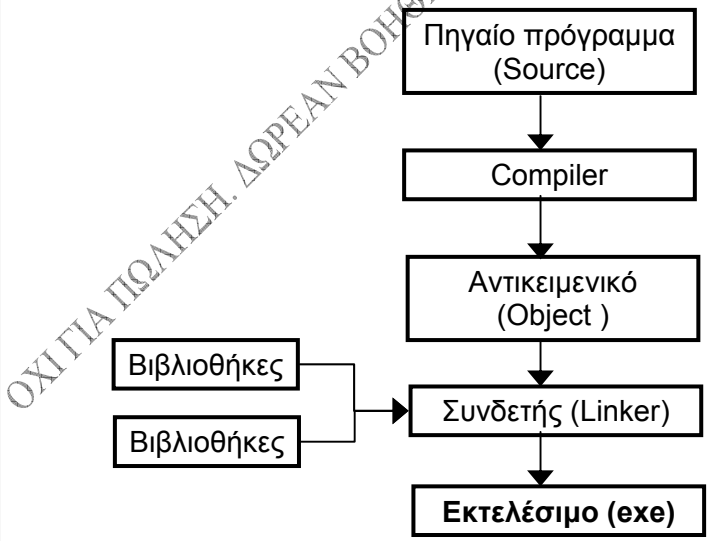
ΓΕΝΙΚΕΣ ΕΝΝΟΙΕΣ

- **Πρόγραμμα: Ομάδα εντολών προς τον υπολογιστή.**
- **Το πρόγραμμα είναι γραμμένο σε μια ειδική γλώσσα, την Γλώσσα Προγραμματισμού.**
- **Υπάρχουν χιλιάδες γλώσσες προγραμματισμού.**
- **Το πρόγραμμα έχει μορφή κειμένου, περιέχει δηλαδή εντολές όπως:**

```
.....
if (x > 0)
      k++;
printf ("%d", k);
.....
```


ΒΗΜΑΤΑ ΔΗΜΙΟΥΡΓΙΑΣ ΚΑΙ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ



ΤΙ ΚΑΝΕΙ ΚΑΙ ΤΙ ΔΕΝ ΚΑΝΕΙ Ο COMPILER (1)

- **Μεταφράζει το πρόγραμμα σε ακολουθίες 0 και 1.**
- **Το πρόγραμμα πρέπει να είναι συντακτικά σωστό, αλλιώς δεν προχωρεί στην μετάφραση.**
- **Το παρακάτω ΔΕΝ είναι συντακτικά σωστό:**

```
……….
if (x > 0)
   k++
printf (“%d”, k);
……….
```

Ούτε και το επόμενο είναι σωστό:

```
……….
if (x > 0)
   k++;
printf (“%d”, k);.
……….
```


ΤΙ ΚΑΝΕΙ ΚΑΙ ΤΙ ΔΕΝ ΚΑΝΕΙ Ο COMPILER (2)

- **Ο compiler προσπαθεί να μας βοηθήσει στην ανίχνευση των συντακτικών λαθών. Μας υποδεικνύει την θέση και το είδος του λάθους, χωρίς όμως να επιτυγχάνει πάντα τα σωστά.**
- **Ο compiler ΔΕΝ ανιχνεύει τα λογικά λάθη. Ένα λογικό λάθος στο παρακάτω τμήμα προγράμματος είναι:**

Θέλαμε

```
……….
if (x > 0)
   k++;
printf (“%d”, k);

……….
```

…και γράψαμε

```
……….
if (x < 0)
   k++;
printf (“%d”, k);

……….
```

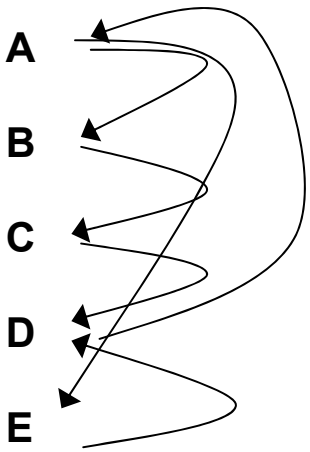
ΜΟΡΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
/* Πρώτο πρόγραμμα */
#include  <stdio.h>
main( )
{
  printf("Γειά σας. Πρώτο πρόγραμμα");
}
```

- Η C είναι γλώσσα **ελευθέρου format**. Προσπαθούμε να κάνουμε το πρόγραμμά μας ευκολοδιάβαστο.
- Η C κάνει **διάκριση μικρών/κεφαλαίων**. Γράφουμε με μικρά.
- Τα **σχόλια**: Δεν μεταφράζονται από τον compiler. Μπορούν να βρίσκονται οπουδήποτε μέσα στο πρόγραμμα. Τίθενται μεταξύ των /* */
- #include: **οδηγία** προς τον **προεπεξεργαστή**. Υπάρχει στην αρχή κάθε προγράμματος C.

ΣΗΜΕΡΙΝΕΣ ΤΕΧΝΙΚΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

- Ένα πρόγραμμα **ΔΕΝ** αποτελείται από πλήθος (εκατοντάδων ή χιλιάδων) εντολών.
- Το πρόγραμμα χωρίζεται σε **τμήματα**.
- Κάθε ένα από τα τμήματα αυτά έχει το **όνομά** του και **είναι ένα μικρό πρόγραμμα** (όχι δηλαδή μια μόνο εντολή. Αποτελείται από μια ομάδα εντολών).
- Το ένα πρόγραμμα καλεί το άλλο.



Τα μέρη στα οποία χωρίζεται το πρόγραμμα λέγονται στις διάφορες γλώσσες προγραμματισμού:

Υποπρογράμματα
Ρουτίνες
Υπορουτίνες

Στη C λέγονται:

ΣΥΝΑΡΤΗΣΕΙΣ

ΜΟΡΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ – ΕΝΤΟΛΕΣ (1)

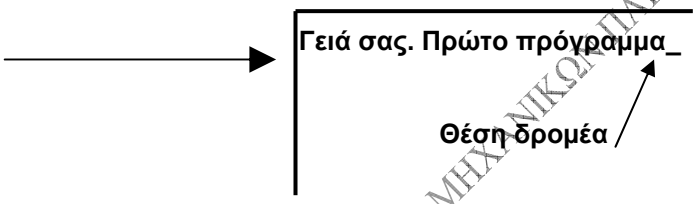
```
/* Πρώτο πρόγραμμα */
#include  <stdio.h>
main( )
{
   printf("Γειά σας. Πρώτο πρόγραμμα");
}
```

- **main( ): κύρια συνάρτηση, υπάρχει πάντα. Σε κάθε πρόγραμμα «κάνει» κάτι άλλο, αλλά υπάρχει πάντα μια συνάρτηση με αυτό το όνομα. Από την πρώτη εντολή της αρχίζει η εκτέλεση του προγράμματος.**
- **Άγκιστρα { }. Τα όρια της main( ).**
- **Η printf εμφανίζει στην οθόνη ό,τι είναι ανάμεσα στα ‘’ ‘’.**
- **Οθόνη προγράμματος και οθόνη αποτελεσμάτων.**
- **Η printf γράφει στην οθόνη (αποτελεσμάτων) στο σημείο που βρίσκεται ο δρομέας.**

ΜΟΡΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ – ΕΝΤΟΛΕΣ (2)

```
/* Πρώτο πρόγραμμα */
#include  <stdio.h>
main( )
{
  printf("Γειά σας. Πρώτο πρόγραμμα");
}
```

Άρα το πρόγραμμα θα εμφανίσει……



ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (1)

```
#include  <stdio.h>
main( )
{
   int  num, art;
   num = 1;
   printf ("Ο αριθμός ισούται με %d\n", num);
   art = num + 1;
   printf ("Προσθέτοντας 1 έχουμε %d\n",  art);
}
```

- **Μορφή: όπως το προηγούμενο.**
- **Εξετάζουμε τι γίνεται στο πρόγραμμα σε δύο φάσεις:**
  - **την φάση της μεταγλώττισης και**
  - **την φάση της εκτέλεσης**

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (2)
ΦΑΣΗ ΜΕΤΑΓΛΩΤΤΙΣΗΣ

```
#include  <stdio.h>
main( )
{
   int  num, art;
   num = 1;
   printf ("Ο αριθμός ισούται με %d\n", num);
   art = num + 1;
   printf ("Προσθέτοντας 1 έχουμε %d\n",  art);
}
```

- **Μεταβλητές: Ονόματα που θα πάρουν τιμές (όπως και οι μεταβλητές στα Μαθηματικά).**
- **Οι μεταβλητές είναι άγνωστες λέξεις στον compiler, γι' αυτό δηλώνονται πριν την χρήση.**
- **Η δήλωση περιλαμβάνει το όνομα της μεταβλητής και το είδος της τιμής που θα πάρει.**

(ΥΠΕΝΘΥΜΙΣΕΙΣ ΓΙΑ ΤΗ ΜΝΗΜΗ)

- **Το πρόγραμμα, όπως και τα αποτελέσματα γράφονται στη μνήμη RAM.**
- **Η μνήμη αποτελείται από ένα σύνολο από «κουτάκια», τα οποία λέγονται θέσεις μνήμης.**
- **Κάθε θέση περιέχει ένα συνδυασμό από 8 μηδενικά και άσσους (bits). Η οκτάδα λέγεται byte.**
- **Κάθε θέση χαρακτηρίζεται από ένα αριθμό, τη διεύθυνση της μνήμης.**

01101100	0
10100011	1
.....	
10000110	15789
00011010	15790
.....	
11011111	652324
10101000	652325
.....	
00000010	15427396
00000000	
.....	
11111101	1356789432

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (2)
ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 1

```
main( )
{
    int  num, art;
    ……..
```

ΟΠΟΥΔΗΠΟΤΕ

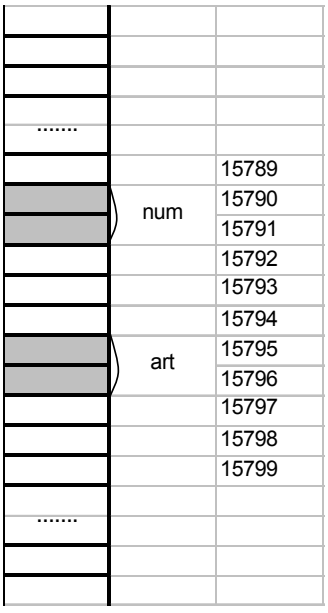
μέσα στο πρόγραμμα συναντήσουμε δηλώσεις μεταβλητών γίνονται οι εξής δύο «ενέργειες»:

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (3)
ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 2

1. **ΔΕΣΜΕΥΣΗ ΧΩΡΟΥ ΜΝΗΜΗΣ, δηλαδή:**
   Εξασφάλιση χώρου (που δεν χρησιμοποιείται από άλλο πρόγραμμα, το οποίο «τρέχει» ταυτόχρονα στον υπολογιστή), για να μπει η τιμή της μεταβλητής.
   Ο χώρος αυτός εξαρτάται από το είδος της μεταβλητής (για κάθε ακέραιο 2 byte).

   ```
   int  num, art;
   ```

   Η δέσμευση γίνεται από το Λειτουργικό Σύστημα, το οποίο έχει «μπροστά του» ένα χάρτη μνήμης.



ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (4)

ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 3

2. ΤΟΠΟΘΕΤΗΣΗ ΕΝΟΣ ΒΕΛΟΥΣ ΣΤΗ ΜΝΗΜΗ.
Ένα βελάκι για κάθε μεταβλητή, το οποίο δείχνει πού θα αποθηκευτεί η τιμή της μεταβλητής.
Τα βελάκια έχουν ονόματα.
Για τις δηλώσεις:

```
int  num, art;
```

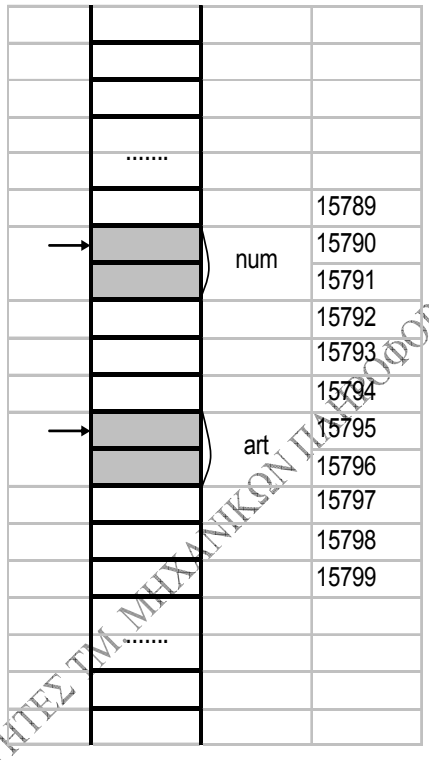
τα βέλακια λέγονται:

```
&num
```

(και σημαίνει «εδώ θα αποθηκευτεί η τιμή του num»)
και

```
&art
```

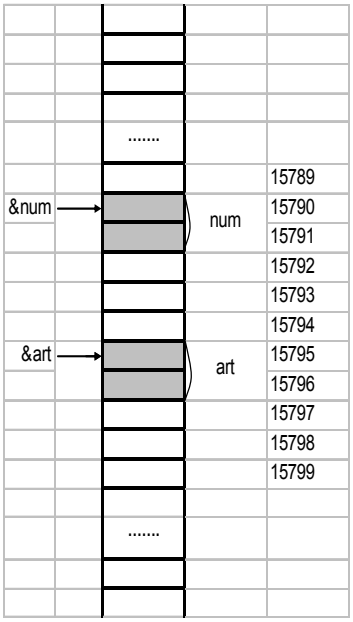
(σημαίνει «εδώ θα αποθηκευτεί η τιμή του art»)



ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (5)

ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 4

- **Η κατάσταση στη μνήμη είναι όπως στο σχήμα.**
- **Κάθε βελάκι λέμε ότι ΔΕΙΧΝΕΙ την μεταβλητή (δηλ. το &num δείχνει ολόκληρη την num και το &art ολόκληρη την art).**
- **Το βελάκι δείχνει ολόκληρη την μεταβλητή, αλλά είναι «κολλημένο» (τοποθετημένο) στο πρώτο από τα byte που δεσμεύτηκαν για την μεταβλητή.**
- **Κάθε ένα από αυτά τα βελάκια είναι ένας…..**



ΔΕΙΚΤΗΣ

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (6)

ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 5

```
.....................................................
num = 1;
printf ("Ο αριθμός ισούται με %d\n", num);
art = num + 1;
printf ("Προσθέτοντας 1 έχουμε %d\n",  art);
.....................................................
```

Το **=** είναι ο **τελεστής εκχώρησης τιμής**. Αποδίδει την τιμή του δεξιού μέλους στο αριστερό. **ΔΕΝ σημαίνει «ίσον».**

Μετά τις εντολές:

```
num = 5;
num = num + 7;
```

το num έχει τιμή:

12

Η printf ( ) εμφανίζει στην οθόνη ό,τι βρίσκεται ανάμεσα στα “ “, όμως το **%d δεν εμφανίζεται** στην οθόνη.

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (7)

ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 6

```
……..
num = 1;
printf ("Ο αριθμός ισούται με %d\n", num);
art = num + 1;
printf ("Προσθέτοντας 1 έχουμε %d\n",  art);
……..
```

Το **%d** λέγεται **προσδιοριστής μορφής των ακεραίων** και καθορίζει πού και πώς θα εμφανιστεί ο ακέραιος που βρίσκεται μετά τα " " και μετά το κόμμα. Δηλαδή:

```
Ο αριθμός ισούται με 1_
```

(Θέση δρομέα μετά το %d)

Η printf δεν έχει τελειώσει….. Δεν μιλήσαμε ακόμα για το \n.

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (8)

ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 7

```
……..
num = 1;
printf ("Ο αριθμός ισούται με %d\n", num);
art = num + 1;
printf ("Προσθέτοντας 1 έχουμε %d\n",  art);
……..
```

Το **\n** θεωρείται **ένας χαρακτήρας**. Είναι ο **χαρακτήρας αλλαγής γραμμής**. Όταν τον συναντήσουμε σε μια printf κατεβάζει τον δρομέα στην αρχή της επόμενης γραμμής. Δηλαδή:

```
Ο αριθμός ισούται με 1
_
```

(Θέση δρομέα μετά το \n)

ΕΝΑ ΠΙΟ ΣΥΝΘΕΤΟ ΠΡΟΓΡΑΜΜΑ (9)

ΦΑΣΗ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ - 8

```
……..
num = 1;
printf ("Ο αριθμός ισούται με %d\n", num);
art = num + 1;
printf ("Προσθέτοντας 1 έχουμε %d\n",  art);
……..
```

Τελικά:

```
Ο αριθμός ισούται με 1
Προσθέτοντας 1 έχουμε 2
_
```

(Θέση δρομέα)

ΠΕΡΙΣΣΟΤΕΡΑ ΓΙΑ ΤΗΝ printf( )

- Μετά το κόμμα μπορεί να υπάρχει ολόκληρη παράσταση.

```
printf ("Άθροισμα = %d\n", num+art);
```

```
Άθροισμα = 3
_
```

- Περισσότερα από ένα %d σε μια printf( ):

```
printf ("Άθροισμα = %d\nΔιαφορά = %d\n", num+art, num-art);
```

Τα %d **αντιστοιχούν ένα προς ένα με την σειρά** στους ακέραιους που βρίσκονται μετά το κόμμα.

```
Άθροισμα = 3
Διαφορά = -1
_
```

ΔΕΔΟΜΕΝΑ

Σταθερές και μεταβλητές.

Κύριοι τύποι:

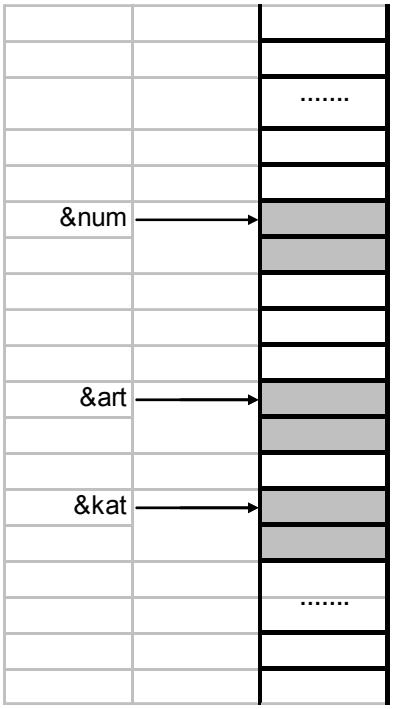
- **Ακέραιοι (int)**
- **Μακρείς ακέραιοι (long)**
- **Μη προσημασμένοι ακέραιοι (unsigned)**
- **Χαρακτήρες (char)**
- **Κινητής υποδιαστολής (float)**
- **Διπλής ακρίβειας (double)**

ΑΚΕΡΑΙΟΙ (1)

- **Δεν έχουν κλασματικό μέρος**. Π.χ. οι αριθμοί 135 και -2456
- Οι ακέραιες μεταβλητές δηλώνονται :

  int num, kat, art;
- Κάθε ακέραιος αποθηκεύεται στη μνήμη σε χώρο **2 byte**.
- Περιοχή τιμών :

  - 32768 έως +32767



ΑΚΕΡΑΙΟΙ (2)

Δυνατοί συνδυασμοί των 16 bits για κάθε ακέραιο:

0000000000000000
0000000000000001
.....
0011100101100010
.....
1100010001111001
.....
1111111111111111

Πλήθος διαφορετικών συνδυασμών:

2^{16} = 65536

Απλουστεύοντας:

32768 συνδυασμοί για τους αρνητικούς
32767 συνδυασμοί για τους θετικούς
1 συνδυασμός για το μηδέν

Σύνολο: 65536

ΑΚΕΡΑΙΟΙ (3)

- Απόδοση **αρχικής τιμής**: `int num = 20;`
  (Ταυτόχρονα με την δήλωση της μεταβλητής)
- **Απεικόνιση** τιμών τύπου int:
  - ✓ Χρησιμοποιείται ο **προσδιοριστής μορφής %d**
  - ✓ Μεταξύ του % και του d **μπορεί να υπάρχει ένας αριθμός**, π.χ:

    `printf ("%3d επί %3d δίνει %5d\n", num, 4, 4*num);`

    Ο αριθμός αυτός καθορίζει **σε πόσες θέσεις στην οθόνη** θα γραφεί η ακέραια τιμή, στοιχημένη στο δεξιό άκρο.

υ20 επί υυ4 δίνει υυυ80

–

ΑΚΕΡΑΙΟΙ (4)

Αν το num έχει τιμή 3156, τότε η printf θα το εμφανίσει στην οθόνη:

Με προσδιοριστή	%5d	ως	υ3156
Με προσδιοριστή	%7d	ως	υυυ3156
Με προσδιοριστή	%3d	ως	?????

Αν ο ακέραιος έχει **περισσότερα ψηφία** από όσα καθορίζει ο αριθμός στον προσδιοριστή (εύρος πεδίου), τότε ο αριθμός αυτός **αγνοείται**

ΑΚΕΡΑΙΟΙ (5)

Άλλοι προσδιοριστές στους int:

Αν το num είναι 38……

η **printf ("%3d x %3d = %5x\n", num, 2, 2*num);** θα εμφανίσει:

υ38 x υυ2 = υυυ4C

%d : προσαρμοστής μορφής των ακεραίων του **δεκαδικού** αριθμητικού συστήματος

%x : προσαρμοστής μορφής των ακεραίων του **δεκαεξαδικού** αριθμητικού συστήματος

Η printf ("%3d + %3d = %5o\n", num, 14, num+14); θα εμφανίσει:

υ38 + υ14 = υυυ64

%o : προσαρμοστής μορφής των ακεραίων του **οκταδικού** αριθμητικού συστήματος

ΧΑΡΑΚΤΗΡΕΣ (1)

- Παίρνουν τιμή ένα από τα **σύμβολα** του πληκτρολογίου (και όχι μόνο αυτά). Π.χ. τα: A d ! ╫ ? κλπ. Δηλαδή μια «εικόνα», μια μη αριθμητική τιμή.
- Στην πραγματικότητα (όπως θα δούμε) είναι **ακέραιος τύπος**.
- Οι μεταβλητές τύπου χαρακτήρα **δηλώνονται**:

  char ch, grm;
- Για κάθε χαρακτήρα δεσμεύεται χώρος **1 byte** στη μνήμη.
- Καταχώρηση τιμών:

  ch = 'K';

  Οι συγκεκριμένοι μεμονωμένοι χαρακτήρες τίθενται μέσα **σε μονά εισαγωγικά**.

ΧΑΡΑΚΤΗΡΕΣ (2)

Οι χαρακτήρες που χρησιμοποιεί ο υπολογιστής τοποθετούνται σε μια **σειρά**, στην οποία **κάθε χαρακτήρας έχει ένα αύξοντα αριθμό**:

<space>	32
!	33
#	35
+	43
A	65
B	66
C	67
a	97
b	98
c	99

Σε κάθε χαρακτήρα αντιστοιχεί ένας μόνο αύξων αριθμός και **σε κάθε αύξοντα αριθμό αντιστοιχεί ένας μόνο χαρακτήρας**

ΧΑΡΑΚΤΗΡΕΣ (3)

Μια τέτοια ενός-προς-ένα αντιστοίχιση χαρακτήρων με αύξοντες αριθμούς λέγεται….

ΚΩΔΙΚΑΣ

Για n διαφορετικά σύμβολα υπάρχουν n! διαφορετικοί κώδικες, π.χ.:

<space>	32	50
!	33	87
#	35	105
+	43	123
A	65	12
B	66	78
C	67	80
a	97	15
b	98	20
c	99	8

κλπ

Ένας από αυτούς είναι διαφορετικούς κώδικες είναι γνωστός στον υπολογιστή και λέγεται….

ΚΩΔΙΚΑΣ ASCII

ΧΑΡΑΚΤΗΡΕΣ (4)

Μπορούμε **ισοδύναμα** να μιλήσουμε για τον 65ο χαρακτήρα του κώδικα ASCII ή για τον χαρακτήρα A, για τον 35ο ή τον χαρακτήρα # κλπ, άρα **κάθε χαρακτήρας μπορεί ισοδύναμα να εκφραστεί με τον αύξοντα αριθμό του στον ASCII**.

ΟΠΟΥ

στο πρόγραμμα χρειάζεται χαρακτήρας **μπορούμε να χρησιμοποιήσουμε τον αντίστοιχο ακέραιο του ASCII**.

