Η εντολή if

Σύνταξη μιας εντολής if

```
if (συνθήκη) /* Η συνθήκη είναι λογική έκφραση */
        {
        εντολή1;
        εντολή2;
        . . .
        εντολήN;
        }
```

Το σώμα της if αποτελείται από ένα σύνολο εντολών οι οποίες θα εκτελεστούν μόνο αν η συνθήκη της if είναι αληθής

Παράδειγμα1

```
if (x <= 10 && x > 5)   {
        a = b + 2;
        y = x;
        }
```

Παράδειγμα2

```
if (x <= 10 && x > 5);
y = x;
```

ΠΡΟΣΟΧΗ : **ΟΧΙ ;** *εκτελείται πάντα*

Πρόγραμμα που εμφανίζει αποτελέσματα συγκρίσεων

```
#include <stdio.h>
int main()
{
  int  x, y;

  printf("Input two integers x and y: \n");
  scanf("%d %d",&x,&y);

  /* Compare x and y and show the result */

  if (x == y)
     printf("x is equal to y\n");

  if (x > y)
     printf("x is greater than y\n");

  if (x < y)
     printf("x is less than y\n");

  return 0;
}
```

```
Input two integers x and y:  25  42 <enter>
x is less than y
```

Η εντολή if – else

Σύνταξη μιας εντολής if - else

```
if (συνθήκη)
        εντολή1; /* απλή εντολή ή σώμα εντολών {. . .} */
else
        εντολή2; /* απλή εντολή ή σώμα εντολών {. . .} */
```

Παράδειγμα υπολογισμού απόλυτης τιμής

```
. . .
int x, y;
. . .
scanf("%d",&x);
if (x < 0)
        y = -x;
else
        y = x;
printf("ABS(%d) = %d\n",x,y);
```

Φωλιασμένες if [1/2]
nested if

Παράδειγμα φωλιασμένης if

```
if (συνθήκη1)
        εντολή1;
else
        if (συνθήκη2)
                εντολή2;
        else
                εντολή3;
```

Παράδειγμα βελτίωσης του προγράμματος σύγκρισης των x, y

```
. . .
if (x == y)
        printf("x is equal to y\n");
else
        if (x > y)
                printf("x is greater than y\n");
        else
                printf("x is less than y\n");
```

Φωλιασμένες if [2/2]
nested if

- Άλλο παράδειγμα φωλιασμένων if με ισοδύναμη επίλυση του προηγούμενου προβλήματος σύγκρισης των x, y

```
if (x >= y)
   if (x > y)
      printf("x is greater than y\n");
   else
      printf("x is equal to y\n");
else
   printf("x is less than y\n");
```

- Γενικά, η κάθε else αντιστοιχίζεται με την πλησιέστερη if εκτός και αν η if απομονώνεται σε σώμα εντολών, όπως στο παρακάτω παράδειγμα:

```
if (x >= y)
{
   if (x > y)
      printf("x-y: positive difference\n");
}
else
   printf("x-y: negative difference\n");
```

Ο τριαδικός υπό συνθήκη τελεστής ? :

- Αντιστοιχεί στην εντολή if - else:

```
if (συνθήκη)
   εντολή1;
else
   εντολή2;
```

⟺

```
συνθήκη ? έκφραση1 : έκφραση2
```

Πρώτα αποτιμάται η συνθήκη. Αν είναι αληθής τότε αποτιμάται η έκφραση1 ειδάλλως η έκφραση2.

- Παραδείγματα

```
y = (x < 0) ? -x : x;    /* y = abs(x) */

mx = (y > z) ? y : z;    /* mx = max(y,z) */

printf("Exete %d antikeimen%c.\n",n, (n==1) ? 'o' : 'a');

(b == 10) ? printf("Ten\n") : printf("Other than ten\n");

int_flag ? scanf("%d",&i) : scanf("%f",&f);
```

Πολλαπλές επιλογές: η εντολή switch

- **Σύνταξη της switch**

```
switch (έκφραση)
{
   case τιμή1:  εντολές1;  break;
   case τιμή2:  εντολές2;  break;
   . . .
   default:     εντολές;
}
```

break: προαιρετική εντολή που τερματίζει τη switch.

