονομάζονται `k1`, `m` και `e`. Αυτοί, όπως είπαμε παραπάνω, είναι δομές που προήλθαν από τον ορισμό δομής `kyklos`:

```
struct kyklos k1, m, e;
```

Τώρα έστω ότι θέλουμε να δώσουμε την τιμή 5 στην ακτίνα του κύκλου `k1`. Πιθανόν να γράψετε την εντολή:

✘
```
aktina = 5;
```

Λάθος!! Πρώτ' απ' όλα, η `aktina` ΔΕΝ είναι μια μεταβλητή που έχει δηλωθεί μόνη της – αντιθέτως ανήκει στον ορισμό δομής `kyklos`, άρα το παραπάνω είναι λάθος.
Εξ' άλλου, με την παραπάνω εντολή λέω στον υπολογιστή ότι η ακτίνα είναι ίση με 5 – αλλά ποια ακτίνα; Εδώ έχουμε 3 κύκλους. Του κύκλου `k1`, του κύκλου `m` ή του κύκλου `e` η ακτίνα είναι ίση με 5; Παρόλα αυτά, κανείς μπορεί να αναρωτηθεί: δηλαδή αν είχαμε δηλώσει μόνο μια μεταβλητή δομής, για παράδειγμα μόνο τον κύκλο `k1`, τότε η παραπάνω εντολή θα ήταν σωστή; (αφού δεν τίθεται θέμα ποιου κύκλου η ακτίνα είναι ίση με 5). Και πάλι η απάντηση είναι: λάθος!
Πρέπει πάντα να προσδιορίζουμε όχι μόνο τη μεταβλητή της δομής αλλά και την ίδια τη δομή στην οποία αναφερόμαστε. Έτσι, αν θέλαμε να κάνουμε ίση με 5 την ακτίνα του κύκλου `k1`, θα γράφαμε:

✓
```
k1.aktina = 5;
```

Παρομοίως, αν θέλαμε να κάνουμε την ακτίνα του κύκλου `m` ίση με 8, θα γράφαμε:

✓
```
m.aktina = 8;
```

Συμπέρασμα: όλοι οι κύκλοι (δηλαδή οι δομές) που προέρχονται από τον ορισμό δομής `kyklos` έχουν ακτίνες – και φυσικό είναι αφού προέρχονται από το ίδιο «καλούπι». Όμως ο κάθε κύκλος έχει τη δική του ξεχωριστή ακτίνα – ΔΕΝ μοιράζονται όλοι οι κύκλοι την ίδια ακτίνα!
Το ίδιο πρέπει να κάνουμε και για τις υπόλοιπες μεταβλητές που υπάρχουν στον ορισμό δομής, δηλαδή τις συντεταγμένες `x`, `y` και το εμβαδόν όταν θέλουμε να τις χρησιμοποιήσουμε σε πράξεις.
Έτσι η γενική μορφή με την οποία θα δίνουμε τιμή σε μια μεταβλητή που περιέχεται σε μια δομή ή θα τη χρησιμοποιούμε σε πράξεις ή θα την εμφανίζουμε στην οθόνη είναι με την τελεία ανάμεσα:
δομή.μεταβλητή
Μερικά παραδείγματα:

```
k1.aktina = 5;
scanf("%d", &k1.aktina);
k1.embadon = 3.14 * k1.aktina * k1.aktina;
printf("H AKTINA EINAI ISH ME %d", k1.aktina);
```

Ακόμη ένα «λεπτό» σημείο: Από απροσεξία, μπορεί κανείς να γράψει μια εντολή τέτοιας μορφής:

✘
```
kyklos.aktina = 5;
```
ή

✘
```
kyklos.embadon = 3.14 * kyklos.aktina * kyklos.aktina;
```
κλπ

Αυτό είναι λάθος!! Ο `kyklos` είναι ορισμός δομής από τον οποία δημιουργούνται κύκλοι, όπως ο `k1`, `m`, `e`. Ο `kyklos` είναι παρόμοιος με το `int`, το `float`, κλπ. Θα γράφατε ποτέ μια εντολή της μορφής:

✘
```
int = 7;
```

ελπίζω πως όχι!!