```
char ch, xr=70;
ch = 'K';
printf("Ο ch έχει τιμή %3c\n", ch);
printf("Στον ASCII ο %c είναι ο %d\n", xr, xr);
```

Στο παραπάνω, το xr είναι μεταβλητή **με αρχική τιμή…**

70 ή F

ΧΑΡΑΚΤΗΡΕΣ (5)

```
char ch, xr=70;
ch = 'K';
printf("Ο ch έχει τιμή %3c\n", ch);
printf("Στον ASCII ο %c είναι ο %d\n", xr, xr);
```

Η πρώτη printf( ) χρησιμοποιεί τον προσδιοριστή **%c**, ο οποίος είναι ο **προσδιοριστής μορφής των χαρακτήρων**. Στην οθόνη θα γραφεί:

Ο ch έχει τιμή υυK

Η δεύτερη printf( ) γράφει το xr **με μορφή χαρακτήρα** (%c), αλλά και **με μορφή ακεραίου** (%d). Στην οθόνη θα γραφεί:

Στον ASCII ο F είναι ο 70

Ένας χαρακτήρας είναι **ακέραιος τύπος**. Η printf( ) μου δίνει τη δυνατότητα να τον εμφανίσω **είτε με μορφή χαρακτήρα είτε με μορφή ακεραίου**.

ΧΑΡΑΚΤΗΡΕΣ (6)

Ακολουθίες διαφυγής: ζευγάρια χαρακτήρων, τα οποία **θεωρούνται ένας χαρακτήρας**. Σε μια printf( ):

Το **\n** κατεβάζει τον δρομέα **στην αρχή της επόμενης γραμμής**

Το **\b** **επιστρέφει** τον δρομέα **μια θέση προς τα πίσω**

Το **\r** επαναφέρει τον δρομέα **στην αρχή της γραμμής στην οποία βρίσκεται**

Το **\t** **μετακινεί** τον δρομέα **όσο το tab**.

Οι εντολές:

```
printf ("\nΚΑΛΗΜΕΡΑ\b\b");
printf ("ΓΙΑΝΝΗ\r");
```

Θα γράψουν στην οθόνη:

ΚΑΛΗΜΕΓΙΑΝΝΗ

(ο δρομέας κάτω από το Κ)

ΚΙΝΗΤΗΣ ΥΠΟΔΙΑΣΤΟΛΗΣ (1)

- Έχουν ακέραιο και κλασματικό μέρος.
- Δύο τύποι: **float** και **double**
- Δηλώνονται :

```
float  pr, tel;
double  art;
```

- **Για κάθε float** δεσμεύεται χώρος **4 byte** και **για κάθε double** δεσμεύεται **χώρος 8 byte**.
- **Καταχώρηση τιμών**:

```
pr = 9.356;
tel = 5.6e-5;
```

  Το e διαβάζεται: «επί 10 εις την»
- **Απεικόνιση** τιμών τύπου float και double: στην printf( ) χρησιμοποιείται ο **προσδιοριστής %f**.

ΚΙΝΗΤΗΣ ΥΠΟΔΙΑΣΤΟΛΗΣ (2)

Απεικόνιση:

```
float  val = 397.83;
printf("ΤΙΜΗ %f\n", val);
```

Πιο συνηθισμένο είναι κάτι τέτοιο:

```
printf("ΤΙΜΗ %5.1f\n", val);
```

Το 5.1 σημαίνει ότι ο float θα γραφτεί σε **5 διαστήματα συνολικά, από τα οποία το ένα για το κλασματικό μέρος**.

ΟΧΙ 5 διαστήματα **και ένα** για το κλασματικό μέρος**.**
Ξεκινάμε πάντα από το κλασματικό μέρος.

Η val με προσδιοριστή %7.2f θα γραφεί ως....	υ397.83
Η val με προσδιοριστή %8.3f θα γραφεί ως....	υ397.830
Το 8523.986 με προσδιοριστή %10.2f θα γραφεί ως....	υυυ8523.99
Το 8523.986 με προσδιοριστή %6.2f θα γραφεί ως....	????????

Όταν το ακέραιο μέρος του float δεν «χωράει» στον χώρο που περισσεύει σύμφωνα με τον προσδιοριστή, **το ακέραιο μέρος γράφεται στόν αριθμό διαστημάτων που χρειάζεται** (δεν κόβεται δηλαδή)

ΣΤΑΘΕΡΕΣ (1)

- Διατηρούν **την ίδια τιμή** σε όλη τη διάρκεια εκτέλεσης του προγράμματος.

- **Ορίζονται** με εντολές προς τον **προεπεξεργαστή**. Προεπεξεργαστής είναι μια ιδεατή «μηχανή», η οποία επεξεργάζεται το πρόγραμμα **πριν τον compiler**. Π.χ.:

```
#define CNT 99.3
```

  Οι ορισμοί τίθενται πριν την main( ).

- Γράφονται συνήθως με **κεφαλαία γράμματα**. Αυτό μας δίνει την δυνατότητα να διακρίνουμε αμέσως τις σταθερές σε ένα πρόγραμμα.

- Μετά την τιμή **δεν υπάρχει το ;**

- Ακολουθεί ένα πρόγραμμα με την χρήση και χωρίς την χρήση σταθερών.

ΣΤΑΘΕΡΕΣ (2)

```
#include <stdio.h>
main()
{
    float tel, prn, pc;
    printf ("Τιμή τηλεόρασης\n");
    scanf ("%f", &tel);
    printf ("Τιμή εκτυπωτή\n");
    scanf ("%f", &prn);
    printf ("Τιμή υπολογιστή\n");
    scanf ("%f", &pc);
    tel = tel + 0.21 * tel;
    prn = prn + 0.21 * prn;
    pc = pc + 0.21 * pc;
    printf ("Τιμές με ΦΠΑ κατά σειρά\n");
    printf ("%f %f %f %f \n", tel, vid, pr, hd);
}
```

```
#include <stdio.h>
#define FPA 0.21
main()
{
    float tel, prn, pc;
    printf ("Τιμή τηλεόρασης\n");
    scanf ("%f", &tel);
    printf ("Τιμή εκτυπωτή\n");
    scanf ("%f", &prn);
    printf ("Τιμή υπολογιστή\n");
    scanf ("%f", &pc);
    tel = tel + FPA * tel;
    prn = prn + FPA * prn;
    pc = pc + FPA * pc;
    printf ("Τιμές με ΦΠΑ κατά σειρά\n");
    printf ("%f %f %f %f \n", tel, vid, pr, hd);
}
```

Η ΣΥΝΑΡΤΗΣΗ scanf( ) (1)

Διαβάζει την τιμή μιάς μεταβλητής **από το πληκτρολόγιο**.

```
……….
int    age;
float hgt;
printf ("Δώστε το ύψος σας σε μέτρα "
        "και την ηλικία σας\n");
scanf ("%f %d", &hgt, &age);
printf ("Είστε %d ετών. Ύψος %5.2f m\n", age, hgt);
……….
```

(Μια λεπτομέρεια για την printf( ): **Τα διπλά εισαγωγικά ανοίγουν και κλείνουν μια φορά, εκτός αν** αυτό που πληκτρολογούμε γράφε-ται σε περισσότερες από μια γραμμές).

Για να γράψουμε στην οθόνη τις τιμές των hgt και age:

```
printf ("%f %d", hgt, age);
```

Για να διαβάσουμε από το πληκτρολόγιο τις τιμές των hgt και age:

```
scanf ("%f %d", &hgt, &age);
```

Η scanf( ) χρησιμοποιεί **ίδιους προσδιοριστές** με την printf( ), αλλά…

Η ΣΥΝΑΡΤΗΣΗ scanf( ) (2)

```
printf ("%f %d", hgt, age);
scanf ("%f %d", &hgt, &age);
```

Η printf( ) χρησιμοποιεί μετά το κόμμα **μεταβλητές**, ενώ η scanf( ) χρησιμοποιεί **δείκτες**.

Η printf( ) «λέει»: **Γράψε στην οθόνη ένα float, τον hgt, και ένα ακέραιο, τον age**, ενώ…

Η scanf( ) «λέει»: **Διάβασε από το πληκτρολόγιο ένα float, και βάλε τον στη μνήμη εκεί που δείχνει ο δείκτης &hgt, διάβασε και ένα ακέραιο και βάλε τον στη μνήμη εκεί που δείχνει ο δείκτης &age.**

ΠΙΝΑΚΕΣ (1)

Είναι **διατεταγμένα** σύνολα στοιχείων του **ίδιου τύπου** (όλα int, όλα float, όλα char κλπ), τα οποία στοιχεία βρίσκονται αποθηκευμένα σε **διαδοχικές θέσεις μνήμης**.

Δήλωση:

```
char name[40];
```

Θα την διαβάζαμε ως εξής:
Πήγαινε στη μνήμη, δέσμευσε **40 διαδοχικές θέσεις**, πές τις και τις 40 με το **όνομα name** και **στην κάθε θέση** να ξέρεις ότι θα μπεί **ένας χαρακτήρας**.

Ο αριθμός μέσα στις αγκύλες ΔΕΝ σημαίνει byte. Σημαίνει **πόσες τιμές του είδους char** περιλαμβάνει ο πίνακας.

ΠΙΝΑΚΕΣ (2)

Άλλες δηλώσεις:

```
char name[40];
int arf [25];
float deb[20];
```

Ο **arf** είναι πίνακας 25 θέσεων, και περιλαμβάνει **25 ακέραιους**.

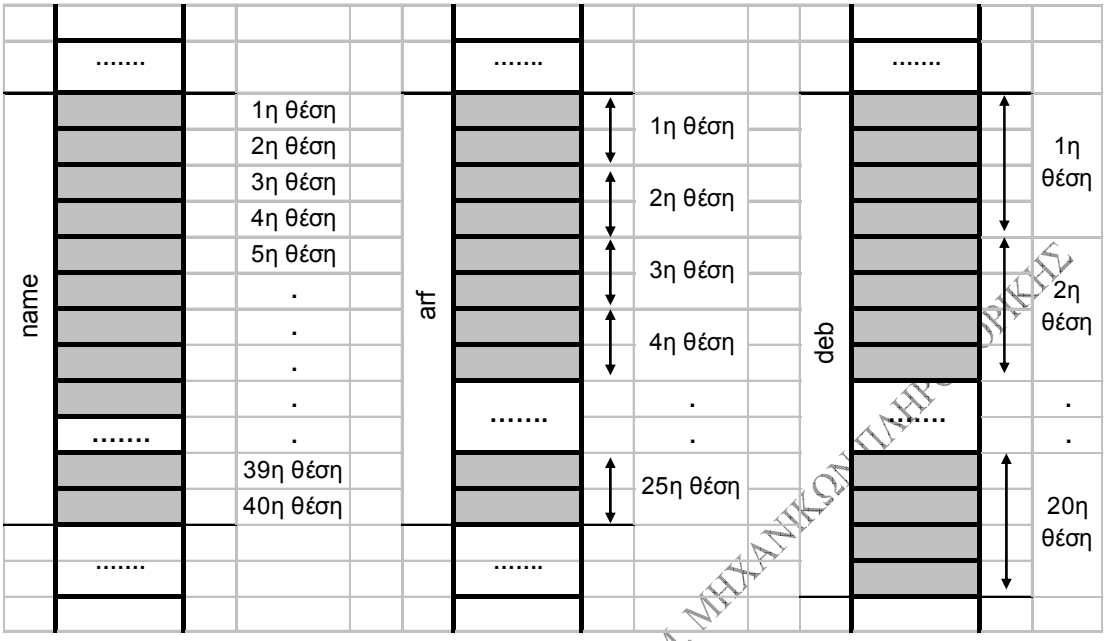
Κάθε ακέραιος χρειάζεται 2 byte στη μνήμη, άρα **ο arf καταλαμβάνει 50 byte στη μνήμη**.

Ο **name** καταλαμβάνει **40 byte** και ο **deb 80 byte**.

Σχηματικά οι θέσεις του κάθε πίνακα……

ΠΙΝΑΚΕΣ (3)

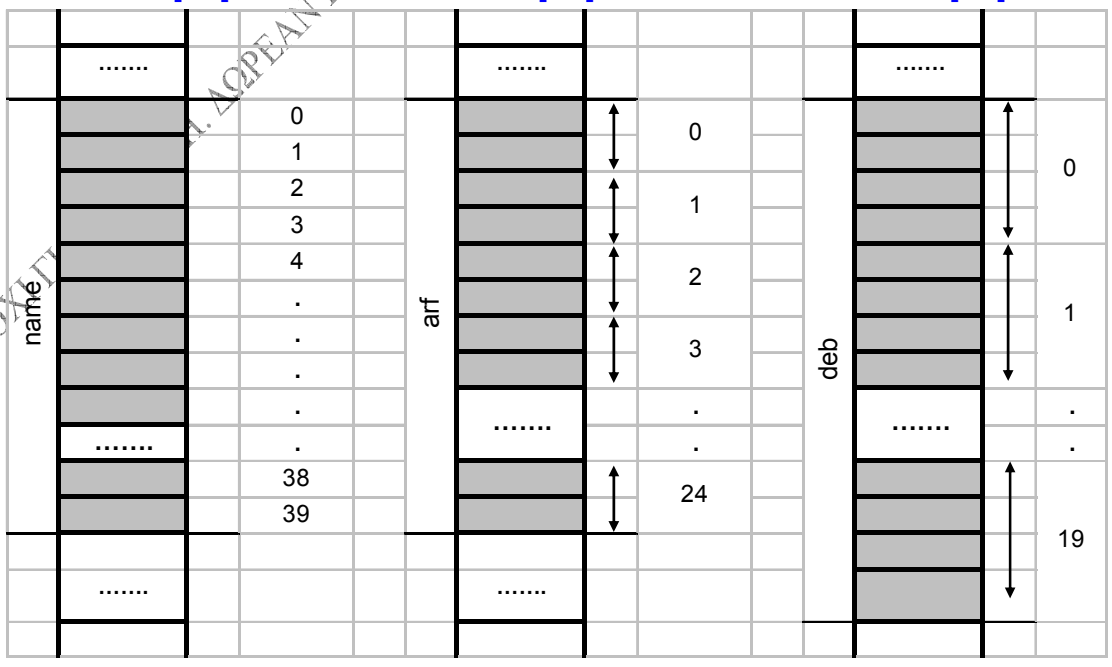
char name[40]; int arf [25]; float deb[20];



ΠΙΝΑΚΕΣ (4)

Οι θέσεις των πινάκων στην C **αριθμούνται ξεκινώντας πάντα από το μηδέν**.

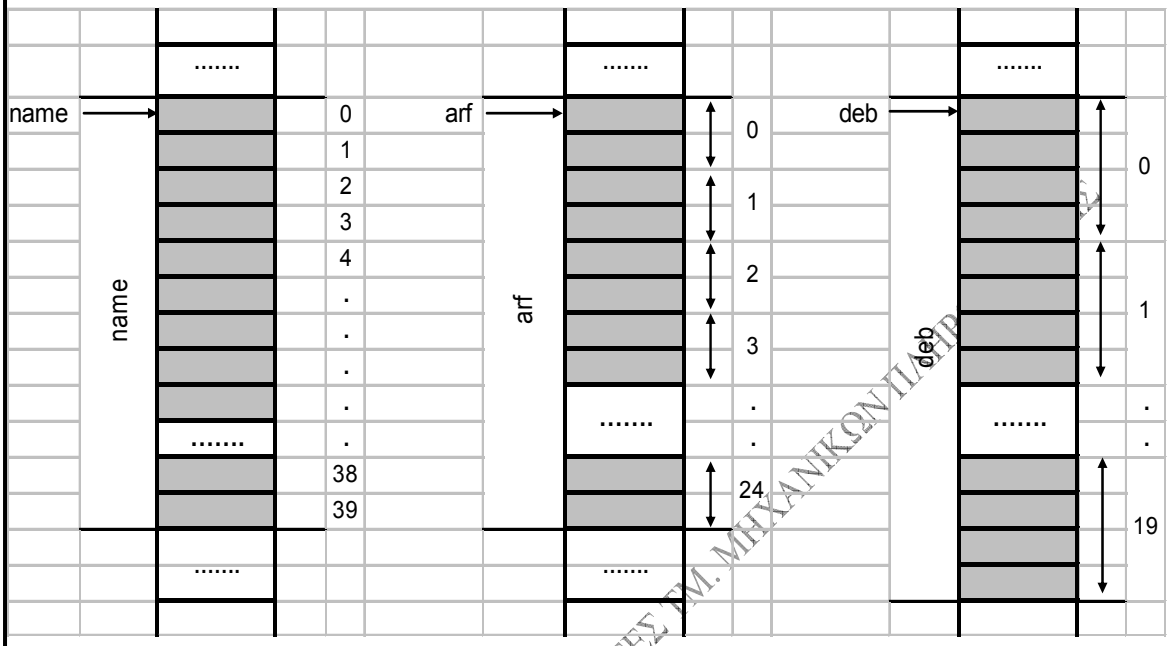
char name[40]; int arf [25]; float deb[20];



ΠΙΝΑΚΕΣ (5)

Ο δείκτης που τοποθετείται στη μνήμη **όταν δηλώνεται πίνακας έχει το ίδιο όνομα με το όνομα του πίνακα**

char name[40]; int arf [25]; float deb[20];



ΠΙΝΑΚΕΣ (6)

char name[40]; int arf [25]; float deb[20];

Αυτό που υπάρχει…
σε κάθε θέση του πίνακα name είναι **ένας χαρακτήρας**
σε κάθε θέση του πίνακα arf είναι **ένας ακέραιος**
σε κάθε θέση του πίνακα deb είναι **ένας float**

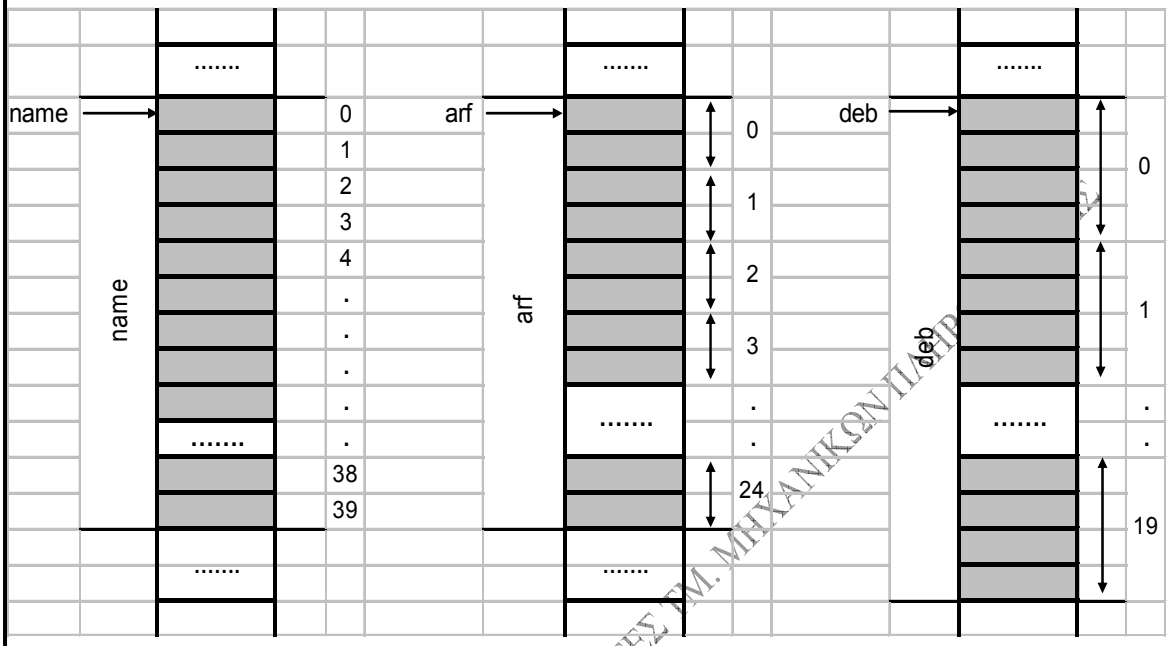
Δηλαδή **στις θέσεις των πινάκων υπάρχουν μεταβλητές.**

Οι μεταβλητές κάθε πίνακα είναι **«οργανωμένες» σε μια δομή** και όχι μεμονωμένες.

Οι μεταβλητές κάθε πίνακα **έχουν τα δικά τους ονόματα**, τα οποία προκύπτουν από **το όνομα του πίνακα και μια αριθμητική ετικέττα μέσα σε ανκύλες**.

ΠΙΝΑΚΕΣ (7)

Για παράδειγμα, τα name[0], name[38] κλπ είναι χαρακτήρες, τα arf [0], arf [1] κλπ είναι ακέραιοι, τα deb[1], deb[19] κλπ είναι float.



ΠΙΝΑΚΕΣ (8)

char name[40]; int arf [25]; float deb[20];

Όταν σε ένα πρόγραμμα έχει δηλωθεί μια μεταβλητή, η ak (οποιουδήποτε τύπου), αυτή αποθηκεύεται εκεί που δείχνει ο δείκτης…

&ak

Αντίστοιχα, (μιλώντας για τον πίνακα arf, αλλά τα ίδια ισχύουν για κάθε πίνακα)…

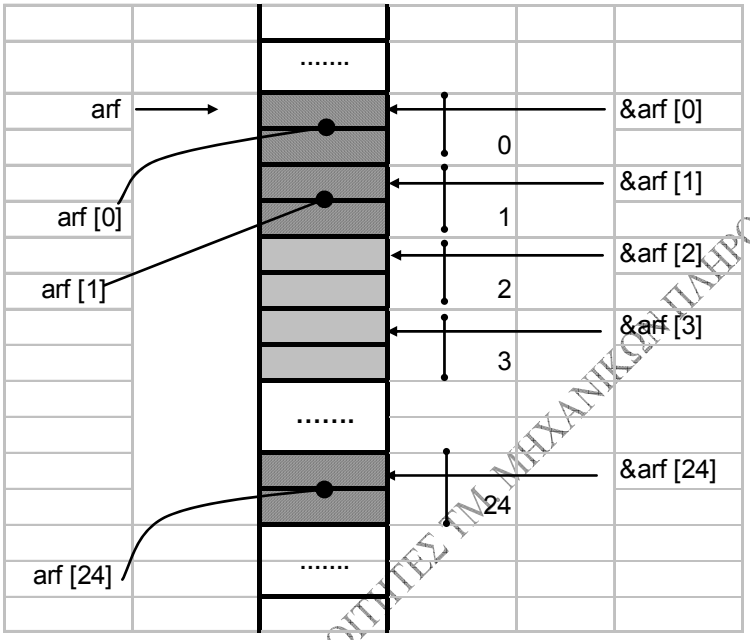
η μεταβλητή arf [0], αποθηκεύεται εκεί που δείχνει ο δείκτης **&arf[0]**
η μεταβλητή arf [1], αποθηκεύεται εκεί που δείχνει ο δείκτης **&arf[1]**
…… ……
η μεταβλητή arf [24], αποθηκεύεται εκεί που δείχνει ο δείκτης **&arf[24]**

Δηλαδή…

Σε κάθε θέση του πίνακα αντιστοιχεί ένας δείκτης

ΠΙΝΑΚΕΣ (9)

Στη μνήμη θα έχουμε την παρακάτω «εικόνα». Ας δεχτούμε προς το παρόν ότι στην πρώτη θέση του πίνακα υπάρχουν δύο δείκτες, ο arf και ο &arf [0]. **Σε μια θέση μνήμης μπορούμε να βάλουμε να δείχνουν όσοι δείκτες θέλουμε.**



ΤΕΛΕΣΤΕΣ (1)

Είναι **σύμβολα** πράξεων και σχέσεων

Τελεστής Καταχώρησης: =

ΔΕΝ σημαίνει ίσον.

Αποδίδει την τιμή του δεξιού μέλους στο αριστερό.

