ΜΑΘΗΜΑ 13

ΣΥΝΑΡΤΗΣΕΙΣ ΚΑΙ ΑΝΑΔΡΟΜΗ 2/2

Γιώργος Διάκος - Full Stack Developer

2. Πως Γράφουμε Συναρτήσεις

4. Κλήση Συνάρτησης

- Αφού γράψουμε την συνάρτησή μας, έχουμε δικαίωμα να την καλέσουμε οπουδήποτε μέσα στο πρόγραμμα μας. Για να την καλέσουμε:
  - Γράφουμε το όνομα της και διοχετεύουμε κατάλληλα ορίσματα που θα είναι:
    - Είτε απευθείας συγκεκριμένες αριθμητικές τιμές.
    - Είτε ονόματα μεταβλητών που χρησιμοποιούμε ήδη στο πρόγραμμα μας. Προσοχή! Απλά γράφουμε τα ονόματα των μεταβλητών ως ορίσματα και όχι τον τύπο δεδομένων
    - Είτε γενικότερα υπολογιζόμενες παραστάσεις (όπως τις ορίσαμε στο μάθημα 2 που μελετήσαμε τον τελεστή εκχώρησης)

- Δείτε το παράδειγμα της επόμενης διαφάνειας και εντοπίστε στις κλήσεις των συναρτήσεων, να «διοχετεύονται» ορίσματα είτε αριθμητικά, είτε με μεταβλητές που χρησιμοποιεί η «καλούσα συνάρτηση»

2. Πως Γράφουμε Συναρτήσεις

4. Κλήση Συνάρτησης

```
/* orismata.c: Anadeikniei pos pairname orismata se sinartiseis */

#include <stdio.h>

int square(int x);

main()
{
   int a=5;
   int b=10;
   int teta,tetb,sum;

   teta=square(a);
   tetb=square(b);
   sum=teta+tetb;
   printf("%d^2 + %d^2 = %d",a,b,sum);
}

int square(int x)
{
  int y;
  y=x*x;
  return y;
}
```

2. Πως Γράφουμε Συναρτήσεις

4. Κλήση Συνάρτησης

- Όπως φαίνεται και από το παράδειγμα, η επιστρεφόμενη τιμή της συνάρτησης γίνεται με την εντολή return:

```
return τιμή;
```

  - Η δεσμευμένη λέξη return ακολουθείται από την τιμή, η οποία μπορεί να είναι οποιαδήποτε υπολογιζόμενη παράσταση (σταθερά, μεταβλητή, υπολογισμός, ή ακόμη και συνάρτηση)
  - Σημαντικό: Η εκτέλεση της εντολής return σταματά επιτόπου την εκτέλεση της συνάρτησης, υπολογίζει την τιμή και επιστρέφει στην καλούσα συνάρτηση.

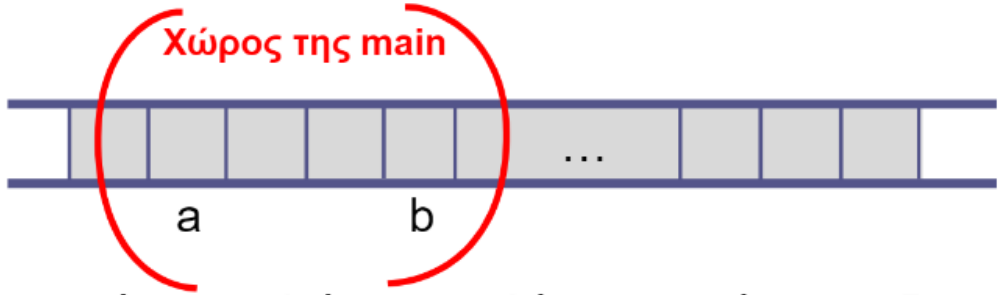
  - Ισχύει επίσης ότι σε μια συνάρτηση που επιστρέφει void, μπορούμε να σταματήσουμε την εκτέλεση της με την εντολή:

```
return;
```

