

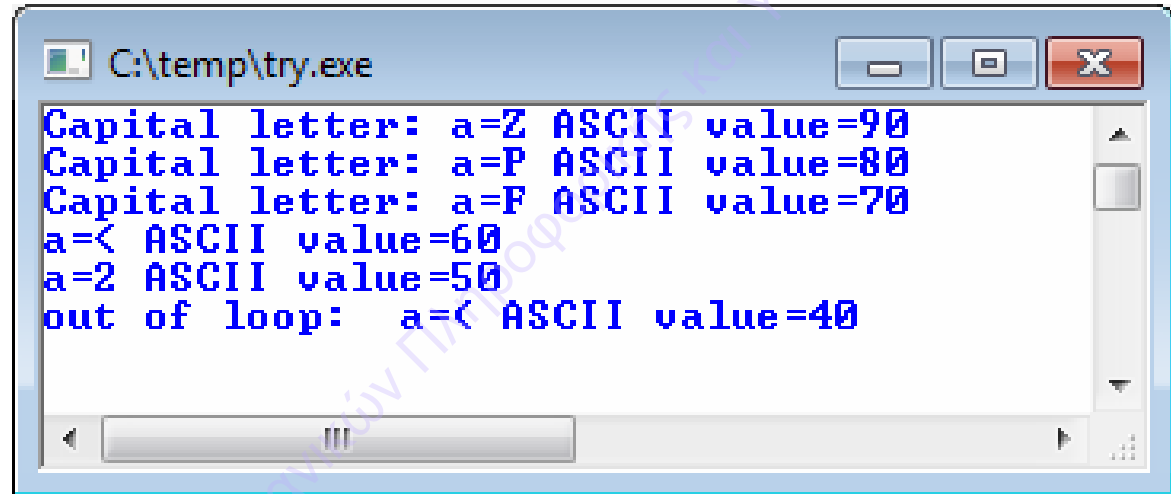
**Άσκηση:** Δίνονται οι παρακάτω δύο προτάσεις: **(α) while (++count<12) Π1** και **(β) while (count++<12) Π1**. Να περιγραφεί ο τρόπος με τον οποίο ο υπολογιστής τις εκτελεί, εντοπίζοντας τη διαφορά τους, εάν υπάρχει.

**Λύση:** Υπάρχει διαφορά μεταξύ των προτάσεων κι αυτή εντοπίζεται στον αριθμό επαναλήψεων. Η **(α)** χρησιμοποιεί την προθεματική σημειογραφία ενώ η **(β)** τη μεταθεματική. Στην **(α)** πρόταση αυξάνεται πρώτα η τιμή της **count** και η νέα τιμή της συγκρίνεται με το **12**, ενώ στη **(β)** πρώτα συγκρίνεται η τιμή της **count** με το **12** και στη συνέχεια αυξάνεται η τιμή της. Αυτό σημαίνει πως η πρόταση **Π1** θα εκτελεσθεί μία φορά παραπάνω στην περίπτωση **(β)**.

## Παράδειγμα 2:

```
#include <stdio.h>
int main ()
{
    char a='Z';

    while(a>40)
    {
        if (a>'A')
        { printf("Capital letter:\t"); }
        printf( "a=%c ASCII value=%d\n",a,a );
        a=a-10;
    }
    printf( "out of loop: a=%c ASCII value=%d\n",a,a );
    return 0;
}
```



```
C:\temp\try.exe
Capital letter: a=Z ASCII value=90
Capital letter: a=P ASCII value=80
Capital letter: a=F ASCII value=70
a=< ASCII value=60
a=2 ASCII value=50
out of loop: a=< ASCII value=40
```

**Παράδειγμα 3:** Να γραφεί πρόγραμμα που να διαβάζει μία σειρά χαρακτήρων από την είσοδο, να μετρά τα κενά και να τυπώνει τον αριθμό τους.