```
val = 589;
```

Αν: `k = 12;`
Μετά την εντολή: `k = k + 1;`
Το k έχει τιμή:

13

ΤΕΛΕΣΤΕΣ (2)

- **Όχι:** 693 = val;
  (Τέτοιο λάθος χαρακτηρίζεται ως Lvalue error)
- Δικαιούμαι να πώ:

  val = temp = met = 153;
- **Πρόσθεσης:** **+**
- **Αφαίρεσης:** **-**
- **Πολλαπλασιασμού:** ***
- **Διαίρεσης:** **/**

Γενικός κανόνας στις πράξεις: Πράξεις μεταξύ τιμών κάποιου τύπου καταλήγουν σε **αποτέλεσμα του ίδιου τύπου**

ΤΕΛΕΣΤΕΣ (3)

4	+	2	δίνει	6
2.0	+	1.3	δίνει	3.3
3.2	+	1.8	δίνει	5.0 και **ΟΧΙ** 5
6	/	2	δίνει	3
7	/	3	δίνει	2 και **ΟΧΙ** 2.5

Διότι
Αυτή είναι μια πράξη μεταξύ ακεραίων, άρα με **ακέραιο αποτέλεσμα**
(ακέραια διαίρεση)

ΤΕΛΕΣΤΕΣ (4)

- **Τελεστής προσήμου:** **+** και **-**

  met = -32;

- **Τελεστής ακεραίου υπολοίπου:** **%**
  - Λέγεται και **modulo**.
  - Δίνει το **υπόλοιπο της ακέραιας διαίρεσης** δύο ακεραίων.
  - **ΔΕΝ** γίνεται με float

6 / 4	δίνει	1
6 % 4	δίνει	2
15 % 3	δίνει	0
3 % 15	δίνει	3

ΤΕΛΕΣΤΕΣ (5)

Στο πρόγραμμα που ακολουθεί δίνουμε από το πληκτρολόγιο ένα αριθμό δευτερολέπτων και υπολογίζεται σε πόσα λεπτά και πόσα δευτερόλεπτα αντιστοιχεί αυτός ο αριθμός.

```
#include <stdio.h>
#define SPM 60
main( )
{
        int sec, min, left;
        printf ("Δώστε αριθμό δευτερολέπτων");
        scanf(''%d'', &sec);                                    192
        min = sec / SPM;          /* Λεπτά */                    3
        left = sec % SPM;         /* Δευτερόλεπτα */            12
        printf ("%d δευτερόλεπτα είναι %d λεπτά και "
              "%d δευτερόλεπτα", sec, min, left);
}
```

ΤΕΛΕΣΤΕΣ (6)

- **Τελεστές αύξησης και μείωσης:** **++** και **- -**
  Μπαίνουν **πριν** ή **μετά από το όνομα μεταβλητής**.
  Και στις δυο περιπτώσεις **αυξάνεται / μειώνεται η τιμή** της μεταβλητής **κατά 1**. Όμως….
  Το **++a** σημαίνει ότι **πρώτα αυξάνουμε την τιμή της μεταβλητής και μετά την χρησιμοποιούμε**, ενώ…
  Το **a++** σημαίνει ότι **πρώτα χρησιμοποιούμε την μεταβλητή** (όπου παίρνει μέρος) **και μετά αυξάνουμε την τιμή της**.
  Αν το a έχει τιμή 18, τότε:

  `p = 2 * ++a;` Το a γίνεται **19** και… το p παίρνει τιμή 2*19, άρα **38**

  `p = 2 * a++;` Το p παίρνει τιμή 2*18, άρα **36** και… το a γίνεται **19**

ΤΕΛΕΣΤΕΣ (7)

```
#include <stdio.h>
main( )
{
    int a=1, b=1, aplus, plusb;
    aplus = a++;
    plusb = ++b;
    printf ("a  aplus   b   plusb   \n");
    printf ("%1d   %5d   %5d   %5d\n", a, aplus, b, plusb);
}
```

Τα a και b έχουν αρχική τιμή 1.
Τα aplus και plusb αρχικά **δεν έχουν τιμή μηδέν**, **ούτε τιμή «τίποτα»** …..
αλλά…
…..**τυχούσα** (άγνωστη) τιμή.

ΤΕΛΕΣΤΕΣ (8)

```
#include <stdio.h>
main( )
{
     int a=1, b=1, aplus, plusb;
     aplus = a++;
     plusb = ++b;
     printf ("a   aplus   b   plusb   \n");
     printf ("%1d   %5d   %5d   %5d\n", a, aplus, b, plusb);
}
```

Η τιμή του a αποδίδεται στο aplus και μετά το a αυξάνει κατά 1, άρα το aplus γίνεται 1 και το a γίνεται 2
Η τιμή του b αυξάνει κατά 1 και μετά αποδίδεται στο plusb, άρα το b γίνεται 2 και το plusb γίνεται 2.

ΤΕΛΕΣΤΕΣ (9)

- **Άλλοι τελεστές καταχώρησης: += -= *= /= %=**

x += 5;	σημαίνει	x = x + 5;
f -= 4;	σημαίνει	f = f - 4;
k *= 7;	σημαίνει	k = k * 7;
p /= 3;	σημαίνει	p = p / 3;
sum %= 15;	σημαίνει	sum = sum % 15;

ΤΕΛΕΣΤΕΣ (10)
ΠΡΟΤΕΡΑΙΟΤΗΤΑ ΤΕΛΕΣΤΩΝ

( )
+ - (πρόσημα) **++ --**
*** / %**
+ - (πρόσθεση, αφαίρεση)
< > <= >=
== !=
&&
||
= *= /= %= += -=

Η προτεραιότητα αυξάνει κατά την φορά του βέλους

ΜΕΤΑΤΡΟΠΕΣ ΤΥΠΩΝ

Στις πράξεις επιτρέπεται η ανάμειξη διαφόρων τύπων. Σε τέτοια περίπτωση, οι τιμές των «ασθενέστερων» τύπων **μετατρέπονται αυτόματα στον ισχυρότερο τύπο**, σύμφωνα με την παρακάτω ιεραρχία τύπων:

Ιεραρχία τύπων:

double — Υψηλότερη
float
long
unsigned int
int
char — Χαμηλότερη

Έτσι, π.χ. στην πράξη 2 + 3.6 το 2 μετατρέπεται σε 2.0 και ακολουθεί η πρόσθεση (αποτέλεσμα 5.6)

«ΕΞΑΝΑΓΚΑΣΜΕΝΕΣ» ΜΕΤΑΤΡΟΠΕΣ ΤΥΠΩΝ

Μετά την επόμενη εντολή…

```
ak = (int) 6.7;
```

το ak παίρνει τιμή 6

Σε παρενθέσεις τίθεται **ο τύπος στον οποίο θέλουμε να καταλήξουμε**.

Αντίστοιχα, μετά την εντολή….

```
fp = (float) 8;
```

το fp παίρνει τιμή 8.0

Γενικά, αν: `int ms; float br;`
….μπορούμε να γράψουμε:

```
ms = (int) br;
```

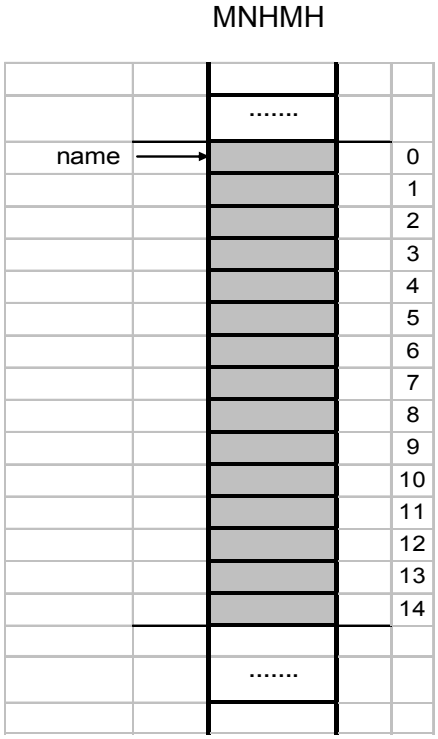

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (1)

```
# include <stdio.h>
main( )
{
    char name[10];
    printf ("Πώς σε λένε;\n");
    scanf("%s", name);
    printf ("Γειά σου %s\n", name);
}
```

ΟΘΟΝΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Πώς σε λένε;

<θέση δρομέα>



ΣΥΜΒΟΛΟΣΕΙΡΕΣ (2)

Εάν επρόκειτο να διαβάσουμε ένα ακέραιο, τον ak, θα είχαμε….

```
scanf ("%d", &ak);
```

Εδώ η `scanf ("%s", name);` χρησιμοποιεί τον προσδιοριστή….

%s

Το %s είναι ο **προσδιοριστής μορφής για τις συμβολοσειρές** (string).

Δηλαδή η scanf ( ) θα περιμένει να διαβάσει…. όχι ένα ακέραιο
….όχι ένα float….
….όχι ένα χαρακτήρα,…. αλλά

Μια ομάδα χαρακτήρων (συμβολοσειρά)

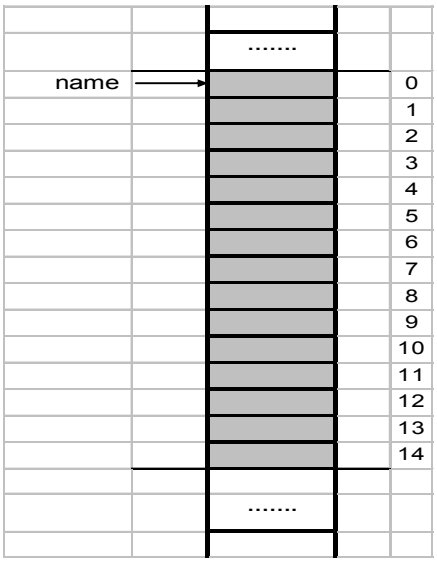
ΣΥΜΒΟΛΟΣΕΙΡΕΣ (3)

Η scanf ( ) **μετά το κόμμα** χρειάζεται **δείκτη**. Εάν επρόκειτο να διαβάσουμε ένα ακέραιο, τον ak, θα είχαμε….

```
scanf ("%d", &ak);
```

Στην `scanf ("%s", name);` υπάρχει δείκτης;

Θυμίζουμε ότι **ο πίνακας και ο δείκτης στην αρχή του έχουν το ίδιο όνομα**, άρα εδώ το name παίζει τον ρόλο του δείκτη και όχι του πίνακα:

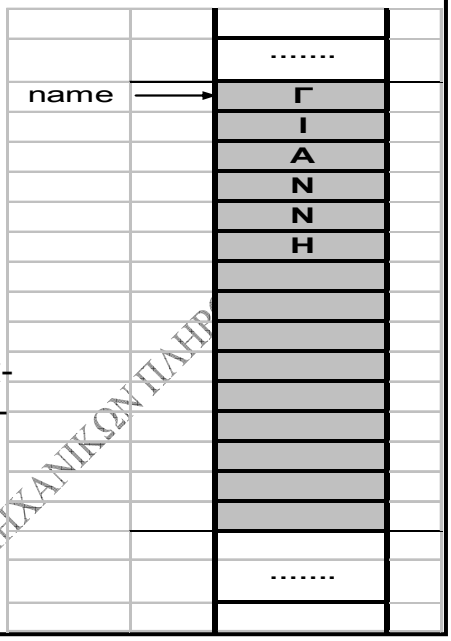


ΣΥΜΒΟΛΟΣΕΙΡΕΣ (4)

scanf ("%s", name);

Η scanf ( ) διαβάζει μια ομάδα χαρακτήρων και την τοποθετεί εκεί που δείχνει ο δείκτης name:

Πώς σε λένε;
ΓΙΑΝΝΗ
_



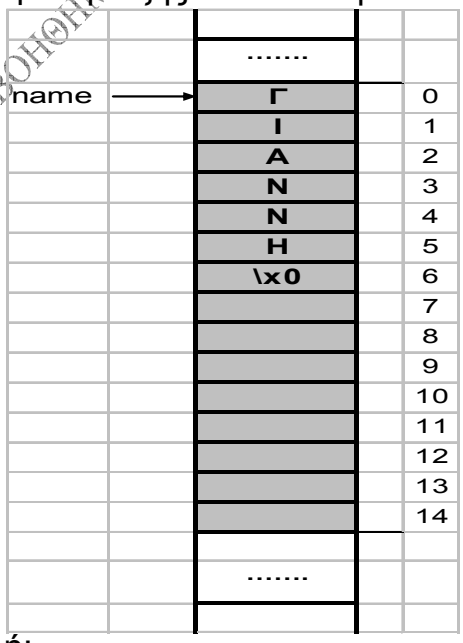
Η scanf ( ) **από μόνη της τοποθετεί** στον πίνακα, μετά τους χαρακτήρες που πληκτρολογήσαμε **ένα ειδικό χαρακτήρα**, τον….

\x0

Αυτός είναι ο **μηδενικός** (ο πρώτος) **χαρακτήρας του κώδικα ASCII**

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (5)

Στη μνήμη τώρα έχουμε την εξής κατάσταση:



Αν δώσουμε την εντολή:

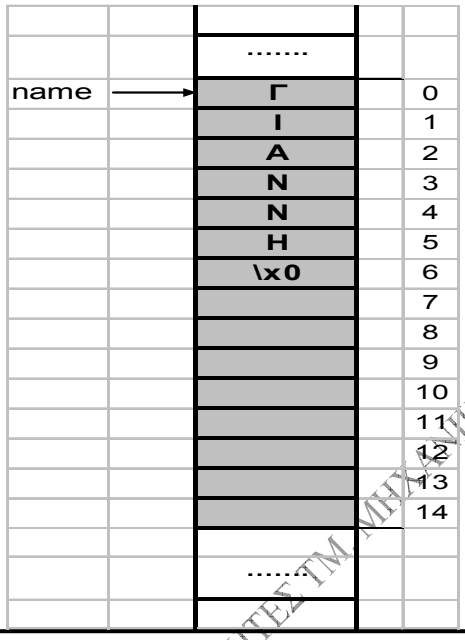
printf ("%s", name);

τι θα συμβεί;

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (6)

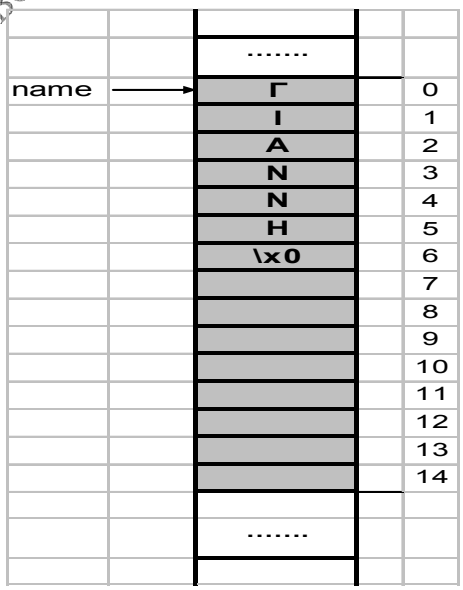
Πρέπει να διακρίνουμε μεταξύ….

- του **πίνακα χαρακτήρων name** (οι θέσεις από την 0 έως και την 14)
- της **συμβολοσειράς name** (οι θέσεις από την 0 έως και την 6)



ΣΥΜΒΟΛΟΣΕΙΡΕΣ (7)

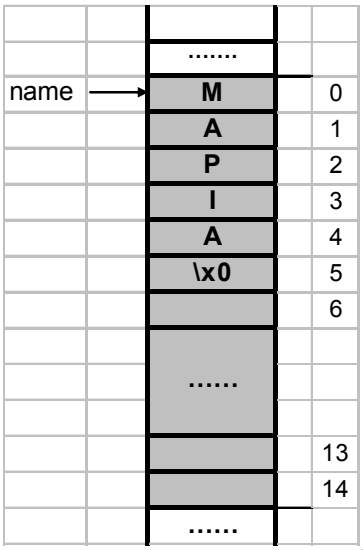
Το \x0 μπήκε από την scanf( ) για να «ξέρει» η printf( ) **μέχρι πού** θα γράψει όταν εμφανίζει συμβολοσειρά (με τη χρήση του προσδιοριστή %s)



Άρα η printf (“ΚΑΛΗΜΕΡΑ %s”, name); θα εμφανίσει:
ΚΑΛΗΜΕΡΑ ΓΙΑΝΝΗ

ΜΗΚΟΣ ΣΥΜΒΟΛΟΣΕΙΡΑΣ (1)

Τι θα κάναμε για να μετρήσουμε **το πλήθος των στοιχείων της συμβολοσειράς** name από την αρχή έως το \x0 (χωρίς το \x0) ;



Ξεκινώντας από το πρώτο, **ελέγχουμε ένα-ένα στοιχείο** εάν είναι το \x0.
Όσο δεν είναι, αυξάνουμε την τιμή μιάς μεταβλητής κατά 1 (μετρητής).
Τελειώνοντας, ο μετρητής έχει τιμή **5**

ΜΗΚΟΣ ΣΥΜΒΟΛΟΣΕΙΡΑΣ (2)

Η C δίνει τη δυνατότητα αυτό το «μέτρημα» να το κάνει μια συνάρτηση, η οποία λέγεται **strlen**.
Θυμίζουμε ότι μια **συνάρτηση** είναι…

ένα **έτοιμο πρόγραμμα**, το οποίο κάνει μια συγκεκριμένη δουλειά.
Το πρόγραμμα αυτό το «φωνάζω», **το καλώ με το όνομά του**.

ΚΑΘΕ

συνάρτηση στη C, **όταν τελειώσει τη δουλειά της γίνεται ίση με κάτι**, δηλαδή…

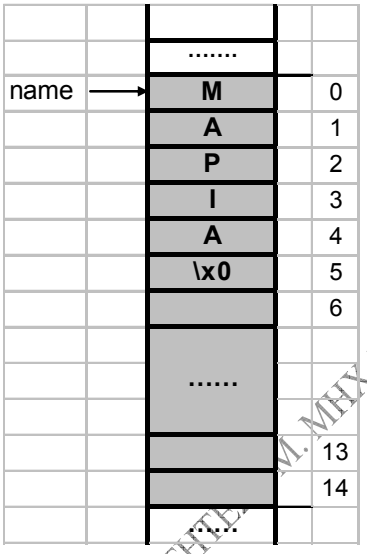
γίνεται ίση με ένα ακέραιο….
ένα float….
ένα χαρακτήρα…. κλπ

Αυτό με το οποίο γίνεται ίση η συνάρτηση λέγεται…
Τιμή επιστροφής της συνάρτησης

ΜΗΚΟΣ ΣΥΜΒΟΛΟΣΕΙΡΑΣ (3)

Η **strlen έχει τιμή επιστροφής ακέραιο**, δηλαδή όταν τελειώσει την δουλειά της γίνεται ίση με ένα ακέραιο. Αυτός ισούται με **το πλήθος των χαρακτήρων της συμβολοσειράς από την αρχή έως το \x0** (χωρίς αυτό).
Η strlen χρειάζεται στις παρενθέσεις της το όνομα της συμβολοσειράς, της οποίας θα μετρήσει το μήκος. Για την συμβολοσειρά name γράφουμε….

strlen (name)

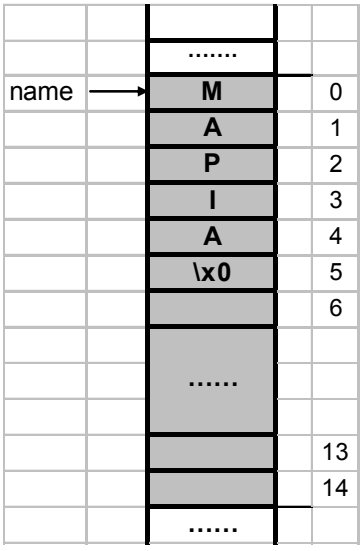


Εδώ το μήκος είναι 5

ΜΗΚΟΣ ΣΥΜΒΟΛΟΣΕΙΡΑΣ (4)

Αν έχουμε: int ak;
Το παρακάτω έχει νόημα;

ak = strlen (name);



Το επόμενο άραγε;;; printf ("%d", strlen(name));

ΜΗΚΟΣ ΣΥΜΒΟΛΟΣΕΙΡΑΣ (5)

Για να μπορέσουμε να χρησιμοποιήσουμε την συνάρτηση strlen( ) πρέπει να ξέρουμε **πού θα βρούμε τον κώδικά της** (τις εντολές της).