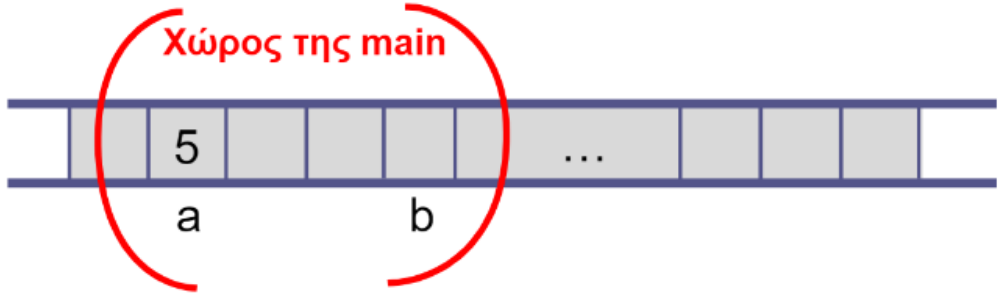
3. Πως Λειτουργούν οι Συναρτήσεις

1. Συναρτήσεις και Χώρος στη Μνήμη

- Είναι σημαντικό να καταλάβουμε ότι κάθε συνάρτηση έχει το δικό της «χώρο» στη μνήμη, στον οποίο αποθηκεύει τις μεταβλητές της.
- Για παράδειγμα έστω το τμήμα κώδικα που φαίνεται στα δεξιά
- Όταν ξεκινάει να εκτελείται ο κώδικας υπάρχει ο χώρος αποθήκευσης μόνο για την main!



- Έπειτα όταν εκτελείται η εντολή αρχικοποίησης a=5, η κατάσταση της μνήμης είναι:

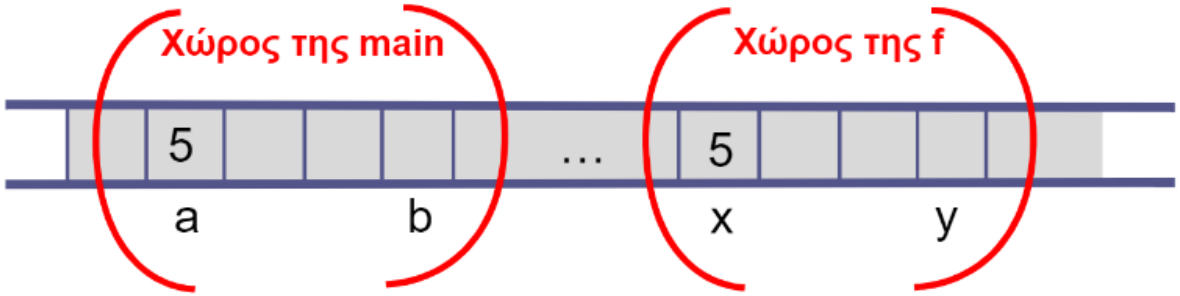



```
int f(int x);

main()
{
  int a=5,b;
  b=f(a);
}

int f(int x)
{
  int y;
  y=x*x;
  return y;
}
```

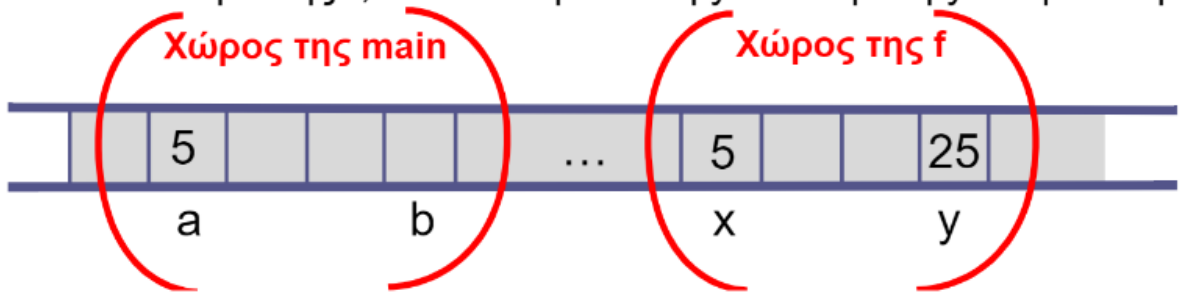
3. Πως Λειτουργούν οι Συναρτήσεις

1. Συναρτήσεις και Χώρος στη Μνήμη

- Έπειτα καλείται η f με όρισμα a.
  - Προσοχή! Αυτό σημαίνει, ότι δημιουργείται χώρος αποθήκευσης για την f.
  - Και στον χώρο αποθήκευσης της f, η μεταβλητή x θα πάρει την τιμή του ορίσματος που βάλαμε, άρα η x θα πάρει την τιμή 5.