- Παράδειγμα

```
...
char letter;     ...    scanf("%c",&letter);
switch (letter)
{
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u': printf("Letter %c is a vowel!\n",letter);  break;
  default:  printf("Letter %c is a consonant!\n",letter);
}
```

Good programming style [1/7]

Γενικά

- Every program you write that you intend to keep around for more than a couple of hours ought to have documentation in it.
- Don't talk yourself into putting off the documentation.
- A program that is perfectly clear today is clear only because you just wrote it.
- Put it away for a few months, and it will most like take you a while to figure out what it does and how it does it. If it takes you a while to figure it out, how long would it take someone else to figure it out?

  Programming style is a term used to describe the effort a programmer should take to make his or her code easy to read and easy to understand.
- Good organization of the code and meaningful variable names help readability, and liberal use of comments can help the reader understand what the program does and why.

- *http://www.cs.arizona.edu/~mccann/style_c.html*

Good programming style [2/7]

INDENTATION, («φώλιασμα»)

- One immediate problem that this program has is that it does not adhere to a consistent indentation pattern. There are dozens of indentation styles that you could adopt, and some general ideas are common to most of them:

- The style is applied consistently throughout the program.
- Code within a block (e.g., inside a loop, or in the body of a subprogram) should be indented.
- If a block is nested within another block the inner block's body should be indented relative to the enclosing block.
- Avoid excessive "stairstep" indentation (such as you often see with groups of nested IF statements) because this will force you to attempt to squeeze code to fit on just the right half of the screen/page. If stairstepping becomes a problem, reduce the number of spaces per indentation (from 8 to 4, for example) or switch to a vertical style temporarily.

- Πάρα πολύ καλό link:
- *http://www.cs.arizona.edu/~mccann/indent_c.html*

Good programming style [3/7]

MEANINGFUL VARIABLE NAMES

- Another impediment to program readability is that the program's identifiers (variable names, subprogram names, etc.) are mostly meaningless. Again, there are some principles :

- Use good, meaningful names, but don't go overboard. If you have a variable in your program that holds the number of hours an employee works in a week, you might call it HOURS, although that name still leaves a lot of doubt. On the other hand, a name such as HOURS_WORKED_IN_A_WEEK is much more descriptive but contains 22 symbols; A compromise such as HOURS_PER_WEEK is a good solution.
- Either use underscores as part of names, as shown above. If not, you can still improve name readability by mixing the case of the letters in the name. For example, 'HoursPerWeek' is much easier to read than 'hoursperweek'.
- Always place a comment with each variable name declaration. The comment should give a brief phrase or sentence that explains the purpose of the variable
- Common abbreviations are often acceptable in variable names. For example: HRS_PER_WEEK. But don't get carried away; HRS_P_WK simply is not a good name.
- Although this isn't really a point about naming, it is related: Don't reuse variables in a subprogram. For example, you may need a loop control integer variable at the top of the block, and you might also need a place to store a value for a while at the bottom of the block. Resist the temptation to reuse that loop control variable as the temporary holder.

Good programming style [4/7]

INTERNAL DOCUMENTATION

- Internal documentation is the set of comments that are included within the code to help clarify algorithms. Some students take internal documentation to mean that they should comment each line of code! This is obviously an example of overdoing a good idea. Any programmer knows how to increment a value in a variable; there's no reason to explain trivial operations such as that. The value of some good internal documentation should be clear by looking at the latest version of our sample program. Even with the good code organization and variable names, the function of this program is still not obvious.
- Complex sections of code and any other parts of the program that need some explanation should have comments just ahead of them or embedded in them
- A critical point: Documentation, and internal documentation in particular, should be written and included in the program as the code is being written. Students tend to get in the habit of writing the code and then tossing in some documentation only if they have time before the due date. This makes documenting seem even more boring and tedious that it already is, and students who rush the documentation at the last minute usually do a very mediocre job. Documentation should be written when the code is being written, and should be typed in as the code is typed in.