```
#include <stdio.h>
#include <string.h>
main()
{
        char name[30];
        printf ("Πώς σε λένε ;\n");
        scanf("%s", name);
        printf ("Το όνομά σου έχει %d γράμματα\n", strlen(name));
}
```

Αρχείο κεφαλίδας **<string.h>**
Η printf( ) και η scanf( ) είναι κι αυτές **συναρτήσεις**. Πού βρίσκεται ο κώδικάς τους (οι εντολές τους);
Αρχείο κεφαλίδας **<stdio.h>**

ΕΙΣΟΔΟΣ-ΕΞΟΔΟΣ ΑΠΛΟΥ ΧΑΡΑΚΤΗΡΑ (1)

Για διάβασμα (είσοδο) **ενός χαρακτήρα** (του ch) από το πληκτρολόγιο:

```
scanf ("%c", &ch);
```

Διάβασμα χαρακτήρα γίνεται και με…..

1. Την **συνάρτηση getchar( )**, η οποία….
   - **Δεν έχει ορίσματα**, δηλαδή καλείται με άδειες παρενθέσεις.
   - Περιμένει να πληκτρολογήσουμε ένα **χαρακτήρα και <Enter>**. Διαβάζει τον χαρακτήρα και **γίνεται ίση με αυτόν**, δηλαδή **έχει τιμή επιστροφής τον χαρακτήρα που διάβασε**.
   - Είναι σωστό το παρακάτω;

     ```
     ch = getchar( );
     ```

2. Την **συνάρτηση getche( )**, η οποία….
   - **Δεν έχει ορίσματα**, δηλαδή καλείται με άδειες παρενθέσεις.
   - Περιμένει να πληκτρολογήσουμε ένα **χαρακτήρα χωρίς <Enter>**. Διαβάζει τον χαρακτήρα και **γίνεται ίση με αυτόν**, δηλαδή **έχει τιμή επιστροφής τον χαρακτήρα που διάβασε**.
   - Π.χ. `ch = getche( );`

ΕΙΣΟΔΟΣ-ΕΞΟΔΟΣ ΑΠΛΟΥ ΧΑΡΑΚΤΗΡΑ (2)

3. Την **συνάρτηση getch( )**, η οποία….

- **Δεν έχει ορίσματα**, δηλαδή καλείται με άδειες παρενθέσεις.
- Περιμένει να πληκτρολογήσουμε ένα **χαρακτήρα χωρίς <Enter>**. Διαβάζει τον χαρακτήρα και **γίνεται ίση με αυτόν**, δηλαδή **έχει τιμή επιστροφής τον χαρακτήρα που διάβασε**. Ο **χαρακτήρας** που πληκτρολογήσαμε **δεν φαίνεται στην οθόνη**.
- Π.χ.

```
ch = getch( );
```

Για εμφάνιση (έξοδο) **ενός χαρακτήρα** (του ch) στην οθόνη:

```
printf (“%c”, ch);
```

Εμφάνιση του χαρακτήρα γίνεται και με την…..

```
putchar (ch);
```

Η **συνάρτηση putchar ( )** χρειάζεται ως **όρισμα** τον **χαρακτήρα**, τον οποίο θέλουμε να γράψουμε στην οθόνη.

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (1)

Εντολή if:

Η **if** μας επιτρέπει **να επιλέξουμε αν θα γίνει ή όχι** κάποια ενέργεια. Συντάσσεται έτσι:

```
if (Έκφραση)
    Εντολή
```

Η Εντολή θα εκτελεστεί αν αυτό που λέει η Έκφραση ισχύει (αν δηλαδή είναι αληθές)

Π.χ.:

```
……
if  (a > 5)
   x++;
……
```

- **Εάν το a είναι μεγαλύτερο του 5**, το x θα αυξηθεί κατά 1. Μετά θα πάμε στην επόμενη εντολή του προγράμματος.
- **Εάν το a δεν είναι μεγαλύτερο του 5**, το x δεν θα αυξηθεί και θα πάμε στην επόμενη εντολή του προγράμματος.

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (2)

Μια παρόμοια εντολή είναι η:

```
if (Έκφραση)
   Εντολή 1
else
   Εντολή 2
```

Η **if…else** μας επιτρέπει **να επιλέξουμε μεταξύ δύο περιπτώσεων.**

Π.χ.:

```
……
if (a > 5)
   x++;
else
   f += 8;
……
```

- **Εάν το a είναι μεγαλύτερο του 5**, το x θα αυξηθεί κατά 1. Μετά θα πάμε στην επόμενη εντολή του προγράμματος.
- **Εάν το a δεν είναι μεγαλύτερο του 5**, το f θα αυξηθεί κατά 8. Μετά θα πάμε στην επόμενη εντολή του προγράμματος.

Θα εκτελεστεί **μόνο η μία** από τις δύο εντολές, αλλά **οπωσδήποτε η μία από τις δύο**

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (3)

ΠΡΟΣΟΧΗ:

- **Πριν το else** υπάρχει **;**
- **Η if…else** είναι ολόκληρη **ΜΙΑ** εντολή.

Σύνθετες προτάσεις:

Θέλω, εάν το a είναι θετικό.....
Να αυξήσω την τιμή του b κατά 1...
Να ελαττώσω την τιμή του c κατά 2…
Να κάνω το d ίσο με το άθροισμα των b και c…
Να γράψω στην οθόνη τις τιμές των b, c και d
Θα γινόταν έτσι;

```
……..
if (a > 0)
   b++;
   c -=2;
   d = b + c;
   printf (“%d%d%d”, b, c, d);
……..
```

Εάν a<0 δεν θα εκτελεστεί το b++, θα εκτελεστούν όμως οι υπόλοιπες εντολές.

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (4)

Όταν θέλουμε **να εκτελείται μια ομάδα εντολών αν ισχύει μια συνθήκη**, αλλά **να μην εκτελείται καμμιά από τις εντολές αν δεν ισχύει η συνθήκη**, τις εντολές αυτές τις ομαδοποιούμε σε ένα…

μπλόκ

δηλαδή:

```
……..
if  (a > 0)
{
     b++;
     c -=2;
     d = b + c;
     printf (“%d%d%d”, b, c, d);
}
……..
```

Μετά το άγκιστρο κλεισίματος σε ένα μπλόκ **ΔΕΝ υπάρχει ;**

Ολόκληρο το μπλόκ θεωρείται **ΜΙΑ εντολή,** η οποία θα εκτελεστεί ολόκληρη ή δεν θα εκτελεστεί καθόλου.

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (5)

Αν πρέπει να διακρίνουμε ανάμεσα **σε περισσότερες από δύο** περιπτώσεις, τότε **δεν επαρκεί μια if…else**. Έτσι…

αν έχουμε **3 περιπτώσεις** χρειαζόμαστε **δύο if…else**

αν έχουμε **4 περιπτώσεις** χρειαζόμαστε **τρείς if…else** κλπ

μιλάμε δηλαδή για **πολλαπλές if…else**

Π.χ:

Θέλουμε, εάν a>0 να γράφεται στην οθόνη Θετικός, εάν a<0 να γράφεται Αρνητικός και εάν a ίσο με μηδέν να γράφεται Μηδέν

```
……..
if  (a > 0)
   printf (“Θετικός”);
else
    if  (a < 0)
        printf (“Αρνητικός”);
    else
        printf (“Μηδέν”);
……..
```

Κάθε else «κολλάει» **με το πιο κοντινό του if προς τα πίσω**, το οποίο δεν είναι ήδη «πιασμένο», δηλαδή δεν χρησιμοποιείται.

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (6)

Στην περίπτωση **πολλών διαδοχικών if….else** προτιμούμε συνήθως την ισοδύναμη μέθοδο της χρήσης **πολλών μεμονωμένων if**. Έτσι, το προηγούμενο παράδειγμα….

```
……..
if  (a > 0)
    printf (“Θετικός”);
else
    if  (a < 0)
        printf (“Αρνητικός”);
    else
        printf (“Μηδέν”);
```

…μπορεί ισοδύναμα να γραφεί έτσι:

```
……..
if  (a > 0)
    printf (“Θετικός”);
if  (a < 0)
        printf (“Αρνητικός”);
if  (a == 0)
        printf (“Μηδέν”);
……..
```


ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (7)

Το προηγούμενο παράδειγμα:

```
……..
if  (a > 0)
    printf (“Θετικός”);
else
    if  (a < 0)
        printf (“Αρνητικός”);
    else
        printf (“Μηδέν”);
……..
```

Θα μπορούσε να γραφεί έτσι;

```
……..
if  (a > 0)
    printf (“Θετικός”);
if  (a < 0)
        printf (“Αρνητικός”);
else
        printf (“Μηδέν”);
……..
```

Αν a >0 στην οθόνη θα γραφεί:

ΘετικόςΜηδέν

…αφού **το else ανήκει μόνο στο τελευταίο if** και **ΟΧΙ και στα δύο**

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (8)

Τα άγκιστρα χρησιμοποιούνται και **για να τροποποιούν** το ποιο else «κολλάει» σε ποιο if.

```
if (num > 6)
    if (num < 12)
        printf ("Πλησιάζεις! \n");
    else
        printf ("Έχασες! \n");
```

Εδώ το else ανήκει στο 2ο if

- Αν το num είναι 4 πάμε στην επόμενη εντολή
- Αν το num είναι 10 γράφει Πλησιάζεις
- Αν το num είναι 15 γράφει Έχασες

```
if (num > 6)
{
    if (num < 12)
        printf ("Πλησιάζεις!
       \n");
}
    else
        printf ("Έχασες! \n");
```

Εδώ το else ανήκει στο 1ο if

- Αν το num είναι 4 γράφει Έχασες
- Αν το num είναι 10 γράφει Πλησιάζεις
- Αν το num είναι 15 πάμε στην επόμενη εντολή

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ if (9)

Τελεστής υπό συνθήκη:

Μια άλλη μορφή του if…else. Το παρακάτω:

```
x = (y > 0)? 8 : -3;
```

είναι ισοδύναμο με:

```
if (y > 0)
    x = 8;
else
    x = -3;
```

Αλήθεια – ψεύδος:

Στην εντολή

```
if (ak > 5)
    x++;
```

Το x θα αυξηθεί κατά 1 εάν το ak είναι μεγαλύτερο του 5, ή αλλιώς **εάν το ak>5 είναι αληθές**. Όμως…..

Τι σημαίνει ΑΛΗΘΕΙΑ;;;;

ΣΧΕΣΕΙΣ – ΣΧΕΣΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (1)

Η **«Αλήθεια»** και το **«Ψέμα»** πρέπει να είναι **μετρήσιμες ποσότητες** για τον υπολογιστή.

Στην C:

Ψευδές είναι ο,τιδήποτε έχει τιμή **μηδέν**.

Ο,τιδήποτε έχει **τιμή διάφορη του μηδενός** θεωρείται **αληθές**.

Σχεσιακοί τελεστές – σχέσεις:

>	Μεγαλύτερο
<	Μικρότερο
>=	Μεγαλύτερο ή ίσον
<=	Μικρότερο ή ίσον
==	Ίσον
!=	Διάφορο

Όπου χρησιμοποιείται ένας σχεσιακός τελεστής υπάρχει και μια…

ΣΧΕΣΕΙΣ – ΣΧΕΣΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (2)

Το παρακάτω…

a > b

είναι μια **σχέση** μεταξύ των a και b, η οποία μπορεί **να ισχύει** ή **να μην ισχύει.**

Κάθε σχέση έχει αποτέλεσμα...

Εάν μια σχέση **δεν ισχύει**, το **αποτέλεσμά** της είναι **0** (ψευδές).

Εάν μια σχέση **ισχύει**, το **αποτέλεσμά** της είναι **1**

ΠΡΟΣΟΧΗ! **Αληθής** είναι **ΚΑΘΕ τιμή διάφορη του μηδενός**. Όμως…

Οι αληθείς σχέσεις έχουν τιμή 1

Άρα:

Αν…	k = 5 > 3;	το k έχει τιμή….	**1**
Αν…	k = 3 < 2 + 1;	το k έχει τιμή….	**0**
Αν…	k = 7 == 5;	το k έχει τιμή….	**0**
Η…	printf (“%d”, 10 > 3);	θα γράψει στην οθόνη….	**1**
Αν το a έχει τιμή 7, η	printf (“%d”, !(a < 8));	θα γράψει στην οθόνη….	**0**

ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (1)

Περίπτωση 1

Διαβάζεται ο ακέραιος ak. Θέλουμε, εάν ο ak είναι…

μεγαλύτερος από 5……**ΚΑΙ**……μικρότερος από 10

η τιμή της μεταβλητής x να αυξάνεται κατά 1.

Γράφουμε την παρακάτω εντολή:

```
if (5 < ak < 10)
  x++;
```

Έστω ότι το ak είναι 7. Τότε… 5 < 7 < 10 → 1 < 10 → **1 (αληθές)**
Έστω ότι το ak είναι 2. Τότε… 5 < 2 < 10 → 0 < 10 → **1 (αληθές)**
Έστω ότι το ak είναι 15. Τότε… 5 < 15 < 10 → 1 < 10 → **1 (αληθές)**

Άρα **η συνθήκη είναι πάντα αληθής**, άσχετα από την τιμή του ak όμως….

ΔΕΝ θέλαμε αυτό

Αυτό συνέβη γιατί **χειριστήκαμε μια ΔΙΠΛΗ σχέση σαν να είναι ΑΠΛΗ**.

ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (2)

Γράφουμε χωριστά τις δύο απλές σχέσεις….

ak > 5 και ak < 10

και τις συνδέουμε με τον λογικό τελεστή…

&& (**AND** ή **ΛΟΓΙΚΟ ΚΑΙ**)

Άρα:

```
if ((ak >5) && (ak < 10))
  x++;
```

Η σχέση στις παρενθέσεις της if **είναι αληθής** όταν **ΚΑΙ ΟΙ ΔΥΟ επιμέρους σχέσεις είναι αληθείς** (το ίδιο ισχύει και για περισσότερες από δύο απλές σχέσεις)

Αντίστροφα:

Αν θέλουμε **μια σχέση να είναι αληθής όταν όλες οι επιμέρους σχέσεις** που την αποτελούν **είναι αληθείς**, τότε αυτές **τις συνδέουμε** μεταξύ τους **με τον λογικό τελεστή &&**

ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (3)

Περίπτωση 2

Διαβάζεται ο ακέραιος ak. Θέλουμε, εάν ο ak είναι…

μικρότερος από 5……**'Η**…… μεγαλύτερος από 10

η τιμή της μεταβλητής x να αυξάνεται κατά 1.

Τώρα χρησιμοποιούμε τον λογικό τελεστή…

|| (**OR** ή **ΛΟΓΙΚΟ 'Η**)

Άρα:

```
if ((ak < 5) || (ak > 10))
    x++;
```

Η σχέση στις παρενθέσεις της if **είναι αληθής** όταν **ΜΙΑ ΤΟΥΛΑΧΙΣΤΟΝ από τις επιμέρους σχέσεις είναι αληθείς** (το ίδιο ισχύει και για περισσότερες από δύο απλές σχέσεις)

Αντίστροφα:

Αν θέλουμε **μια σχέση να είναι αληθής όταν μία τουλάχιστον από τις επιμέρους σχέσεις** που την αποτελούν **είναι αληθείς**, τότε αυτές **τις συνδέουμε** μεταξύ τους **με τον λογικό τελεστή ||**

ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (4)

Η…

```
if ((ak > 5) && (ak < 10))
    x++;
```

και η…

```
if ((ak < 5) || (ak > 10))
    x++;
```

Είναι άραγε ισοδύναμες με τις →

```
if (ak > 5 && ak < 10)
    x++;
```

και…

```
if (ak < 5 || ak > 10)
    x++;
```

Οι τελεστές **&&** και **||** έχουν **μεγαλύτερη προτεραιότητα από** τον **>** και τον **<**, άρα **είναι ισοδύναμες**.

Τα παρακάτω είναι προφανώς συντακτικά λάθη:

```
if (ak > 5) && (ak < 10)
    x++;
```

και…

```
if (ak < 5) || (ak > 10)
    x++;
```

ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (5)

Περίπτωση 3

Το k στην παρακάτω εντολή...

```
k = 5 > 3;
```

ισούται με... **1**

Το k στην παρακάτω εντολή...

```
k = !(5 > 3);
```

ισούται με... **0**

Το **!** είναι ο τελεστής **NOT** ή **Λογικό ΟΧΙ** και χρησιμοποιείται **ΜΟΝΟ** σε **λογικές πράξεις.**

Άρα η παράσταση...

```
! (5 – 3) + !2 – 3
```

έχει τιμή....

```
!2 + !2 – 3
0 + 0 – 3
-3
```

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ switch (1)

Η εντολή **switch** παίζει τον ρόλο **πολλών αμοιβαία αποκλειόμενων if**. **Πολλών** δηλαδή **if** από τις οποίες **θα εκτελεστεί μόνο η μία**.

```
int day;
printf ("Δώστε ένα αριθμό από 1 έως 7 \n");
scanf ("%d", &day);
switch (day)
{
    case 1:  printf ("Κυριακή.\n");
             break;
    case 2:  printf ("Δευτέρα.\n");
             break;
    case 3:  printf ("Τρίτη.\n");
             break;
    case 4:  printf ("Τετάρτη.\n");
             break;
    case 5:  printf ("Πέμπτη.\n");
             break;
    case 6:  printf ("Παρασκευή.\n");
             break;
    case 7:  printf ("Σάββατο.\n");
             break;
    default:  printf ("Λάθος\n");
             break;
}
printf ("Καλημέρα σας");
```

Παρατηρήσεις:

- Η ύπαρξη του default **δεν είναι υποχρεωτική**. Αν δεν υπάρχει, και το day είναι για παράδειγμα 9, **δεν θα εκτελεστεί κανένα case**.
- **Οι τιμές** στα διάφορα case **εξαρτώνται από το πρόγραμμα** (δεν είναι αναγκαστικά 1, 2, 3,… κλπ) και **δεν είναι αναγκαστικά στη σειρά**.

Πώς θα γινόταν το διπλανό πρόγραμμα με απλές if;

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ switch (2)

```
int day;
printf ("Δώστε ένα αριθμό από 1 έως 7 \n");
scanf ("%d", &day);
switch (day)
{
     case 1:  printf ("Κυριακή.\n");
          break;
     case 2:  printf ("Δευτέρα.\n");
          break;
     case 3:  printf ("Τρίτη.\n");
          break;
     case 4:  printf ("Τετάρτη.\n");
          break;
     case 5:  printf ("Πέμπτη.\n");
          break;
     case 6:  printf ("Παρασκευή.\n");
          break;
     case 7:  printf ("Σάββατο.\n");
          break;
     default:  printf ("Λάθος\n");
          break;
}
printf ("Καλημέρα σας");
```

Έτσι;;;;;

```
if (day == 1)
     printf ("Κυριακή.\n");
if (day == 2)
     printf (" Δευτέρα.\n");
if (day == 3)
     printf ("Τρίτη.\n");
if (day == 4)
     printf ("Τετάρτη.\n");
if (day == 5)
     printf ("Πέμπτη.\n");
if (day == 6)
     printf ("Παρασκευή.\n");
if (day == 7)
     printf ("Σάββατο.\n");
else
     printf ("Λάθος\n");
printf ("Καλημέρα σας");
```


ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ switch (3)

```
switch (day)
{
    case 1:  printf ("Κυριακή.\n");
             break;
    case 2:  printf ("Δευτέρα.\n");
             break;
    case 3:  printf ("Τρίτη.\n");
             break;
    case 4:  printf ("Τετάρτη.\n");
             break;
    case 5:  printf ("Πέμπτη.\n");
             break;
    case 6:  printf ("Παρασκευή.\n");
             break;
    case 7:  printf ("Σάββατο.\n");
             break;
    default:  printf ("Λάθος\n");
             break;
}
```

Το σωστό (μια εκδοχή) θα ήταν:

```
if (day == 1)
     printf ("Κυριακή.\n");
if (day == 2)
     printf (" Δευτέρα.\n");
if (day == 3)
     printf ("Τρίτη.\n");
if (day == 4)
     printf ("Τετάρτη.\n");
if (day == 5)
     printf ("Πέμπτη.\n");
if (day == 6)
     printf ("Παρασκευή.\n");
if (day == 7)
     printf ("Σάββατο.\n");
if (day!=1 && day!=2 && day!=3 &&
    day!=4 && day!=5 && day!=6 &&
    day!=7)
          printf ("Λάθος\n");
```

ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ – ΕΝΤΟΛΗ switch (4)

```
char let;
scanf ("%c", &let);
switch (let)
{
   case 'A':
   case 'E': printf ("Φωνήεν");
             break;
   case 'B':
   case 'Γ':
   case 'Δ': printf ("Σύμφωνο");
             break;
   default:
      printf ("Οχι στα 5 πρώτα γράμματα");
             break;
}
```

Παρατηρήσεις:

- **Άν μετά την : κάποιου case δεν υπάρχουν εντολές ή υπάρχουν εντολές χωρίς break**, η εκτέλεση περνάει στις **εντολές του επόμενου ή των επόμενων case μέχρι να συναντήσουμε break**.
- Το διπλανό switch στην πραγματικότητα **περιέχει δύο OR**, δηλαδή:

```
if (let=='A' || let =='E')
    printf ("\nΦωνήεν.\n");
else
   if (let=='B' || let =='Γ' || let =='Δ' )
       printf ("Σύμφωνο");
   else
       printf ("Οχι στα 5 πρώτα γράμματα");
```


ΕΠΑΝΑΛΗΠΤΙΚΕΣ ΕΝΤΟΛΕΣ

Οι **επαναληπτικές εντολές** εκτελούν ένα **κομμάτι του προγράμματος πολλές φορές**.

Η C έχει **τρείς** επαναληπτικές εντολές….

- την **for**
- την **while** και….
- την **do…while**

Όταν διαπιστώσουμε ότι σε ένα πρόγραμμα χρειαζόμαστε επαναληπτική εντολή πρέπει να αποφασίσουμε, δηλαδή να γράψουμε με εντολές, **τι θα κάνει το πρόγραμμά μας τη μία φορά**. Μετά, αυτές τις εντολές θα τις βάλουμε «μέσα» στην επανάληψη.

ΕΝΤΟΛΗ for (1)

Η for συντάσσεται όπως παρακάτω:

```
……..
for (met=1; met<=10; met++)
     printf (“ΚΑΛΗΜΕΡΑ\n”);
……..
```

Στις παρενθέσεις υπάρχουν **τρείς προτάσεις** που **χωρίζονται με το ;**
Το for εκτελείται ως εξής:

- Την **πρώτη φορά** που φτάνουμε στο for **εκτελείται η πρώτη πρόταση**. Αυτό γίνεται **ΜΟΝΟ** φτάνοντας στο for για πρώτη φορά.
- **Ελέγχουμε αν ισχύει** αυτό που λέει **η δεύτερη πρόταση**. **Αν ισχύει, εκτελείται η εντολή που ακολουθεί**, αλλιώς το for τερματίζεται.
- **Εκτελείται** αυτό που λέει **η τρίτη πρόταση**.
- **Ελέγχουμε** πάλι **αν ισχύει η δεύτερη πρόταση**. **Αν ισχύει, εκτελείται η εντολή που ακολουθεί**, αλλιώς το for τερματίζεται.
- **Εκτελείται** αυτό που λέει **η τρίτη πρόταση** κλπ, και ο κύκλος επαναλαμβάνεται **μέχρι να πάψει να ισχύει η δεύτερη πρόταση**.

ΕΝΤΟΛΗ for (2)

Παραδείγματα:

1.
```
for (num=0; num<=10; num++)
     printf ("%5d %8d\n", num, num*num*num);
```

2.
```
for (num=0; num<=10; num=num+2)
     printf ("%5d %8d\n", num, num*num*num);
```

3.
```
for (n=10; n>0; n--)
     printf ("%d επαναλήψεις ακόμη\n",n);
printf ("Εκκίνηση!");
```

4.
```
for (k=1; k<=10000; k++)
     ;
```

5.
```
for (k=15; k<=10; k++)
    x++;
```

6.
```
for (ch='a'; ch<='z'; ch++)
     printf ("Η ASCII τιμή του %c είναι %d\n", ch, ch);
```

ΕΝΤΟΛΗ for (3)

Παραδείγματα:

7.
```
int y=55, x;
for (x=1; y<=75; y=++x*5+50)
    printf ("%5d %5d\n", x, y);
```

8.
```
int y=55, x;
for (x=1; y<=75; y=x++*5+50)
    printf ("%5d %5d\n", x, y);
```

9.
```
int n, ak;
for (n=1; n<=10; n++)
{
    scanf (“%d”, &ak);
    if (ak % 2 == 0)
       printf (”Ο %d είναι άρτιος\n", ak);
    else
       printf (”Ο %d είναι περιττός\n", ak);
}
```

Οι εντολές μέσα στα άγκιστρα αποτελούν μια **σύνθετη εντολή.**

ΕΝΤΟΛΗ for (4)

Παραδείγματα:

10. Θέλουμε να **αθροίσουμε** τους αριθμούς από το 1 έως το 100. Πρέπει να γράψουμε με εντολές **τι θα κάνει το πρόγραμμα ΚΑΘΕ φορά**…

```
s = s + k;
```

Αυτό πρέπει να γίνεται **για όλα τα k**, άρα…

```
……….
for (k=1; k<=100; k++)
    s = s + k;
……….
```

Το s είναι μια μεταβλητή που χρησιμοποιείται ως…

αθροιστής

Στους αθροιστές δίνουμε πάντα **αρχική τιμή μηδέν**, άρα χρειάζονται οι δηλώσεις…

```
int k, s=0;
```

Αρχική τιμή μηδέν δίνουμε και στους **μετρητές**, δηλαδή σε μεταβλητές, οι οποίες **σε κάθε επανάληψη αυξάνονται κατά 1**, αλλιώς….

οι μεταβλητές αυτές έχουν **ΤΥΧΟΥΣΑ αρχική τιμή**

ΕΝΤΟΛΗ for (5)

Παραδείγματα:

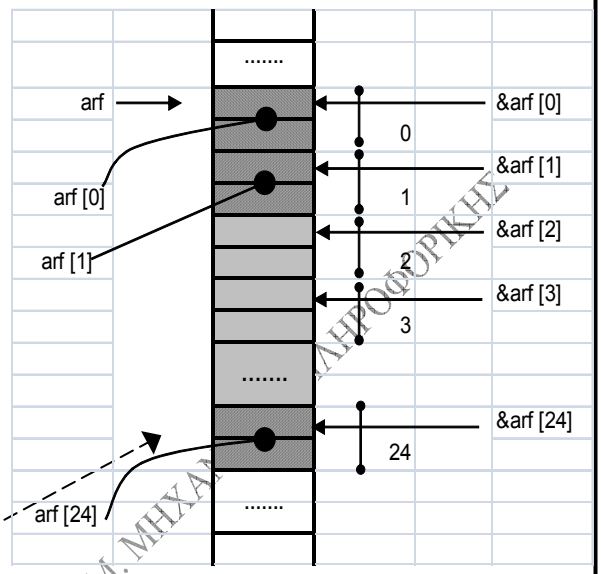
11. Θέλουμε να **γεμίσουμε** από το πληκτρολόγιο ένα πίνακα ακεραίων 25 θέσεων, τον arf. Θυμίζουμε ότι…
αν σε ένα πρόγραμμα έχουμε μια μεταβλητή, την ak, τότε αυτή **βρίσκεται αποθηκευμένη** εκεί που δείχνει ο δείκτης…

&ak

Ο arf είναι μια «συλλογή» 25 μεταβλητών, των…

arf [0], arf [1], ….., arf [23], arf [24]

Η κατάσταση της μνήμης φαίνεται δίπλα…

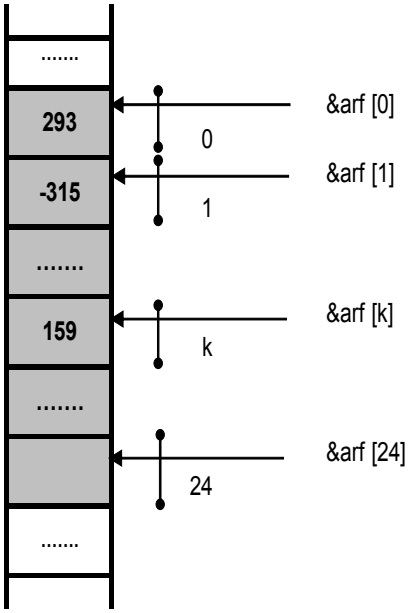


ΕΝΤΟΛΗ for (6)

Παραδείγματα:

11. (Συνέχεια)
Διαβάζουμε το πρώτο στοιχείο του πίνακα….
scanf (“%d”, &arf[0]);
Διαβάζουμε το δεύτερο στοιχείο του πίνακα….
scanf (“%d”, &arf[1]);
Διαβάζουμε το k στοιχείο….
scanf (“%d”, &arf[k]);
Για να διαβάσουμε **όλα** τα στοιχεία του πίνακα, **το k πρέπει να μεταβάλλεται** από 0 έως και 24, άρα:

```
for (k=0; k<25; k++)
    scanf (“%d”, &arf[k]);
```



ΕΝΤΟΛΗ for (7)

Παραδείγματα:

11. (Συνέχεια)

Τα παρακάτω είναι ισοδύναμα;

```
for (k=0; k<25; k++)
    scanf ("%d", &arf[k]);
```

και

```
for (k=1; k<=25; k++)
    scanf ("%d", &arf[k]);
```

Το **πλήθος** των στοιχείων που διαβάζονται είναι **το ίδιο**, ΟΜΩΣ....

Στην δεύτερη περίπτωση αφ' ενός μεν **έχουμε «χάσει» ένα στοιχείο του πίνακα (το στοιχείο 0)**, αφ' ετέρου δε, **το τελευταίο στοιχείο (το 25ο) είναι εκτός των ορίων του πίνακα).**

ΕΝΤΟΛΗ for (8)

Παραδείγματα:

12. Πώς μπορούμε **να αθροίσουμε μεταξύ τους** όλα τα στοιχεία ενός πίνακα ακεραίων, 25 θέσεων, του arf:

Κάθε φορά το πρόγραμμα πρέπει να κάνει:

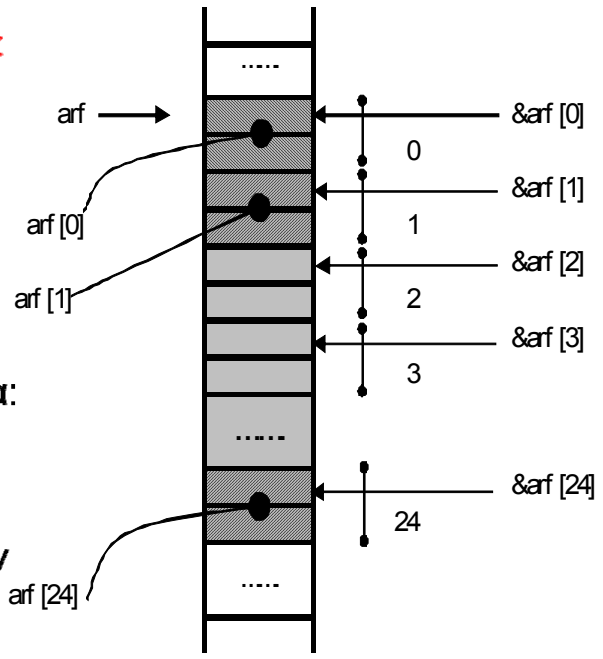
```
s = s + arf [k];
```

Για όλα τα στοιχεία του πίνακα:

```
for (k=0; k<=24; k++)
    s = s + arf [k];
```

Το **s είναι αθροιστής**, άρα, πριν το for πρέπει:

```
s = 0;
```



ΕΝΤΟΛΗ while (1)

```
int  index;
……..
if (index < 5)
     printf ("ΚΑΛΗΜΕΡΑ\n");
```

Εάν το index είναι μικρότερο του 5, γράφεται: ΚΑΛΗΜΕΡΑ
Η εντολή **while**:

```
while (index < 5)
     printf ("ΚΑΛΗΜΕΡΑ\n");
```

κάνει αντίστοιχο έλεγχο, δηλαδή εάν το index είναι μικρότερο του 5, γράφεται: ΚΑΛΗΜΕΡΑ

ΟΜΩΣ …

Ο έλεγχος γίνεται πάλι… και πάλι… και πάλι… και **η printf εκτελείται συνεχώς μέχρι κάποια στιγμή η συνθήκη του while να γίνει ψευδής**, οπότε θα σταματήσει η επανάληψη

ΕΝΤΟΛΗ while (2)

Πόσες φορές θα εκτελεστεί το παρακάτω;

```
index = 3;
while (index < 8)
printf ("Τι νέα;\n");
```

Η επανάληψη **δεν τελειώνει ποτέ**, διότι **η συνθήκη παραμένει πάντα αληθής**, άρα έχουμε ένα…

ατέρμονα βρόχο

Στο επόμενο η συνθήκη γίνεται κάποια στιγμή ψευδής με την διαρκή αύξηση του index:

```
index = 10;
while (++index < 15)
{
     printf ("%d", k);
     printf ("Συνεχίστε !\n");
}
```

ΕΝΤΟΛΗ while (3)

```
index = 10;
while (++index < 15)
{
     printf ("%d", k);
     printf ("Συνεχίστε !\n");
}
```

Index	Επανάληψη	Οθόνη
10		
11	ΝΑΙ	11 Συνεχίστε
12	ΝΑΙ	12 Συνεχίστε
13	ΝΑΙ	13 Συνεχίστε
14	ΝΑΙ	14 Συνεχίστε
15	ΟΧΙ	

Δείτε τι θα κάνει:

Παρατηρείστε την διαφορά:

```
index = 10;
while (index++ < 15)
{
     printf ("%d", k);
     printf ("Συνεχίστε !\n");
}
```

Επανάληψη	Index	Οθόνη
	10	
ΝΑΙ	11	11 Συνεχίστε
ΝΑΙ	12	12 Συνεχίστε
ΝΑΙ	13	13 Συνεχίστε
ΝΑΙ	14	14 Συνεχίστε
ΝΑΙ	15	15 Συνεχίστε
ΟΧΙ	16	

ΕΝΤΟΛΗ while (4)

Στο παρακάτω πρόγραμμα **διαβάζουμε συνεχώς ακεραίους** από το πληκτρολόγιο, **όσο** οι ακέραιοι αυτοί **είναι θετικοί**.

```
int ak, k=0;
……..……
scanf (“%d”, &ak);
while (ak > 0)
{
    printf ("%d", ak+1);
    k++;
    scanf (“%d”, &ak);
}
```

ak	while	printf ( )	k
5	ΝΑΙ	6	1
10	ΝΑΙ	11	2
17	ΝΑΙ	18	3
6	ΝΑΙ	7	4
-3	ΟΧΙ		

Ακριβέστερα…..

Θυμόμαστε ότι….

Αν σε μια μεταβλητή δεν έχουμε δώσει τιμή, αυτή έχει τυχούσα τιμή

ΕΝΤΟΛΗ while (5)

Τι θα συνέβαινε **αν δεν υπήρχε η πρώτη scanf( )** στο προηγούμενο παράδειγμα; Δηλαδή:

```
int ak, k=0;
……….
while (ak > 0)
{
    printf ("%d", ak+1);
    k++;
    scanf (“%d”, &ak);
 }
```

ak	while	printf ( )	k	Παρατήρηση
8	ΝΑΙ	9	1	Μη επιλεγμένο
10	ΝΑΙ	11	2	Κανονικό
17	ΝΑΙ	18	3	Κανονικό
6	ΝΑΙ	7	4	Κανονικό
-3	ΟΧΙ			Κανονικό

Έστω ότι το ak έχει αρχικά (τυχούσα) τιμή 8

Έστω ότι το ak έχει αρχικά (τυχούσα) τιμή -3. Τότε…

Η επανάληψη δεν θα εκτελεστεί καμμιά φορά, διότι η συνθήκη είναι από την αρχή ψευδής

ΕΝΤΟΛΗ while (6)

Πρώτη παρατήρηση στο while

Το τι θα γίνει **την πρώτη φορά** το έχουμε ρυθμίσει

ΠΡΙΝ

φτάσουμε στην επανάληψη. Αλλιώς έχουμε αφήσει τα πράγματα στην τύχη τους και **η επανάληψη μπορεί να μην εκτελεστεί καμμιά φορά.**

Παρατηρείστε το παρακάτω:

Είχαμε …

```
int ak, k=0;
……….
scanf (“%d”, &ak);
while (ak > 0)
{
    printf ("%d", ak+1);
    k++;
    scanf (“%d”, &ak);
 }
```

Τώρα …

```
int ak, k=0;
……….
scanf (“%d”, &ak);
while (ak > 0)
{
    printf ("%d", ak+1);
    k++;
 }
```

ΕΝΤΟΛΗ while (7)

```
int ak, k=0;
……….
scanf (“%d”, &ak);
while (ak > 0)
{
     printf ("%d", ak+1);
     k++;
}
```

Αν με τη scanf( ) διαβάσουμε π.χ. τιμή -5 για το ak, η επανάληψη δεν εκτελείται. Όμως…

ak		while	printf ( )	k
8		ΝΑΙ	9	1
8	(δεν διαβάζεται άλλη τιμή)	ΝΑΙ	9	2
8	(δεν διαβάζεται άλλη τιμή)	ΝΑΙ	9	3

Συνεχίζεται επ’ άπειρον διότι η συνθήκη δεν γίνεται **ΠΟΤΕ** ψευδής

ΕΝΤΟΛΗ while (8)

Δεύτερη παρατήρηση στο while

Στις εντολές του while πρέπει να υπάρχει κάποια (ή κάποιες), η οποία θα κάνει τη συνθήκη του while ψευδή, ώστε η επανάληψη να σταματήσει. Αλλιώς, αν η επανάληψη ξεκινήσει, δεν θα σταματήσει ποτέ.

Τι θα κάνει το παρακάτω πρόγραμμα;

```
char  ch;
while ((ch = getche()) !=´*´)
   putchar(ch);
```

Γράφει στην οθόνη τον χαρακτήρα που διαβάζει, **μέχρι να δοθεί το ***.
Πώς γίνεται αυτό;

Το παρακάτω κάνει το ίδιο;

```
while (ch = getche() !=´*´)
   putchar(ch);
```

ΕΝΤΟΛΗ do-while (1)

Γενική μορφή, όπως στο παράδειγμα:

```
int ak, count=0;
……..
do
{
    scanf ("%d", &ak);
    count++;
}
while (ak != 9);
```

ak	count	do-while
;	0	ΝΑΙ
8	1	ΝΑΙ
17	2	ΝΑΙ
-6	3	ΝΑΙ
9	4	ΟΧΙ

ΕΝΤΟΛΗ do-while (2)

```
int ak, count=0;
……..
do
{
   scanf ("%d", &ak);
   count++;
}
while (ak != 9);
```

Παρατηρείστε ότι **αν ως πρώτη-πρώτη τιμή του ak δώσετε το 9** το do-while **θα εκτελεστεί μία φορά** και μετά θα τερματιστεί. Αυτό γιατί **ο έλεγχος γίνεται στο τέλος της επανάληψης**.

Άρα, σε κάθε περίπτωση…

Οι εντολές του do-while εκτελούνται τουλάχιστον μια φορά.

ΕΠΑΝΑΛΗΨΕΙΣ ΓΕΝΙΚΑ

Κάθε επαναληπτική εντολή μπορεί να κάνει **ό,τι και οποιαδήποτε** άλλη.
Επιλέγουμε εκείνη την εντολή που «μας βολεύει» περισσότερο, ανάλογα με το πρόβλημα. Έτσι…

Αν είναι **εκ των προτέρων γνωστός ο αριθμός των επαναλήψεων** μας βολεύει συνήθως η **for**.

Αν **δεν είναι εκ των προτέρων γνωστός ο αριθμός των επαναλήψεων** μας βολεύει συνήθως η **while** ή η **do-while**.
Ποια από τις δυό;

Αν η επανάληψη ξέρουμε ότι **πρέπει να γίνει τουλάχιστον μια φορά**, τότε η **do-while**.

Αν η επανάληψη **μπορεί και να μη γίνει καμμιά φορά**, τότε η **while**.

Στατιστικά, το **do-while** μας βολεύει **μόνο στο 5%** περίπου των επαναλήψεων.

ΦΩΛΙΑΣΜΕΝΕΣ ΕΠΑΝΑΛΗΨΕΙΣ (1)

Μέσα σε μια επαναληπτική εντολή μπορώ να έχω οποιαδήποτε εντολή, ακόμα και **μια άλλη επαναληπτική**. Π.χ. for μέσα σε for, while μέσα σε for, do-while μέσα σε while κλπ.

Πώς θα βγάλω το παρακάτω στην οθόνη;

```
*****
*****
*****
```

Στοιχειώδης ενέργεια: το **να γράψω ένα αστεράκι** στην οθόνη, δηλαδή…

```
putchar ('*');
```

Για τα πέντε αστεράκια κάθε γραμμής:

```
int j;
for (j=1; j<=5; j++)
    putchar ('*');
```

ΦΩΛΙΑΣΜΕΝΕΣ ΕΠΑΝΑΛΗΨΕΙΣ (2)

Θέλω:

```
*****
*****
*****
```

Και για τις 3 γραμμές:

```
int j, k;
for (k=1; k<=3; k++)
{
    for (j=1; j<=5; j++)
        putchar (´*´);
}
```

Έχει το ζητούμενο αποτέλεσμα;

k=1						k=2						k=3						k=4
j=1	j=2	j=3	j=4	j=5	j=6	j=1	j=2	j=3	j=4	j=5	j=6	j=1	j=2	j=3	j=4	j=5	j=6	Τέλος
*	*	*	*	*		*	*	*	*	*		*	*	*	*	*		

Δεν είναι σωστό διότι **υπάρχει λογικό λάθος**

ΦΩΛΙΑΣΜΕΝΕΣ ΕΠΑΝΑΛΗΨΕΙΣ (3)

ΔΕΝ θέλαμε 3 φορές να γράψουμε 5 αστεράκια, αλλά…

3 φορές να γράψουμε 5 αστεράκια **ΚΑΙ να αλλάξουμε γραμμή**, δηλαδή:

```
int j, k;
for (k=1; k<=3; k++)
{
    for (j=1; j<=5; j++)
        putchar (´*´);
    putchar (´\n´);
}
```

Πώς θα εμφανίσουμε το παρακάτω στην οθόνη;

```
ABCDEF
ABCDEF
ABCDEF
ABCDEF
ABCDEF
ABCDEF
```

ΦΩΛΙΑΣΜΕΝΕΣ ΕΠΑΝΑΛΗΨΕΙΣ (4)

```
int k;
char ch;
for (k=0; k<6; k++)
{
     for (ch='A'; ch< 'G'; ch++)
          putchar (ch);
     putchar ('\n');
 }
```

Τροποποιώντας ελαφρά τις παραπάνω εντολές, μπορείτε να εμφανίσετε το παρακάτω στην οθόνη;

ABCDEF
ABCDE
ABCD
ABC
AB
A

ΔΥΟ ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ ΕΙΔΙΚΗΣ ΧΡΗΣΗΣ (1)

Η πρόταση **break**:
Την έχουμε συναντήσει στην switch. Είναι ιδιαίτερα χρήσιμη διότι…

Διακόπτει «βίαια» την εκτέλεση των προτάσεων **for, while και do-while**.

Λέγοντας **«βίαια»** εννοούμε ότι **η επανάληψη διακόπτεται ενώ η συνθήκη** του for, του while ή του do-while **εξακολουθεί να ισχύει**.