  - Είναι σημαντικό να καταλάβουμε ότι από εδώ και πέρα η x δεν έχει καμία σχέση με την a. Έχει τον δικό της χώρο μνήμης και την διαχειρίζεται η f.
- Πλέον στο σώμα της f, καλείται η εντολή y=x*x άρα η y παίρνει την 25

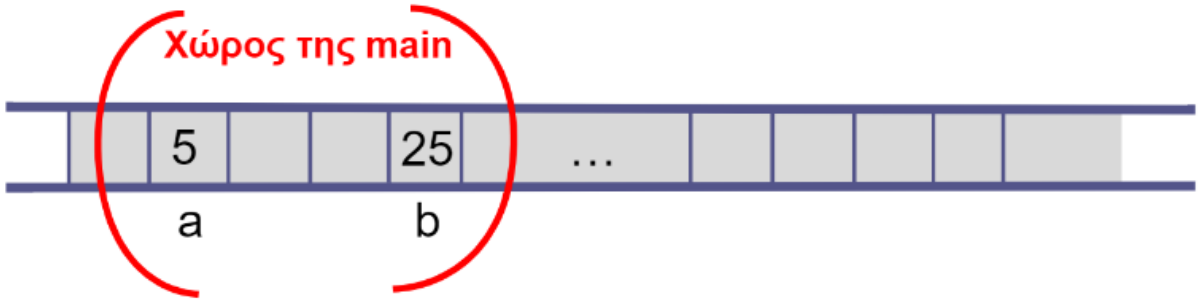



```
int f(int x);

main()
{
  int a=5,b;
  b=f(a);
}

int f(int x)
{
  int y;
  y=x*x;
  return y;
}
```

3. Πως Λειτουργούν οι Συναρτήσεις

1. Συναρτήσεις και Χώρος στη Μνήμη

- Καλείται η εντολή return y. Αυτό σημαίνει ότι επιστρέφουμε στην main!
  - Πλέον η επιστρεφόμενη τιμή (25) αποθηκεύεται στην μεταβλητή b.
  - Είναι σημαντικό ότι μετά την επιστροφή τιμής ο χώρος της f, απελευθερώνεται για να μπορεί να χρησιμοποιηθεί από άλλες συναρτήσεις:




```
int f(int x);

main()
{
  int a=5,b;
  b=f(a);
}

int f(int x)
{
  int y;
  y=x*x;
  return y;
}
```

- Το παράδειγμα αυτό αναδεικνύει δύο σημαντικά θέματα!
  - Κάθε κλήση συνάρτησης δημιουργεί τον δικό της χώρο στην μνήμη!
  - Ο μόνος δίαυλος επικοινωνίας με την καλούσα συνάρτηση είναι τα ορίσματα (στην αρχή) και η επιστρεφόμενη τιμή (στο τέλος)

3. Πως Λειτουργούν οι Συναρτήσεις

2. Περίπλοκα Ορίσματα

- Στην πραγματικότητα η διοχέτευση ορίσματος:
  - είναι απλά, να αρχικοποιηθεί η τιμή του ορίσματος με την τιμή που δίνουμε από την καλούσα συνάρτηση!
- Έτσι έχουμε το δικαίωμα να θέσουμε ως ορίσματα οποιαδήποτε παράσταση της C
  - Τότε απλά θα έχουμε ότι θα υπολογιστεί η τιμή του ορίσματος, και έπειτα η μεταβλητή του ορίσματος θα πάρει την κατάλληλη τιμή.
- Για παράδειγμα στο τμήμα κώδικα:

```
int a=5;
int b;
b=f(2*a);
```

  - Υπολογίζεται πρώτα το όρισμα (είναι η τιμή 10)
  - Έπειτα καλείται η f με όρισμα την τιμή 10

- Ενώ αν η f είναι η συνάρτηση που είδαμε προηγουμένως τότε έχουμε δικαίωμα να γράψουμε ακόμη και το εξής:

```
int a=5;
int b;
b=f(f(a));
```

  - Όπου υπολογίζεται πρώτα το εσωτερικό f(a) (υπολογίζεται σε 25) και μετά υπολογίζεται το f(25) άρα αποθηκεύεται στο b η τιμή 625.

