

*Τμήμα Εφαρμοσμένης Πληροφορικής και
Πολυμέσων*

Προγραμματισμός - Γλώσσα Προγραμματισμού C

2002-2003 Εαρινό Εξάμηνο

Παναγιώτης Καλογεράκης

Πρόβλημα:

Θέλουμε να τυπώσουμε έναν πίνακα που να δείχνει τις θερμοκρασίες Φαρενάιτ από 0 έως 300 βαθμούς, ανά 20 βαθμούς, και δίπλα από κάθε θερμοκρασία Φαρενάιτ την αντίστοιχη θερμοκρασία Κελσίου.

Τύπος μετατροπής:

$$\text{Βαθμός Κελσίου} = 5 / 9 \times (\text{Βαθμός Φαρενάιτ} - 32)$$

$^{\circ}\text{F}$	$^{\circ}\text{C}$
0	-17
20	-6
40	4
60	15
80	26
100	37
120	48
140	60
160	71
180	82
200	93
220	104
240	115
260	126
280	137
300	148

Παράδειγμα μετατροπής, $40^{\circ}\text{F} = 5 / 9 \times (40 - 32) = 4,4$

Λύση του προβλήματος με πρόγραμμα C:

```
#include <stdio.h>

/* Εμφάνιση πίνακα Φαρενάιτ-Κελσίου
για θερμοκρασίες Φαρενάιτ = 0, 20,
...,300 */
main()
{
    int fahr, celsius;
    int lower, upper, step;

    lower = 0;      /* κάτω όριο του
                     πίνακα θερμοκρασιών */
    upper = 300;    /* άνω όριο */
    step = 20;      /* μέγεθος βήματος */

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

Ένα πρόγραμμα C αποτελείται από μία ή περισσότερες **συναρτήσεις (functions)**

Έχει δηλαδή την ακόλουθη μορφή:

Συνάρτηση 1

Συνάρτηση 2

....

Συνάρτηση N

Κάθε συνάρτηση

- ❖ έχει ένα **όνομα** το οποίο δίδει ελεύθερα ο προγραμματιστής.
- ❖ και περιέχει **εντολές** που προσδιορίζουν την λειτουργία της. Οι εντολές μιάς συνάρτησης περικλείονται σε αγκύλες.

Ειδική σημασία για ένα πρόγραμμα C έχει η συνάρτηση με το όνομα **main**:

- Το πρόγραμμα αρχίζει να εκτελείται από την αρχή της **main**
- Κάθε πρόγραμμα πρέπει να περιέχει κάπου μία **main**.

Η **main** συνήθως καλεί άλλες συναρτήσεις για να την βοηθήσουν να εκτελέσει το έργο της.

```
main()  
{  
    <Εντολή 1>;  
    <Εντολή 2>;  
    ...  
}
```

Σε κάθε συνάρτηση ορίζουμε μία λίστα από παραμέτρους μέσω των οποίων η συνάρτηση επικοινωνεί με το υπόλοιπο πρόγραμμα κατά την κλήση της. Οι παράμετροι αυτοί ονομάζονται **ορίσματα** της συνάρτησης.

Παρέχεται έτσι ένας μηχανισμός μεταφοράς δεδομένων μεταξύ συναρτήσεων.

Τα ορίσματα μίας συνάρτησης καθορίζονται μέσα στο ζευγάρι από παρενθέσεις που ακολουθεί το όνομα της.

Μια συνάρτηση μπορεί να μην έχει ορίσματα. Σε αυτήν την περίπτωση, δίπλα στο όνομα της, ακολουθεί κενή λίστα: () Δεν υπάρχει τίποτα μεταξύ των παρενθέσεων.

Μια συνάρτηση καλείται με το όνομα της, ακολουθούμενο από μια λίστα από τιμές μέσα σε παρενθέσεις. Οι τιμές αυτές μια προς μία μεταφέρονται στις παραμέτρους της συνάρτησης.

Η κάθε συνάρτηση, σύμφωνα με τον ορισμό της, περιμένει συγκεκριμένα ορίσματα, σε συγκεκριμένη σειρά, καθένα από τα οποία έχει προκαθορισμένο ρόλο.

Μια συνάρτηση, με την ολοκλήρωση εκτέλεσης της, έχει την δυνατότητα:

- είτε να επιτρέπει μια τιμή στην καλούσα της συνάρτηση
- είτε να μην επιστρέφει τίποτα.

Στην δεύτερη περίπτωση η συνάρτηση ορίζεται ως `void` (γράφουμε την λέξη `void` πριν από το όνομα της).

Τις συναρτήσεις είτε τις γράφουμε εμείς στο πρόγραμμα μας, είτε τις παίρνουμε έτοιμες από βιβλιοθήκες που μας διατίθενται. Με την εντολή `#include <ονομα βιβλιοθήκης>` συμπεριλαμβάνουμε στο πρόγραμμα μας μια βιβλιοθήκη. Σε ένα πρόγραμμα μας μπορούμε να συμπεριλάβουμε όσες βιβλιοθήκες θέλουμε, δίδοντας αντίστοιχες εντολές `#include`.

```
#include <stdio.h>

void main()
{
    printf("Καλημέρα");
}
```

Με τις συναρτήσεις θα δίδεται ο μηχανισμός να σπάσουμε σύνθετες, περίπλοκες εργασίες σε μικρότερες και απλούστερες, καθώς και να οικοδομήσουμε πάνω σε αυτά που έχουν φτιάξει άλλοι, ή εμείς σε άλλα μας προγράμματα, αντί να αρχίζουμε από την αρχή.

Μεταβλητές

Μηχανισμός με τον οποίο, κατά την διάρκεια εκτέλεσης ενός προγράμματος, μας δίδεται η δυνατότητα αποθήκευσης και χρήσης τιμών.

Μια μεταβλητή αντιστοιχεί σε μία θέση μνήμης, στην οποία κατά την διάρκεια εκτέλεσης του προγράμματος, μπορούμε να αποθηκεύουμε τιμές συγκεκριμένου τύπου (π.χ. ακέραιες τιμές). Τονίζουμε ότι οι μεταβλητές δεν παρέχουν μόνιμο τρόπο αποθήκευσης τιμών: με τον τερματισμό λειτουργίας του προγράμματος οι πληροφορίες που υπάρχουν στις μεταβλητές χάνονται.

Κάθε στιγμή εκτέλεσης του προγράμματος κάθε μεταβλητή έχει μία και μόνο τιμή.

Η τιμή της μεταβλητής μπορεί να μεταβληθεί κατά την διάρκεια εκτέλεσης του προγράμματος (γιαυτό λέγεται μεταβλητή), μέσω εκχώρησης νέας τιμής σε αυτήν.

Η εντολή εκχώρησης τιμής σε μεταβλητή είναι η ακόλουθη:

<Όνομα μεταβλητής> = <έκφραση>

π.χ. `num = 100; k = 10 + 2;`

Όταν εκχωρείται μια νέα τιμή σε μία μεταβλητή η παλιά τιμή χάνεται.

Κατά την διάρκεια εκτέλεσης του προγράμματος έχουμε την δυνατότητα να χρησιμοποιήσουμε την τιμή μίας μεταβλητής, με απλή αναφορά στο όνομα της.

π.χ. για να υπολογιστεί η τιμή της έκφρασης `num + 10`, το πρόγραμμα πρώτα "διαβάζει" την τιμή που είναι αποθηκευμένη στην μεταβλητή `num` και στην συνέχεια εκτελεί τους αριθμητικούς υπολογισμούς.

Στα προγράμματα μας θα πρέπει να φροντίζουμε να έχουμε δώσει τιμή σε μια μεταβλητή πριν την χρησιμοποιήσουμε για πρώτη φορά.

Βήματα χρήσης μίας μεταβλητής στην C:

A. Δήλωση της μεταβλητής

Κάθε μεταβλητή θα πρέπει να την κοινοποιήσουμε στο πρόγραμμα μας δηλώνοντας την.

ΟΛΕΣ ΟΙ ΜΕΤΑΒΛΗΤΕΣ ΠΡΕΠΕΙ ΝΑ ΔΗΛΩΝΟΝΤΑΙ ΠΡΙΝ ΧΡΗΣΙΜΟΠΟΙΗΘΟΥΝ.

Στην δήλωση μίας μεταβλητής ορίζουμε το όνομα της και τον τύπο της ως εξής:

<τύπος μεταβλητής> <όνομα μεταβλητής>;

π.χ.

```
int num;
```

Εάν θέλουμε σε μία γραμμή μπορούμε να δηλώσουμε περισσότερες από μία μεταβλητές.

π.χ.

```
int num, sum, product;
```

Μια μεταβλητή μπορεί να πάρει αρχική τιμή κατά την δήλωση της,

π.χ.

```
int num=1, sum, product=1;
```

B. Εκχώρηση τιμής σε μεταβλητή

<Όνομα μεταβλητής> = <Εκφραση>

Προσοχή: η <έκφραση> θα πρέπει να δίδει τιμή του τύπου της μεταβλητής.

Γ. Καταχώρηση σε μεταβλητή τιμής που δίδει ο χρήστης

Κατά την διάρκεια εκτέλεσης ενός προγράμματος, όταν κληθεί η συνάρτηση `scanf`, το πρόγραμμα περιμένει από την χρήστη να πληκτρολογήσει μια τιμή. Η τιμή αυτή εκχωρείται στην μεταβλητή της οποίας η διεύθυνση περνιέται ως παράμετρος στην `scanf`.

π.χ. εάν θέλουμε η ακέραια τιμή που θα πληκτρολογήσει ο χρήστης να καταχωρηθεί στην ακέραια μεταβλητή `num`, θα πρέπει να καλέσουμε την `scanf` ως εξής:

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

Το `%d` υποδηλώνει ότι περιμένουμε να διαβάσουμε ακέραια τιμή.

Προσοχή η `scanf` θέλει ως δεύτερη παράμετρο την διεύθυνση της μεταβλητής (`&`) στην οποία θέλουμε να εκχωρηθεί η τιμή.

Δ. Χρήση τιμής μεταβλητής

Με απλή αναφορά στο όνομα της.

π.χ.

θέλουμε να τυπώσουμε την τιμή της μεταβλητής `num`

```
printf ("%d\n", num) ;
```

θέλουμε να χρησιμοποιήσουμε την τιμή της `num` σε μία έκφραση:

```
printf ("%d\n", num + 1) ;
```

Τι κάνουν οι εντολές:

```
num = num + 1 ;
```

```
sum = sum + k ;
```

Ονόματα μεταβλητών

- ✓ Αποτελούνται από γράμματα του αγγλικού αλφαβήτου και ψηφία, με τον πρώτο χαρακτήρα να είναι γράμμα.
- ✓ Ο χαρακτήρας υπογράμμιση "_" θεωρείται γράμμα (χρήσιμος για να βελτιώσει την αναγνωσιμότητα μεγάλων ονομάτων μεταβλητών).
Συμβουλή: Μην αρχίζετε ονόματα μεταβλητών με χαρακτήρα υπογράμμισης (χρησιμοποιείται συνήθως από συναρτήσεις βιβλιοθήκης).
- ✓ Γίνεται διάκριση σε μικρά (πεζά) και κεφαλαία γράμματα.
Παραδοσιακή πρακτική: χρησιμοποιούμε μικρά γράμματα για ονόματα μεταβλητών.
- ✓ Τουλάχιστον οι 31 χαρακτήρες ενός ονόματος είναι σημαντικοί.
- ✓ Λέξεις κλειδιά όπως **int**, **if**, **else** είναι δεσμευμένες και δεν μπορούμε να τις χρησιμοποιήσουμε στα ονόματα μεταβλητών.
- ✓ Καλό είναι να επιλέγουμε ονόματα μεταβλητών σχετικά με τον σκοπό που εξυπηρετεί η κάθε μεταβλητή.

