

C: Από τη Θεωρία στην Εφαρμογή

Κεφάλαιο 6° Βρόχοι Επανάληψης

Ο βρόχος for

- Η εντολή **for** χρησιμοποιείται για τη δημιουργία **επαναληπτικών βρόχων** στη C
- Επαναληπτικός βρόχος καλείται το τμήμα του κώδικα μέσα σε ένα πρόγραμμα, το οποίο εκτελείται από την αρχή και επαναλαμβάνεται όσο μία συνθήκη παραμένει **αληθής (true)**
- Γενική σύνταξη της εντολής **for**:

```
for (αρχική_έκφραση; συνθήκη; τελική_έκφραση)
{
/* ομάδα εντολών (ή αλλιώς "σώμα" του βρόχου) που
   εκτελείται όσο η συνθήκη παραμένει αληθής. */
}
```

Τα βήματα εκτέλεσης της `for`

1. Εκτελείται η αρχική_έκφραση

- Η αρχική_έκφραση εκτελείται **μόνο μία φορά**, όταν αρχίζει η εκτέλεση της `for` εντολής και μπορεί να είναι οποιαδήποτε έγκυρη έκφραση της C
- Συνήθως, είναι μία εντολή εκχώρησης που αρχικοποιεί κάποια μεταβλητή, η οποία θα χρησιμοποιηθεί από τις άλλες δύο εκφράσεις

2. Γίνεται έλεγχος της τιμής της συνθήκης

- Η συνθήκη είναι συνήθως μία σχεσιακή έκφραση
- Αν είναι ψευδής, τότε ο `for` βρόχος **τερματίζεται** και η εκτέλεση του προγράμματος συνεχίζει με την **πρώτη εντολή** που υπάρχει **μετά το άγκιστρο κλεισίματος** της `for` εντολής
- Αν είναι αληθής, τότε εκτελείται η ομάδα των εντολών που ονομάζεται και «σώμα του βρόχου»

3. Εκτελείται η τελική_έκφραση

- Συνήθως, η τελική_έκφραση **αλλάζει** την τιμή κάποιας μεταβλητής που χρησιμοποιείται στη συνθήκη

4. Επαναλαμβάνονται συνεχώς τα βήματα (2) και (3), μέχρι η τιμή της συνθήκης να γίνει ψευδής

Παράδειγμα

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

    for (a = 0; a < 5; a++)
    {
        printf("%d ", a);
    }
    return 0;
}
```

Έξοδος: 0 1 2 3 4

Παρατηρήσεις (I)

- Όταν **γνωρίζουμε** εκ των προτέρων **τον αριθμό** των επαναλήψεων που επιθυμούμε να εκτελεστούν, τότε χρησιμοποιούμε συνήθως την εντολή **for** και όχι κάποια άλλη επαναληπτική μέθοδο
- Όπως και στην περίπτωση της **if-else** δομής, αν το μπλοκ εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν

Π.χ. το προηγούμενο παράδειγμα θα μπορούσε να γραφεί:

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

    for(a = 0; a < 5; a++)
        printf("%d ", a);
    return 0;
}
```

Παρατηρήσεις (II)



Μην βάζετε το ελληνικό ερωτηματικό ; στο τέλος της **for** εντολής, γιατί το ερωτηματικό θεωρείται ξεχωριστή πρόταση, η οποία σημαίνει ότι δεν υπάρχει ομάδα εντολών για εκτέλεση (**null statement**)

- Π.χ. η εντολή:

```
for (a = 0; a < 1000; a++) ;
```

αυξάνει την τιμή του **a** χίλιες φορές και δεν κάνει τίποτα άλλο

- Συνήθως, **for** βρόχοι με «κενή ομάδα εντολών» χρησιμοποιούνται σαν βρόχοι εισαγωγής χρονικής καθυστέρησης, δηλαδή «για να περάσει η ώρα» μέχρι να γίνει κάποια ενέργεια...
- Σε περίπτωση, πάντως, που όντως θέλετε να δημιουργήσετε έναν «κενό επαναληπτικό βρόχο», τοποθετήστε το ερωτηματικό στην επόμενη σειρά, έτσι ώστε να είναι ξεκάθαρο