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

```
int main() {
```

```
    int num_spaces=0; char ch;
```

```
    printf( "Give a sentence\n" );
```

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

```
    while (ch!='\n')
```

```
    {
```

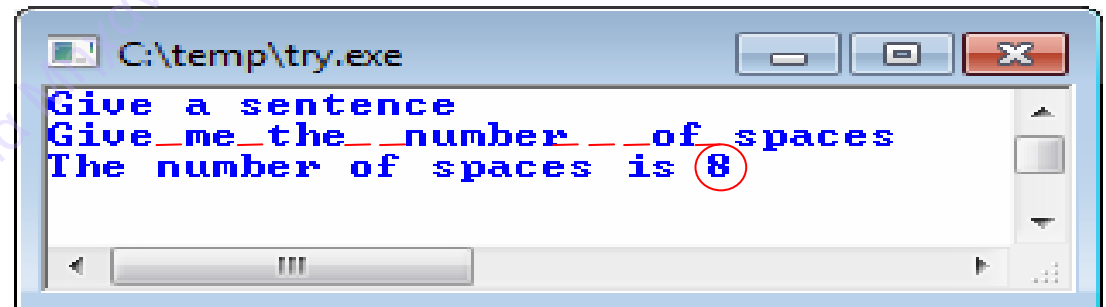
```
        if (ch == ' ') num_spaces++;
```

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

```
    }
```

```
    printf("The number of spaces is %d\n",num_spaces);
```

```
    return 0; }
```



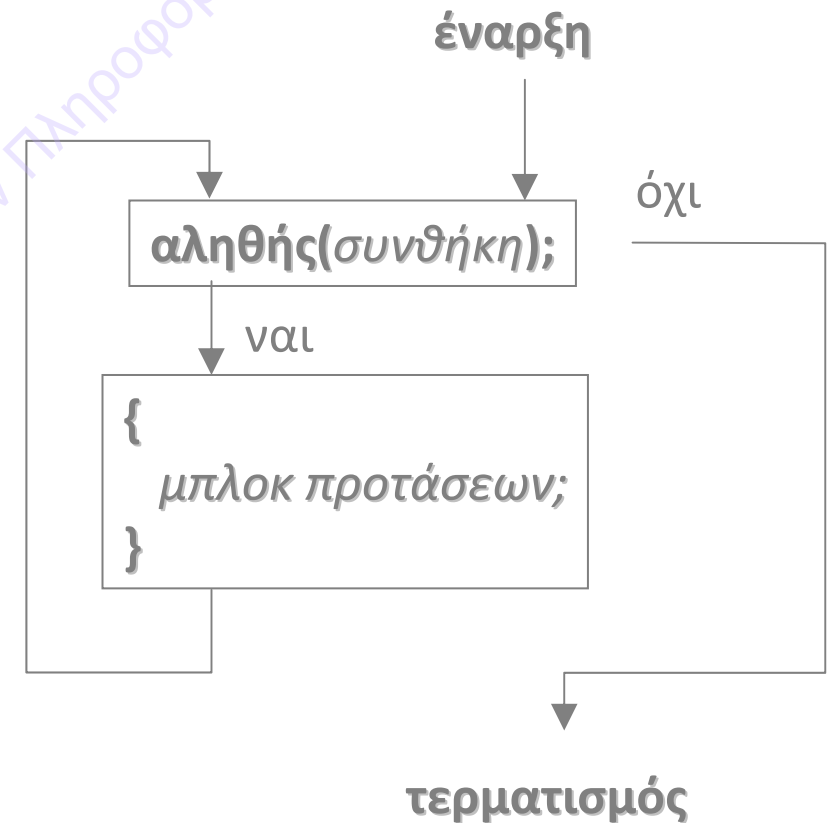
**Παρατήρηση:** Μία συμπαγής γραφή της ανάγνωσης χαρακτήρα και ελέγχου του περιεχομένου του – που χρησιμοποιείται ευρέως στη βιβλιογραφία – είναι η ενσωμάτωση της `getchar()` μέσα στη `while`. Μέσα στη συνθήκη ελέγχου πρώτα εκτελείται η πρόταση στα αριστερά και μετά ο έλεγχος. Σε αυτήν την περίπτωση προκύπτει ο ακόλουθος λειτουργικά ισοδύναμος κώδικας:

```
#include <stdio.h>
int main()
{
    int numSpaces=0;
    char ch;
    printf( "Give a sentence:\n" );
    while ( (ch=getchar()) != '\n' )
    {
        if (ch==' ') numSpaces++;
    }
    printf( "\n\nThe number of spaces is %d\n",numSpaces);

    return 0;
}
```