```
__a1
a
12ab
ab12
a12345678901234567890123456789012344
den_yparxei
ok
if
den#yparxei
ok=test
den yparxei
a1215235n
```

Τύποι δεδομένων

A) Ακέραιοι

Βασικός τύπος: **int**

Επιπλέον τύποι:

short int ή απλά **short** (μικροί ακέραιοι)

long int ή απλά **long** (μεγάλοι ακέραιοι)

unsigned int (πάντα θετικοί ή μηδέν)

Συνήθως:

int φυσικό μέγεθος ακεραίου

short 16bit

long 32bit

Κανόνας:

short, int τουλάχιστο 16bit

long τουλάχιστον 32bit

short <= int <= long

	μέγεθος (bytes)	Από	Εως
short int	2	SHRT_MIN -2^{15} -32.768	SHRT_MAX $2^{15} - 1$ 32.767
int	4	INT_MIN -2^{31} -2.147.483.648	INT_MAX $2^{31} - 1$ 2.147.483.647
long int	4	LONG_MIN -2^{31} -2.147.483.648	LONG_MAX $2^{31} - 1$ 2.147.483.647
unsigned int	4		UINT_MAX $2^{32} - 1$ 4.294.967.295

Τα μεγέθη των ακεραίων τύπων ορίζονται ως σταθερές στο αρχείο <limits.h> (θα πρέπει: #include <limits.h>)

Πώς βρίσκουμε το μέγεθος μνήμης μίας μεταβλητής:

π.χ.

```
int k;
```

```
unsigned int s;
```

```
s = sizeof k;
```

Ακέραιες Σταθερές:

Μια ακέραια σταθερά είναι int.

π.χ. 1560

Σταθερά long, γράφεται με l ή L στο τέλος. π.χ 1560L

Σταθερά unsigned, γράφεται με u ή U στο τέλος. π.χ 1560U

Ακέραιος πολυ μεγάλος για int, εκλαμβάνεται αυτόματα ως long.

Καθορισμός τιμής στο οκταδικό σύστημα:

0 (μηδέν) μπροστά από τον αριθμό

Καθορισμός τιμής στο δεκαεξαδικό σύστημα:

0x ή 0X (μηδέν και X ή x) μπροστά από τον αριθμό

k = 012;

printf("%d\n", k); $2 \times 8^0 + 1 \times 8^1 = 2 \times 1 + 1 \times 8 = 10$

k = 0x12;

printf("%d\n", k); $2 \times 16^0 + 1 \times 16^1 = 2 \times 1 + 1 \times 16 = 18$

k = 0x1f ή 0x1F 31

Παράμετροι printf

%d για int

%o για int στο οκταδικό σύστημα

%x για int στο δεκαεξαδικό σύστημα με μικρά abcdef

%X για int στο δεκαεξαδικό σύστημα με κεφαλαία ABCDEF

%u για unsigned int

```
long n = 31;
printf("d:%5d, o:%5o, x:%5x, X:%5X\n", n,n,n,n);
d:___31,  o:___37,  x:___1f,  X:___1F
```

B) Κινητής υποδιαστολής (πραγματικοί αριθμοί)

float	Απλής ακρίβειας
double	Διπλής ακρίβειας
long double	Εκτεταμένης ακρίβειας

float 4 bytes, double 8 bytes, long double 8 bytes

Σταθερές:

Περιέχουν μια υποδιαστολή ή μία δύναμη.

123.4 (τελεία όχι κόμμα)

1e-2

Είναι double εκτός εάν έχουν κατάληξη:

f, F για float

l, L για long double

Παράμετροι printf

%f για double δεκαδικός συμβολισμός της μορφής
[-]m.dddddd

%e για double δεκαδικός συμβολισμός της μορφής
[-]m.dddddexx

```
double n = 12.45;
printf("%f\n", n);      12.450000
printf("%6.2f\n", n);  _12.45
printf("%.1f\n", n);   12.4
printf("%3.1f\n", n);  12.4
printf("%e\n", n);     1.245000e+001
printf("%6.2e\n", n);  1.24e+001
```

Αριθμητικοί τελεστές

Δυναδικοί αριθμητικοί τελεστές (με δύο τελεστέους):

+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση Εάν διαίρεση μεταξύ ακεραίων, τότε αποκόβεται το κλασματικό μέρος (πηλίκο διαίρεσης), διαφορετικά κανονική διαίρεση. 5/2 ισούται με 2 5/2.0 ή 5.0/2 ή 5.0/2.0 ισούται με 2.5
%	Υπόλοιπο της διαίρεσης Μόνο μεταξύ int (όχι float ή double)

Μοναδικοί αριθμητικοί τελεστές (με ένα τελεστέο):

+	Συν
-	Μείον

Προτεραιότητες αριθμητικών τελεστών

ΙΣΧΥΡΗ	+ -	Μοναδικοί τελεστές
∨ ∨	* / %	Πολλαπλ, διαίρεση, υπόλοιπο
ΧΑΜΗΛΗ	+ -	Πρόσθεση, αφαίρεση

Οι αριθμητικοί τελεστές προσεταιρίζουν από αριστερά προς τα δεξιά.

Παραδείγματα αριθμητικών εκφράσεων:

$$2 + 3 * 5$$

$$2 + 15$$

$$17$$

$$(2+3) * 5$$

$$5 * 5$$

$$25$$

$$(12+6) / 3 * 2$$

$$18 / 3 * 2$$

$$6 * 2$$

$$12$$

$$10 \% 3 \% 2$$

$$1 \% 2$$

$$1$$

$$8 / 3 / 3$$

$$2 / 3$$

$$0$$

$$-3 + 8 / -2$$

$$-3 + -4$$

$$-7$$

Η συνάρτηση printf

Γράφει στην standard έξοδο (συνήθως στην οθόνη μας) τις τιμές που δέχεται ως παραμέτρους, με τον εξής τρόπο:

Η πρώτη της παράμετρος είναι μια συμβολοσειρά μορφοποίησης, η οποία περιέχει:

- κοινούς χαρακτήρες οι οποίοι αντιγράφονται ως έχουν στην έξοδο,
- και προδιαγραφές μετατροπών που καθεμιά προκαλεί την μετατροπή και εμφάνιση του επόμενου ορίσματος της printf. Κάθε προδιαγραφή μετατροπής αρχίζει με τον χαρακτήρα % και τελειώνει με έναν χαρακτήρα μετατροπής.

Παραδείγματα:

```
int n= 10;
printf("n =n\n");
printf("n =%d\n", n);
printf("n =%3d\n", n);
printf("n =%3d,%d\n", n, n*2);
```

```
n =n
n =10
n = 10
n = 10,20
```

Ακολουθίες διαφυγής:

\n νέα γραμμή
\t στηλογνώμονας
\a χαρακτήρας προειδοποίησης (κουδούνι)

```
printf("test\a\atest\n");
printf("test\ttest\n");
printf("\nt\nest");
```

```
testtest
test    test
```

```
t
est
```


Γ) Χαρακτήρας

char Μέγεθος: ένα μόνο byte.
Παίρνει ως τιμή έναν χαρακτήρα του συνόλου χαρακτήρων του μηχανήματος (ASCII).
Στην πραγματικότητα είναι ακέραιος τύπος.

Πίνακας ASCII

Πίνακας 256 θέσεων, από την θέση 0 έως την 255.

Σε κάθε θέση έχει αντιστοιχηθεί ένας χαρακτήρας.

Π.χ.

στη θέση 36 έχει αντιστοιχηθεί ο χαρακτήρας δολάριο\$,

στη θέση 48 ο χαρακτήρας μηδέν 0,

στην θέση 65 ο χαρακτήρας κεφαλαίο Αγγλικό Α κοκ

Κανόνες που πρέπει να ξέρουμε:

- Τα ψηφία βρίσκονται σε διαδοχικές θέσεις, αρχίζοντας από την θέση 48, με την εξής σειρά: '0', '1', '2', ... , '9'
- Οι κεφαλαίοι αγγλικοί χαρακτήρες είναι σε διαδοχικές θέσεις, αρχίζοντας από την θέση 65, σύμφωνα με την αλφαβητική τους σειρά: 'A', 'B', 'C', ... , 'Z'
- Οι μικροί αγγλικοί χαρακτήρες είναι σε διαδοχικές θέσεις, αρχίζοντας από την θέση 97, επίσης σύμφωνα με την αλφαβητική τους σειρά: 'a', 'b', 'c', ... , 'z'

Σε μια μεταβλητή τύπου χαρακτήρα μπορεί να εκχωρηθεί μια σταθερά χαρακτήρα, δηλαδή ένας χαρακτήρας μέσα σε μονά εισαγωγικά (π.χ. '0', '#', 'a', 'B', ...).

Στην πραγματικότητα εκχωρούμε ως τιμή του την θέση του χαρακτήρα στον πίνακα ASCII.

Είναι ισοδύναμο να καταχωρήσουμε τον χαρακτήρα '0' είτε την αριθμητική τιμή 48 (*προσοχή ο χαρακτήρας '0' είναι διαφορετικός από την ακέραια τιμή 0*).

Παράμετρος στην printf για χαρακτήρες: %c

```
#include <stdio.h>

void main()
{
    char ch;
    int nch;

    ch = 'A';
    nch = ch;

    printf("ASCII= %d, char=[%c]\n", nch, ch);

    nch = 66;
    ch = nch;
    printf("ASCII= %d, char=[%c]\n", nch, ch);

    printf("ASCII= %d, char=[%c]\n", ch, ch);
}

/*
ASCII= 65, char=[A]
ASCII= 66, char=[B]
ASCII= 66, char=[B]
*/
```

```
#include <stdio.h>
void main()
{
    char ch;
    int num;

    ch = '5';
    num = ch - '0';

    printf("char=[%c], %d\n", ch, num);
    ch = ch + 1;
    printf("next char=[%c]\n", ch);
}