Good programming style [5/7]

EXTERNAL DOCUMENTATION [1/2]

- In a professional programmer's shop, large projects are documented in great detail, not only with comments in the code but with descriptions that are maintained separately from the code. In such an environment, programmers are often asked to fix problems in code that they didn't write. Many times, the author of the code isn't even with the company any longer.
- External documentation doesn't deal with details of the code. Instead, it serves as a general description of the project, including such information as what the code does, who wrote it and when, which common algorithms it uses, upon which other programs or libraries it is dependent, which systems it was designed to work with, what form and source of input it requires, the format of the output it produces, etc. Often the external documentation will include structure charts of the outline of the program that were produced when the program was being designed.

Good programming style [6/7]

EXTERNAL DOCUMENTATION [2/2]

- In most programming classes, (έτσι και στη δικιά μας), it is impractical for instructors to require large amounts of external documentation for programs that are only a few hundred lines long. Instead, it is common for instructors to require that a small amount of external documentation be included at the top of the program in the form of a large block comment. This condensed version should include at least the following pieces of information:
  - Your name, the course name, assignment name/number, instructor's name, and due date.
  - Description of the problem the program was written to solve.
  - Approach used to solve the problem. This should always include a brief description of the major algorithms used, or their names if they are common algorithms.
  - The program's operational requirements: Which language system you used, special compilation information, where the input can be located on disk, etc.
  - Required features of the assignment that you were not able to include.
  - Known bugs should be reported here as well. If a particular feature does not work correctly, it is in your best interest to be honest and complete about your program's shortcomings.

Good programming style [7/7]

A FINAL WORD

In a programming class, instructors don't want you to write the most efficient programs; they'd much rather you learn the material well and learn good programming style at the same time. Never pursue efficiency at the expense of clarity. An efficient program is better than an inefficient one, of course, but it is also true that a slow, correct program is better than a fast, buggy one. Clear, well-designed programs are more likely to be correctly functioning programs. Get the program working before you worry too much about making it work quickly.

Είσοδος/έξοδος χαρακτήρων: getchar() , putchar()

- Η συνάρτηση `getchar()` παίρνει έναν χαρακτήρα από το πληκτρολόγιο και τον αποδίδει στο εκτελούμενο πρόγραμμα.
- Η συνάρτηση `putchar()` παίρνει έναν χαρακτήρα από το εκτελούμενο πρόγραμμα και τον εμφανίζει στην οθόνη.
- Παράδειγμα

```
#include <stdio.h>
int main()
{
  char  ch;
  ch = getchar();   /* anti tou  scanf("%c",&ch); */
  putchar(ch);      /* anti tou  printf("%c",ch); */
  return 0;
}
```

```
g <enter>
g
```

- Η `getchar()` δεν έχει ορίσματα. Η τιμή που επιστρέφει είναι ο χαρακτήρας που παίρνει από το πληκτρολόγιο. Αυτόν τον χαρακτήρα στη συνέχεια αποδίδει στη μεταβλητή `ch`.
- Η `putchar()` έχει ένα όρισμα που είναι κάποιος χαρακτήρας ή ακολουθία διαφυγής ή κάποια μεταβλητή ή συνάρτηση της οποίας η τιμή είναι ένας μοναδικός χαρακτήρας.
- Π.χ. `putchar('A'); putchar('\n'); putchar(ch); putchar(getchar());`

Η σύνταξη των τριών βρόχων της C

ΒΡΟΧΟΣ	ΠΑΡΑΔΕΙΓΜΑΤΑ
`while ( συνθήκη )` `  εντολή;`	`while ( i++ < 20 )` `  q *= 2;`

`do`
`  εντολή;`
`while ( συνθήκη )`
	`do`
	`  printf("Hello!\n");`
	`while ( i++ < 10 )`

`for(αρχικοποίηση; συνθήκη; αναπροσαρμογή)` `  εντολή;`	`for(n=0; n<10; n++)` `  printf("%d %d\n",n,2*n-1);`