# Βρόχος με συνθήκη εισόδου στη C, οδηγούμενος από μετρητή: **for**

```
for (αρχική; συνθήκη; μετρητής)  
{  
    προτάσεις;  
}
```



## Βρόχος με συνθήκη εισόδου στη C, οδηγούμενος από μετρητή: *for*

Η λειτουργία της πρότασης επανάληψης *for* μπορεί να μορφοποιηθεί σε δομημένα Ελληνικά ως εξής:

Αρχικοποίησε τον μετρητή

Έλεγε τη συνθήκη

Εάν είναι αληθής

Εκτέλεσε τις προτάσεις

Ενημέρωσε τον μετρητή

Επάνελθε στον έλεγχο της συνθήκης

Αλλιώς ενημέρωσε τον μετρητή και σταμάτησε

## Βρόχος με συνθήκη εισόδου στη C, οδηγούμενος από μετρητή: *for*

Ο βρόχος *for* στη γλώσσα C παρέχει μεγάλη ευελιξία καθώς οι εκφράσεις μέσα στις παρενθέσεις μπορούν να έχουν πολλές παραλλαγές:

- Μπορεί να χρησιμοποιηθεί ο τελεστής μείωσης για μέτρηση προς τα κάτω:

```
for (n=10; n>0; n- -) printf( "n=%d\n",n );
```

- Το βήμα καθορίζεται από τον χρήστη:

```
for (n=0; n<60; n=n+13) printf( "n=%d\n",n );
```

- Ο μετρητής μπορεί να αυξάνει κατά γεωμετρική πρόοδο:

```
for (n=2; n<60.0; n=1.2*n) printf("n=%f\n",n );
```

## Βρόχος με συνθήκη εισόδου στη C, οδηγούμενος από μετρητή: *for*

- Χρησιμοποιώντας την ιδιότητα ότι κάθε χαρακτήρας του κώδικα ASCII έχει μία ακέραια τιμή, ο μετρητής μπορεί να είναι μεταβλητή χαρακτήρα. Το παρακάτω τμήμα κώδικα θα τυπώνει τους χαρακτήρες από το 'a' έως το 'z' μαζί με τον ASCII κωδικό τους:

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

- Θα πρέπει να σημειωθεί ότι εάν το σώμα του βρόχου αποτελείται από μία πρόταση, δεν απαιτούνται {}. Ωστόσο προτείνεται η χρήση των αγκίστρων σε κάθε περίπτωση, ανεξάρτητα από τον αριθμό των προτάσεων που απαρτίζουν το σώμα του βρόχου.



## Βρόχος με συνθήκη εισόδου στη C, οδηγούμενος από μετρητή: **for**

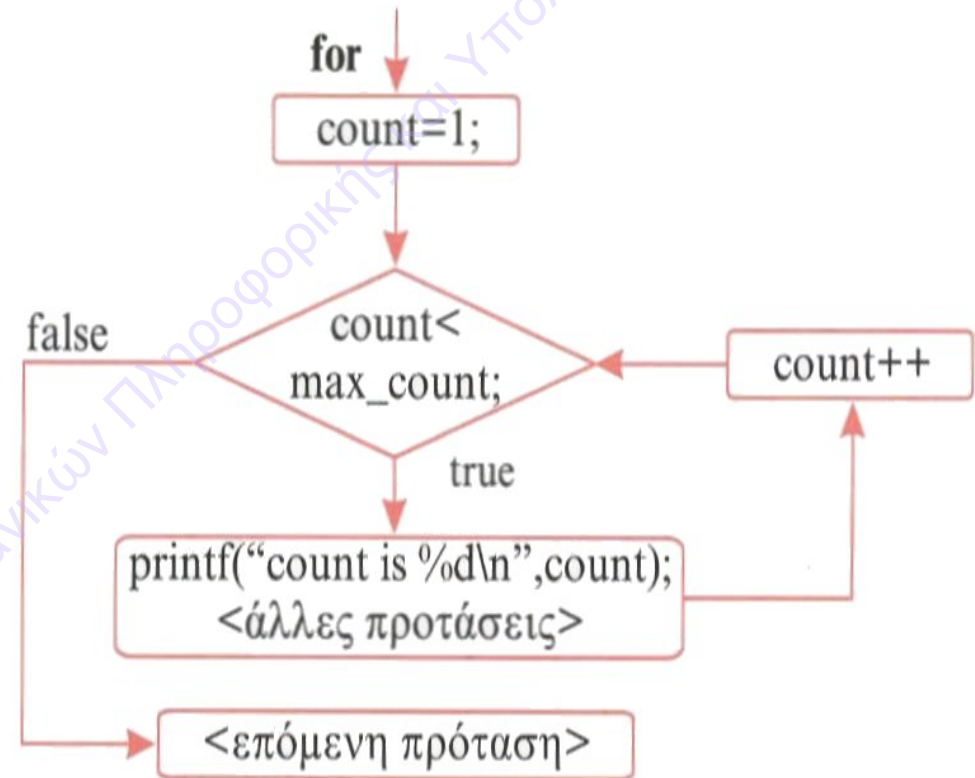
Στο πρότυπο C99 ο ο μετρητής μπορεί να δηλωθεί μέσα στη δήλωση της **for**. Ωστόσο απαιτείται ιδιαίτερη προσοχή γιατί μία τέτοια μεταβλητή δεν μπορεί να χρησιμοποιηθεί πουθενά αλλού παρά μόνο μέσα στο μπλοκ προτάσεων της **for**. Δηλαδή η ακόλουθη επαναληπτική πρόταση είναι σωστή:

```
for (int count=0; count<10; count++) printf(  
    "count=%d\n", count );
```