/*
char=[5], 5
next char=[6]
*/
```

Συναρτήσεις εισόδου - εξόδου χαρακτήρων

Η είσοδος ή έξοδος χαρακτήρων, ανεξάρτητα από την προέλευση ή τον προορισμό της, αντιμετωπίζεται σαν ρεύμα χαρακτήρων.

α) getchar()

Κάθε φορά που καλείται διαβάζει τον επόμενο χαρακτήρα της εισόδου και τον επιστρέφει σαν τιμή.

Δεν δέχεται ορίσματα. Ανήκει στην βιβλιοθήκη <stdio.h>

Π.χ., μετά την εκτέλεση της εντολής:

```
c = getchar();
```

η μεταβλητή **c** θα περιέχει τον επόμενο χαρακτήρα της εισόδου. Οι χαρακτήρες κανονικά προέρχονται από το πληκτρολόγιο.

Προσοχή: Η getchar χρειάζεται <Enter> για να εκτελεστεί.

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

```
void main()
{
    char c;
    c = getchar();
    printf("%d", c);
    c = getchar();
    printf("%d", c); printf("\n");
    c = getchar();
    printf("#%d", c);
}
```

είσοδος:	έξοδος:
01234<Enter>	01234
	<u>4849</u>
	#50

β) putchar()

Δέχεται ως όρισμα έναν χαρακτήρα τον οποίο εμφανίζει στην έξοδο (συνήθως στην οθόνη), σαν χαρακτήρα. Οι συναρτήσεις putchar και printf μπορεί να εναλλάσσονται.

π.χ. putchar(c) όπου c μεταβλητή τύπου char,
putchar('x'); putchar(getchar());

γ) getch()

Λειτουργεί όπως και η getchar, με τις εξής διαφορές:
ΔΕΝ χρειάζεται <Enter> για να εκτελεστεί, εκτελείται μόλις πληκτρολογηθεί ένας χαρακτήρας.

Ορίζεται στην βιβλιοθήκη <conio.h>, όχι στην <stdio.h>.

δ) getch()

Όπως η getch, με την διαφορά ότι ο χαρακτήρας που διαβάζεται δεν εμφανίζεται στην οθόνη.

```
#include <stdio.h>
#include <conio.h>
void main()
{   char c;
    c = getch(); printf("%d", c);
    c = getch(); printf("%d", c); printf("\n");
    c = getch(); printf("#%d", c);
}
```

είσοδος:	έξοδος:
0	<u>48</u>
1	<u>4849</u>
8	4849 <u>#56</u>

```
#include <stdio.h>
#include <conio.h>
void main()
{   char c;
    c = getche(); printf("%d", c);
    c = getche(); printf("%d", c); printf("\n");
    c = getche(); printf("#%d", c);
}
```

είσοδος:	έξοδος:
0	<u>048</u>
1	<u>048149</u>
8	048149 <u>8#56</u>

Συσχετιστικοί τελεστές

>	Μεγαλύτερο
>=	Μεγαλύτερο ή ίσο
<	Μικρότερο
<=	Μικρότερο ή ίσο
==	Ισότητα
!=	Ανισότητα

Έχουν χαμηλότερη προτεραιότητα από τους αριθμητικούς τελεστές.

Έτσι π.χ. $10 < 9 + 2$ εκλαμβάνεται ως: $10 < (9 + 2)$

Στην C, οι τιμές ΑΛΗΘΕΣ και ΨΕΥΔΕΣ αναπαριστούνται από τις αριθμητικές τιμές 1 (ΕΝΑ) και 0 (ΜΗΔΕΝ). Δεν υπάρχει λογικός τύπος δεδομένων όπως π.χ. ο τύπος Boolean της Pascal, ούτε ειδικές τιμές της μορφής TRUE/FALSE.

Συνεπώς, το αποτέλεσμα μίας πράξης με τελεστές συσχέτισης είναι:

- 1, εάν η συσχέτιση είναι αληθής
- 0, σε αντίθετη περίπτωση.

Παραδείγματα:

$10 == 6 + 6$

$10 == 12$

0

$(10 == 6 + 4) * 2$

$(10 == 10) * 2$

$1 * 2$

2

$10 > 5 > 3$

$1 > 3$

0

Λογικοί τελεστές

& &	Λογικό ΚΑΙ
 	Λογικό Η
!	Μοναδικός τελεστής άρνησης. Μετατρέπει έναν μη-μηδενικό τελεστέο σε 0, και έναν μηδενικό σε 1.

Οι παραστάσεις που σχετίζονται με λογικούς τελεστές υπολογίζονται από αριστερά προς τα δεξιά, και ο υπολογισμός σταματάει μόλις εξακριβωθεί το αληθές ή το ψευδές του αποτελέσματος.

Η προτεραιότητα των λογικών τελεστών είναι χαμηλότερη από των συσχετιστικών τελεστών. Μεταξύ τους, ο τελεστής && έχει υψηλότερη προτεραιότητα από τον | |.

Παραδείγματα:

10> 2+7 && 5+6>3

10>9 && 11>3

1 && 1

1

!(10!=4)

!1

0

!!5

!0

1

Τελεστής μετατροπής

Με τον τελεστή αυτό επιβάλλουμε ρητές μετατροπές τύπων.

Λέγεται προσαρμογή (cast), είναι ένας μοναδικός τελεστής και ορίζεται ως εξής:
(όνομα τύπου) παράσταση

Η μετατροπή float σε int προκαλεί την αποκοπή τυχόν κλασματικού μέρους.

Παραδείγματα:

```
(int) 5.7 + 4  
5 + 4  
9
```

```
(float) 10 / 3  
10.0 / 3  
3.333
```

```
(float) (10/3)  
(float) 3  
3.0
```

Τελεστής μοναδιαίας αύξησης και μείωσης

Η C διαθέτει δύο ασυνήθιστους τελεστές για την αύξηση και μείωση μεταβλητών:

++

--

Μπορούν να χρησιμοποιηθούν σαν:

- προθεματικοί, πριν την μεταβλητή, στην μορφή ++n
- μεταθεματικοί, μετά την μεταβλητή, στην μορφή n++

Και στις δύο περιπτώσεις το αποτέλεσμα είναι η μοναδιαία αύξηση της μεταβλητής n.

Η διαφορά τους:

- Η παράσταση ++n, αυξάνει την n ΠΡΙΝ χρησιμοποιηθεί η τιμή της.
- Η παράσταση n++, αυξάνει την n αφού έχει χρησιμοποιηθεί η τιμή της.

```
int a=1;
```

```
printf("%d\n", 2 * ++a);
```

Θα εμφανίσει την τιμή 4 και η τιμή του a θα γίνει 2.

```
int a=1;
```

```
printf("%d\n", 2 * a++);
```

Θα εμφανίσει την τιμή 2 και η τιμή του a θα γίνει 2.

Τελεστές εκχώρησης

Παραστάσεις της μορφής:

$i = i + 2$

όπου η μεταβλητή του αριστερού μέρους επαναλαμβάνεται στην αρχή του δεξιού μέρους, μπορούν να γραφούν στην μορφή

$i += 2$

Τελεστές της μορφής += λέγονται τελεστές εκχώρησης.

Οι περισσότεροι δυαδικοί τελεστές έχουν τον αντίστοιχο τους τελεστή εκχώρησης:

+	-	*	/	%
+=	-=	*=	/=	%=

Γενικότερα, αν παρ_1 και παρ_2 είναι παραστάσεις, και τελ τελεστής, τότε:

$\text{παρ}_1 \text{τελ} = \text{παρ}_2$

είναι ισοδύναμη με:

$\text{παρ}_1 = (\text{παρ}_1) <\text{τελ}> (\text{παρ}_2)$

$x *= y + 1;$ σημαίνει $x = x * (y+1);$

Π.χ.