```
int  ak, sum=0,
count=0;
……..
scanf ("%d", &ak);
while (ak != 9)
{
    sum += ak;
    if (sum > 100)
       break;
    count++;
    scanf ("%d", &ak);
}
```

ak	while	sum	count
;		0	0
10	ΝΑΙ	10	1
30	ΝΑΙ	40	2
40	ΝΑΙ	80	3
25	ΝΑΙ	105	Τέλος επανάληψης

(Ενώ η συνθήκη του while εξακολουθεί να ισχύει)

ΔΥΟ ΠΡΟΤΑΣΕΙΣ ΕΛΕΓΧΟΥ ΕΙΔΙΚΗΣ ΧΡΗΣΗΣ (2)

Η πρόταση **continue**:
Έχει σαν αποτέλεσμα τη **μη εκτέλεση του τμήματος του βρόχου από την continue μέχρι το τέλος** και την **επανάληψή του βρόχου από την αρχή**.

```
char ch;
int    num=0;
……..
while ((ch=getchar( )) != ´*´)
{
      if (ch == 'A')
            continue;
      num++;
      printf ("%3d", num);
}
```

ch	while	ch=='A'	num	printf
;			0	
K	ΝΑΙ	Ψευδές	1	uu1
X	ΝΑΙ	Ψευδές	2	uu2
F	ΝΑΙ	Ψευδές	3	uu3
A	ΝΑΙ	Αληθές		
S	ΝΑΙ	Ψευδές	4	uu4
A	ΝΑΙ	Αληθές		
*	ΟΧΙ			

Τέλος

Το πρόγραμμα **διαβάζει συνεχώς χαρακτήρες** από το πληκτρολόγιο, τους μετρά και γράφει το αποτέλεσμα στην οθόνη, **εκτός από τα A**

Και τώρα…..

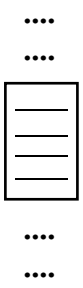
ΣΥΝΑΡΤΗΣΕΙΣ

ΣΥΝΑΡΤΗΣΕΙΣ (1)

Συνάρτηση είναι **αυτοδύναμη μονάδα κώδικα** προγράμματος που έχει σχεδιαστεί για να εκτελεί μια συγκεκριμένη εργασία.

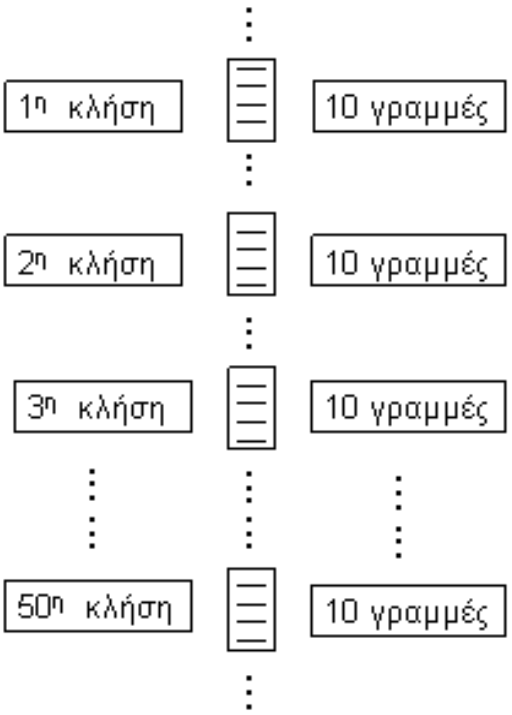
- **Μονάδα κώδικα**, δηλαδή όχι μια εντολή, αλλά ομάδα εντολών
- **Αυτοδύναμη**, δηλαδή περιέχει ό,τι της χρειάζεται. Όμως…
  δεν δουλεύει μόνη της, πρέπει να «δεθεί» κατάλληλα με το υπόλοιπο πρόγραμμα.

Μας γλυτώνουν από τον **επαναληπτικό προγραμματισμό**.
Π.χ. Υπολογισμός τετραγωνικής ρίζας: 10 εντολές



Αν χρειαζόμουνα την τετραγωνική ρίζα σε πενήντα θέσεις;

ΣΥΝΑΡΤΗΣΕΙΣ (2)



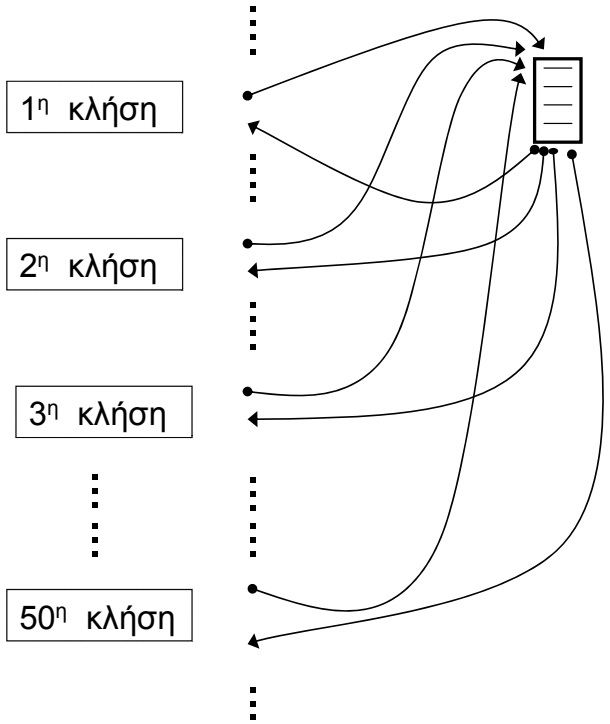
Δηλαδή... 500 γραμμές

ΜΟΝΟ

για τη συνάρτηση (ίδιος κώδικας κάθε φορά)

Υπάρχει άλλη προσέγγιση;

ΣΥΝΑΡΤΗΣΕΙΣ (3)



10 γραμμές

Δηλαδή…

10 γραμμές για τη συνάρτηση

και επιπλέον 50 γραμμές για την κλήση της, αφού…

κάθε συνάρτηση έχει όνομα και καλείται με αυτό

ΣΥΝΑΡΤΗΣΕΙΣ (4)

Χρησιμοποιούμε συναρτήσεις ακόμα και αν εργασίες γίνονται **μια μόνο φορά** μέσα στο πρόγραμμα, διότι κάνουν το πρόγραμμα πιο εύκολο στο **διάβασμα** και την **τροποποίηση**.

Κάθε συνάρτηση της C σε ένα πρόγραμμα είναι **ισοδύναμη** με τις άλλες.

Μία συνάρτηση, ως όνομα, **υπάρχει σε όλα τα προγράμματα της C**, η

main( )

Η μόνη ιδιαιτερότητά της main( ): από αυτήν **αρχίζει** και σε αυτήν **τελειώνει** η εκτέλεση του προγράμματος.

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (1)

```
#include  <stdio.h>
void  starbar( );

main( )
{
    int  k;
    starbar( );
    scanf ("%d", &k);
    printf ("k+1 = %d\n", k+1);
    starbar( );
}
```

Κλήση συνάρτησης που λέγεται starbar

Όνομα με παρενθέσεις (άδειες ή όχι) και ; στο τέλος: **κλήση συνάρτησης**

Ένας διαχωρισμός των συναρτήσεων: **συναρτήσεις της γλώσσας** και **συναρτήσεις του χρήστη.**

Αν έχουμε συνάρτηση της γλώσσας (π.χ. η printf εδώ) πρέπει…

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (2)

…να έχουμε κάνει include το κατάλληλο .h αρχείο

```
#include  <stdio.h>
void  starbar( );

main( )
{
    int  k;
    starbar( );
    scanf ("%d", &k);
    printf ("k+1 = %d\n", k+1);
    starbar( );
}
```

Εδώ το .h αρχείο που απαιτείται είναι το: stdio.h

Αν έχουμε συνάρτηση του χρήστη, πρέπει (σε μικρής κλίμακας προγραμματισμό, όπως εδώ) **…**

η συνάρτηση να είναι γραμμένη κάτω από την main( )

Δηλαδή:

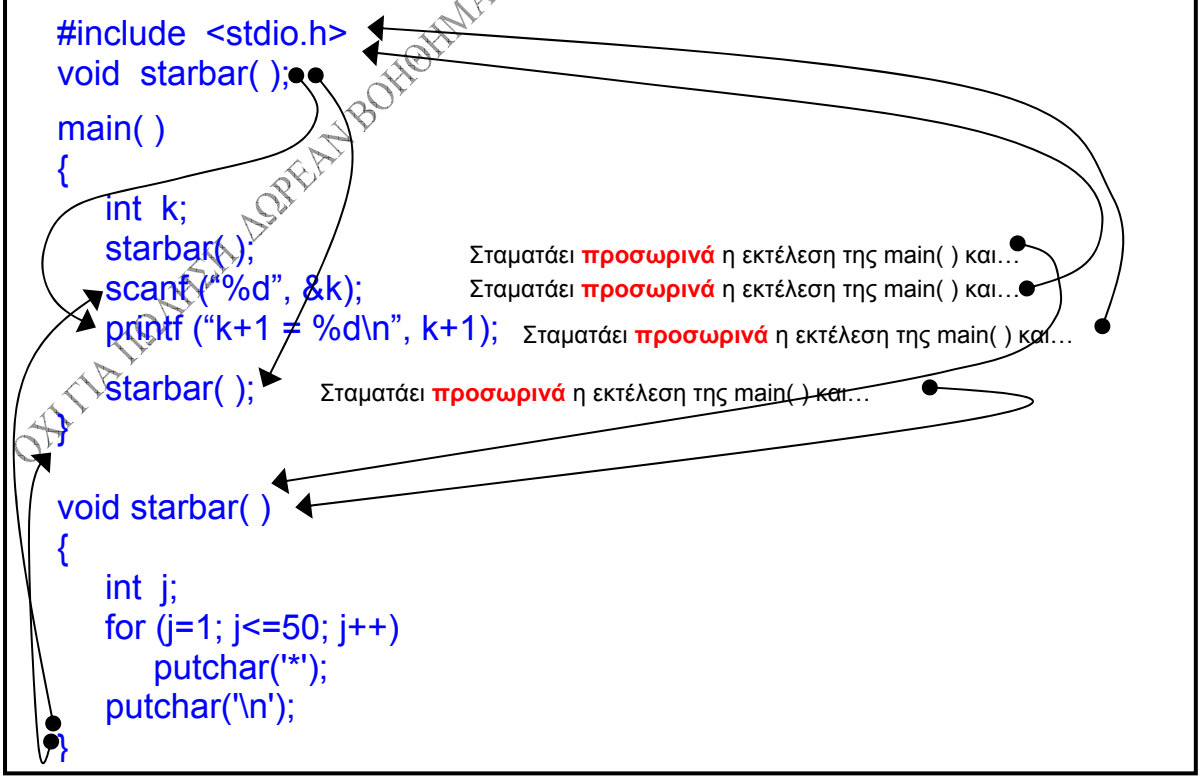
ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (3)

```
#include  <stdio.h>
void  starbar( );

main( )
{
    int  k;
    starbar( );
    scanf (“%d”, &k);
    printf (“k+1 = %d\n”, k+1);
    starbar( );
}

void starbar( )
{
    int  j;
    for (j=1; j<=50; j++)
        putchar('*');
    putchar('\n');
}
```

Οι εντολές από τις οποίες αποτελείται η συνάρτηση:

Ορισμός της συνάρτησης

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (4)



ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (5)

Θέλουμε:
Να γράψουμε 50 αστεράκια στην οθόνη και μετά να αλλάξουμε γραμμή.

Πώς;

```
for (j=1; j<=50; j++)
    putchar('*');
putchar('\n');
```

Τι λείπει;

```
int j;
for (j=1; j<=50; j++)
    putchar('*');
putchar('\n');
```

← Αυτό!

Πώς μπορούμε τα παραπάνω **να τα κάνουμε συνάρτηση**;

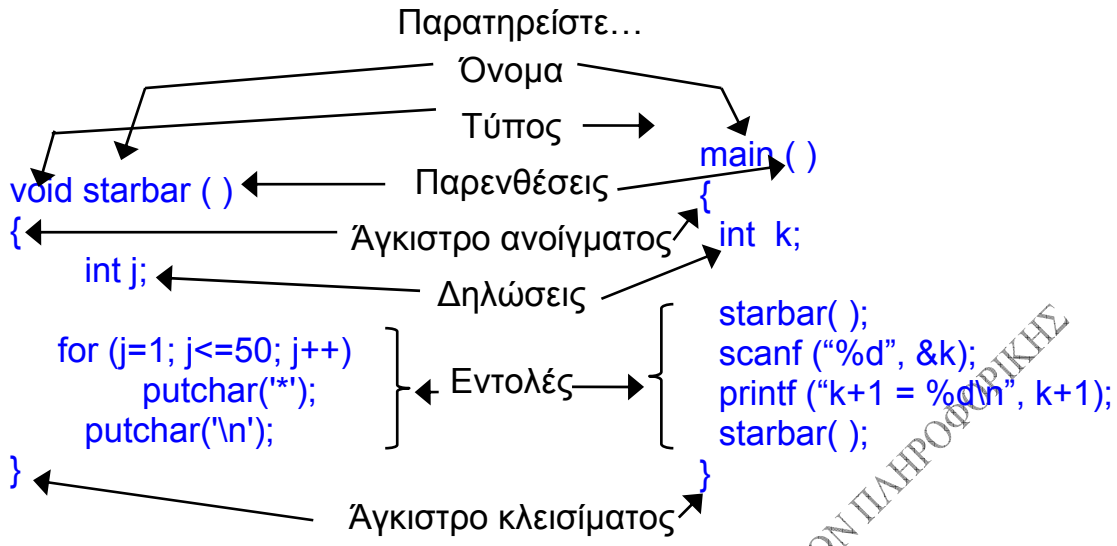
ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (6)

- Επιλέγουμε το **όνομα** που θέλουμε για την συνάρτηση. Π.χ. starbar
- Βάζουμε **ό,τι χρειάζεται** η συνάρτηση για να δουλέψει, δηλαδή τις **δηλώσεις** και τις **εντολές** της **ανάμεσα σε άγκιστρα** { }.
- Γράφουμε σαν **επικεφαλίδα το όνομα** που επιλέξαμε, με την λέξη **void μπροστά** και **παρενθέσεις μετά το όνομα**.
- Μετά τις παρενθέσεις **δεν υπάρχει** ελληνικό ερωτηματικό (;)

Δηλαδή:

```
void starbar( )
{
    int j;
    for (j=1; j<=50; j++)
        putchar('*');
    putchar('\n');
}
```

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (7)



Ορισμός της starbar **Ορισμός της main**

Άρα…

Όλες οι συναρτήσεις στην C έχουν την ίδια μορφή

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (8)

```
void starbar( )
{
    int j;
    for (j=1; j<=50; j++)
        putchar('*');
    putchar('\n');
}
```

Η μεταβλητή j που δηλώνεται μέσα στην starbar( ) λέγεται **τοπική μεταβλητή** της συνάρτησης.

ΠΡΟΣΟΧΗ!!!!

Οι τοπικές μεταβλητές είναι γνωστές ΜΟΝΟ μέσα στη συνάρτηση, στην οποία έχουν δηλωθεί

Δηλαδή…

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (9)

```
#include  <stdio.h>
void  starbar( );

main( )
{
    int  k;
    starbar( );
    scanf (“%d”, &k);
    printf (“k+1 = %d\n”, k+1);
    starbar( );
}

void starbar( )
{
    int  j;
    for (j=1; j<=50; j++)
        putchar('*');
    putchar('\n');
}
```

- Τοπική μεταβλητή της main( ). **Γνωστή ΜΟΝΟ στη main( ).** Δεν την γνωρίζει η starbar( ).
- Τοπική μεταβλητή της starbar( ). **Γνωστή ΜΟΝΟ στη starbar( ).** Δεν την γνωρίζει η main( ).

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (10)

```
#include  <stdio.h>
void  starbar( );

main( )
{
    int  k;
    starbar( );
    scanf (“%d”, &k);
    printf (“k+1 = %d\n”, k+1);
    starbar( );
}

void starbar( )
{
    int  j;
    for (j=1; j<=50; j++)
        putchar('*');
    putchar('\n');
}
```

- **Δήλωση** της συνάρτησης. Υπάρχει **; στο τέλος**
- Τοπική μεταβλητή της main( ). **Γνωστή ΜΟΝΟ στη main( ).** Δεν την γνωρίζει η starbar ( ).
- Τοπική μεταβλητή της starbar( ). **Γνωστή ΜΟΝΟ στη starbar( ).** Δεν την γνωρίζει η main( ). Η k της main( ) είναι **άλλη μεταβλητή** και το πρόγραμμα δεν τις συγχέει.

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ (11)

```
#include  <stdio.h>
void  starbar( );

main( )
{
    int  k;
    starbar( );
    scanf (“%d”, &k);
    printf (“k+1 = %d\n”, k+1);
    starbar( );
}

void starbar( )
{
    int  j;
    for (j=1; j<=50; j++)
        putchar('*');
    putchar('\n');
}
```

Αν μια συνάρτηση **χρειάζεται πληροφορίες** για να δουλέψει, αυτές **της διαβιβάζονται μέσω των παρενθέσεών της**.

Οι πληροφορίες λέγονται **ορίσματα της συνάρτησης**.

Η starbar( ) δεν χρειάζεται πληροφορίες. Καλείται με **άδειες παρενθέσεις**.

Και εδώ **οι παρενθέσεις είναι άδειες**, η συνάρτηση **δεν έχει παραμέτρους**.

Αυτή είναι η **εξαίρεση**. Οι συναρτήσεις **συνήθως χρειάζονται πληροφορίες**, άρα **οι παρενθέσεις σπάνια είναι άδειες**

ΠΙΟ ΣΥΝΘΕΤΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (1)

```
#include  <stdio.h>
#include  <string.h>
#define  NAME  "TEI KPHTHΣ"
void  starbar( );
void  space(int);
main( ) {
    Int x;
    starbar( );
    space(12);
    printf("%s\n", NAME);
    x = strlen(NAME);
    space(x);
    starbar( ); }

void starbar( ) {
    int  j;
    for (j=1; j<=LIM; j++)
        putchar('*');
    putchar('\n'); }

void space(int  num) {
    int  j;
    for (j=1; j<=num; j++)
        putchar(' ');
}
```

Ο χρήστης έχει γράψει...
δύο συναρτήσεις

Η starbar( ) δεν χρειάζεται πληροφορίες. Καλείται με **άδειες παρενθέσεις.**

Η space( ) χρειάζεται πληροφορίες. Αυτές **της στέλνονται κατά την κλήση της μέσω των παρενθέσεων** (ο αριθμός 12 εδώ)

ΠΙΟ ΣΥΝΘΕΤΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (2)

Θέλουμε τις εντολές για να **γράφουμε 10 κενά** στην οθόνη και μετά να αλλάζουμε γραμμή:

```
int j;

for (j=1; j<=10; j++)
        putchar(' ');
putchar('\n ');
```

Να γράφουμε **όχι 10, αλλά 35 κενά**:

```
for (j=1; j<=35; j++)
        putchar(' ');
```

Να γράφουμε **όχι 35, αλλά 28 κενά**:

```
for (j=1; j<=28; j++)
        putchar(' ');
```

ΠΙΟ ΣΥΝΘΕΤΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (3)

Θέλουμε να γράφουμε **όχι 28, αλλά num κενά**, όπου num ένας ακέραιος:

```
int j;

for (j=1; j<=num; j++)
        putchar(' ');
putchar('\n ');
```

Οι εντολές αυτές **να γίνουν συνάρτηση.**

ΠΡΟΣΟΧΗ!! Θέλουμε **το num να παίρνει τιμή τη στιγμή που καλείται η συνάρτηση από την main( ):**

```
void space (int  num)
{
    int  j;
    for (j=1; j<=num; j++)
        putchar(' ');
}
```

Παράμετρος της συνάρτησης.

ΘΥΜΗΘΕΙΤΕ:

Η συνάρτηση είναι φτιαγμένη να γράφει num κενά στην οθόνη, όσο κι αν είναι το num.

ΠΙΟ ΣΥΝΘΕΤΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (4)

```
#include  <stdio.h>
#include  <string.h>
#define  NAME  "TEI KPHTHΣ"
void  starbar( );
void  space( int);
main(  ) {
    int x;
    starbar( );
    space(12);
    printf("%s\n", NAME);
    x = strlen(NAME);
    space(x);
    starbar( ); }

void starbar( ) {
     int  j;
     for (j=1; j<=LIM; j++)
          putchar('*');
     putchar('\n'); }

void space(int  num) {
     int  j;
     for (j=1; j<=num; j++)
          putchar(' ');
}
```

Σταματάει προσωρινά η main( ), πάμε στην starbar( ), γράφει 50 αστεράκια και επιστρέφουμε.

Σταματάει προσωρινά η main( ), πάμε στην space( )

Πριν αρχίσει να δουλεύει η space( )....

...η τιμή 12 περνάει στο num (το num γίνεται ίσο με 12)

Η space( ) είναι φτιαγμένη να γράφει num κενά, το num έγινε ίσο με 12, άρα γράφει 12κενά στην οθόνη.

Επιστρέφουμε στην printf( )

ΠΙΟ ΣΥΝΘΕΤΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (5)

```
#include  <stdio.h>
#include  <string.h>
#define  NAME  "TEI KPHTHΣ"
void  starbar( );
void  space(int);
main(  ) {
    int x;
    starbar( );
    space(12);
    printf("%s\n", NAME);
    x = strlen(NAME);

    space(x);
    starbar( ); }
void starbar( ) {
     int  j;
     for (j=1; j<=LIM; j++)
          putchar('*');
     putchar('\n'); }
void space(int  num) {
     int  j;
     for (j=1; j<=num; j++)
          putchar(' ');
}
```

Συνέχεια στις επόμενες εντολές

Σταματάει προσωρινά η main( ), πάμε στην printf( ), γράφει ΤΕΙ ΚΡΗΤΗΣ. Επιστρέφουμε στην...

x = strlen (NAME);

Σταματάει προσωρινά η main( ), πάμε στην strlen( ), η οποία γίνεται ίση με 10.

Προσέξτε πού επιστρέφει... στο ίσον

Άρα το x γίνεται ίσο με 10

Πάμε στην space(). Πριν αρχίσει να δουλεύει η space( ), η τιμή 10 περνάει στο num

Η space( ) γράφει num κενά, το num έγινε 10, άρα γράφει 10 κενά στην οθόνη. Επιστρέφουμε στη main( )

Πάμε στην starbar().

Επιστρέφουμε στην main( ). Τέλος προγράμματος.

ΤΙΜΗ ΕΠΙΣΤΡΟΦΗΣ ΣΥΝΑΡΤΗΣΗΣ (1)

Η παρακάτω συνάρτηση δέχεται ως παραμέτρους δύο ακεραίους. Υπολογίζει τον μέγιστο από τους δύο ακεραίους:

```
int  max (int x, int y)
{
    int  meg;

    if (x>y)
        meg = x;
    else
        meg = y;
    return meg;
}
```

Μία **συνάρτηση τερματίζει** τη δουλειά της με ένα από τους εξής δύο τρόπους:

- **Τέλειωσαν οι εντολές της**
- Συνάντησε την **εντολή return**

Μετά το return πρέπει να υπάρχει μια τιμή (int, float, char, double κλπ)

Η συνάρτηση, όταν συναντήσει το return τερματίζει την δουλειά της, επιστρέφει εκεί απ' όπου κλήθηκε και **γίνεται ολόκληρη ίση με την τιμή που υπάρχει μετά το return**.
Στο παράδειγμά μας η συνάρτηση γίνεται ίση με int, πράγμα που δείχνεται **εδώ**!!!

Η τιμή αυτή λέγεται **ΤΙΜΗ ΕΠΙΣΤΡΟΦΗΣ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ**

ΤΙΜΗ ΕΠΙΣΤΡΟΦΗΣ ΣΥΝΑΡΤΗΣΗΣ (2)

Αν μια συνάρτηση τερματιστεί επειδή τέλειωσαν οι εντολές της (δηλαδή **δεν έχει return**) η συνάρτηση έχει τιμή επιστροφής…

ΤΙΠΟΤΑ

Αυτό το «τίποτα» λέγεται…

void

```
void starbar( )
{
    int k;
    for (k=1; k<=50; k++)
        putchar('*');
    putchar('\n');
}
```

Μια συνάρτηση **void δεν σημαίνει ότι δεν κάνει τίποτα**. Σημαίνει ότι **δεν επιστρέφει τίποτα** εκεί απ' όπου κλήθηκε, ή αλλιώς, **τελειώνοντας δεν γίνεται ίση με κάτι**

Δήλωση της starbar: `void starbar( );`
Δήλωση της max : `int max (int, int);`

ΤΙΜΗ ΕΠΙΣΤΡΟΦΗΣ ΣΥΝΑΡΤΗΣΗΣ (3)

Προσοχή στα παρακάτω:

1. Ο **τερματισμός** μιάς συνάρτησης **με το return** γίνεται **ακόμη και εάν το return δεν είναι η τελευταία εντολή** της συνάρτησης.
   Τι έχετε να πείτε για την printf της παρακάτω συνάρτησης;

```
int  max (int x, int y)
{
    int  meg;

    if (x>y)
        meg = x;
    else
        meg = y;
    return meg;
    printf ("%d", meg);
}
```

Δεν είναι συντακτικό λάθος, αλλά…
Δεν θα εκτελεστεί ποτέ!!

ΤΙΜΗ ΕΠΙΣΤΡΟΦΗΣ ΣΥΝΑΡΤΗΣΗΣ (4)

Προσοχή στα παρακάτω:

2. Ούτε το παρακάτω είναι συντακτικό λάθος, αλλά…

```
int  min_max (int x, int y)
{
    int  meg, mik;

    if (x>y) {
        meg = x; mik = y;}
    else {
        meg = y; mik = x;}
    return meg;
    return mik;
}
```

…το δεύτερο return
δεν θα εκτελεστεί ποτέ!!

3. Αυτό **είναι συντακτικό λάθος**:

```
int  min_max (int x, int y)
{
    int  meg, mik;

    if (x>y) {
        meg = x; mik = y;}
    else {
        meg = y; mik = x;}
    return meg, mik;
}
```

Η συνάρτηση **μπορεί να επιστρέψει μόνο μία τιμή**

ΤΙΜΗ ΕΠΙΣΤΡΟΦΗΣ ΣΥΝΑΡΤΗΣΗΣ (5)

Προσοχή στα παρακάτω:

4. Μια συνάρτηση πρέπει **να έχει σε κάθε περίπτωση την ίδια τιμή επιστροφής**. Το παρακάτω θα δημιουργήσει προβλήματα:

```
int  max (int x, int y)
{
    if (x>y)
        return x;
    else
        printf ("ΛΑΘΟΣ");
}
```

Εδώ έχουμε τιμή επιστροφής, ενώ…

εδώ όχι

Θυμηθείτε συναρτήσεις που ξέρουμε:

getchar( )	με τιμή επιστροφής	**char**
getche( )	-//-	**char**
strlen( )	-//-	**int**

ΠΑΛΙΕΣ ΔΙΑΦΑΝΕΙΕΣ (ενδεικτικές)

Αναδρομικότητα συναρτήσεων:

Εννοούμε την **κλήση** μιας συνάρτησης **από τον εαυτό της** (recursion). Π.χ.:

```
#include <stdio.h>

void up_and_down(int);

main( )
{
   up_and_down(1);
}

void up_and_down(int n)
{
   printf ("Στάδιο %d\n", n);
   if (n < 4)
      up_and_down(n+1);
   printf ("Στάδιο %d\n", n);
}
```

Στην οθόνη θα εμφανισθεί:

```
Στάδιο 1
Στάδιο 2
Στάδιο 3
Στάδιο 4
Στάδιο 4
Στάδιο 3
Στάδιο 2
Στάδιο 1
```

Κάθε **επίπεδο κλήσης** της συνάρτησης έχει τις **δικές του μεταβλητές**.

Τι θα δώσει στην οθόνη το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int summing(int);

main()
{
   int k=0;

   k = k + summing(1);
   printf("%d\n", k);
}
```

```
int summing(int n)
{
      int j=0;

      if (n++ < 4)
          j = summing(n)+ n;
      return j;
}
```

ΔΙΕΥΘΥΝΣΕΙΣ. Ο ΤΕΛΕΣΤΗΣ &

Ο τελεστής **&** μας δίνει τη **διεύθυνση**, στην οποία αποθηκεύεται μια μεταβλητή.

Εάν π.χ.:

```
int  ak = 10;
```

και ο ak είναι αποθηκευμένος στη θέση μνήμης 25439, τότε, η εντολή:

```
printf ("%d  %p\n", ak, &ak);
```

βγάζει στην οθόνη:

```
10   25439
```

ΔΕΙΚΤΕΣ

Ξέρουμε ότι είναι δυνατό να περάσουμε πολλές τιμές σε μια συνάρτηση. Όμως **επιστρέφεται μια μόνο τιμή** από τη συνάρτηση. Για να επιστρέψουμε περισσότερες τιμές χρησιμοποιούμε τους **δείκτες**.

Ένα απλό πρόβλημα:

Θέλουμε να εναλλάξουμε τις τιμές δύο μεταβλητών, των x και y.

Γίνεται έτσι;

```
x = y;
y = x;
```

Δεν λειτουργεί.

Το παρακάτω δουλεύει;

```
temp = x;
x = y;
y = temp;
```

Ναι.

Μπορούμε να βάλουμε αυτές τις εντολές σε μια συνάρτηση; Π.χ.:

```
#include <stdio.h>
void interchange(int, int);
main( )
{
    int x=5; y=10;

    printf("x = %d, y=%d\n", x, y);
    interchange(x, y);
    printf("x = %d, y=%d\n", x, y);
}

void interchange(int u, int v)
{
    int temp;

    temp = u;
    u = v;
    v = temp;
}
```


Διαπιστώσεις:

- Κάτι **δεν πάει καλά**.
- Η interchange( ) λειτουργεί σωστά, όμως οι τιμές των μεταβλητών **δεν επιστρέφουν** στη main( ).
- Θα μπορούσαμε να πούμε;

```
        return u;                       (στην interchange)
και     x = interchange(x, y);  (στη main)
```

Ναι, αλλά έτσι **επιστρέφεται** στη main( ) **μια μόνο τιμή**. Η άλλη τι γίνεται;

Πέρασμα τιμών σε συναρτήσεις:

Πριν δούμε τη λύση του προβλήματος, ας εξετάσουμε τι συμβαίνει όταν περνάμε τιμές σε μια συνάρτηση. Π.χ.:

```
main()
{
   int x=4; y=7;

   pass(x,y);
}

void pass(xx, yy)
int xx, yy;
{
   printf ("Πρώτη τιμή %d, δεύτερη %d", xx, yy);
}
```

Δεσμεύεται χώρος στη μνήμη για τις xx και yy. Η συνάρτηση μπορεί να δουλέψει με τις xx και yy, **χωρίς να επηρεάσει τις x και y.**

Περνώντας διευθύνσεις σε συνάρτηση:

```
#include <stdio.h>

void rets(int *, int *);

main()
{
   int x=5; y=9;

   rets(&x, &y);
   printf ("Πρώτη τιμή %d, δεύτερη %d", x, y);
}

void rets(int *px, int *py)
{
   *px = 3;
   *py = 18;
}
```

Για τις διευθύνσεις δεν υπάρχει άλλος τύπος. Χρησιμοποιούμε:

int *px, *py;

Οι αστερίσκοι λένε ότι οι μεταβλητές αυτές περιέχουν **διευθύνσεις** και ότι **δείχνουν σε μεταβλητές τύπου ακεραίου.**

Δηλαδή

```
main( )
{
  int x, y;

  rets(&x, &y);
……….
}
```

```
void rets(int *px, int *py)
{
   ……..
}
```

Πώς όμως θα **προσπελάσω τα περιεχόμενα των px και py;**

Ξανά με το *, το οποίο όμως χρησιμοποιείται τώρα **με διαφορετικό τρόπο από ό,τι στις δηλώσεις δείκτη:**

*px = 3;

Δηλαδή:

- **Τα περιεχόμενα του px να γίνουν ίσα με 3**

ή αλλιώς:

- **Ανάθεσε στη μεταβλητή που δείχνεται από το px την τιμή 3**

Ξαναγυρνάμε στο πρόγραμμα ανταλλαγής τιμών:

Για να δούμε πώς θα επιλυθεί με τους δείκτες:

```
#include <stdio.h>

void interchange(int *, int *);
```

```
main( )
{
    int x=5; y=10;

    printf("x = %d, y=%d\n", x, y);
    interchange(&x, &y);
    printf("x = %d, y=%d\n", x, y);
}

void interchange(int *u, int *v)
{
    int temp;

    temp = *u;
    *u = *v;
    *v = temp;
}
```

Παρατηρείστε ότι:

- Στην interchange( ) μεταδώσαμε διευθύνσεις.
- Τα u και v έχουν τιμές διευθύνσεις, άρα πρέπει να δηλωθούν σαν δείκτες.
- x, y ακέραιοι, άρα: u, v δείκτες σε ακεραίους.
- temp = *u; Θυμόμαστε ότι το u έχει τιμή &x, άρα το temp παίρνει τιμή x.

Συνοψίζοντας:

Περάσαμε στη συνάρτηση τις **διευθύνσεις** των μεταβλητών. Με τη χρήση του * η συνάρτηση έχει **έλεγχο** αυτών των μεταβλητών.

Ανακεφαλαίωση στους δείκτες:

Αν η pok είναι μια ακέραια μεταβλητή, και ο ptr είναι δείκτης σε ακέραιο, τότε τα παρακάτω είναι σωστά:

- Η &pok είναι ένας **δείκτης στην pok**.
- Μπορώ να πώ:

```
ptr = &pok;
```

Τώρα ο ptr **δείχνει** στην pok.

- Έστω: ptr = &bak;

  Τότε το:

  val = *ptr;

  σημαίνει **τα περιεχόμενα της διεύθυνσης, στην οποία δείχνει ο ptr**. Δηλαδή:

  val = bak;

  Παράδειγμα:

```
nurse = 22;
ptr = &nurse;
val = *ptr;
```

  Η val εδώ έχει τιμή 22.

Το ptr στο πιο πάνω είναι μια **μεταβλητή**. Πώς δηλώνουμε μια τέτοια μεταβλητή;

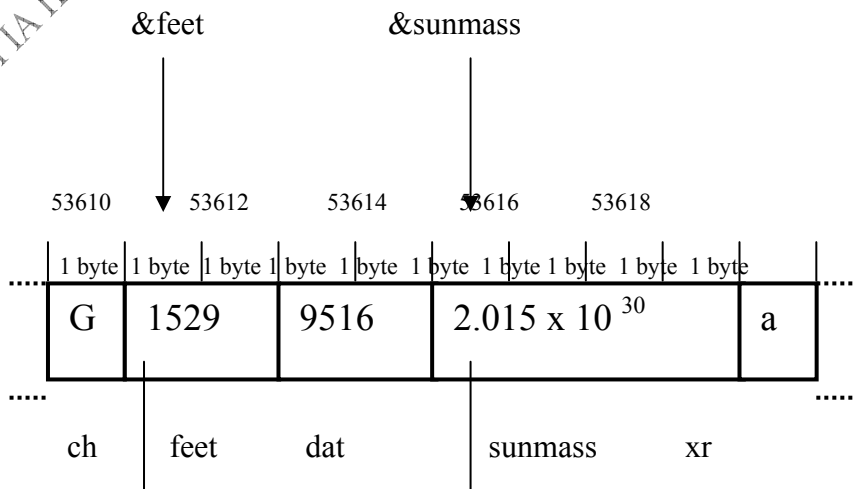
Κάτι τέτοιο:

pointer ptr;

δεν είναι αρκετό. Χρειάζεται να πούμε και **τι είδος (τύπος) είναι η μεταβλητή στην οποία δείχνει ο δείκτης**. Άρα:

```
int  *pi;
char *pc;
float *pf, *pg;
```

Σχηματικά:



```
   int  *pfeet              float  *psun;
pfeet = &feet;          psun = &sunmass;
*pfeet                  *psun
```

Κλήσεις συναρτήσεων:

α)

```
function1(x);
```

σημαίνει και :

```
function1(int num);
```

β)

```
function2(&x);
```

σημαίνει και :

```
function2(int *ptr);
```


ΚΑΤΗΓΟΡΙΕΣ ΜΝΗΜΗΣ

Οι κατηγορίες μνήμης επιτρέπουν να καθορίζουμε ποιες συναρτήσεις **γνωρίζουν ποιες μεταβλητές** και **πόσο θα υπάρχει μια μεταβλητή** σε ένα πρόγραμμα.

α) Αυτόματες μεταβλητές:

- Είναι εξ ορισμού οι μεταβλητές που δηλώνονται **μέσα σε μια συνάρτηση**.

- Έχουν **τοπική εμβέλεια**, είναι δηλαδή γνωστές **μόνο μέσα στη συνάρτηση**, μέσα στην οποία ορίζονται.

- Άλλες συναρτήσεις μπορούν να χρησιμοποιούν μεταβλητές με το ίδιο όνομα, οι οποίες όμως είναι **ανεξάρτητες** και αποθηκεύονται σε **διαφορετικές θέσεις** μνήμης.

- **Αρχίζουν να υπάρχουν** μόλις κληθεί η συνάρτηση που τις περιέχει. Μόλις η συνάρτηση ολοκληρώσει την εργασία της και επιστρέψει τον έλεγχο, **χάνονται**.

- Η **απόδοση αρχικών τιμών** σε αυτές γίνεται μόνο εάν εμείς την κάνουμε. Δηλαδή:

```
main()
{
   int  ak;
   int  pr=7;
```

Το pr έχει τιμή 7. Το ak έχει τυχούσα τιμή.

β) Εξωτερικές μεταβλητές:

- Ορίζονται **έξω** από μια συνάρτηση.

- Στα παραδείγματα που ακολουθούν φαίνονται διάφοροι συνδυασμοί δηλώσεων μεταβλητών:

1)

```
int  hocus;
int  magic();
main()
{
   int  ak;
   ……….
}

int magic()
{
   int  ms;
   ……….
}
```

Εδώ, η εξωτερική μεταβλητή hocus είναι γνωστή και στη main() και στη magic().

2)

```
int  hocus;
int  magic();
main()
```

```
{
   int  hocus;
   ……….
}

int magic()
{
   int  ak;
   ……….
}
```

Εδώ, η εξωτερική hocus δεν είναι γνωστή στη main(), αλλά είναι γνωστή στη magic( ) και σε οποιαδήποτε άλλη συνάρτηση που δεν έχει δικιά της hocus.

- Οι **εξωτερικές μεταβλητές** υπάρχουν **όσο υπάρχει το πρόγραμμα**.

- Η **απόδοση αρχικών τιμών** σε αυτές γίνεται μόνο μια φορά, **όταν ορίζονται**. Αν δεν τους αποδώσουμε αρχική τιμή, παίρνουν τιμή **μηδέν**.

γ) Στατικές μεταβλητές:

- Έχουν την ίδια εμβέλεια με τις αυτόματες, αλλά **δεν χάνονται** μόλις τελειώσει το έργο της συνάρτησης που τις περιέχει. Ο Η/Υ **θυμάται τις τιμές τους** από τη μια κλήση της συνάρτησης μέχρι την επόμενη.

- Στο παράδειγμα που ακολουθεί, η μεταβλητή fade είναι τύπου int, ενώ η stay είναι static int. Τι θα εμφανιστεί στην οθόνη;

```
#include <stdio.h>
void  trystat( );
main( )
{
   int count;

   for (count=1; count<=3; count++)
        {
           printf ("Επανάληψη %d:\n", count);
           trystat();
        }
}

void trystat( )
{
   int fade=1;
```

```
    static int stay=1;

    printf ("fade = %5d. stay = %5d\n", fade++, stay++);
}
```

Η fade παίρνει αρχική τιμή κάθε φορά που καλείται η trystat( ), ενώ η stay παίρνει αρχική τιμή μια φορά, όταν μεταγλωττίζεται η trystat( ).

- Αν **δεν τους αποδώσουμε αρχική τιμή**, παίρνουν τιμή **μηδέν**.

ΠΙΝΑΚΕΣ

Θυμίζουμε πώς δηλώνονται:

```
char  pin[30];
int   mat[20];
float arr[38];
```

Πρώτο στοιχείο του pin είναι το pin[0] και **τελευταίο** το pin[29].

Κατηγορίες πινάκων:

Αντίστοιχα με τις μεταβλητές που έχουμε ήδη δεί, μιλάμε για **αυτόματους**, **εξωτερικούς**, **στατικούς** πίνακες.

Παράδειγμα:

```
#define  MAX  5

int  mat[5] = {3, 6, 2, -1, 0};

int  zer[7];

main()
{
    int pin[7] = {1, 7, 4, -5, 0, 5, 2};
```

```
    int  table[MAX] = {50, 25, 10, 5, 8};

    static char qrt[6] = {'Γ', 'Ι', 'Ο', 'Ρ', 'Τ', 'Η'};

    int  arr[ ] = {8, 7, -3, 6};

    char  sline[ ] = {'c', 'a', 't', '\0'};

    char  aline[ ] = "cat";
    ……..……
}
```

Όλα τα στοιχεία του zer είναι μηδέν.

Ο arr είναι πίνακας 4 θέσεων. Δεν χρειάζεται το 4 στις [ ].

Οι πίνακες sline[ ] και aline[ ] είναι ισοδύναμοι.

Δείκτες και πίνακες:

Το **όνομα** ενός πίνακα είναι και **δείκτης** που δείχνει **το πρώτο στοιχείο του**.

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

#define SIZE 4

main( )
{
    int  dates[SIZE], *pti, index;
    float   bills[SIZE], *ptf;

    pti = dates;
    ptf = bills;
    for (index=0; index<SIZE; index++)
    printf  ("Δείκτες  +  %d:  %10p    %10p\n",  index,  pti+index,
              ptf+index);
}
```

Θα εμφανιστεί κάτι τέτοιο:

```
Δείκτες + 0:    56014      56026
Δείκτες + 1:    56016      56030
Δείκτες + 2:    56018      56034
Δείκτες + 3:    56020      56038
```

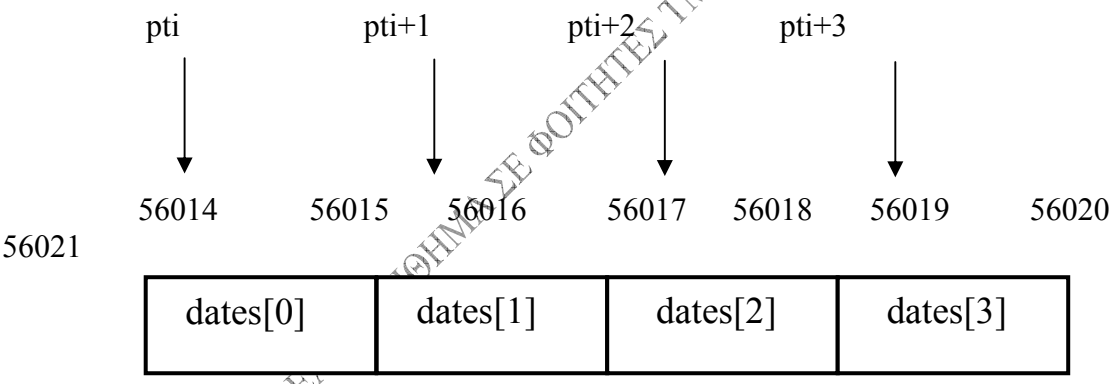
Δηλαδή:

56014 + 1 = 56016
και
56026 + 1 = 56030

Ακούγεται **λογικό**; **Ναι**, διότι:

Όταν λέμε **πρόσθεσε 1 στον δείκτη**, εννοούμε **μια μονάδα αποθήκευσης και όχι ένα byte**. Για τους int μονάδα αποθήκευσης είναι τα 2 byte και για τους float είναι τα 4. Γι αυτό πρέπει να ξέρουμε **σε τι τύπο μεταβλητής δείχνει ο δείκτης**.

Σχηματικά:



int dates[4], *pti;
pti = dates; ή ισοδύναμα pti = &dates[0];

Είναι ισοδύναμα τα παρακάτω;

dates + 2	και	&dates[2]
*(dates + 2)	και	dates[2]
*(dates + 2)	και	*dates + 2

Πίνακες σαν ορίσματα συναρτήσεων:

Η C **δεν επιτρέπει** να περνούν πίνακες σαν ορίσματα συναρτήσεων.

Αυτό που εφαρμόζεται είναι να περνούν σαν ορίσματα **δύο παράμετροι**:

- ένας **δείκτης** στο πρώτο στοιχείο του πίνακα και
- ένας **ακέραιος** που δηλώνει το μέγεθος του πίνακα.

Το παρακάτω πρόγραμμα χρησιμοποιεί τη συνάρτηση sump για να αθροίσει τα στοιχεία του πίνακα marbles[ ] (ποια είναι αλήθεια τα περιεχόμενά του;) :