Το παρακάτω πρόγραμμα δέχεται χαρακτήρες από το πληκτρολόγιο και τους εμφανίζει στην οθόνη μέχρι να πληκτρολογήσουμε '*'.

```
#include <stdio.h>
#define  STOP '*'
int main()
{
  char ch;

  ch = getchar();
  while (ch != STOP)
  {
    putchar(ch);
    ch = getchar();
  }
  return 0;
}
```

```
while ((ch=getchar()) != STOP)
  putchar(ch);
```

Με τον παρακάτω βρόχο καθαρίζουμε το buffer εισόδου από υπολείμματα (π.χ. τον χαρακτήρα νέας γραμμής) πριν το διάβασμα νέων δεδομένων.

```
while( getchar() != '\n');  /* Empty input buffer */
```

Μορφοποίηση δεδομένων κινητής υποδιαστολής

```
#include <stdio.h>
int main()
{
  float a,b,c,d;

  a = 21.234;  b = 5467.1;
  c = .99;   d = 130.25;
  printf("Numbers:\n%f %f\n",a,b);
  printf("%f %f\n",c,d);
  return 0;
}
```

χωρίς μορφοποίηση εξόδου

```
Numbers:
21.234 5467.1
.99 130.25
```

Μορφοποίηση δεκαδικού μέρους

```
#include <stdio.h>
int main()
{
  float a,b,c,d;

  a = 21.234;  b = 5467.1;
  c = .99;    d = 130.25;
  printf("Numbers:\n%.3f %.3f\n",a,b);
  printf("%.3f %.3f\n",c,d);
  return 0;
}
```

3 δεκαδικά ψηφία

```
Numbers:
21.234 5467.100
.990 130.250
```

Λεπτομερής καθορισμός τρόπου εμφάνισης πραγματικών αριθμών

```
#include <stdio.h>
int main()
{
  float a,b,c,d;

  a = 21.234;  b = 5467.1;
  c = .99;    d = 130.25;
  printf("Numbers:\n%8.3f %8.3f\n",a,b);
  printf("%8.3f %8.3f\n",c,d);
  return 0;
}
```

χώρος 8 θέσεων: μία για υποδιαστολή, τρεις για τα δεκαδικά ψηφία και 4 για ακέραιο μέρος

```
Numbers:
  21.234 5467.100
    .990  130.250
```

ΣΥΝΑΡΤΗΣΕΙΣ [1/3]

- **Η συνάρτηση είναι μια αυτοδύναμη μονάδα του κώδικα, που είναι σχεδιασμένη για την επίτευξη κάποιου συγκεκριμένου σκοπού.**
- **Μερικές συναρτήσεις απλώς εκτελούν κάποιες ενέργειες χωρίς να επιστρέφουν κάποια τιμή στο πρόγραμμα. Για παράδειγμα, η printf() απλώς εμφανίζει δεδομένα στην οθόνη του υπολογιστή.**
- **Άλλες συναρτήσεις υπολογίζουν κάποια τιμή που θα χρησιμοποιηθεί από το πρόγραμμα που τις κάλεσε. Για παράδειγμα, η cos() υπολογίζει, για λογαριασμό του προγράμματος που την κάλεσε, το συνημίτονο μιας γωνίας.**
- **Γενικά, μια συνάρτηση μπορεί, ταυτόχρονα, να εκτελεί κάποιες ενέργειες και να υπολογίζει τιμές, τις οποίες επιστρέφει στο πρόγραμμα που την κάλεσε.**

ΣΥΝΑΡΤΗΣΕΙΣ [2/3]

- **Γενική μορφή μιας C-συνάρτησης:**

```
τύπος όνομα-συνάρτησης (λίστα με διακηρύξεις ορισμάτων)
        {
        σώμα συνάρτησης
        }
```

- **Οι συναρτήσεις μας γλυτώνουν από επαναλήψεις. Για παράδειγμα, αν πρέπει να επαναλαμβάνουμε συχνά κάποια διαδικασία, είναι καλύτερο να γράψουμε μια κατάλληλη συνάρτηση γι'αυτό το σκοπό και να έχουμε το πρόγραμμα ή τα προγράμματα να καλούν αυτή τη συνάρτηση όποτε τη χρειάζονται.**
- **Ακόμη και αν δεν πρόκειται να χρησιμοποιηθεί πολλές φορές μια συνάρτηση, αξίζει να δομούμε τα προγράμματά μας με συναρτήσεις επειδή αυτά γίνονται πιο κατανοητά και εύκολα σε αλλαγές ή βελτιώσεις.**