Μπορούμε να δημιουργήσουμε από έναν ορισμό δομής **πίνακες δομών**. Στο παράδειγμα με τον κύκλο, μπορούμε να δημιουργήσουμε έναν πίνακα δομών, δηλαδή έναν πίνακα κύκλων, με την εντολή:

```
struct kyklos k[10];
```

Τώρα ο k είναι πίνακας που περιλαμβάνει 10 κύκλους. Έτσι αποφεύγουμε τον κόπο να δηλώσουμε ξεχωριστά 10 διαφορετικούς κύκλους, όπως k1, m, e, g, κλπ.

Και πως αναφερόμαστε στις μεταβλητές του κάθε κύκλου τώρα; Για παράδειγμα, αν θέλω να κάνω την ακτίνα του τελευταίου κύκλου ίση με 20, ποια θα ήταν η εντολή; Η απάντηση είναι:

✓
```
k[9].aktina = 20;
```

Μπορούμε να δημιουργήσουμε και **δείκτες που να προέρχονται από έναν ορισμό δομής**. Ένας τέτοιος δείκτης θα μπορεί να δείχνει σε δομές μέσα στη μνήμη του υπολογιστή. Για παράδειγμα, ο δείκτης k που δηλώνεται παρακάτω:

```
struct kyklos *k;
```

μπορεί να δείχνει σε κύκλους μέσα στη μνήμη του υπολογιστή. Φυσικά, θα πρέπει να έχετε δεσμεύσει μνήμη με τη συνάρτηση malloc για να μπορέσει ο δείκτης να «δείξει» σ' αυτή τη μνήμη. Η παρακάτω εντολή δεσμεύει μνήμη αρκετή για 4 κύκλους:

```
k = (kyklos *)malloc(sizeof(kyklos) * 4);
```

Ο δείκτης k δείχνει στον 1ο κύκλο από τους τέσσερις.

Τώρα όμως πως θα αναφερόμαστε στις μεταβλητές που περιέχει η κάθε δομή-κύκλος; Όταν έχουμε δείκτη δομής, χρησιμοποιούμε συνήθως τον συμβολισμό με το «βελάκι» αντί για την τελεία. Δείτε πως:

```
k->aktina = 5;
scanf("%d", &k->aktina);
k->embadon = 3.14 * k->aktina * k->aktina;
printf("H AKTINA EINAI ISH ME %d", k->aktina);
```

Προσέξτε, ότι λόγω του συμβολισμού αυτού, χρειάζεται το σύμβολο & στην scanf, ενώ είχαμε μάθει ότι όταν έχουμε δείκτες δεν χρειάζεται. ***Εναλλακτικός συμβολισμός*** είναι ο εξής (εδώ το & δεν χρειάζεται γιατί παραλείπουμε το αστεράκι στην scanf):

```
(*k).aktina = 5;
scanf("%d", k.aktina);
(*k).embadon = 3.14 * (*k).aktina * (*k).aktina;
printf("H AKTINA EINAI ISH ME %d", (*k).aktina);
```

Μέχρι εδώ είδατε ότι δηλώνουμε τον ορισμό δομής χωριστά από τις μεταβλητές τύπου δομής που προέρχονται από αυτόν. Δείτε τώρα πως μπορείτε να δηλώσετε μεταβλητές-δομές μαζί με τον ορισμό από τον οποίο προέρχονται:

```
struct kyklos {
   int x;
   int y;
   int aktina ;
   float embadon;
} k1, m, e;
```

Αυτός είναι ένας εναλλακτικός τρόπος δήλωσης ισοδύναμος με εκείνον που είδαμε παραπάνω.