Π.χ.

```
for (a = 0; a < 1000; a++)  
    ;
```

Παρατηρήσεις (III)

- Τα τμήματα της `for` εντολής, αρχική_έκφραση, συνθήκη και τελική_έκφραση μπορεί να αποτελούνται από μία μόνο εντολή, αλλά και από περισσότερες
- Στην περίπτωση που αποτελούνται από περισσότερες από μία εντολές, τότε αυτές χωρίζονται μεταξύ τους με τον τελεστή κόμμα (,).

■ Π.χ.:

```
for (i = 0, j = 10; i+j<15; i++, j--)
```

αρχική_έκφραση

συνθήκη

τελική_έκφραση

Παρατηρήσεις (III)

- Στη θέση των αρχική_έκφραση, συνθήκη και τελική_έκφραση μπορεί να μπει οποιαδήποτε έγκυρη έκφραση της C

Π.χ.

```
for (printf("Yes\n"); συνθήκη; τελική_έκφραση)
```

Με την παραπάνω εντολή, τυπώνεται στην οθόνη Yes και το πρόγραμμα συνεχίζει με τον έλεγχο της συνθήκης της **for**...

Παρατηρήσεις (IV)

- Σε μία **for** εντολή μπορεί να λείπουν κάποια από τα 3 τμήματά της ή ακόμη και όλα
- Π.χ. στην εντολή:

```
for(; a < 5; a++)
```

λείπει η αρχική_έκφραση

- Ωστόσο, το ελληνικό ερωτηματικό ; εξακολουθεί να υπάρχει και να λειτουργεί σαν διαχωριστικό μεταξύ των τμημάτων
- Στην εντολή:

```
for(;;)
```

λείπουν και τα 3 τμήματα

Παρατηρήσεις (V)

- Όταν σε μία `for` εντολή λείπει η συνθήκη ή η συνθήκη είναι πάντα αληθής, τότε αυτός ο `for` βρόχος ονομάζεται ατέρμονος βρόχος, γιατί δεν τερματίζεται ποτέ
- Π.χ. ο βρόχος:

```
for (a = 0; 0 < 1; a++)
```

είναι ατέρμονος, γιατί η συνθήκη $0 < 1$ είναι πάντα αληθής

- Επίσης, ο βρόχος:

```
for ( ; ; )
```

είναι και αυτός ατέρμονος, αφού λείπει η συνθήκη

Παρατηρήσεις (VI)

- Αν η συνθήκη είναι εξ'αρχής **ψευδής**, τότε δεν θα εκτελεστεί ποτέ το μπλοκ εντολών της **for**
- Π.χ. ο παρακάτω **for** βρόχος και το μπλοκ εντολών του δεν θα εκτελεστεί ποτέ, αφού η συνθήκη $a > 10$ είναι εξ'αρχής ψευδής (αφού η τιμή του a είναι 0)

```
for (a = 0; a > 10; a++)  
{  
    printf ("%d\n", a) ;  
    printf ("Yes\n") ;  
}
```

Παρατηρήσεις (VII)

- Αν αρχική_έκφραση και η τελική_έκφραση λείπουν, ο `for` βρόχος είναι ισοδύναμος με τον αντίστοιχο `while` βρόχο, όπως θα δούμε παρακάτω
- Π.χ. ο παρακάτω `for` βρόχος:

```
for (; a < 5;)
```

Είναι ισοδύναμος με τον `while` βρόχο:

```
while (a < 5)
```

Παραδείγματα (I)

- Γράψτε ένα πρόγραμμα που να εμφανίζει τους ακέραιους αριθμούς από το 1 έως το 10

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 1; i <= 10; i++)
        printf("%d\n",i);
    return 0;
}
```

- Γράψτε ένα πρόγραμμα που να εμφανίζει τους ακέραιους αριθμούς από το 1 έως το 10, αλλά με ανάποδη σειρά...

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 10; i >= 1; i--)
        printf("%d\n",i);
    return 0;
}
```