```
int x = 5, y = 1;
x *= 2;           /* x = x * 2 */
printf("%d\n", x) /* θα τυπώσει 10 */
x %= 2 + y;       /* x = x % (2+y) */
printf("%d\n", x) /* θα τυπώσει 1 */
x /= 2;           /* x = x / 2 */
printf("%d\n", x) /* θα τυπώσει 0 */
```

Παραστάσεις υπό συνθήκη

Έχουν την μορφή:

$\text{παρ}_1 \text{ ? παρ}_2 \text{ : παρ}_3$

Υπολογίζεται πρώτα η παράσταση παρ_1 .

Σε περίπτωση που η τιμή της είναι μη-μηδενική (Αληθής), τότε υπολογίζεται η παράσταση παρ_2 και η τιμή της παρ_2 γίνεται η τιμή της παράστασης υπό συνθήκη.

Αλλιώς, εάν δηλαδή η παρ_1 είναι μηδενική (Ψευδής), τότε υπολογίζεται η παράσταση παρ_3 και η τιμή της παρ_3 γίνεται η τιμή της παράστασης.

Παράδειγμα:

$z = (a > b) \text{ ? } a \text{ : } b;$

Στην μεταβλητή z εκχωρούμε το μέγιστο των τιμών των a και b .

Ισοδυναμεί με τον ψευδοκώδικα:

Εάν $a > b$ τότε

$z = a$

αλλιώς

$z = b$

Προτεραιότητα τελεστών

ΜΕΓΑΛΗ	()	Παρενθέσεις
	! ++ -- + - (τύπος)	ΜΟΝΑΔΙΑΙΟΙ ΤΕΛΕΣΤΕΣ: λογικός τελεστής άρνησης Αύξησης Μείωσης Πρόσημα Αλλαγής τύπου
	* / %	ΔΥΑΔΙΚΟΙ, ΑΡΙΘΜΗΤΙΚΟΙ Πολλαπλασιασμός Διαίρεση Υπόλοιπο διαίρεσης
	+ -	ΔΥΑΔΙΚΟΙ, ΑΡΙΘΜΗΤΙΚΟΙ Πρόσθεση Αφαίρεση
	< <= > >=	ΣΥΣΧΕΤΙΣΗΣ Μικρότερο Μικρότερο ή ίσο Μεγαλύτερο Μεγαλύτερο ή ίσο
	== !=	ΣΥΣΧΕΤΙΣΗΣ Ισότητα Ανισότητα
	&&	Λογικό ΚΑΙ
		Λογικό Η
	? :	Παράσταση υπό συνθήκη
ΜΙΚΡΗ	= += -= *= /= %=	Εκχώρησης Τελεστές εκχώρησης

Οι τελεστές που βρίσκονται στο ίδιο κελί του παραπάνω πίνακα έχουν ίδια προτεραιότητα.

Συμβολικές σταθερές

#define <όνομα> <κείμενο αντικατάστασης>

Παραδείγματα:

#define MAX 1000

#define YES 1

Οποιαδήποτε εμφάνιση του <όνομα> στο κείμενο του προγράμματος θα αντικαθίσταται από το κείμενο αντικατάστασης. Αντικατάσταση δεν γίνεται στις περιπτώσεις που το <όνομα> βρίσκεται μέσα σε εισαγωγικά ή όταν είναι μέρος άλλου ονόματος.

Το <όνομα> έχει την μορφή ενός ονόματος μεταβλητής.

Προσοχή: Στο τέλος της γραμμής #define δεν μπαίνει ελληνικό ερωτηματικό (;)

Παράδειγμα χρήσης συμβολικού ονόματος σε πρόγραμμα:

Μπορούμε να γράψουμε:

if (value > MAX)

αντί για:

if (value > 1000)

Παραδείγματα που δεν γίνεται αντικατάσταση:

printf("YES.\n");

YESMAN = 10;

Πλεονεκτήματα συμβολικών σταθερών:

Κάνουν τα προγράμματα μας πιο ευανάγνωστα και πιο εύκολα στην συντήρησή τους. Είναι κακή πρακτική να γράφω σταθερές τιμές σε οποιονδήποτε σημείο του προγράμματος. Προτιμώ να τις συγκεντρώνω σε ένα

σημείο και να τους δίδω συμβολικά ονόματα που να με βοηθούν στην κατανόηση του προγράμματος.

Συνηθίζω να γράφω τα συμβολικά ονόματα με κεφαλαία γράμματα για να διακρίνονται εύκολα.

Προεπεξεργαστής: ένα ξεχωριστό πρώτο βήμα μεταγλώττισης. Μεταβάλλει το κείμενο του προγράμματος πριν την κανονική του μεταγλώττιση (δεν γνωρίζει C!). Στο δεύτερο βήμα θα γίνει η κανονική μεταγλώττιση του κειμένου όπως αυτό θα έχει διαμορφωθεί από τον προεπεξεργαστή.

Βασικές εντολές προς τον προεπεξεργαστή:

#include <όνομα αρχείου> ή #include "όνομα αρχείου"	Συμπεριλαμβάνει τα περιεχόμενα του αρχείου στη μεταγλώττιση. Εάν το όνομα αρχείου είναι μέσα σε διπλά εισαγωγικά τότε ο προεπεξεργαστής θα το αναζητήσει πρώτα στον κατάλογο του πηγαίου κώδικα.
#define	Αντικατάσταση ενός λεκτικού συμβόλου με μια αυθαίρετη ακολουθία χαρακτήρων.

Συχνά όλα τα #define μιας εφαρμογής τα γράφουμε σε ανεξάρτητο αρχείο, το οποίο στην συνέχεια κάνουμε #include.

Προτάσεις ελέγχου: if

<code>if (έκφραση) εντολή₁</code>	Μας επιτρέπει να επιλέξουμε αν θα γίνει ή όχι μια ενέργεια
<code>if (έκφραση) εντολή₁ else εντολή₂</code>	Μας επιτρέπει να επιλέξουμε μεταξύ δύο ενεργειών
<code>if (έκφραση) εντολή₁ else if (έκφραση) εντολή₂ else if (έκφραση) εντολή₃ ... else εντολή_N</code>	Επιλογή μεταξύ περισσοτέρων από δύο ενεργειών Το else συνδυάζεται με το πιο κοντινό if (εκτός εάν υπάρχουν αγκύλες)

Στα παραπάνω εντολή, είναι:

- είτε μιά απλή εντολή,
- είτε ένα μπλοκ εντολών, μια ομάδα δηλαδή εντολών που βρίσκονται μέσα σε άγκιστρα { }.

Σημειώνουμε ότι κάθε εντολή στην C τελειώνει με Ελληνικό ερωτηματικό, γι' αυτό και θα βάζουμε πάντα ερωτηματικό ακόμη και εάν ακολουθεί else ή άγκιστρο.

Σε μία εντολή `if` υπολογίζεται πρώτα η έκφραση που ακολουθεί της `if`. Η έκφραση αυτή γράφεται πάντα μέσα σε παρενθέσεις `()`.

Εάν είναι αληθής (δηλαδή εάν η τιμή της είναι διαφορετική του μηδενός), τότε εκτελείται η εντολή (ή το μπλοκ εντολών) που ακολουθεί του `if`.

Εάν είναι ψευδής (δηλαδή εάν η τιμή της είναι μηδέν), τότε εκτελείται η εντολή (ή το μπλοκ εντολών) που ακολουθεί το `else` (εφόσον βέβαια υπάρχει τέτοια).

Συνήθως στην έκφραση της `if` θα συναντάμε σχεσιακούς τελεστές ή/και λογικούς τελεστές.

π.χ.

```
if (a>10)
```

```
if (ch >='a' && ch <='b')
```

```
if ((k>10) || (k<5))
```

Μπορούμε όμως να έχουμε οποιαδήποτε έκφραση:

π.χ.

if (k) Θα εκτελεστεί η εντολή του `if` εάν το `k` είναι διαφορετικό του μηδενός

if (k%3) Θα εκτελεστεί η εντολή του `if` εάν το `k` δεν είναι πολλαπλάσιο του 3.

if (1) Σε κάθε περίπτωση θα εκτελεστεί η εντολή του `if`.

Σημειώνουμε ότι η τιμή μίας πρότασης εκχώρησης είναι η τιμή της αριστερής πλευράς.

π.χ. **if** (k = 5 + a)

Θα υπολογιστεί πρώτα η παράσταση `5+a` και η τιμή της θα εκχωρηθεί στην μεταβλητή `k`. Η τιμή του `k`, όπως αυτή διαμορφωθεί, θα είναι και η τιμή της έκφρασης. Εάν είναι διάφορη του 0 θα εκτελεστεί η εντολή του `if`.

Προτάσεις ελέγχου: switch

Η εντολή **switch** είναι μια διακλαδωμένη απόφαση που ελέγχει αν μία παράσταση ταυτίζεται με μια τιμή από ένα σύνολο *σταθερών τιμών*, και διακλαδίζεται ανάλογα:

```
switch (παράσταση) {  
    case σταθερή1: εντολές  
    case σταθερή2: εντολές  
    ...  
    default: εντολές  
}
```

Σε μία εντολή **switch**, πρώτα υπολογίζεται η τιμή της παράστασης.

Αν η τιμή αυτή είναι ίση με την τιμή μίας περίπτωσης (*case*), τότε η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που ακολουθεί την *case*.

Σημειώνουμε ότι εκτελούνται όλες οι υπόλοιπες εντολές (ακόμη και αυτές των *case* που ακολουθούν), εκτός εάν υπάρχει η εντολή **break** η οποία διακόπτει την εκτέλεση της *switch*. *Θεωρούμε ότι οι case εξυπηρετούν απλώς ως ετικέτες, δεν είναι εντολές που εκτελούνται.*

Σημειώνουμε επίσης ότι οι τιμές των *case* θα πρέπει να είναι διαφορετικές.

Στην περίπτωση που η τιμή της παράστασης δεν υπάρχει σε καμία *case* τότε εκτελούνται οι εντολές που ακολουθούν της προαιρετικής ετικέτας **default** (εάν δεν υπάρχει *default* δεν γίνεται καμία ενέργεια της *switch*).

Παράδειγμα προγράμματος με switch:

```
#include <stdio.h>

void main()
{
    char let;

    printf("Πληκτρολόγησε ένα γράμμα:");
    scanf("%c", &let);
    switch (let)    {
    case 'Α':
    case 'Ε':
        printf("\nΦΩΝΗΕΝ.\n");
        break;
    case 'Β':
    case 'Γ':
    case 'Δ':
        printf("\nΣΥΜΦΩΝΟ.\n");
        break;
    default:
        printf("ΟΧΙ ΣΤΟΥΣ ΠΡΩΤΟΥΣ ΠΕΝΤΕ.");
        break;
    }
}
```

Πίνακας

Πίνακας είναι μια διατεταγμένη ακολουθία στοιχείων κάποιου τύπου που βρίσκονται αποθηκευμένα σε γειτονικές (διαδοχικές) θέσεις μνήμης.

Το κάθε στοιχείο διακρίνεται από το άλλο με μία αριθμητική ετικέτα. Το πρώτο στοιχείο του πίνακα έχει πάντα την ετικέτα 0 (μηδέν).

Όλα τα στοιχεία ενός πίνακα είναι του ίδιου τύπου.

π.χ.

int **pin[10]** ; Πίνακας 10 ακεραίων.

0		pin[0] το πρώτο στοιχείο του πίνακα
1		pin[1]
2		
3		
4		
5		
6		
7		
8		
9		pin[9] το τελευταίο στοιχείο του πίνακα

Συμβολοσειρά (string)

Είναι πίνακας τύπου `char` που τερματίζεται με τον ΕΙΔΙΚΟ χαρακτήρα `'\0'` (backslash μηδέν), ο οποίος υποδηλώνει και το τέλος του.

π.χ.

```
char lexi[] = "TEST";
```

Τα διπλά εισαγωγικά απλά οριοθετούν την συμβολοσειρά.

`lexi`

<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
T	E	S	T	\0

```
char lexi2[8];
```

```
lexi2[0] = 'E';
```

```
lexi2[1] = 'N';
```

```
lexi2[2] = 'A';
```

```
lexi2[3] = '\0';
```

`lexi2`

<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
E	N	A	\0				

Ανάγνωση συμβολοσειρών με την `scanf`:

- Χρησιμοποιούμε τον προσδιοριστή `%s`
- Η `scanf` διαβάζει απλές λέξεις και όχι φράσεις (διαβάζει μέχρι να συναντήσει τον χαρακτήρα κενό).
- Στο τέλος της συμβολοσειράς τοποθετεί τον ειδικό χαρακτήρα `\0`.

Εκτύπωση συμβολοσειράς γίνεται με την `printf`, με τον προσδιοριστή `%s`.

Παράδειγμα:

```
#include <stdio.h>

void main()
{
    char    name[30];
    printf("Δώσε όνομα:\n");
    scanf("%s", name);
    printf("Σε λένε %s.\n", name);
}
```

Προσοχή: η name δεν παίρνει & (γιατί είναι πίνακας).

Παράδειγμα εκτέλεσης του προγράμματος:

Δώσε όνομα:

Παναγιώτης Καλογεράκης

Σε λένε Παναγιώτης.

Μήκος συμβολοσειράς

Το δίδει η συνάρτηση **strlen**, η οποία έχει οριστεί στην βιβλιοθήκη **string.h**.

Η συνάρτηση αυτή παίρνει ως παράμετρο μια συμβολοσειρά και επιστρέφει το μήκος της (μια ακέραια τιμή): *Μετράει τους χαρακτήρες της συμβολοσειράς μέχρι τον ειδικό χαρακτήρα \0 (χωρίς αυτόν).*

Στο προηγούμενο παράδειγμα προσθέτουμε τον κώδικα:

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

Στο τέλος του προγράμματος:

```
printf("%d\n", strlen(name) );      /* 10 */
printf("%d\n", sizeof name );      /* 30 */
```

Η `sizeof name` επιστρέφει το πλήθος των bytes που καταλαμβάνει στην μνήμη η μεταβλητή `name`.

Βρόγχοι

Με τους βρόγχους (loop) μας δίδεται η δυνατότητα να εκτελέσουμε ένα κομμάτι του προγράμματος μας πολλές φορές.

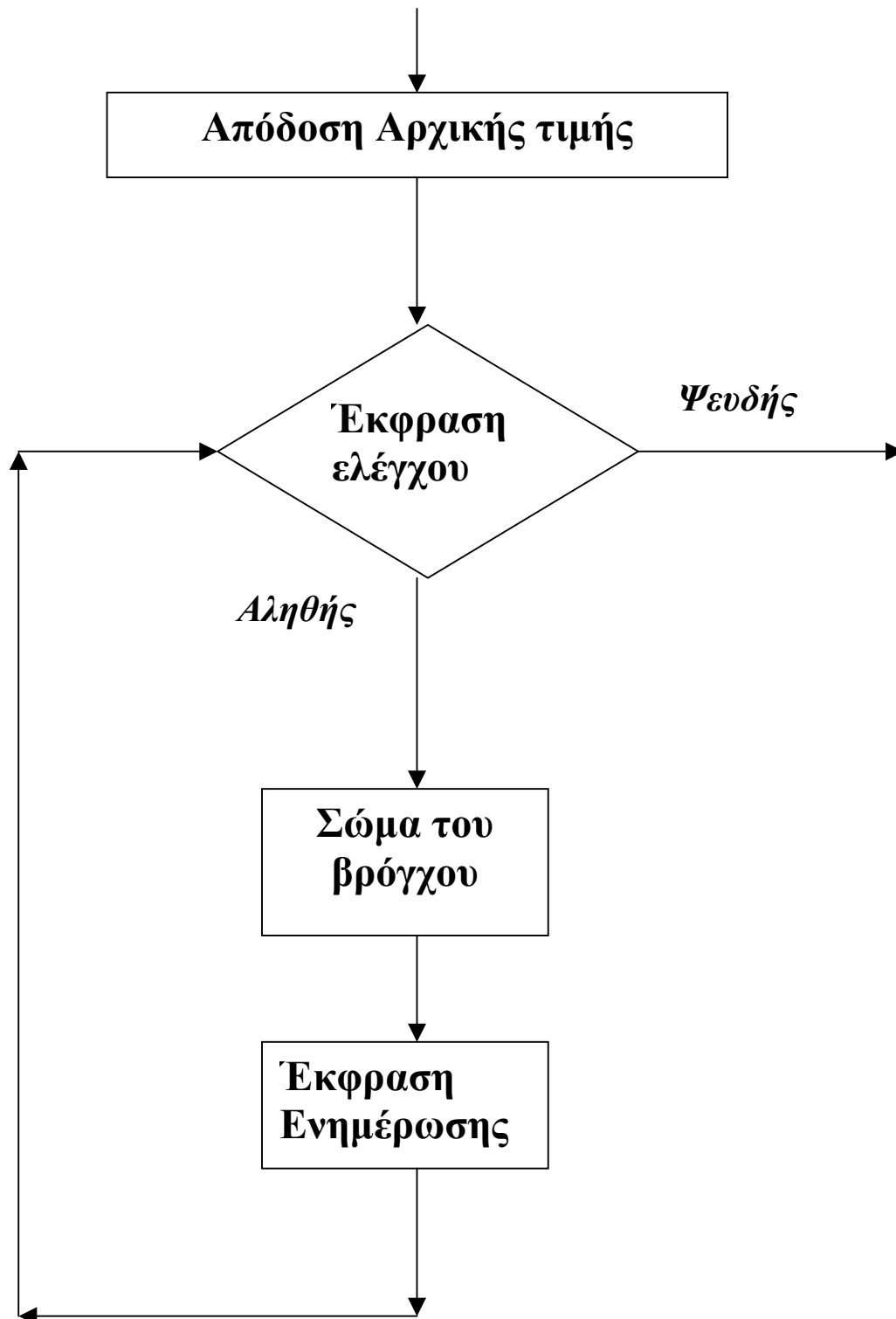
Βρόγχοι: for

Η εντολή for χρησιμοποιεί τρεις εκφράσεις, οι οποίες γράφονται μετά την λέξη for μέσα σε παρενθέσεις, και χωρίζονται μεταξύ τους με ελληνικά ερωτηματικά.

Οι εκφράσεις αυτές είναι:

Έκφραση απόδοσης αρχικής τιμής	Εκτελείται ΜΙΑ φορά, πριν από οποιαδήποτε άλλη πρόταση του βρόγχου.
Έκφραση ελέγχου	Υπολογίζεται πριν αρχίσουμε να εκτελούμε τις εντολές του βρόγχου. Εάν είναι αληθής (διαφορετική του μηδενός) τότε εκτελείται το σώμα του βρόγχου. Διαφορετικά ο βρόγχος τερματίζεται.
Έκφραση ενημέρωσης	Εκτελείται μετά την εκτέλεση του σώματος του βρόγχου. Στην συνέχεια ξαναυπολογίζεται η έκφραση ελέγχου και ανάλογα με το εάν αυτή είναι αληθής ή όχι επαναλαμβάνεται η εκτέλεση του σώματος του βρόγχου ή τερματίζεται ο βρόγχος.

Ο βρόγχος for:



Παράδειγμα χρήσης του βρόγχου for:

```
for (met= 1; met <= 10; met++)  
    printf("Καλημέρα\n") ;
```

met = 1	Απόδοση αρχικής τιμής Εκτελείται μία μόνο φορά πριν από οποιαδήποτε άλλη πρόταση του βρόγχου.
met <= 10	Έκφραση ελέγχου Εάν είναι αληθής, εκτελείται το σώμα του βρόγχου
met++	Έκφραση ενημέρωσης Υπολογίζεται στο τέλος κάθε επανάληψης, και στην συνέχεια εξετάζεται πάλι η έκφραση ελέγχου
printf("Καλημέρα\n")	Το σώμα του βρόγχου.

Στο παραπάνω παράδειγμα θέλουμε να εκτελέσουμε ένα συγκεκριμένο πλήθος επαναλήψεων (10 επαναλήψεις). Η αντιμετώπιση του προβλήματος με την for είναι η εξής:

- Χρησιμοποιούμε μια int μεταβλητή η οποία θα παίζει τον ρόλο του μετρητή (την met).
- Στην έκφραση απόδοσης αρχικής τιμής εκχωρούμε στην μεταβλητή αυτή την τιμή 1 (met = 1).
- Στην έκφραση ελέγχου συγκρίνουμε την τιμή του μετρητή με το πλήθος των επαναλήψεων με τον σχεσιακό τελεστή μικρότερο ή ίσο (met <= 10).
- Στην έκφραση ενημέρωσης αυξάνουμε την τιμή του μετρητή κατά 1 (σημείωση: ο μετρητής ΔΕΝ αυξάνει αυτόματα).

Παραδείγματα for:

1)

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

```
void main()
```

```
{
```

```
int num;
```

```
for (num=0; num<=10; num++)
```

```
    printf("%5d %8d\n", num, num*num*num);
```

```
}
```

2)

```
for (num=0; num<=10; num += 2)
```

```
    printf("%5d %8d\n", num, num*num*num);
```

3)

```
int n;
```

```
for (n=10; n>0; n--)
```

```
    printf("%d επαναλήψεις ακόμη\n", n);
```

```
printf("Εκκίνηση!\n");
```

4)

```
int k;
```

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

```
printf("telos!\n");
```

Προσοχή το σώμα του βρόγχου είναι κενό!!!

5)

```
for (k=15; k<=10; k++)
```

```
    x++;
```

Πόσες φορές θα εκτελεστεί το σώμα του βρόγχου;

6)

```
char ch;
for (ch='a'; ch<='z'; ch++)
    printf("ASCII %c = %d\n", ch, ch);
```

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", x, y);
    printf("\n");
}
```

for και πίνακες

9) Μηδενισμός πίνακα

```
int pin[28], j;
for (j=0; j<28; j++)
    pin[j] = 0;
```

10) Ανάγνωση πίνακα, χειρισμός του

```
#include <stdio.h>
void main()
{
    int pin[8], j;
    for (j=0; j<8; j++)
        scanf("%d", &pin[j]);
    for (j=0; j<8; j++)
        printf("%d\t%d\n", pin[j], pin[j]*pin[j]);
}
```

```
11)
#include <stdio.h>
#include <conio.h>

void main()
{
    int k;
    char    str[10];
    for (k=0; k<6;k++)
        str[k] = getche();
    str[6] = '\0';
    printf("\n$s%s", str);
}
```

Στο παραπάνω παράδειγμα δημιουργούμε μια συμβολοσειρά (string) με τους χαρακτήρες που μας δίνει ο χρήστης, και στην συνέχεια την εκτυπώνουμε ανάμεσα σε δολάρια.

Προσοχή: είναι απαραίτητο να τερματίσουμε την συμβολοσειρά με τον Ειδικό χαρακτήρα \0.

Εάν έλλειπε από το πρόγραμμα η εντολή:

```
str[6] = '\0';
```

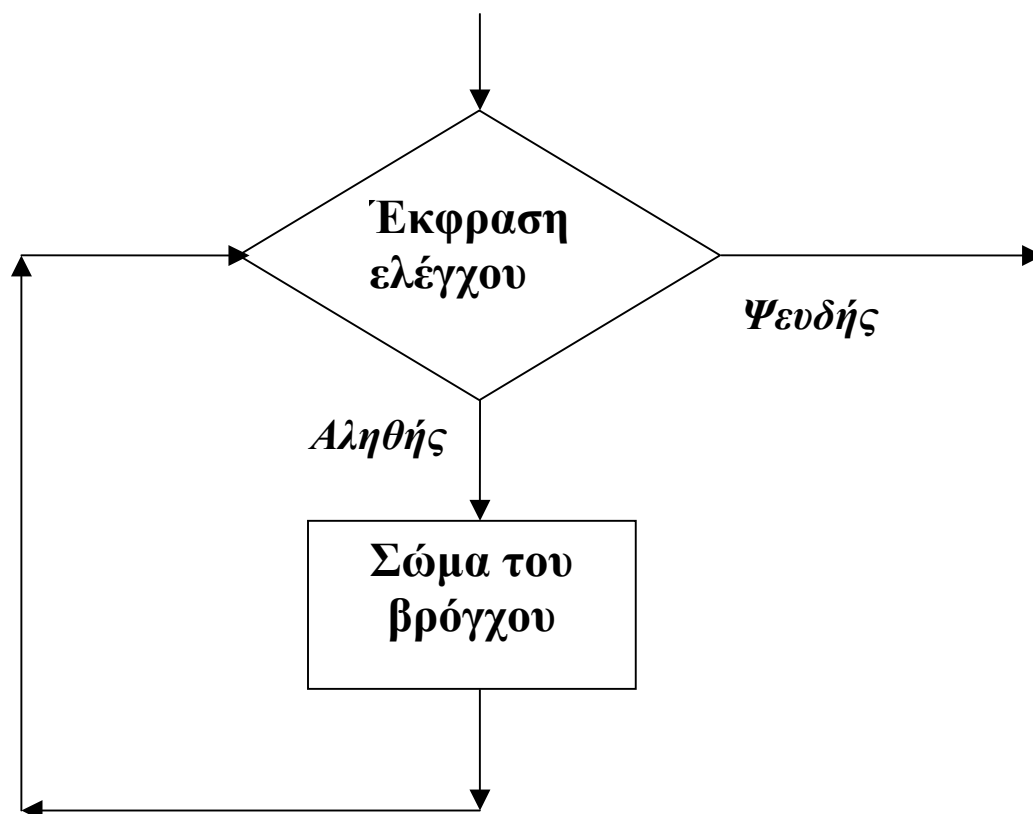
τότε ο str δεν θα ήταν συμβολοσειρά αλλά ένας απλός πίνακας χαρακτήρων.