```
#include <stdio.h>

#define SIZE 10

long sump(int *, int);

main()
{
    int marbles[SIZE]={20, 10, 5, 39, 4, 16};
    long answer;

    answer = sump (marbles, SIZE);
    printf ("Το άθροισμα των στοιχείων είναι %d.\n", answer);
}

long sump(int *ar, int n)
{
    int j;
    long total=0;

    for (j=0; j<n; j++)
        {
            total += *ar;
            ar++;
        }
    return total;
}
```

ΘΕΜΑΤΑ ΣΥΜΒΟΛΟΣΕΙΡΩΝ

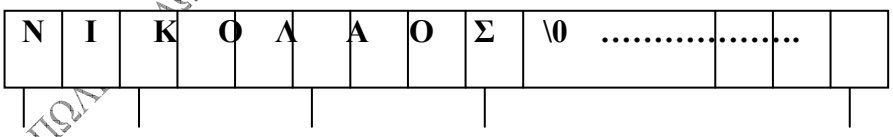
Συμβολοσειρές είναι **πίνακες τύπου char** που **τερματίζονται** με ένα μηδενικό χαρακτήρα, τον **'\0'**.

Σταθερές συμβολοσειράς χαρακτήρα:

Είναι όλοι οι χαρακτήρες που βρίσκονται ανάμεσα σε **διπλά εισαγωγικά**, μαζί με τον χαρακτήρα '\0'. Π.χ, η συμβολοσειρά

'' ΝΙΚΟΛΑΟΣ''

είναι αποθηκευμένη έτσι:



pin[0] pin[2] pin[5] pin[8] pin[29]

Αυτά τα pin[k] είναι **χαρακτήρες**.

Ολόκληρη η φράση μέσα στα εισαγωγικά δρα σαν δείκτης εκεί που αποθηκεύεται η συμβολοσειρά.

Τι θα βγάλει στην οθόνη το παρακάτω πρόγραμμα;

```
#include <stdio.h>
#define PIN "ΚΑΛΗΜΕΡΑ"
```

```
main( )
{
   char  ch, xr;

   xr = *PIN;
   printf("%c\n", xr);
   ch = * "ΚΑΛΗΜΕΡΑ";
   printf("%c\n", ch);
   ch = *("ΚΑΛΗΜΕΡΑ" + 1);
   printf("%c\n", ch);
}
```


Απόδοση αρχικών τιμών σε συμβολοσειρές:

```
char  mat[ ] = "ΜΙΑ  ΓΡΑΜΜΗ";
```

Αλλιώς:

```
char  mat[ ] = {'M','I','A',' ','Γ','P','A','M','M','H','\0'};
```

Δείτε κι αυτό:

```
char  *ptr = "ΜΙΑ  ΓΡΑΜΜΗ";
```

Οι δυο μορφές **δεν είναι ισοδύναμες**: το mat είναι **σταθερά δείκτη** (δεν αλλάζει), ενώ το ptr είναι **μεταβλητή δείκτη** (μπορεί να αλλάξει). Έτσι:

α)

```
for (j=0; j<3; j++)
     putchar (*(mat + j));
putchar ('\n');
for (j=0; j<3; j++)
     putchar (*(ptr + j));
putchar ('\n');
```

Και τα δυο for βγάζουν στην οθόνη:

ΜΙΑ

β) Το παρακάτω **επιτρέπεται**:

```
while  (*(ptr) != '\0')
```

```
putchar (*(ptr ++));
```

ενώ αυτό όχι:

```
while  (*(mat) != '\0')
    putchar (*(mat ++));
```

γ) Μπορώ να πώ:

```
ptr = mat;
```

(τι θα συμβεί τότε;)

αλλά **όχι**:

```
mat = ptr;
```

Είσοδος συμβολοσειράς:

Πρώτα **δημιουργούμε χώρο**, όπου θα αποθηκευτεί η συμβολοσειρά αφού διαβαστεί. Π.χ.

```
char  name[80];
```

Μετά, χρησιμοποιούμε για το διάβασμα την gets( ) ή την scanf( ).

α) Η συνάρτηση gets( ) :

- **Διαβάζει συμβολοσειρές** από το πληκτρολόγιο. **Τερματίζεται** όταν "δει" τον χαρακτήρα **'\n'** (που δημιουργείται με το πάτημα του Enter).
- Ο '\n' αγνοείται. Προσθέτει στο τέλος της σειράς των χαρακτήρων το '\0'.
- **Μπορεί** να διαβάζει συμβολοσειρές με κενά (η scanf( ) όχι).
- Δέχεται σαν **όρισμα δείκτη σε χαρακτήρα** (π.χ. το όνομα της συμβολοσειράς)

Τι θα κάνει το πρόγραμμα:

```
#include <stdio.h>
main( )
{
   char  name[80];

   printf ("Δώσε το όνομά σου");
   gets (name);
   printf ("Χαιρετίσματα, %s.\n", name);
}
```

- Η **τιμή επιστροφής** της gets( ) είναι η διεύθυνση της συμβολοσειράς που διάβασε (ένας δείκτης). Δηλαδή, η gets( ) μοιάζει έτσι:

```
char  *gets(char *s)
{
   ……..
   return (s);
}
```


β) Η συνάρτηση scanf( ) :

- Χρησιμοποιεί τον **προσδιοριστή %s** για το διάβασμα συμβολοσειρών.
- Διαφέρει από την gets( ) στον τρόπο που καταλαβαίνει το τέλος της συμβολοσειράς. Για τη scanf( ) η συμβολοσειρά **ξεκινά** με τον πρώτο μη λευκό χαρακτήρα και:

✧ Φθάνει στον επόμενο λευκό χαρακτήρα (χρήση του %s)

ή

✧ Λαμβάνει υπ' όψιν εύρος πεδίου ή τον επόμενο λευκό χαρακτήρα (π.χ. %10s).

Δείτε το παρακάτω πρόγραμμα:

```
#include<stdio.h>
main( )
{
    char  name1[10], name2[10];
    int  count;

    printf ("Δώστε 2 ονόματα");
    count = scanf ("%5s %10s", name1, name2);
    printf ("Διάβασα τα %d ονόματα %s και %s\n", count, name1,
            name2);
}
```

Τι θα βγεί στην οθόνη με τους παρακάτω συνδυασμούς ονομάτων:

α)

ΝΙΚΟΣ ΜΑΡΙΑ

Διάβασα τα 2 ονόματα ΝΙΚΟΣ και ΜΑΡΙΑ

β)

ΚΙΚΗ ΠΑΠΑΔΟΠΟΥΛΟΥ

Διάβασα τα 2 ονόματα ΚΙΚΗ και ΠΑΠΑΔΟΠΟΥΛ

γ)

ΜΑΝΩΛΗΣ ΚΑΝΑΚΗΣ

Διάβασα τα 2 ονόματα ΜΑΝΩΛ και ΗΣ

Έξοδος συμβολοσειράς:

α) Η συνάρτηση puts( ) :

- Δέχεται ένα **όρισμα** που είναι **δείκτης σε χαρακτήρα**.
- Εμφανίζει τη συμβολοσειρά αυτή στην οθόνη.
- Η puts( ) σταματά να παίρνει χαρακτήρες όταν βρεί το '\0'. Τότε κάνει την **αντικατάσταση του '\0' με '\n'** και στέλνει το αποτέλεσμα στην οθόνη, άρα η puts( ) συνεπάγεται και **αλλαγή γραμμής**.

Π.χ.:

Τι θα δώσει στην οθόνη το πιο κάτω πρόγραμμα;

```
#include <stdio.h>
#define SYM "ΚΑΛΗΜΕΡΑ"
main( )
{
   char  str[ ]= "ΤΙ ΚΑΝΕΙΣ ΣΗΜΕΡΑ;";
   char  *pro ="ΕΙΜΑΙ ΠΟΛΥ ΚΑΛΑ";

   puts ("ΔΟΚΙΜΗ ΕΞΟΔΟΥ");
   puts (SYM);
   puts (str);
   puts (pro);
   puts (&str[4]);
   puts (pro+3);
}
```


β) Η συνάρτηση printf( ) :

- Χρήση του προσδιοριστή **%s**.

- Ιδιαίτερα χρήσιμη, αν θέλουμε να συνδυάσουμε σε μια γραμμή πολλές συμβολοσειρές.

Άλλες συναρτήσεις συμβολοσειράς:

α) Η συνάρτηση strlen( ):

Βρίσκει το **μήκος** μιας συμβολοσειράς (χωρίς να συμπεριλαμβάνεται το '\0').

β) Η συνάρτηση strcpy( ) :

- Δέχεται **δύο δείκτες** σε χαρακτήρα ως **ορίσματα**.
- **Αντιγράφει** τα περιεχόμενα της δεύτερης συμβολοσειράς στην πρώτη.
- Πρέπει να έχουμε μεριμνήσει ώστε ο πίνακας προορισμού να έχει αρκετό χώρο για την εισερχόμενη συμβολοσειρά.
- Είναι **τύπου *char**. Επιστρέφει την τιμή του πρώτου ορίσματος.

Π.χ.

```
#include <stdio.h>
#include <string.h>
#define SYM "ΑΔΥΝΑΤΑ"
#define SIZE 40
main( )
{
   static char  *orig = SYM;
   static char  copy[SIZE]= "ΒΑΛΕ ΤΑ ΔΥΝΑΤΑ ΣΟΥ";
   char  *ps;

   puts (orig);
   puts (copy);
   ps = strcpy (copy + 5, orig);
   puts (copy);
```

```
    puts (ps);
}
```

Τι θα εμφανίσει στην οθόνη;

```
ΑΔΥΝΑΤΑ
ΒΑΛΕ ΤΑ ΔΥΝΑΤΑ ΣΟΥ
ΒΑΛΕ ΑΔΥΝΑΤΑ
ΑΔΥΝΑΤΑ
```

γ) Η συνάρτηση strcat( ) :

- Δέχεται **δύο** συμβολοσειρές (δείκτες σε χαρακτήρα) σαν **ορίσματα**.
- Ένα αντίγραφο της **δεύτερης** τοποθετείται **στο τέλος της πρώτης** και οι δύο μαζί παίρνουν **τη θέση της πρώτης**.
- Η **δεύτερη** συμβολοσειρά **δεν αλλάζει**.
- Πρέπει να έχουμε μεριμνήσει ώστε ο πρώτος πίνακας να έχει αρκετό χώρο για να χωρέσει και ο δεύτερος.
- Είναι **τύπου *char**. Επιστρέφει την τιμή του πρώτου ορίσματος.

```
……..
char f[20], s[10];
……..
strcpy(f, "ΓΕΙΑ");
strcpy(s, " ΧΑΡΑ");
strcat (f, s);
puts(f);
```

Το f θα έχει τιμή:

ΓΕΙΑ ΧΑΡΑ

δ) Η συνάρτηση strcmp( ) :

- Δέχεται **δύο** συμβολοσειρές (δείκτες σε χαρακτήρα) σαν **ορίσματα**.

- **Επιστρέφει**:
  - ✧ **0** εάν οι συμβολοσειρές είναι **ίδιες.**
  - ✧ **Αρνητικό αριθμό** εάν η πρώτη συμβολοσειρά προηγείται αλφαβητικά της δεύτερης.
  - ✧ **Θετικό αριθμό** εάν η δεύτερη συμβολοσειρά προηγείται αλφαβητικά της πρώτης.

Μετατροπές συμβολοσειράς σε αριθμό:

Οι αριθμοί μπορούν να αποθηκευτούν είτε σε αριθμητική μορφή, είτε ως συμβολοσειρές. Π.χ., ο αριθμός 356 μπορεί να αποθηκευτεί σε ένα πίνακα χαρακτήρα, σαν ψηφία:

'3', '5', '6', '\0'

Η συνάρτηση **atoi( )** δέχεται ως **όρισμα** μια **συμβολοσειρά** (δείκτη σε χαρακτήρα) και επιστρέφει την αντίστοιχη **ακέραια τιμή**. Π.χ.:

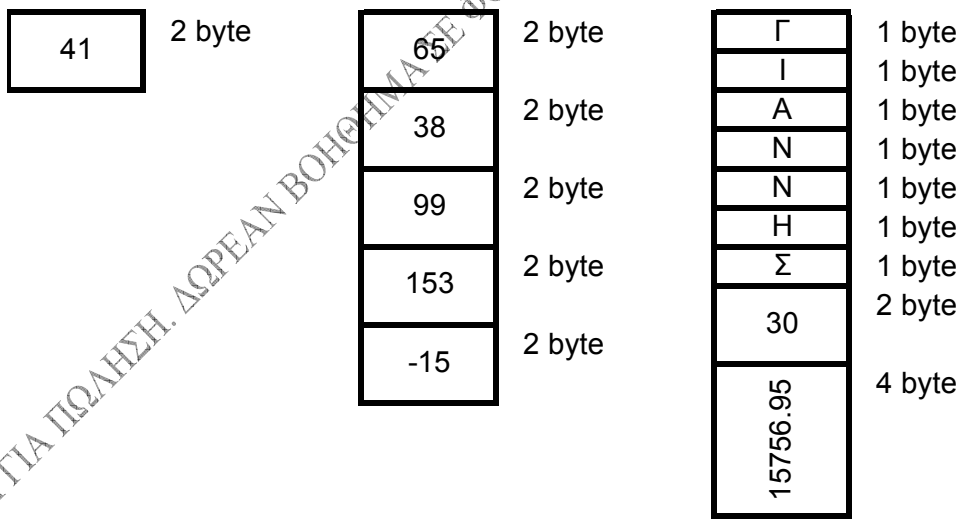
```
#include <stdio.h>
#include <stdlib.h>
main( )
{
      static char copy[5];
      int  ak;

      gets(copy);
      ak = atoi(copy);
      if (ak < 1000)
          printf("MIKROTEROS TOY 1000");
      else
           printf("MEGALYTEROS TOY 1000");
}
```


Άλλες αντίστοιχες συναρτήσεις είναι η **atof( )** που μετατρέπει μια συμβολοσειρά σε **double** και η **atol( )** που μετατρέπει μια συμβολοσειρά σε **long**.

ΔΟΜΕΣ

Συχνά θέλουμε να λειτουργούμε πάνω σε **πληροφορίες διαφορετικού τύπου δεδομένων** η κάθε μια, **τις οποίες να χειριζόμαστε σαν ομάδα**.

Δεν είναι κατάλληλες οι μεμονωμένες μεταβλητές και οι πίνακες.



Για να λύσουμε τέτοια προβλήματα την **δομή (structure)**.
Μια δομή αποτελείται από ένα πλήθος στοιχείων δεδομένων, τα οποία δεν χρειάζεται να είναι του ίδιου τύπου.

Πολύ σημαντική χρήση των δομών: η δημιουργία **νέων μορφών δεδομένων**, όπως λίστες, δέντρα, σωροί.

Δομές **συνδεδεμένες μεταξύ τους**.

ΜΙΑ ΑΠΛΗ ΔΟΜΗ

```
main( )
{
    struct easy
    {
        int num;
        char ch;
    };

    struct easy dom;

    dom.num = 5;
    dom.ch  = 'A';
    printf("dom.num=%d, dom.ch=%c", dom.num, dom.ch);
}
```


Σχόλια:

1. **Πρέπει να πούμε στον compiler πώς είναι αυτή η δομή πριν χρησιμοποιηθούν μεταβλητές αυτού του τύπου**.

2. Η δήλωση:

```
struct easy
{
```

```
    int num;
    char ch;
};
```

δηλώνει ένα **νέο τύπο δεδομένων**, ο οποίος είναι **δομή** και λέγεται easy.

3. Κάθε μεταβλητή του τύπου easy θα αποτελείται από μια ακέραια μεταβλητή και μια μεταβλητή χαρακτήρα.

4. **Το παραπάνω** δεν αναφέρεται σε συγκεκριμένες μεταβλητές, άρα **δεν δεσμεύει χώρο στη μνήμη**. **Λέει απλώς ποια είναι η μορφή των δεδομένων του είδους «δομή easy»**.

5. Δέσμευση γίνεται όταν αναφέρουμε και το όνομα της μεταβλητής.

6. Αφού δηλώσουμε τον τύπο της δομής, μπορούμε να δηλώσουμε μια ή περισσότερες μεταβλητές αυτού του τύπου. Στο:

```
struct easy dom;
```

δηλώνεται μια μεταβλητή, η dom, που είναι του τύπου struct easy. Εδώ **δεσμεύεται χώρος στη μνήμη**. Δεσμεύεται χώρος, **3 byte**.

7. Για να αναφερθώ στα τμήματα της δομής dom:

```
dom.num = 8;
dom.ch = 'K';
```

Ο τελεστής τελείας (.) ενώνει το όνομα μεταβλητής δομής με ένα μέλος της δομής.

8. Μπορεί να υπάρχει οποιοδήποτε πλήθος μεταβλητών ενός δεδομένου τύπου δομής.

```
struct easy dom;
struct easy str;

……

dom.num = 6;
dom.ch = 'F';
str.num = 32;
str.ch = 'P';
```

9. Μπορούμε να συνδυάσουμε σε μια εντολή τη δήλωση του τύπου δομής και των μεταβλητών δομής:

```
struct easy
{
    int num;
    char ch;
} dom, str;
```

ΔΕΔΟΜΕΝΑ ΣΤΙΣ ΔΟΜΕΣ

```
void main( )
{
   struct person
   {
      char name[30];
      int code;
   };
   struct person persa, persb;
   char strcode[10];

   gets (persa.name);
   gets (strcode);
   persa.code = atoi (strcode);
   gets (persb.name);
   gets (strcode);
   persb.code = atoi (strcode);
   puts (“ΠΕΡΙΕΧΟΜΕΝΑ ΔΟΜΩΝ”);
   printf (“%s\n”, persa.name);
   printf (“%05d\n”, persa.code);
   printf (“%s\n”, persb.name);
   printf (“%05d\n”, persb.code);
}
```


Αρχικές τιμές.

```
struct person persa = {“ΓΙΑΝΝΗΣ“, 15};
```

```
struct person persb = {“ΝΙΚΟΣ“, 10};
```

Απόδοση τιμής.

Ισχύει (όχι στις αρχικές εκδόσεις της C):

```
persa = persb;
```

Δηλαδή:

Μπορούμε **να αναθέσουμε με μια εντολή τα στοιχεία μιας δομής σε μια άλλη**, ακόμη κι αν πρόκειται π.χ. για μεγάλους πίνακες.

ΦΩΛΙΑΣΜΕΝΕΣ ΔΟΜΕΣ.

Δομές που περιέχουν άλλες δομές.

```
struct person
{
    char name[30];
    int code;
};
struct couple
{
    struct person chief;
    struct person memb;
} first, second;
```


Δομή (εγγραφή) couple:

Περιέχει δύο στοιχεία, **καθένα από τα οποία είναι μια δομή τύπου person**.

Το πρώτο στοιχείο της first:

```
first.chief
```

Είναι μεταβλητή δομής τύπου person, της οποίας το δεύτερο στοιχείο είναι το:

```
first.chief.code
```

Μπορώ να πώ:

```
first.chief.code = 158;
```

ΔΟΜΕΣ ΚΑΙ ΣΥΝΑΡΤΗΣΕΙΣ


```
#include <stdio.h>

struct person
{
    char name[30];
    int code;
};

struct person newpers( );      /*Δήλωση της newpers( )*/
void disppers(struct person);  /*Δήλωση disppers( )*/


void main(void)
{
    struct person persa;
    struct person persb;
    persa = newpers( );
    persb = newpers( );
}

struct person newpers( );          /*Ορισμός newpers( )*/
{
    struct person temp;
    char  ch;

    gets(temp.name);
    scanf("%d", &temp.code);
    scanf("%c", &ch);
    return(temp);
}

void disppers(struct person dpers) /*Ορισμός disppers( )*/
{
    puts ("Στοιχεία προσώπου: ");
    printf ("Όνομα : %s\n", dpers.name);
```

```
        printf ("Κωδικός : %05d\n", dpers.code);
}
```

Η συνάρτηση newpers( ) διαβάζει από το πληκτρολόγιο στοιχεία μιας μεταβλητής δομής και τα καταχωρεί σε μεταβλητές της main( ).

Η newpers( ) **επιστρέφει την τιμή που διάβασε, δηλαδή ολόκληρη τη μεταβλητή δομής**. Είναι τύπου struct person.

Η temp είναι μια τοπική μεταβλητή της newpers( ).

Η συνάρτηση disppers( ) **εμφανίζει στην οθόνη** τα στοιχεία μιας δομής την οποία δέχεται σαν όρισμα.

Και οι συναρτήσεις και το κυρίως πρόγραμμα πρέπει να ξέρουν τη μορφή της δομής person, για τον λόγο δε αυτό η δήλωσή της γίνεται **εξωτερική**, τοποθετώντας την **έξω από όλες τις συναρτήσεις**, πριν από τη main( ).

Η scanf("%c", &ch); μέσα στην disppers( ) χρησιμοποιείται για την **«κατανάλωση» του Enter** μετά το διάβασμα του κωδικού.

ΠΙΝΑΚΑΣ ΔΟΜΩΝ

Πίνακας με **κάθε στοιχείο** του **δομή**. Έστω:

```
struct employee
{
    char name[30];
    int  code;
    float pos;
};

struct employee pinax[30];


gets (pinax[2].name);
```

ΔΕΙΚΤΕΣ ΚΑΙ ΔΟΜΕΣ

```
void main(void)
{
    struct simple
    {
        int   num;
        char ch;
    };
    struct simple  dok;
    struct simple  *ptr;

    ptr = &dok;
    ptr -> num = 303;
    ptr -> ch = 'Q';
}
```

Μπορούμε να **προσπελάσουμε τα στοιχεία της δομής χρησιμοποιώντας** αντί για το όνομα της δομής **τον δείκτη σε αυτή**.

ptr.num : Δεν είναι έγκυρο, διότι **ο τελεστής τελεία (.) χρειάζεται όνομα δομής στο αριστερό του μέρος**, ενώ ο ptr δεν είναι δομή, αλλά δείκτης σε δομή.

Θα ήταν σωστή η έκφραση: (*ptr).num

Τι προβλήματα τυχόν παρουσιάζει το πιο κάτω πρόγραμμα:

```
#include <stdio.h>

struct person
{
```

```
    char name[30];
    int code;
};
struct couple
{
    struct person chief;
    struct person memb;
};


void main(void)
{
    struct couple frost;
    struct couple *sand;

    frost.chief.name = "ΓΙΑΝΝΗΣ";              /* 1 */
    frost.chief.code = 1999;                   /* 2 */
    sand ->chief.name = "ΜΑΡΙΑ";               /* 3 */
    sand ->chief.code = 1958;                  /* 4 */
    frost.memb->name = "ΚΩΣΤΑΣ";               /* 5 */
    sand ->memb->name = "ΝΙΚΟΛΑΟΣ";            /* 6 */
}
```