Σαν μια σύνοψη των πιο βασικών σημείων που αφορούν στις δομές, δείτε παρακάτω τη δήλωση ενός ορισμού δομής με όνομα circle, τη δήλωση μιας μεταβλητής-δομής που προέρχεται από αυτόν τον ορισμό και ονομάζεται k, και τις εντολές που επιτρέπονται ή που είναι λανθασμένες:

```
struct circle {
  int x, y;
  float aktina;
  char color[20];
```

```
};

void main() {
  struct circle  k;
  float p, e;

✓ k.x = 5;

  printf("DOSE TIMH AKTINAS ");
✓ scanf("%f", &k.aktina);

✓ p = 2 * 3.14 * k.aktina;

  printf("DOSE XROMA KYKLOY ");
✓ scanf("%s", k.color);

✘ circle.y = 4;                                    k.y = 4;

✘ e = 3.14 * circle.aktina * circle.aktina;
                          e = 3.14 * k.aktina * k.aktina;

✘ x.k = 7 ;                                        k.x = 7 ;

✘ circle.k = 7 ;
}
```

Συγκρίνετε τη χρήση τριών διαφορετικών μορφών δομής, με τη βοήθεια του παρακάτω πίνακα.

Απλή δομή k

```
struct circle {
  int x, y;
  float aktina ;
  char color[20] ;
} ;


void main() {
  struct circle  k;
  float p;



  k.x = 5;




  printf("DOSE TIMH AKTINAS ");
  scanf("%f", &k.aktina);


  p = 2 * 3.14 * k.aktina;
}
```

Δομή – πίνακας k[3]

```
struct circle {
  int x, y;
  float aktina ;
  char color[20] ;
} ;


void main() {
  struct circle  k[3];
  float p;



  k[0].x = 5;
  k[1].x = 9;          } ή με for
  k[2].x = 7;

  for(i=0; i<=2; i++) {
    printf("DOSE TIMH AKTINAS ");
    scanf("%f", &k[i].aktina);
  }

  p = 2 * 3.14 * k[0].aktina;
  p = 2 * 3.14 * k[1].aktina;   } ή με for
  p = 2 * 3.14 * k[2].aktina;
}
```

Δείκτης σε δομή *k

```
struct circle {
  int x, y;
  float aktina ;
  char color[20] ;
} ;


void main() {
  struct circle  *k;
  float p;

  k = (circle*)malloc(3*sizeof(circle));

  k->x = 5;
  (k+1)->x = 9;          } ή με for
  (k+2)->x = 7;

  for(i=0; i<=2; i++) {
    printf("DOSE TIMH AKTINAS ");
    scanf("%f", &(k+i)->aktina);
  }

  p = 2 * 3.14 * k->aktina;
  p = 2 * 3.14 * (k+1)->aktina;   } ή με for
  p = 2 * 3.14 * (k+2)->aktina;
}
```

Άσκηση 26. Χρησιμοποίηση δομών (structures).

- Να δημιουργήσετε έναν **ορισμό δομής** ο οποίος να περιέχει ***το όνομα, τον αριθμό ημερών εργασίας*** και ***τον μισθό*** ενός εργαζόμενου. Να επιλέξτε κατάλληλα τους τύπους των παραπάνω μεταβλητών.

Να γράψετε ένα πρόγραμμα, το οποίο:

1. Να ζητά και διαβάζει από το πληκτρολόγιο το **όνομα** και τον **αριθμό ημερών εργασίας** ενός εργαζόμενου και να τα αποθηκεύει σε μια μεταβλητή-δομή που θα έχετε ήδη δηλώσει και θα προέρχεται από τον παραπάνω ορισμό.
2. Έπειτα, να υπολογίζει τον μισθό που αναλογεί στον εργαζόμενο αυτό, λαμβάνοντας υπ' όψη τον αριθμό των ημερών εργασίας και το ημερομίσθιο (που δίνεται παρακάτω). Ο μισθός να αποθηκευθεί στην αντίστοιχη μεταβλητή της δομής.
3. Να τυπώνει το όνομα, τον μισθό (με 2 δεκαδικά ψηφία) και τον αριθμό των ημερών εργασίας για τον εργαζόμενο, σε χωριστές γραμμές και με ανάλογα μηνύματα.