ΣΥΝΑΡΤΗΣΕΙΣ [3/3]

- Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να γράψουμε ένα πρόγραμμα που:
- Να διαβάζει μια λίστα αριθμών
- Να τους τοποθετεί σε αύξουσα διάταξη
- Να βρίσκει τη μέση τιμή τους
- Να εμφανίζει ένα γράφημα με μπάρες
- Θα μπορούσαμε να χρησιμοποιήσουμε το παρακάτω πρόγραμμα:

```
int main()
{
        float list[50];
        readlist(list);
        sort(list);
        average(list);
        bargraph(list);
        return 0;
}
```

ΜΙΑ ΑΠΛΗ ΣΥΝΑΡΤΗΣΗ ΧΩΡΙΣ ΟΡΙΣΜΑΤΑ [1/2]

```
#include <stdio.h>
#define EPWNYMIA "ETAIREIA EPISTHMONWN PLHROFORIKHS"
#define DIEYTHYNSH "Plhroforikhs 18"
#define XWRA "ELLADA"

void starbar(); /* Αρχέτυπο συνάρτησης που θα χρησιμοποιηθεί
                   αργότερα από το πρόγραμμά μας */
int main()
{
  starbar();
  printf("%s\n%s\n%s\n",EPWNYMIA,DIEYTHYNSH,XWRA);
  starbar();
  return 0;
}

#define LIMIT 50
void starbar()
{
  int i;
  for(i=0; i<LIMIT; i++)
    putchar('*');
  putchar('\n');
}
```

ΜΙΑ ΑΠΛΗ ΣΥΝΑΡΤΗΣΗ ΧΩΡΙΣ ΟΡΙΣΜΑΤΑ [2/2]

Εξοδος προγράμματος:

```
**************************************************
ETAIREIA EPISTHMONWN PLHROFORIKHS
Plhroforikhs 18
ELLADA
**************************************************
```

ΣΥΝΑΡΤΗΣΗ ΜΕ ΟΡΙΣΜΑΤΑ [1/2]

```
#include <stdio.h>
float square(float x);
float cube(float x);

int main()
{
  float num, num2, num3;

  printf("Pliktrologhse enan arithmo: ");
  scanf("%f",&num);

  num2 = square(num);
  num3 = cube(num);
  printf("%.1f^2 = %.3f, %.1f^3 = %.3f\n",num,num2,num,num3);
  return 0;
}

float square(float x)  /* Υπολογίζει το τετράγωνο μιας τιμής */
{
  float res;

  res = x * x;
  return res;
}
```

ΣΥΝΑΡΤΗΣΗ ΜΕ ΟΡΙΣΜΑΤΑ [2/2]

```
float cube(float x)  /* Υπολογίζει τον κύβο μιας τιμής */
{
  return (x * x * x);
}
```

Παράδειγμα εκτέλεσης προγράμματος:

```
Pliktrologhse enan arithmo: 2.5 <enter>
2.5^2 = 6.250, 2.5^3 = 15.625
```

ΦΩΛΙΑΣΜΕΝΕΣ ΚΛΗΣΕΙΣ ΣΥΝΑΡΤΗΣΕΩΝ

Τα ορίσματα μιας συνάρτησης μπορούν να είναι οποιεσδήποτε έγκυρες εκφράσεις της C (τιμές, μεταβλητές, αριθμητικές ή λογικές παραστάσεις ή μια άλλη συνάρτηση με τιμή επιστροφής). Οι παρακάτω φωλιασμένες κλήσεις συναρτήσεων είναι απολύτως νόμιμες:

```
x = half(third(square(half(y))));  ⟺  { a = half(y);
                                        b = square(a);
                                        c = third(b);
                                        x = half(c);
```