αλλά η μεταβλητή **count** δεν μπορεί να χρησιμοποιηθεί παρακάτω στο πρόγραμμα.

## Παράδειγμα:

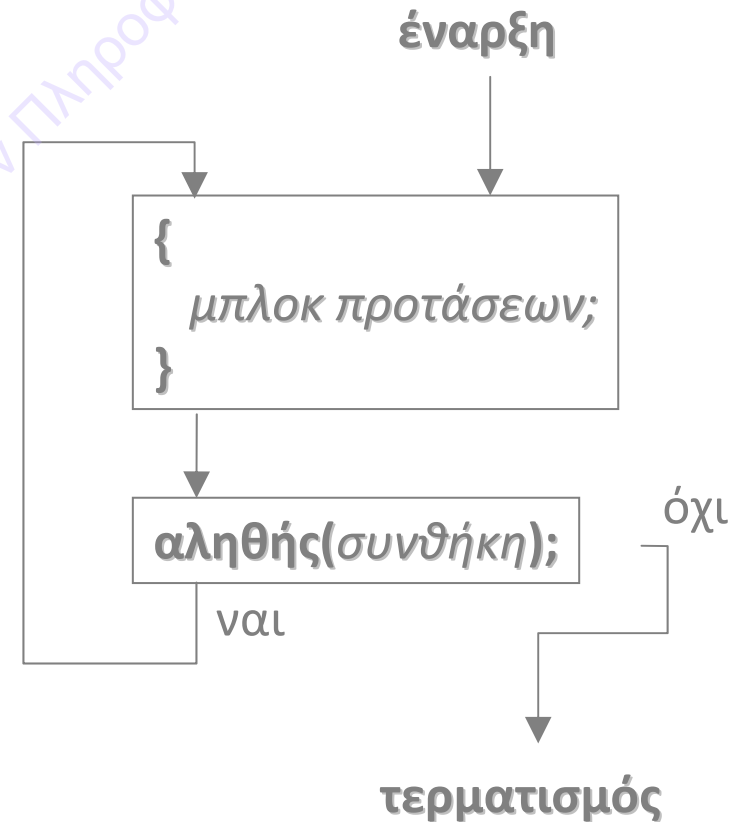
```
int count, max_count=30;  
for  
(count=1;count<max_count;count++)  
{  
    printf("count is %d\n",count);  
    <άλλες προτάσεις>;  
}  
<επόμενη πρόταση>;
```



## Βρόχος με συνθήκη εξόδου στη C, οδηγούμενος από γεγονός/μετρητή: **do{}while()**

```
do  
{  
    προτάσεις στις οποίες αλλάζει η  
    συνθήκη;  
}  
while (συνθήκη);
```