3. Πως Λειτουργούν οι Συναρτήσεις

3. Παραπάνω του ενός ορίσματα

- Αντίστοιχοι είναι οι κανόνες όταν έχουμε μια συνάρτηση που δέχεται πολλά ορίσματα. Πχ. Αν έχουμε ορίσει την συνάρτηση:

```
void func(int x, int y, int z)
```

- Και την καλέσουμε ως εξής:

```
func(1,5,10)
```

  - Τότε η τιμή που θέσαμε πρώτη αποθηκεύεται στην μεταβλητή του 1ου ορίσματος (δηλαδή το x παίρνει την τιμή 1)
  - Η τιμή που θέσαμε δεύτερη αποθηκεύεται στην μεταβλητή του 2ου ορίσματος (δηλαδή το y παίρνει την τιμή 5)
  - Η τιμή που θέσαμε τρίτη αποθηκεύεται στην μεταβλητή του 3ου ορίσματος (δηλαδή το z παίρνει την τιμή 10)

1. Κλήση Συνάρτησης μέσα σε Συνάρτηση

- Όπως είδαμε αφού ορίσουμε μια συνάρτηση μπορούμε να την καλέσουμε οπουδήποτε στον κώδικα.
- Άρα μπορούμε να καλέσουμε μια συνάρτηση που έχουμε ορίσει μέσα σε μια άλλη συνάρτηση!
- Π.χ. ας ορίσουμε μια συνάρτηση που υπολογίζει την $f(x)=2x^2$

```
int f(int x)
{
  int y;
  y=2*x*x;
  return y;
}
```

- Μπορούμε να ορίσουμε την συνάρτηση $g(x)=2x^2+x+1$ ως εξής:

```
int g(int x)
{
  int y;
  y=f(x)+x+1;
  return y;
}
```

2. Αναδρομικές Συναρτήσεις

- Ο όρος **αναδρομή αναφέρεται σε μια συνάρτηση που καλεί τον εαυτό της**!
  - Αυτό είναι απόλυτα νόμιμο και στην C, αφού απλά καλούμε μια συνάρτηση μέσα σε μια συνάρτηση!
- Άρα μια συνάρτηση που στο σώμα της καλεί τον εαυτό της, θα ονομάζεται **αναδρομική συνάρτηση.**
- Η δημιουργία μιας αναδρομικής συνάρτησης είναι πολύ χρήσιμη, ιδίως όταν κατασκευάζουμε πράγματα που ορίζονται αναδρομικά!

- Ας δούμε ένα παράδειγμα:
  - Το παραγοντικό του φυσικού αριθμού n ορίζεται ως:
    - $n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots 2 \cdot 1$
    - π.χ. έχουμε $1! = 1,\ \ 2! = 2 \cdot 1,\ \ 3! = 3 \cdot 2 \cdot 1 = 6,\ \ 4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$ κ.ο.κ.
  - Το παραγοντικό ορίζεται ωστόσο και αναδρομικά ως εξής:
    - $n! = n \cdot (n-1)!$ αν $n > 1$
    - $n! = 1,$ αν $n = 1$

2. Αναδρομικές Συναρτήσεις

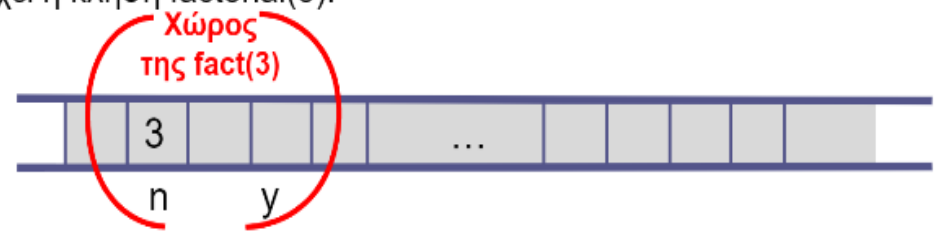
1. Υπολογισμός Παραγοντικού

```
/* factorial.c: Ypologizei to paragontiko
   enos fysikoy */

#include <stdio.h>

int factorial(int n);

main()
{
   int x;
   int res;

   printf("Dwste ton fysiko: ");
   scanf("%d",&x);

   res=factorial(x);

   printf("%d!=%d",x,res);
}
```