ΧΡΗΣΗ ΣΥΝΑΡΤΗΣΕΩΝ ΣΕ ΕΚΦΡΑΣΕΙΣ

```
/* Έλεγχος αν ένα τρίγωνο είναι ορθογώνιο. Τα ορίσματα της
   συνάρτησης είναι τύπου int και αντιστοιχούν στις πλευρές
   του τριγώνου. Η συνάρτηση επιστρέφει TRUE/FALSE. */
int orthogonal_triangle(int a, int b, int c)
{
  if (square(a) == square(b) + square(c) ||
      square(b) == square(c) + square(a) ||
      square(c) == square(a) + square(b))
     return 1;
  else
     return 0;
}
```

Παράδειγμα χρήσης της συνάρτησης:

Έστω ότι έχουμε δώσει ακέραιες τιμές στις μεταβλητές x, y, z

```
if orthogonal_triangle(x, y, z)
   printf("H triada (%d %d %d) ikanopoiei to Pythagoreio
           theorhma\n", x, y, z);
```

ΠΙΝΑΚΕΣ [1/3]

- Ένας πίνακας είναι μια συλλογή δεδομένων που αποτελείται από στοιχεία του ιδίου τύπου (π.χ. πίνακας ακεραίων).
- Τα στοιχεία ενός πίνακα αποθηκεύονται σε συνεχόμενες θέσεις μνήμης
- Στη C ο δείκτης που προσδιορίζει τη θέση του πρώτου στοιχείου ενός πίνακα N στοιχείων είναι το 0 και του τελευταίου στοιχείου το N-1.
- Αντιθέτως, στην Pascal και στο MATLAB, οι δείκτες ξεκινούν από το 1.
- Παράδειγμα δήλωσης πίνακα 10 ακεραίων με όνομα **a** και πίνακα 5 πραγματικών αριθμών με όνομα **weight**:

  int a[10];

  float weight[5];

  Τα στοιχεία του πίνακα a είναι τα: **a[0], ..., a[9].**

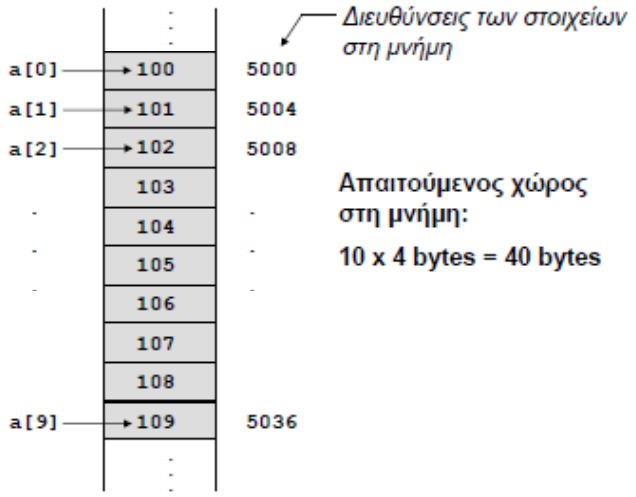
  Παράδειγμα άμεσης προσπέλασης στοιχείων: **weight[2] = 31.25;**

ΠΙΝΑΚΕΣ [2/3]

Η δήλωση ενός πίνακα δεσμεύει συνεχόμενες θέσεις στη μνήμη.

Παράδειγμα:

```
int a[10];
```



ΠΙΝΑΚΕΣ [3/3]

- Με τους πίνακες μπορούμε να ομαδοποιούμε πολλές μεταβλητές με το ίδιο όνομα. Π.χ. Έστω ότι θέλουμε να διαβάσουμε 100 πραγματικούς αριθμούς σε ένα πρόγραμμα, τότε αντί να δηλώσουμε 100 μεταβλητές (**float ar1, ar2, ar3, …, ar100;**) και να επαναλαμβάνουμε την είσοδο/έξοδο και επεξεργασία των αριθμών με πανομοιότυπο τρόπο, ορίζουμε έναν πίνακα τύπου float με τόσες θέσεις όσοι και οι αριθμοί: **float ar[100];**
- Τώρα, η είσοδος, η έξοδος και η επεξεργασία των αριθμών μπορεί να γίνει κομψά και αποτελεσματικά με χρήση των εντολών επανάληψης (βρόχων).
- Προσπέλαση στοιχείων πίνακα:
- **a[0] = 0;/* εκχώρηση τιμής στο 1ο στοιχείο του πίνακα a */**
- **j = 5; a[j] = j; /* εκχώρηση στο 6ο στοιχείο του a */**