*Εναλλακτικά η while μπορεί να  
τοποθετηθεί αμέσως μετά το }*



## Βρόχος με συνθήκη εξόδου στη C, οδηγούμενος από γεγονός/μετρητή: `do{}while()`

Η λειτουργία της πρότασης επανάληψης `do-while` μπορεί να μορφοποιηθεί σε δομημένα Ελληνικά ως εξής:

**Εκτέλεσε τις προτάσεις**

**Έλεγε τη συνθήκη**

**Εάν είναι αληθής**

**Ξεκίνησε από την αρχή**

**Αλλιώς σταμάτησε**

## Παράδειγμα 1:

```
int count=30;
```

```
int limit=40;
```

```
do
```

```
{
```

```
    count++;
```

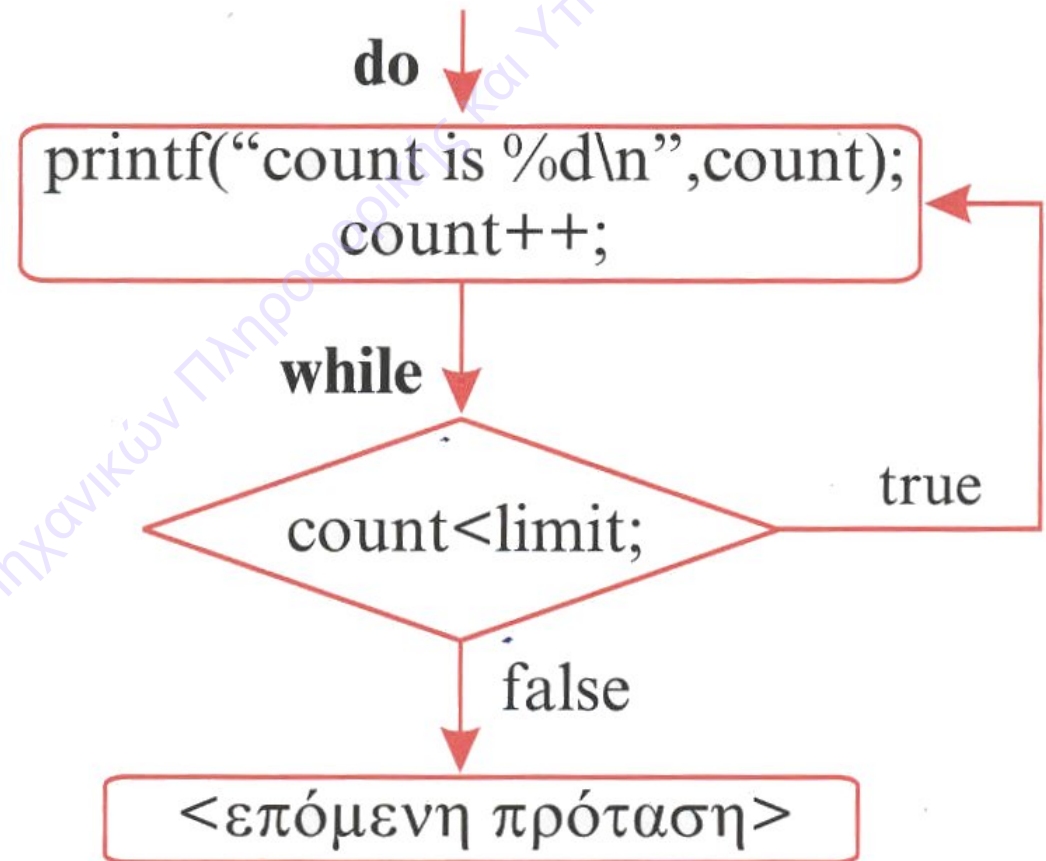
```
    printf("count is %d\n",count);
```

```
}
```

```
while (count<limit);
```

```
<επόμενη πρόταση>;
```

/\*Το σώμα του βρόχου εκτελείται μία φορά, έστω κι αν αρχικά count>=limit.\*/



## Παράδειγμα 2:

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

```
int main() {
```

```
    int iteration=0, count=516, limit=40;
```

```
    do {
```

```
        count++;
```

```
        iteration++;
```