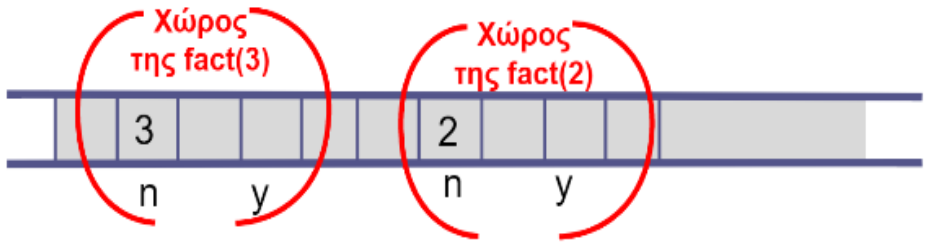
```
int factorial(int n)
{
  int y;

  if (n==1)
    return 1;
  else
  {
    y=factorial(n-1);
    return n*y;
  }
}
```

```
int factorial(int n)
{
  int y;

  if (n==1)
    return 1;
  else
  {
    y=factorial(n-1);
    return n*y;
  }
}
```

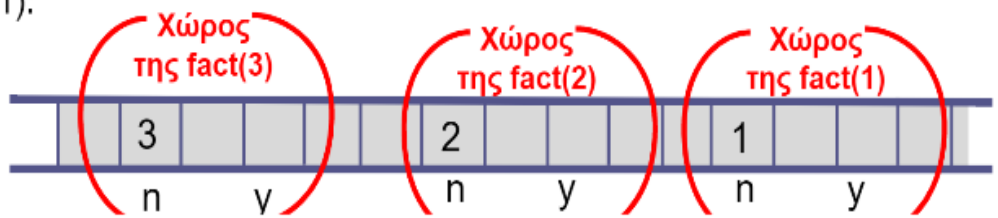
- Στον υπολογισμό μίας αναδρομικής συνάρτησης, καθε αναδρομικη κληση εχει και το δικο της χώρο στη μνήμη.
- Ας δούμε πως τρέχει η κλήση factorial(3):



- Καλεί την factorial(2):

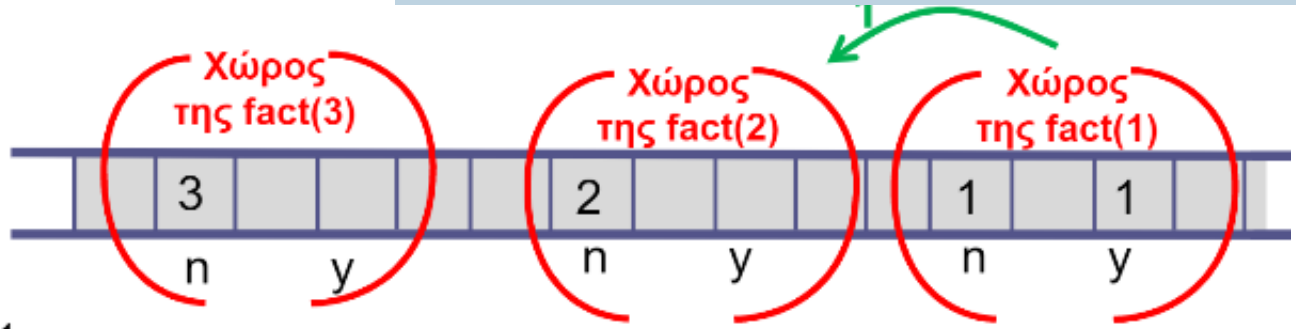


- Καλεί την factorial(1):

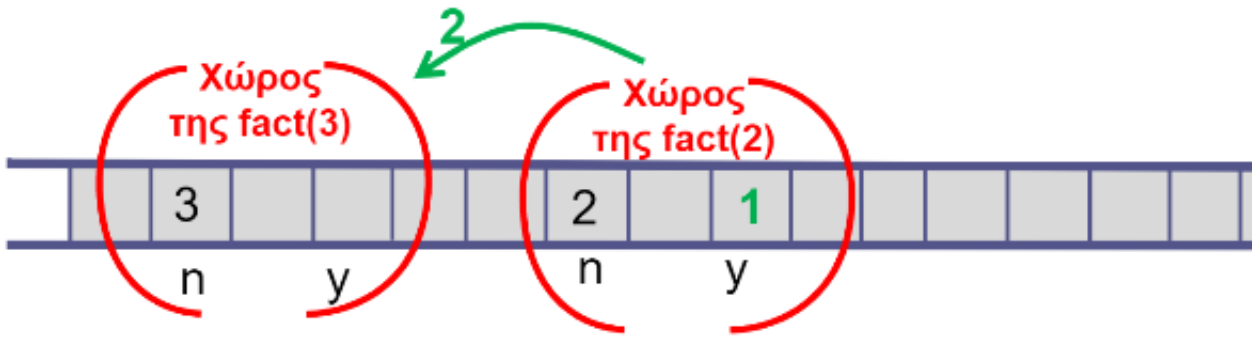


```
int fact (int n)
{
  int y;
  if (n--1) return 1;
  else{
    y=factorial(n-1);
    return n*y;
  }
}
```