Το ημερομίσθιο να δηλωθεί σαν ***σταθερά*** με τιμή 40,5 ευρώ.

```
#include <stdio.h>
#define HMER_MISTH  40.5


struct stoix_erg {
  char onoma[25];
  int imeres_erg;
  float misthos;
};


void main() {
  struct  stoix_erg  ergaz;

  printf("DOSE TO ONOMA TOY ERGAZOMENOY: ");
  gets(ergaz.onoma);

  printf("DOSE HMERES ERGASIAS: ");
  scanf("%d", &ergaz.imeres_erg);

  ergaz.misthos = ergaz.imeres_erg * HMER_MISTH;

  printf("\nO ERGAZOMENOS %s\n", ergaz.onoma);
  printf("PLHRONETAI %.2f EYRO\n", ergaz.misthos);
  printf("GIA %d HMERES ERGASIAS", ergaz.imeres_erg);
}
```

Άσκηση 27. Πίνακες 2 διαστάσεων.

Να γραφεί ένα πρόγραμμα στη γλώσσα C που:

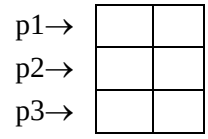
- να ζητά από τον χρήστη τα στοιχεία (αριθμούς) ενός δυδιάστατου πίνακα ακεραίων και διαστάσεων 3x2. Ο χρήστης θα δώσει τους αριθμούς έναν-έναν, κατά γραμμές. Στη συνέχεια να κάνει το ίδιο για έναν πίνακα διαστάσεων επίσης 3x2.
- Στη συνέχεια, το πρόγραμμα **να προσθέτει** τους δύο πίνακες, σύμφωνα με τους κανόνες των μαθηματικών, δηλαδή τα αντίστοιχα στοιχεία τους, και να αποθηκεύει τα αθροίσματα σε έναν τρίτο πίνακα.
- Τέλος, να τυπώνονται και οι 3 πίνακες, σε δυδιάστατη μορφή, ο ένας κάτω από τον άλλο. Προσέξτε ώστε οι αριθμοί του κάθε πίνακα να είναι στοιχισμένοι κατακόρυφα.

```
#define  N  3    /* Orizontia diastasi pinakon */
#define  M  2    /* Katakorifi diastasi pinakon */
#include <stdio.h>

void main() {
 int x[N][M], y[N][M], z[N][M], i, j;

 for(i=0; i<N; i++)                /* Anagnosi stoixeion 1ou pinaka */
  for(j=0; j<M; j++) {
   printf("DOSE STOIXEIO (%d,%d) 1ou PINAKA: ", i, j);
   scanf("%d", &x[i][j]) ;
  }

 for(i=0; i<N; i++)                /* Anagnosi stoixeion 2ou pinaka */
  for(j=0; j<M; j++) {
   printf("DOSE STOIXEIO (%d,%d) 2ou PINAKA: ", i, j);
   scanf("%d", &y[i][j]) ;
  }

 for(i=0; i<N; i++)                /* Ypologismos athroismatos */
  for(j=0; j<M; j++)
   z[i][j] = x[i][j] + y[i][j];

 for(i=0; i<N; i++) {              /* Ektyposi 1ou pinaka */
  for(j=0; j<M; j++)
    printf("%7d", x[i][j]);
  printf("\n");
 }
 printf("\n\n");                   /* Afinoume keno anamesa stous pinakes */

 for(i=0; i<N; i++) {              /* Ektyposi 2ou pinaka */
  for(j=0; j<M; j++)
    printf("%7d", y[i][j]);
  printf("\n");
 }
 printf("\n\n");                   /* Afinoume keno anamesa stous pinakes */

for(i=0; i<N; i++) {               /* Ektyposi 3ou pinaka */
  for(j=0; j<M; j++)
    printf("%7d", z[i][j]);
  printf("\n");
 }
```