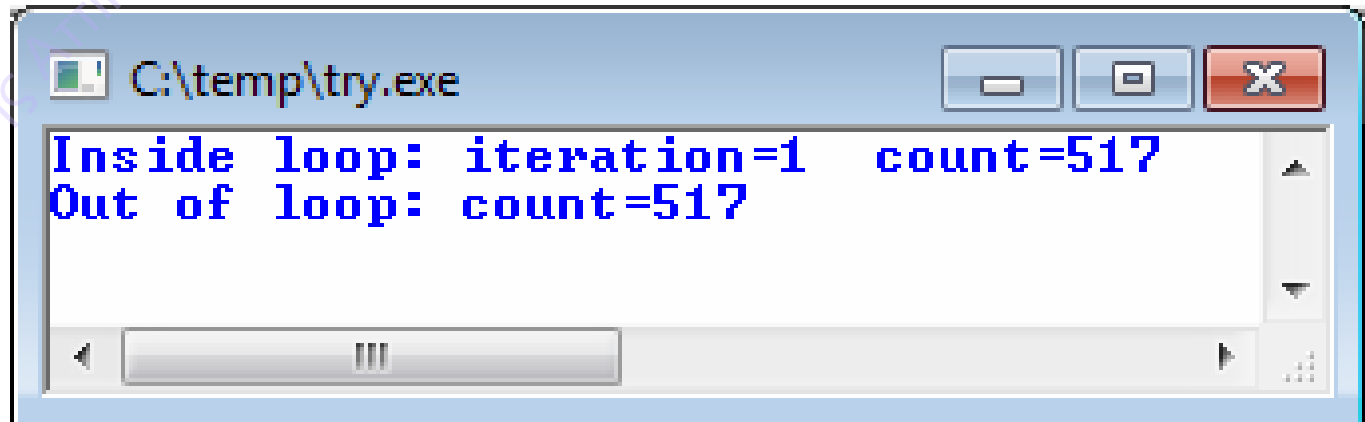
```
        printf("Inside loop: iteration=%d count=%d\n",iteration,count);
```

```
    } while (count<limit);
```

```
    printf("Out of loop: count=%d\n",count);
```

```
    return 0;
```

```
}
```



## Ένθετοι βρόχοι (nested loops)

Φώλιασμα (nesting): Τοποθέτηση ενός βρόχου μέσα σε άλλον. Ο εσωτερικός βρόχος είναι μία πρόταση μέσα στον εξωτερικό.

```
printf("\n");  
for (i=0; i<4; i++)  
{  
    for (j=0; j<3; j++)  
    {  
        printf(" %d.%d ", i, j);  
    }  
    printf("\n");  
}
```

### Αποτέλεσμα:

```
>  
0,0 0,1 0,2  
1,0 1,1 1,2  
2,0 2,1 2,2  
3,0 3,1 3,2  
>
```

**Παράδειγμα 1:** Στο πρόγραμμα που ακολουθεί δημιουργούνται ένθετοι βρόχοι, όπου χρησιμοποιούνται και τα τρία είδη προτάσεων επανάληψης:

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

```
int main() {
```

```
    int i,j;
```

```
    float x;
```

```
    for (i=0;i<=2;i++) {
```

```
        x=0.0;
```

```
        while( x<2) {
```

```
            printf( "\ni=%d x=%f",i,x );
```

```
            x++;
```

```
            do {
```

```
                printf( "\nGive an integer: " ); scanf( "%d",&j );
```

```
            } while (j<4);
```

```
        } // τέλος της while
```

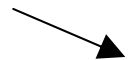
```
    } // τέλος της for
```

```
    return 0; } // τέλος της main
```