Η εντολή printf που ακολουθεί, όπως και κάθε άλλη συνάρτηση που χειρίζεται συμβολοσειρές, ΔΕΝ ΘΑ ΔΟΥΛΕΥΕ ΣΩΣΤΑ.

Βρόγχοι: while

Η εντολή while χρησιμοποιεί μία μόνο έκφραση, η οποίες γράφεται μετά την λέξη while μέσα σε παρενθέσεις.

while (έκφραση) εντολή	Υπολογίζεται η έκφραση και στην περίπτωση που είναι αληθής (διαφορετική του μηδενός) εκτελείται το σώμα του βρόγχου. Διαφορετικά ο βρόγχος τερματίζεται. Με την ολοκλήρωση του σώματος του βρόγχου υπολογίζεται και πάλι η έκφραση του βρόγχου και όσο είναι αληθής εκτελείται το σώμα του βρόγχου.
----------------------------------	---



Παραδείγματα while:

1)

```
int index = 10;
while (index<15)
    printf("%d\n", index);
```

Τι κάνω λάθος στο παραπάνω πρόγραμμα;

2)

```
int index = 10;
while (index++ < 15)
    printf("%d, ", index);
/* θα τυπώσει: 11, 12, 13, 14, 15, */
```

3)

```
int index = 10;
while (++index < 15)
    printf("%d, ", index); /* 11..14 */
/* θα τυπώσει: 11, 12, 13, 14, */
```

4)

```
#define STOP '*'
void main()
{
    char ch;
    ch = getche();
    while (ch != STOP){
        putchar(ch);
        ch = getche();
    }
}
```

5)

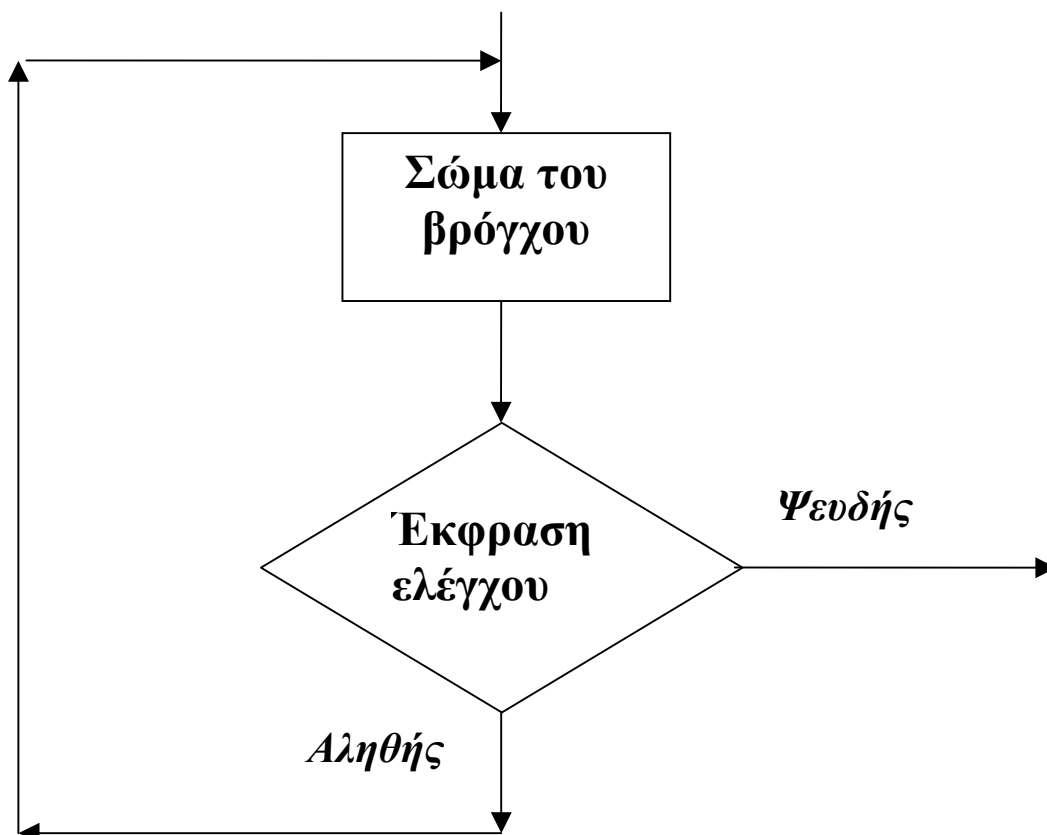
```
#define STOP '*'
void main()
{
    char ch;
    while ((ch = getche()) != STOP)
        putchar(ch);
}
```

Βρόγχοι: do .. while

Η εντολή do while χρησιμοποιεί μία μόνο έκφραση, η οποίες γράφεται μετά την λέξη while μέσα σε παρενθέσεις στο τέλος του βρόγχου.

do εντολη while (έκφραση)	Εκτελείται το σώμα του βρόγχου και στην συνέχεια υπολογίζεται η έκφραση. Όσο η έκφραση είναι αληθής εκτελείται το σώμα του βρόγχου.
---	---

Προσοχή: ο έλεγχος της συνθήκης εξόδου γίνεται μετά την εκτέλεση του σώματος του βρόγχου, άρα το σώμα του βρόγχου θα εκτελεστεί τουλάχιστο μία φορά.



Παραδείγματα do .. while:

```
#include <stdio.h>

#define STOP '*'

void main()
{
    int count = 0;
    do
    {
        printf("%d", count);
        count++;
    }
    while (count != 9);
    printf("\n");
}
```

Προτάσεις ελέγχου ειδικής χρήσης

break	Διακόπτει την εκτέλεση των προτάσεων: switch, for, while, do while
continue	Έχει σαν αποτέλεσμα τη μη εκτέλεση του υπολοίπου τμήματος του σώματος του βρόχου

Παραδείγματα:

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

Αντί να γράφουμε περίπλοκες συνθήκες στην *while*, μπορούμε να δώσουμε επιπλέον δυνατότητες τερματισμού ενός βρόγχου με *break* μέσα σε *if* που θα προσθέσουμε μέσα στον σώμα του βρόγχου.

```
char ch; int cnt = 0;
while ((ch=getche()) != '.'){
    if (ch>='a' && ch<='z')
        continue;
    cnt++;
}
printf("%d", cnt);
```

ΣΥΝΑΡΤΗΣΕΙΣ

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

Πλεονεκτήματα των συναρτήσεων:

- ✓ Οι συναρτήσεις μας γλιτώνουν από τον επαναληπτικό προγραμματισμό.
- ✓ Κάνουν το πρόγραμμα πιο εύκολο στο διάβασμα και στην τροποποίηση.
- ✓ Με την βοήθεια των συναρτήσεων μετατρέπουμε ένα σύνθετο πρόβλημα σε πολλά απλά. Στην συνέχεια αρκεί να επιλύσουμε, ανεξάρτητα το ένα από τα άλλα, καθένα από αυτά τα απλά προβλήματα.
- ✓ Η γλώσσα προγραμματισμού μου παρέχει ισχυρές αλλά χαμηλού επιπέδου εντολές. Οι συναρτήσεις μου παρέχουν πολύ πιο σύνθετα και ισχυρά εργαλεία στα οποία μπορώ να βασιστώ για να χτίσω ευκολότερα τις εφαρμογές μου. Οι συναρτήσεις εμπλουτίζουν την γλώσσα προγραμματισμού.

main()

Ιδιαίτερη συνάρτηση ενός προγράμματος C: από αυτήν αρχίζει η εκτέλεση του προγράμματος.

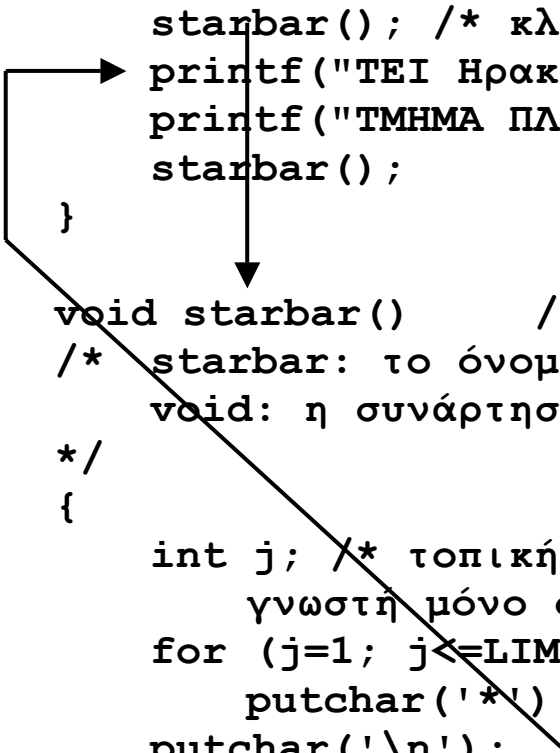
Η C μου δίδει την δυνατότητα, εκτός από το να καλούμε συναρτήσεις όπως κάναμε μέχρι τώρα (printf, scanf, getch κοκ), να ορίζουμε δικές μου συναρτήσεις.


```
#include <stdio.h>
#define LIM 50

void starbar();
/* δήλωση συνάρτησης, η συνάρτηση πρέπει να
δηλωθεί πριν χρησιμοποιηθεί */

void main()
{
    starbar(); /* κλήση συνάρτησης */
    printf("ΤΕΙ Ηρακλείου\n");
    printf("ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ\n");
    starbar();
}

void starbar() /* ορισμός συνάρτησης */
/* starbar: το όνομα της συνάρτησης
void: η συνάρτηση δεν επιστρέφει τιμή
*/
{
    int j; /* τοπική μεταβλητή,
γνωστή μόνο στην starbar */
    for (j=1; j<=LIM; j++)
        putchar('*');
    putchar('\n');
}
```



Όταν η εκτέλεση του προγράμματος φθάσει στην `starbar()`, εκτελούνται οι εντολές που βρίσκονται εκεί. Μετά η εκτέλεση επιστρέφει στην επόμενη γραμμή του καλούντος προγράμματος.

Η συνάρτηση `starbar`, κάνει μια απλή συγκεκριμένη δουλειά: κάθε φορά που καλείται εκτυπώνει στην οθόνη 50 αστεράκια και μετά αλλάζει γραμμή.

Αυτό δεν μας αρέσει, προτιμάμε περισσότερο γενικές και ευέλικτες συναρτήσεις.