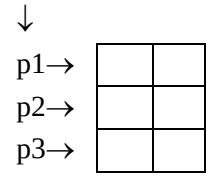
```
}
```

Ωστόσο, μπορεί να μην ξέρουμε πόσους αριθμούς θα καταχωρήσουμε σε έναν δυδιάστατο πίνακα. Σ' αυτή την περίπτωση μάθαμε ότι μπορούμε να κάνουμε **δυναμική δέσμευση μνήμης**, όταν θα τρέξει το πρόγραμμα ο χρήστης. Η δυναμική δέσμευση μνήμης γίνεται με χρήση της συνάρτησης malloc (ή κάποιας συγγενικής της όπως η calloc, κλπ) και ενός δείκτη. Είδαμε ήδη πως γίνεται να χρησιμοποιήσουμε δυναμική δέσμευση μνήμης στη θέση ενός μονοδιάστατου πίνακα, αλλά τι γίνεται με τους δυδιάστατους πίνακες;

Ας υποθέσουμε ότι θέλουμε να δεσμεύσουμε μνήμη για ένα πίνακα ακεραίων 3x2, δηλαδή 3 γραμμές και 2 στήλες. Το σκεπτικό είναι απλό: θα δεσμεύσουμε μνήμη για κάθε γραμμή, η οποία είναι σαν ένας μονοδιάστατος πίνακας. Αλλά έτσι θα χρειαστούμε 3 απλούς δείκτες, όπου ο καθένας θα δείχνει και σε μια γραμμή του πίνακα (βλέπε διπλανό σχήμα). Όμως έτσι θα πρέπει να χρησιμοποιούμε στις εντολές διαφορετικό όνομα πίνακα (ή δείκτη) αναλόγως σε ποια γραμμή αναφερόμαστε. Π.χ. για το πάνω αριστερά στοιχείο του πίνακα θα είχαμε: p1[0], ενώ θα θέλαμε να αναφερόμαστε σ' αυτό σαν p[0][0], όπου p να ονομάζεται όλος ο πίνακας.



Η λύση έρχεται από την ίδια γλώσσα C, η οποία επιτρέπει τη δήλωση **δείκτη σε δείκτη**. Ένας απλός δείκτης δείχνει σε δεδομένα (σε τιμές). Ένας δείκτης σε δείκτη, όπως λέει και το όνομά του, θα δείχνει σε άλλους δείκτες και όχι σε δεδομένα. Αν λοιπόν έχω έναν τέτοιο δείκτη p, μπορώ να τον κάνω να δείξει στους 3 δείκτες p1, p2 και p3 (βλέπε διπλανό σχήμα). Τότε μπορώ να αναφερθώ στο πάνω αριστερά στοιχείο του πίνακα ως εξής: p[0][0], και γενικότερα, για οποιοδήποτε στοιχείο του πίνακα θα έχω: p[i][j].

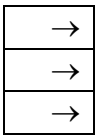


Επιπλέον, μ' αυτόν τον τρόπο δεν χρειάζεται να δηλώσω τους 3 δείκτες που θα δείχνουν στις αντίστοιχες γραμμές (δηλαδή τον p1, p2 και p3). Απλώς θα χρησιμοποιήσω τον δείκτη σε δείκτη p για να δεσμεύσω μνήμη για 3 απλούς δείκτες.

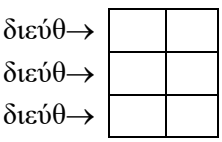
Δείτε τις εντολές που χρειάζονται παρακάτω:

1. Αρχικά, θα πρέπει να δημιουργήσουμε έναν δείκτη σε δείκτη. Αυτός δηλώνεται βάζοντας πριν το όνομά του δύο αστεράκια αντί για ένα.
   ```
   int **p;
   ```

2. Έπειτα, χρησιμοποιώντας τον p, θα δεσμεύσουμε μνήμη για 3 δείκτες, αφού ο ζητούμενος πίνακας έχει 3 γραμμές (βλέπε διπλανό σχήμα). Προσέξτε: οι 3 δείκτες δεν δείχνουν ακόμη σε καμμία θέση μνήμης.
   ```
   p = (int**)malloc(3 * sizeof(int*));
   ```