Εξωτερικός βρόχος

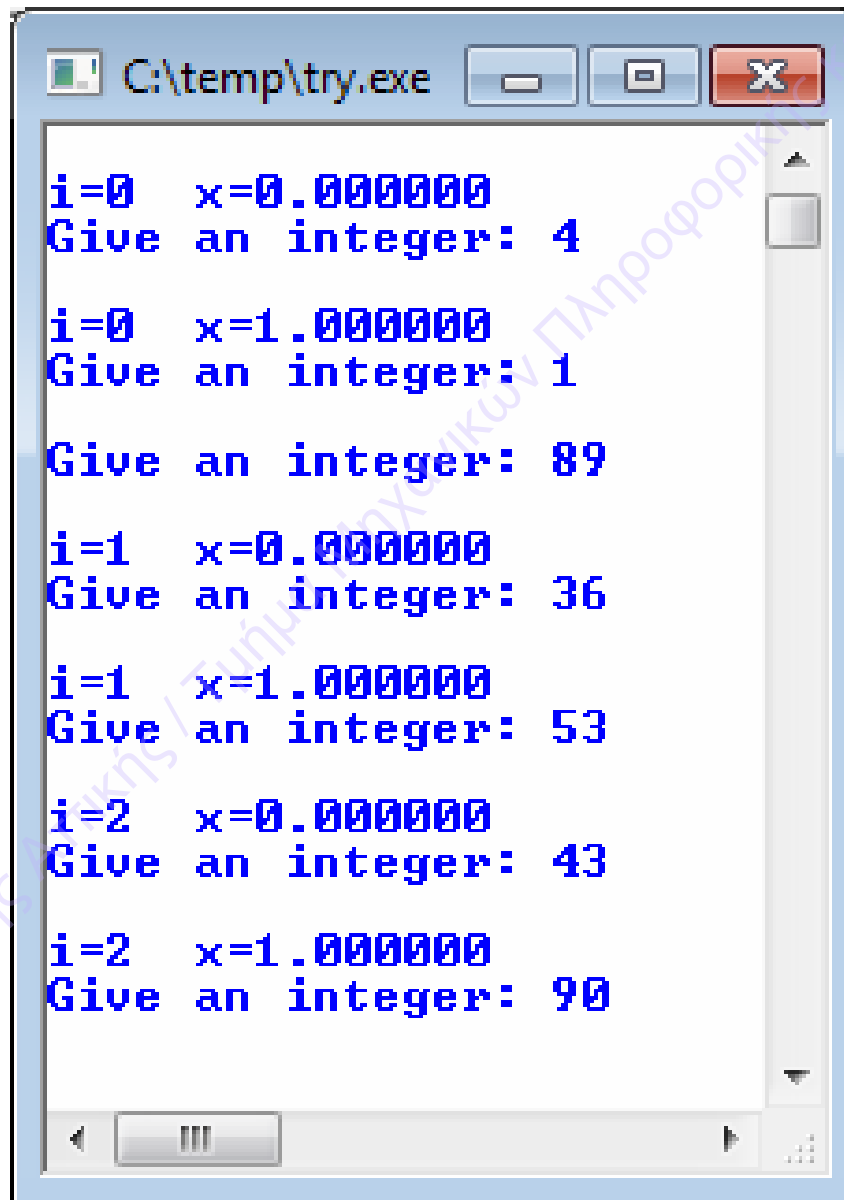
Πρώτο επίπεδο ένθεσης

Δεύτερο επίπεδο ένθεσης





## Αποτελέσματα:



```
i=0  x=0.000000
Give an integer: 4

i=0  x=1.000000
Give an integer: 1

Give an integer: 89

i=1  x=0.000000
Give an integer: 36

i=1  x=1.000000
Give an integer: 53

i=2  x=0.000000
Give an integer: 43

i=2  x=1.000000
Give an integer: 90
```

**Παράδειγμα 2:** Να καταστρωθεί πρόγραμμα που θα δέχεται διαδοχικά **N** αριθμούς, κάθε φορά θα ζητά εκ νέου τον αριθμό εφόσον αυτός δεν υπάγεται σε συγκεκριμένο και δοθέν διάστημα **[A,B]**. Αρχικά ο χρήστης θα δίνει έναν ακέραιο για εκθέτη δύναμης. Το πρόγραμμα θα υψώνει κάθε φορά το δοθέντα αριθμό στη δοθείσα δύναμη και θα τυπώνει το αποτέλεσμα.

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

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

```
#define N 3
```

```
#define A -2
```

```
#define B 6
```

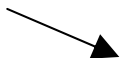
```
#define MY_ZERO 1E-6
```

```
int main() {
```

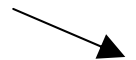
```
    int i,j,pow_coef;
```

```
    float x,y;
```