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

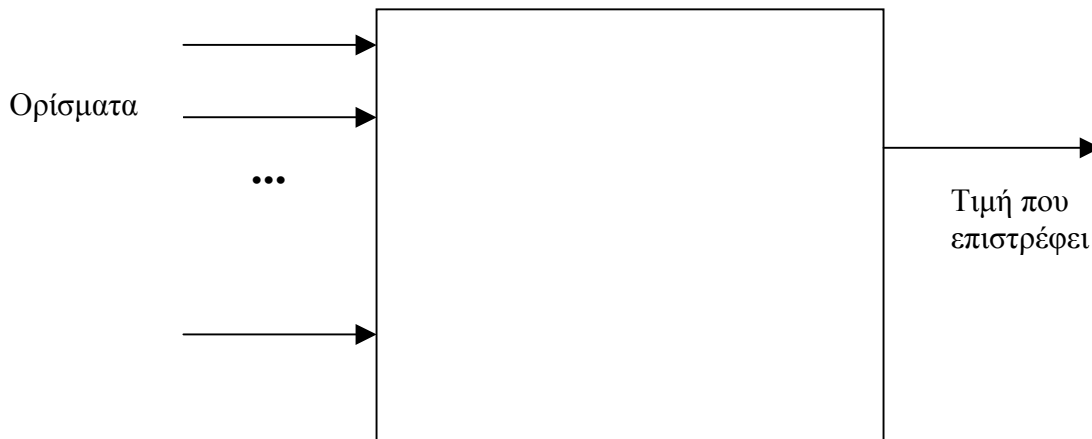
void starbar(int); /* δήλωση */
void main()
{
    char line1[] = "ΤΕΙ Ηρακλείου";
    char line2[] = "ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ";
    int x;

    printf("%s\n", line1);
    x = strlen(line1);
    starbar(x);
    printf("%s\n", line2);
    x = strlen(line2);
    starbar(x);
}

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

Τώρα η συνάρτηση **starbar** είναι πιο γενική. Δεν τυπώνει πάντα 50 αστεράκια, αλλά όσα θελήσουμε. Και πάλι έχει ένα μειονέκτημα: τυπώνει μόνο αστεράκια! Εάν θέλουμε να τυπώσουμε δολάρια, δεν μπορεί να μας βοηθήσει...

Σύνδεση συνάρτησης με το υπόλοιπο πρόγραμμα



Σε κάθε συνάρτηση ορίζεται μία λίστα από παραμέτρους: **τα ορίσματα της**. Η κάθε παράμετρος έχει συγκεκριμένο τύπο. Όταν καλούμε την συνάρτηση γίνονται τα ακόλουθα:

- Υπολογίζονται οι τιμές των ορισμάτων της.
- Εκτελείται ο κώδικας της συνάρτησης. Μέσα στην συνάρτηση μπορούν να χρησιμοποιούνται τα ορίσματα σαν να ήταν τοπικές μεταβλητές.
- Τελειώνοντας την εκτέλεση της, η συνάρτηση επιστρέφει μια τιμή συγκεκριμένου τύπου στη συνάρτηση που την κάλεσε. Υπάρχει η ειδική περίπτωση, μια συνάρτηση να μην επιστρέφει καμία τιμή. Η συνάρτηση αυτή λέμε ότι είναι **void**.

Σημείωση: προκειμένου να μπορέσω να χρησιμοποιήσω σωστά μια συνάρτηση, θα πρέπει να έχω σαφείς οδηγίες και να έχω κατανοήσει το πώς αυτή λειτουργεί. Δεν θα πρέπει να κοιτάζω τον κώδικα της για να την χρησιμοποιήσω. Θα πρέπει να έχω στο μυαλό μου ότι κάθε συνάρτηση είναι ένα εξειδικευμένο εργαλείο το οποίο μου έχουν πει πώς λειτουργεί, και ευθύνη μου είναι να το χρησιμοποιήσω σωστά.

Ο ορισμός μιας συνάρτησης γίνεται ως εξής:

```
<Τύπος> <όνομα συνάρτησης> (<ορίσματα>)  
{  
    <εντολές συνάρτησης>  
}
```

όπου :

<Τύπος> είναι ο τύπος της τιμής που επιστρέφει ή void στην περίπτωση που δεν επιστρέφει καμία τιμή.

<ορίσματα> είναι μία λίστα της μορφής:

```
<τύπος ορίσματος1> <όνομα ορίσματος1>,  
<τύπος ορίσματος2> <όνομα ορίσματος2>,  
...  
<τύπος ορίσματοςN> <όνομα ορίσματοςN>
```

Παραδείγματα ορισμού συναρτήσεων:

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

```
int my_sum(int n1, int n2)  
{  
    int k;  
    k = n1 + n2;  
    return k;  
}
```

Επιστροφή τιμής από συνάρτηση: Εντολή return

```
#include <stdio.h>

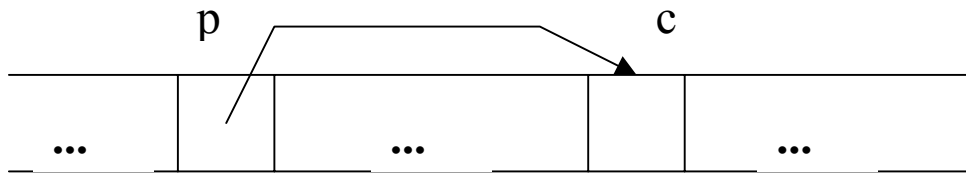
int max(int x, int y)
{
    int meg;

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

void main()
{
    printf("%d\n", max(10, 18));
}
```

Δείκτες (pointers)

Δείκτης είναι μία μεταβλητή που έχει ως τιμή της την διεύθυνση μίας μεταβλητής.



π.χ. **c** μεταβλητή τύπου **char**
 p δείκτης σε **char**

& δίδει την διεύθυνση ενός αντικειμένου

`p = &c`

Η `p` λέμε ότι δείχνει στην `c`.

* Τελεστής έμμεσης αναφοράς

Όταν εφαρμόζεται σε δείκτη, προσπελάζει το αντικείμενο στο οποίο δείχνει ο δείκτης.

```
int x = 1, y = 2;
```

```
int *ip;
```

```
ip = &x;
```

```
y = *ip;
```

```
*ip = 0;
```

Αν ο `ip` δείχνει την
ακέραια μεταβλητή `x`,
τότε η `*ip` μπορεί να
εμφανίζεται σε
οποιοδήποτε σημείο θα
μπορούσε να εμφανιστεί
και η `x`

Κάθε δείκτης δείχνει ένα συγκεκριμένο τύπο δεδομένων. Αυτός καθορίζεται κατά την δήλωση του δείκτη.

Πώς δηλώνουμε έναν δείκτη:

<τύπος δεδομένων> *<όνομα μεταβλητής>;

π.χ.

int *ip; ip είναι δείκτης σε int

char *pc;

float *pf;

Ο <τύπος δεδομένων> είναι ο τύπος δεδομένων στον οποίο δείχνει ο δείκτης.

Παράδειγματα:

int num, val, *ptr;

num = 22;

ptr = #

val = *ptr;

int ak = 10, *ip;

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

/* 10 1245052 */

ip = &ak;

***ip = 12;**

printf("%d %d\n", ak, *ip);

/* 12 12 */

Πέρασμα τιμών σε συνάρτηση

- Όλα τα ορίσματα των συναρτήσεων μεταβιβάζονται "κατ' αξία" (by value).
- Τα ορίσματα είναι τοπικές μεταβλητές που παίρνουν τις αρχικές τιμές τους κατά την κλήση της συνάρτησης.

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

```
void pass(int, int);
```

```
void main()
```

```
{  
    int x =4, y =7;  
    printf("main 1:%d, %d\n", x, y);  
    pass(x, y);  
    printf("main 2:%d, %d\n", x, y);  
}
```

```
void pass(int xx, int yy)
```

```
{  
    printf("pass1:%d, %d\n", xx, yy);  
    xx = 12;  
    yy = 2 * xx;  
    printf("pass2:%d, %d\n", xx, yy);  
}
```

Καλώντας την συνάρτηση pass, ΔΕΝ συνδέω την μεταβλητή x με την μεταβλητή xx.

Απλά υπολογίζω την τιμή της έκφρασης x την στιγμή της κλήσης της συνάρτησης (στο παράδειγμα μας την τιμή 4) και αυτήν την τιμή θέτω ως αρχική τιμή του ορίσματος xx στην pass.

Θέλω να επιλύσω το εξής πρόβλημα: Να ορίσω την συνάρτηση **interchange** η οποία να δέχεται ως παραμέτρους δύο μεταβλητές τύπου **int**, και να αντιμετωπίζει τις τιμές τους.

Εάν π.χ. η μεταβλητή *x* έχει την τιμή 5 και η *y* την τιμή 10, μετά την κλήση της συνάρτησης:

interchange(x, y);

θα ήλελα η μεταβλητή *x* να έχει την τιμή 10 και η μεταβλητή *y* την τιμή 5.

Πρώτη προσπάθεια:

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

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

```
void 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;
}
```

Μάλλον που δεν τα καταφέραμε!!!

Δεύτερη προσπάθεια:

```
#include <stdio.h>

void interchange(int*, int*);

void 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 *pu, int *pv)
{
    int temp;
    temp = *pu;
    *pu = *pv;
    *pv = temp;
}
```

Περνάω στην συνάρτηση την διεύθυνση των μεταβλητών του main. Άρα μέσα από την συνάρτηση interchange αποκτώ έμμεσα πρόσβαση σ' αυτές και μπορώ συνεπώς και να μεταβάλω τις τιμές τους.

Δεν δουλεύω με αντίγραφα των τιμών των μεταβλητών όπως έκανα στην προηγούμενη προσπάθεια, αλλά με τις θέσεις μνήμης των ίδιων των μεταβλητών.

Δείκτες σε πίνακες

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

π.χ.

```
int pin[4]= {10, 20, 30, 40};
```

Ο `pin` είναι δείκτης που δείχνει στον πρώτο στοιχείο του πίνακα, συνεπώς `*pin` έχει την τιμή 10.

`pin+1` δείχνει το επόμενο στοιχείο του πίνακα

`pin+i` δείχνει το **`i`** στοιχείο του πίνακα

`*(pin+1)` είναι ισοδύναμο με **`pin[1]`**

`*(pin+i)` είναι ισοδύναμο με **`pin[i]`**

Εάν `pa` είναι δείκτης σε `int` (έχει δηλωθεί: **`int *pa`**) τότε τα ακόλουθα είναι ισοδύναμα:

➤ **`pa = &pin[0]`**

➤ **`pa = pin;`**

Όταν προσθέτουμε 1 σε ένα δείκτη, εννοούμε μία μονάδα αποθήκευσης του τύπου στον οποίο δείχνει ο δείκτης και όχι π.χ. ένα byte.