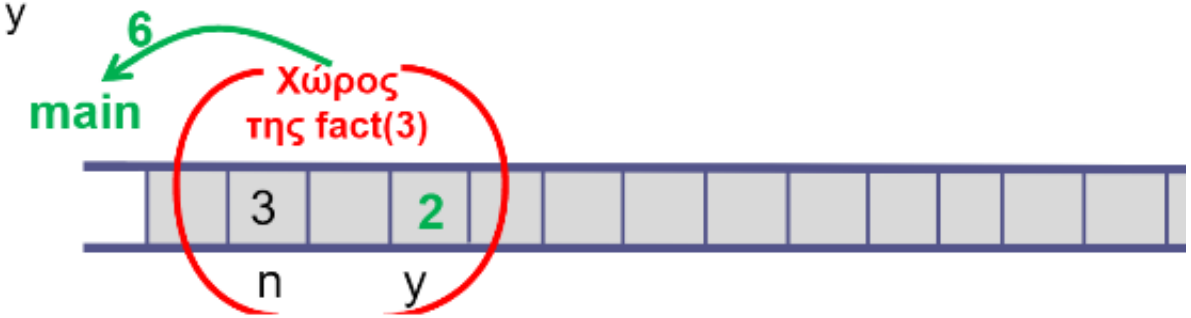
- Η factorial(1) επιστρέφει 1:



- Η factorial(2) αποθηκεύει το 1 στο y
  - έπειτα επιστρέφει 2*1.



- Η factorial(2) αποθηκεύει το 1 στο y
  - έπειτα επιστρέφει 3*2



```
int fact (int n)
{
  int y;
  if (n--1) return 1;
  else{
    y=factorial(n-1);
    return n*y;
  }
}
```

- Για να απεικονίσουμε τις αναδρομικές κλήσεις που γίνονται προτιμάται μία παράσταση όπου κάθε αναδρομική κλήση στοιχίζεται λίγο δεξιότερα.
- Με τον τρόπο αυτό μπορούμε να παρακολουθήσουμε αρκετά ικανοποιητικά την εκτέλεση ενός αναδρομικού κώδικα:

```
ΚΛΗΣΗ fact(3)
(3==1) ΟΧΙ
y=fact(2)
    ΚΛΗΣΗ fact(2)
    (2==1) ΟΧΙ
    y=fact(1)
        ΚΛΗΣΗ fact(1)
        (1==1) ΝΑΙ
        return 1
    y=1
    return 2*1=2
y=2
return 3*2=6
```

ΕΦΑΡΜΟΓΗ 1.

- Όρισε την συνάρτηση :
- Int get_integer(int start , int finish): Θα λαμβάνει ως είσοδο ενα εύρος τιμών [start...finish] και θα διαβάζει εναν ακέραιο σε αυτό το εύρος. Θα επιστρέφει τον αριθμό που διαβάστηκε.
- Όρισε την συνάρτηση main να διαβάζει δυο ακεραίους a ,b στο διάστημα 1...10 και εναν ακέραιο n στο διάστημα 2..5 και θα υπολογίζει την ποσότητα n*(a-b) και θα χρησιμοποιεί τη συνάρτηση που όρισες.

ΕΦΑΡΜΟΓΗ 2.