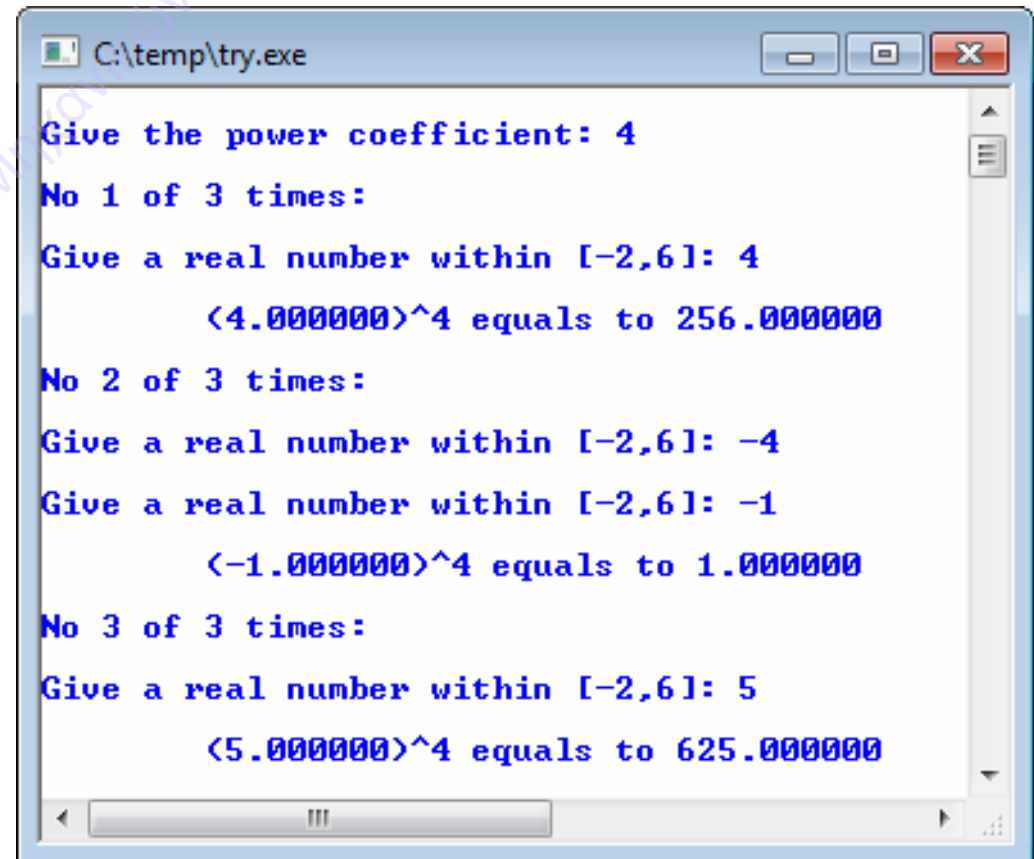
```
    printf("\nGive the power coefficient: "); scanf("%d",&pow_coef);
```



```
for (i=0;i<N;i++) {  
    printf("\nNo %d of %d times:\n",i+1,N);  
    do {  
        printf("\nGive a real number within [%d,%d]: ",A,B);  
        scanf("%f",&x);  
    } while ((x<A) || (x>B));  
    if (pow_coef == 0) y=1.0;  
    else if (fabs(x)<MY_ZERO) {  
        if (pow_coef>0) y=0.0;  
        else {  
            printf("\n\nERROR!! The result can't be defined\n\n");    getch();  
            break;  
        }  
    }  
    // end of internal ELSE-IF  
    else {  
        y=x;
```



```
for (j=1;j<=(abs(pow_coef)-1);j++) {  
    y=y*x;  
}  
if (pow_coef<0) y=1/y;  
} // end of external ELSE-IF  
printf("\n\t(%f)^%d equals to %f\n",x,pow_coef,y);  
} // end of FOR(i)  
return 0;  
}
```



```
C:\temp\try.exe  
Give the power coefficient: 4  
No 1 of 3 times:  
Give a real number within [-2,6]: 4  
      (4.000000)^4 equals to 256.000000  
No 2 of 3 times:  
Give a real number within [-2,6]: -4  
Give a real number within [-2,6]: -1  
      (-1.000000)^4 equals to 1.000000  
No 3 of 3 times:  
Give a real number within [-2,6]: 5  
      (5.000000)^4 equals to 625.000000
```

**Υλοποίηση του προηγούμενου παραδείγματος  
με ανάγνωση όλων των παραμέτρων κατά τον χρόνο εκτέλεσης**

**fully\_parameterized\_various\_nested\_loops.c**