Αν ο `pa` δείχνει σε μια θέση ενός πίνακα ακεραίων, και προσθέσω 1 στον `pa`, τότε ο `pa` θα δείχνει στην επόμενη θέση του πίνακα.

Παράδειγμα δεικτών σε πίνακα:

```
#include <stdio.h>
#define N 5

void main()
{
    int pin[N], i, *pi;

    for (i=0; i<N; i++)
        pin[i] = i*2;

    pi = pin;

    for (i=0; i<N; i++){
        printf("%d ", *pi);
        printf("%d ", *(pin+i) );
        printf("%d\n", pin[i] );
        pi++;
    }
}
```

Πίνακες σαν ορίσματα συναρτήσεων

Η C δεν επιτρέπει να περνούν πίνακες σαν ορίσματα συναρτήσεων.

Αντιμετωπίζω το πρόβλημα περνώντας έναν δείκτη στο πρώτο στοιχείο του πίνακα, και σε ένα δεύτερο όρισμα το μέγεθος του πίνακα.

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

```
long sump(int*, int);
```

```
void main()
```

```
{
    int pin[]={2,4,-1, 5}, i;
    long result;

    result = sump(pin, 4);
    printf("Sum= %d\n", result);
    for (i=0; i<4; i++)
        printf("%d%d\n", i, sump(pin+i, 4-i));
    printf("%d\n", sump(pin+2, 4) );
}
```

```
long sump(int* ar, int n)
```

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

Κατηγορίες μνήμης

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

α) Αυτόματες μεταβλητές

- Δηλώνονται μέσα στην συνάρτηση
- Έχουν τοπική εμβέλεια: είναι γνωστές μόνο μέσα στη συνάρτηση στην οποία ορίζονται.
- Άλλες συναρτήσεις μπορούν να χρησιμοποιούν μεταβλητές με το ίδιο όνομα: αυτές είναι ανεξάρτητες και αποθηκεύονται σε διαφορετικές θέσεις μνήμης.
- Διάρκεια ζωής τους:
 - αρχίζει μόλις κληθεί η συνάρτηση
 - χάνονται μόλις επιστρέψει τον έλεγχο
- Εάν δεν δώσουμε αρχικές τιμές έχουν 'σκουπίδια'. Πρέπει εμείς να φροντίζουμε στην απόδοση αρχικών τιμών.

β) Εξωτερικές μεταβλητές

- Ορίζονται έξω από τις συναρτήσεις
- Μια εξωτερική μεταβλητή θα πρέπει να δηλωθεί και στην συνάρτηση που την καλεί.

```
int err;
```

```
main()
```

```
{  
    extern int err;  
    ...  
}
```

- Οι εξωτερικές μεταβλητές υπάρχουν όσο υπάρχει το πρόγραμμα.
- Αν δεν τους αποδώσουμε αρχική τιμή με τον ορισμό τους, παίρνουν αυτόματα ως αρχική τιμή την τιμή μηδέν.

γ) Στατικές μεταβλητές

- Έχουν ίδια εμβέλεια με τις αυτόματες αλλά οι τιμές τους ΔΕΝ χάνονται από την μία κλήση της συνάρτησης μέχρι την επόμενη.
- Αν δεν τους αποδώσουμε αρχική τιμή, παίρνουν τιμή μηδέν.

Παράδειγμα στατικών μεταβλητών:

```
#include <stdio.h>
void trystat();
void main()
{
    int count;
    for (count=1; count <= 3; count++){
        printf("loop %d:", count);
        trystat();
    }
}

void trystat()
{
    int fade = 1;
    static int stay = 1;
    printf("fade= %5d, stay %5d\n",
           fade++, stay++);
}
/*
loop 1: fade =      1, stay =      1
loop 1: fade =      1, stay =      2
loop 1: fade =      1, stay =      3  */
```


Αναδρομικότητα συναρτήσεων (recursion)

Η κλήση μίας συνάρτησης από τον εαυτό της.

Πολλές φορές ο αλγόριθμος λύσης ενός προβλήματος μπορεί να περιγραφεί απλούστερα με αναδρομή.

Π.χ. θέλουμε να αθροίσουμε τους αριθμούς από το 1 έως το 100:

- πρόσθεσε το 100 συν το άθροισμα μέχρι το 99.
- Για να αθροίσω τους αριθμούς από το 1 έως το 99, πρόσθεσε το 99 στο άθροισμα 1 έως 98.
- ...
- Συνέχισε αυτήν την διαδικασία μέχρι να φτάσεις στο 1, οπότε το άθροισμα είναι ίσο με 1.

Η λύση του προβλήματος στην C με χρήση αναδρομής:

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

```
int my_sum(int) ;
```

```
void main()
```

```
{  
    printf("%d\n", my_sum(100) );  
}
```

```
int my_sum(int n)
```

```
{  
    if (n == 1)  
        return 1;  
    else  
        return n + my_sum(n-1) ;  
}
```

Ειδικά θέματα συμβολοσειρών

α) Απόδοση αρχικών τιμών σε συμβολοσειρές

Τα παρακάτω είναι ισοδύναμα:

```
char mat[] = "ENA";  
char mat[] = { 'E', 'N', 'A', '\0' };
```

Στα παραπάνω η **mat** είναι σταθερά δείκτη: η τιμή της ΔΕΝ αλλάζει.

Μια άλλη δυνατότητα:

```
char *ptr = "ENA";
```

Το **ptr** είναι μεταβλητή δείκτη. Αρχική τιμή της είναι η διεύθυνση της συμβολοσειράς **"ENA"**. Στην συνέχεια μέσα στο πρόγραμμα μας μπορούμε να αλλάξουμε την τιμή της.

Σημείωση: Ολόκληρη η φράση μέσα στα εισαγωγικά δρα σαν δείκτης της θέσης που αποθηκεύεται η συμβολοσειρά.

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

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

Δεν μπορώ όμως να γράψω το ακόλουθο:

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

β) Είσοδος / έξοδος συμβολοσειράς

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 διαβάσει συμβολοσειρές με κενά (η scanf ΔΕΝ μπορεί).
- Δέχεται σαν όρισμα δείκτη (π.χ. το όνομα της συμβολοσειράς).
- Η τιμή επιστροφής της gets() είναι η διεύθυνση της συμβολοσειράς που διάβασε.

Η συνάρτηση puts()

- Δέχεται ένα όρισμα που είναι δείκτης σε συμβολοσειρά.
- Εμφανίζει τη συμβολοσειρά αυτή στην οθόνη.
- Η puts σταματά να εμφανίζει χαρακτήρες όταν βρεί '\0'. Τότε κάνει την αντικατάσταση του '\0' με '\n' και στέλνει το αποτέλεσμα στην οθόνη.

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

```
void main()
```

```
{   char name[80];  
    printf("Δώσε το όνομα σου:");  
    gets(name);  
    printf("Χαιρετίσματα ");  
    puts(name);  
}
```

γ) Άλλες συναρτήσεις συμβολοσειρών

(συμπεριλαμβάνονται στην βιβλιοθήκη *string.h*)

strlen	Επιστρέφει το μήκος μίας συμβολοσειράς, χωρίς να συμπεριλαμβάνεται το '\0'
strcpy	Δέχεται δύο δείκτες συμβολοσειράς ως ορίσματα και αντιγράφει τα περιεχόμενα της δεύτερης συμβολοσειράς στην πρώτη. Επιστρέφει την τιμή του πρώτου ορίσματος. <i>Προσοχή: πρέπει να έχουμε μεριμνήσει ώστε ο πίνακας προορισμού να έχει αρκετό χώρο για την εισερχόμενη συμβολοσειρά.</i>
strcat	Δέχεται δύο δείκτες συμβολοσειράς ως ορίσματα. Η δεύτερη συμβολοσειρά αντιγράφεται στο τέλος της πρώτης. Η δεύτερη συμβολοσειρά δεν αλλάζει. Επιστρέφει την τιμή του πρώτου ορίσματος. <i>Προσοχή: Πρέπει να έχουμε μεριμνήσει ώστε ο πρώτος πίνακας να έχει αρκετό χώρο για να χωρέσει και ο δεύτερος.</i>
strcmp	Δέχεται δύο δείκτες συμβολοσειρές ως ορίσματα. Επιστρέφει: <ul style="list-style-type: none"> • 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.

Παραδείγματα χρήσης συναρτήσεων συμβολοσειράς

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[] = "ENA ΔΥΟ";
    char s2[] = "ΤΡΙΑ";
    char *ps;
    puts(s1);puts(s2);
    ps = strcpy(s1, s2);
    puts(s1); puts(s2); puts(ps);
}
/*
ENA ΔΥΟ
ΤΡΙΑ
ΤΡΙΑ
ΤΡΙΑ
ΤΡΙΑ
*/
```

```
void main()
{
    char s1[] = "ena";
    char s2[] = "dyo";
    char s[20], *ps;
    s[0] = '\0';
    ps = strcat(s, s1);
    ps = strcat(s, ", ");
    ps = strcat(s, s2);
    puts(s);
}
/* ena, dyo */
```

Ορισμός συναρτήσεων συμβολοσειρών

Για να χειριστεί μια συνάρτηση μια συμβολοσειρά χρειάζεται μόνο ένα όρισμα: έναν δείκτη στην αρχή της. Το τέλος της συμβολοσειράς μπορεί να το εντοπίσει η συνάρτηση ελέγχοντας για τον ειδικό χαρακτήρα '\0'.

Υπενθυμίζουμε ότι στο αντίστοιχο πρόβλημα για γενικούς πίνακες θέλαμε δύο παραμέτρους: τον δείκτη στον πίνακα, καθώς και το πλήθος των στοιχείων του πίνακα.

Παραδείγματα ορισμού συναρτήσεων:

```
int my_strlen(char *s)
{
    int n;
    for (n=0; *s != '\0'; s++)
        n++;
    return n;
}
```

```
int my_strlen(char *s)
{
    char *p = s;
    while ( *p != '\0')
        p++;
    return p - s;
}
```

```
int my_strcmp(char *s, char *t)
{
    int i;
    for (; *s == *t; s++, t++)
        if (*s == '\\0')
            return 0;
    return *s - *t;
}
```

```
void my_strcpy(char *s, char *t)
{
    while ((*s = *t) != '\\0') {
        s++;
        t++;
    }
}
```

```
void my_strcpy(char *s, char *t)
{
    while ((*s++ = *t++) != '\\0')
        ;
}
```

Παράδειγμα προγράμματος με συμβολοσειρές

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

void main()
{
    char name[] = "nikolaos", *s;

    s = name + 4;
    printf("%s \n", s);
    printf("MHKOS = %d", strlen(name));
    printf(", %d\n", strlen(s));
    name[3] = '\0';
    printf("%s\n", name);
}

/*
laos
MHKOS 8, 4
nik
*/
```

Προσέξτε πώς εκχωρώντας σε μία θέση του πίνακα name τον ειδικό χαρακτήρα '\0' (στο παράδειγμα μου name [3] = '\0') περιορίζω την συμβολοσειρά μου μέχρι την θέση αυτή!