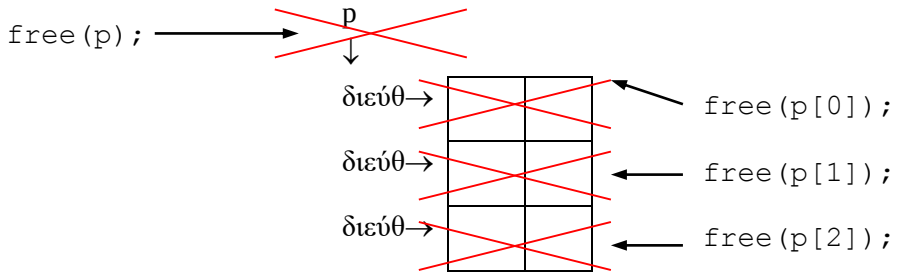
3. Στη συνέχεια, θα γεμίσουμε την μνήμη που δεσμεύσαμε παραπάνω με 3 διευθύνσεις, η καθεμιά από τις οποίες θα δείχνει σε μνήμη που να χωρά 2 ακέραιους, αφού κάθε γραμμή του ζητούμενου πίνακα θα έχει 2 στήλες. Θα χρειαστούμε επαναληπτική εντολή, αφού θα γίνει χωριστή δέσμευση για κάθε γραμμή του πίνακα. Η καθεμιά από τις 3 διευθύνσεις στις οποίες δείχνει ο p συμβολίζονται σαν p[0], p[1] και p[2].
   ```
   for(i=0; i<3; i++)
       p[i] = (int*)malloc(2 * sizeof(int));
   ```



4. Τώρα μπορούμε να χρησιμοποιήσουμε τον δείκτη p σαν να ήταν δυδιάστατος πίνακας και να καταχωρήσουμε τιμές σ' αυτόν, π.χ. `scanf("%d", &p[i][j]).`

5. Όταν δεν χρειάζεται άλλο τη μνήμη, θα την αποδεσμεύσουμε σταδιακά, όπως και τη δεσμεύσαμε. Αρχικά θα απελευθερώσουμε επαναληπτικά τη μνήμη για κάθε γραμμή του πίνακα και έπειτα θα αποδεσμεύσουμε και τη μνήμη των δεικτών προς κάθε γραμμή.

```
for(i=0; i<3; i++)
    free(p[i]);
free(p);
```



Αντί να γράψουμε το πρόγραμμα 27 χρησιμοποιώντας δυναμική δέσμευση μνήμης (κάτι που μπορείτε να επιχειρήσετε μόνοι σας), θα κάνουμε ένα πιο απλό πρόγραμμα. Έστω ότι θα δεσμεύσουμε μνήμη για ένας δυδιάστατο πίνακα (ο χρήστης θα ερωτηθεί για τις διαστάσεις του) στον οποίο θα αποθηκεύσουμε ακέραιους. Στη συνέχεια θα αντικαταστήσουμε τους αριθμούς με τα τετράγωνά τους (στον ίδιο πίνακα) και τελικά θα τα τυπώσουμε στην οθόνη. Φυσικά, στο τέλος η μνήμη θα πρέπει να αποδεσμευτεί.

```
#include <stdio.h>
#include <stdlib.h>
main() {
 int **p, i, j, n, m;

 printf("DOSE DIASTASEIS (n,m) PINAKA: ");
 scanf("%d %d", &n, &m);

 p = (int**)malloc(n * sizeof(int*));   /* Desmeusi mnimis */
 for(i=0; i<n; i++)
   p[i] = (int*)malloc(m * sizeof(int));

 for(i=0; i<n; i++)            /* Anagnosi apo to pliktrologio */
   for(j=0; j<m; j++) {
      printf("DOSE STOIXEIO (%d,%d) PINAKA: ", i, j);
      scanf("%d", &p[i][j]);
    }

 for(i=0; i<n; i++)            /* Ypologismos tetragonon */
   for(j=0; j<m; j++)
      p[i][j] = p[i][j]*p[i][j];

 for(i=0; i<n; i++) {           /* Ektyposi pinaka */
   for(j=0; j<m; j++)
     printf("%7d", p[i][j]);
   printf("\n");
 }

}   /* Telos tou main */
```