- Όρισε τις συναρτήσεις :
- Int is_even (int n): Θα επιστρέφει 0 ή 1 ανάλογα με το αν ο αριθμός είναι άρτιος.
- Int is_odd(int n): Θα επιστρέφει 0 ή 1 ανάλογα με το αν ο αριθμός είναι περιττός.
- Int is_square(int n) : Θα επιστρέφει 0 ή 1 ανάλογα με το αν ο αριθμός είναι τετράγωνο ενος φυσικού.
- Int is_cube(int n) : Θα επιστρέφει 0 ή 1 ανάλογα με το αν ο αριθμός είναι κύβος ενος φυσικού.
- Όρισε την main που θα ζητάει από τον χρήστη είτε να εισάγει εναν αριθμό και θα εξετάζει αν ο αριθμός έχει κάποιες από αυτές τις ιδιότητες.
- Output example :

- **Eisagete ton arithmo : 8**
- **Einai Artios**
- **Einai kivos arithmou**

- **Eisagete ton arithmo : 9**
- **Einai perittos!**
- **Einai tetragono arithmou!**

ΕΦΑΡΜΟΓΗ 3.

- Ένας Φυσικός αριθμός λέμε ότι είναι πρώτος αν διαιρείται (ακριβώς) μόνο με τον ευατό του και την μονάδα. Το 1 θεωρείται ότι δεν είναι πρώτος.
- Κατασκεύασε ενα πρόγραμμα το οποίο :
- Θα ορίζει μια συνάρτηση με όνομα isprime() η οποία θα δέχεται ως όρισμα εναν ακέραιο αριθμό. , θα εξετάζει αν είναι πρώτος και θα επιστρέφει 1 αν είναι πρώτος και 0 αν δεν είναι.
- Η main θα διαβάζει δυο φυσικούς (ελέγχοντας στην είσοδο να είναι >0) που θα ορίζουν την αρχή και το τέλος ενός κλειστού διαστήματος (π.χ a=5 , b=8) και θα τυπώνει τους φυσικούς σε αυτό το διάστημα που είναι πρώτοι.
- Output example :
- **Eisagete tin arxi tou diastimatos : 5**
- **Eisagete to peras tou diastimatos :15**
- **To 5 einai prwtos**
- **To 7 einai prwtos**
- **To 11 einai prwtos**
- **To 13 einai prwtos**

ΕΦΑΡΜΟΓΗ 4.

- Η ακολουθία fibonacci ορίζεται ως :
- Fn = Fn-1 + Fn-2 , για n>2
- F2=1
- F1=1
- Για παράδειγμα έχουμε F1 = 1.F2=1,F3=2,F4=3,F5=5,F6=8 κ.ο.κ
- Όρισε την συνάρτηση int fibonacci(int n) που δέχεται ως όρισμα εναν φυσικό και επιστρέφει το n-οστο fibonacci.
- Έπειτα κατασκεύασε μια main που να διαβάζει απο τον χρήστη έναν ακέραιο και υπολογίζει και επιστρέφει τον αριθμό fibonacci του αριθμού που εισήγαγε ο χρήστης.

ΕΦΑΡΜΟΓΗ 5.

- Ο Αλγόριθμος του Ευκλείδη για την εύρεση του Μέγιστου Κοινού Διαιρέτη δυο (φυσικών) αριθμών :
- Ξεκινά με ενα ζεύγος φυσικών και σχηματίζει ενα νέο ζευγάρι με τον μικρότερο αριθμό και την διαφορά του μικρότερου από τον μεγαλύτερο αριθμό.
- Η διαδικασία επαναλαμβάνεται εωσότου οι αριθμοί γίνουν ίσοι. Ο αριθμός αυτός είναι ο ΜΚΔ των αρχικών αριθμών.
- Μαθηματικά ο ΜΚΔ(a,b) όπου a,b είναι φυσικοί :
- Είναι ίσο με a αν a=b
- Είναι ίσο με ΜΚΔ(a ,b-a), αν a<b
- Είναι ίσο με ΜΚΔ(a-b ,b) , αλλιώς
- Κατασκεύασε ενα πρόγραμμα που θα υλοποιεί με μια αναδρομική συνάρτηση τον υπολογισμό του ΜΚΔ και μια συνάρτηση main που θα ζητάει από τον χρήστη να εισάγει τους δυο φυσικούς , θα κάνει κατάλληλη κλήση της συνάρτησης και θα τυπώνει τον ΜΚΔ των αριθμών.

ΤΕΛΟΣ ΜΑΘΗΜΑΤΟΣ

ΣΥΝΑΡΤΗΣΕΙΣ ΚΑΙ ΑΝΑΔΡΟΜΗ 2/2

Γιώργος Διάκος - Full Stack Developer