Αρχικοποίηση πίνακα

```
/* Ypologismos plithous psifiwn apo metaglwttisth */
#include <stdio.h>
int main()
{
  int i;
  int days[]={31,28,31,30,31,30,31,31,30,31,30,31};

  for(i=0; i<12; i++)
    printf("O mhnas %d exei %d meres\n",i+1,days[i]);
  return 0;
}
```

```
O mhnas 1 exei 31 meres
O mhnas 2 exei 28 meres
O mhnas 3 exei 31 meres
...
O mhnas 12 exei 31 meres
```

Ταξινόμηση αριθμών

```
#include <stdio.h>
int main()
{
 int i,j;
 float m;
 float a[6] = {-1.5, 3.4, -1.6e2, 9., .2, 2.7e-1};

 for(i=0; i<5; i++)
   for(j=i+1; j<6; j++)      /* φωλιασμένη for */
     if(a[i] > a[j])
     {
       m = a[i];
       a[i] = a[j];
       a[j] = m;
     }

 for(i=0; i<6; i++)
   printf("%7.2f ",a[i]);
 printf("\n");

 return 0;
}
```

```
-160.00   -1.50    0.20    0.27    3.40    9.00
```

Εισαγωγή δεδομένων σε πίνακες [1/4]

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main()
{
  int   i, pin1[10], pin2[10];
  int   counter, flag=TRUE;

  printf("EISAGWGH DEDOMENWN\n");

     counter = 0;
  while (flag)     /* while (flag == TRUE) */
  {
    printf("Input 2 integer values for ");
    printf("pin1[%d], pin2[%d]:\n",counter,counter);
    scanf("%d %d",&pin1[counter],&pin2[counter]);
    counter++;
    if (counter == 10)
      flag = FALSE;
  }
  ...
}
```

όχι καλός τρόπος

Εισαγωγή δεδομένων σε πίνακες [2/4]

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main()
{
  int   i, pin1[10], pin2[10];
  int   counter, flag=TRUE;

  printf("EISAGWGH DEDOMENWN\n");

    counter = 0;
  while (counter < 10)
  {
    scanf("%d %d",&pin1[counter],&pin2[counter]);
    counter++;
  }
  ...
}
```

καλύτερος τρόπος

Εισαγωγή δεδομένων σε πίνακες [3/4]

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main()
{
  int   i, pin1[10], pin2[10];
  int   counter, flag=TRUE;

  printf("EISAGWGH DEDOMENWN\n");

    counter = 0;
  do
  {
    scanf("%d %d",&pin1[counter],&pin2[counter]);
    counter++;
  } while (counter < 10)
  ...
}
```

Όχι καλός τρόπος. Ο βρόχος εκτελείται τουλάχιστον 1 φορά!

Εισαγωγή δεδομένων σε πίνακες [4/4]

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main()
{
  int   i, pin1[10], pin2[10];
  int   counter, flag=TRUE;

  printf("EISAGWGH DEDOMENWN\n");

    for(counter = 0; counter < 10; counter++)
   scanf("%d %d",&pin1[counter],&pin2[counter]);

  ...
}
```

Ο καλύτερος τρόπος.

ΠΙΝΑΚΑΣ ΩΣ ΟΡΙΣΜΑ ΣΥΝΑΡΤΗΣΗΣ

```
#include <stdio.h>
#define  DIM  5
float average(float pin[], int dim);
int main()
{
  int i;
  float a[DIM];

  for(i=0; i<DIM; i++)
  {  printf("\nPlease input a float for a[%d]: ",i);
     scanf("%f",&a[i]); }
  printf("Average of the %d numbers: %f\n",i,average(a,DIM));
  return 0;
}
float average(float pin[], int dim)
{
  int i;
  float sum;

  for(i=0, sum=0.0; i<dim; i++)
      sum += pin[i];
  return (sum/dim);
}
```