Παραδείγματα (II)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει 10 ακέραιους αριθμούς και να εμφανίζει κάθε φορά το τριπλάσιο του αριθμού, μόνο αν αυτός είναι μικρότερος του 10 ή μεγαλύτερος του 20

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

    for(i = 0; i < 10; i++)
    {
        printf("Enter number: ");
        scanf("%d",&num);

        if(num < 10 || num > 20)
            printf("Num = %d\n",num * 3);
    }
    return 0;
}
```

Παραδείγματα (III)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i;
    unsigned int j;

    for(i = 12; i > 2; i-=5)
        printf("%d ", i);
        printf("End = %d\n", i);

    for(j = 5; j >= 0; j--)
        printf("Test\n");
    return 0;
}
```

Έξοδος: 12 7 End = 2

Test

Test

Test... (ατέρμων βρόχος)

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 0; i < 3; i++)
        printf("%d ", i);
        printf("\nOne\n");
    return 0;
}
```

Έξοδος: 0 1 2

One

Παραδείγματα (V)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i = 20, j = 5;
    for (i = 0; i+j == 5; j++)
    {
        printf("One\n");
        i = 4;
        j = 1;
    }
    printf("Val1 = %d  Val2 = %d\n", i, j);
    return 0;
}
```

Έξοδος: One

Val1 = 4 Val2 = 2

Η εντολή break

- Η εντολή `break` χρησιμοποιείται για τον άμεσο τερματισμό ενός επαναληπτικού βρόχου (π.χ. `for`, `while` ή `do-while`) ή για τον τερματισμό μίας εντολής `switch`
- Στους επαναληπτικούς βρόχους, μετά την εκτέλεση της εντολής `break` το πρόγραμμα **συνεχίζει** με την **εκτέλεση της πρώτης** εντολής **μετά** τον βρόχο
- Ωστόσο, όπως θα δούμε στη συνέχεια, η εκτέλεση της εντολής `break` μέσα σε έναν **ένθετο** επαναληπτικό βρόχο προκαλεί τον τερματισμό μόνο του βρόχου στον οποίο η ίδια περιέχεται
- Επίσης, όπως είδαμε στην εντολή `switch`, η εκτέλεση της εντολής `break` μέσα σε μία `switch` προκαλεί επίσης τον άμεσο τερματισμό της λειτουργίας της

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            break;

        printf("%d ", i);
    }
    printf("\ni = %d\n", i);
    return 0;
}
```

Έξοδος: 1 2 3 4

i = 5

Η εντολή `continue`

- Η εντολή `continue` χρησιμοποιείται μέσα σε έναν επαναληπτικό βρόχο (π.χ. `for`, `while` ή `do-while`)
- Η εκτέλεση της εντολής `continue` μέσα σε έναν επαναληπτικό βρόχο προκαλεί την **άμεση διακοπή** της εκτέλεσης της ομάδας των εντολών της τρέχουσας επανάληψης και την **έναρξη** της επόμενης επανάληψης
- Άρα, οι εντολές ανάμεσα στην εντολή `continue` και στο τέλος του βρόχου **δεν εκτελούνται** για την τρέχουσα επανάληψη

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            continue;

        printf("%d ", i);
    }
    printf("\ni = %d\n", i);
    return 0;
}
```

Έξοδος: 1 2 3 4 6 7 8 9 10
i = 11

Ένθετοι for βρόχοι

- Ένας επαναληπτικός βρόχος (π.χ. `for`, `while` ή `do-while`) μπορεί να είναι **ένθετος** στο εσωτερικό κάποιου άλλου
- Π.χ. στην παρακάτω γενική περίπτωση, βλέπουμε δύο ένθετα `for`, στα οποία για να συμβεί μία επανάληψη του εξωτερικού βρόχου πρέπει πρώτα να τερματίσει η εκτέλεση του εσωτερικού βρόχου

Εξωτερικός `for` βρόχος

```
for (αρχική_έκφραση_1; συνθήκη_1; τελική_έκφραση_1)
{
    for (αρχική_έκφραση_2; συνθήκη_2; τελική_έκφραση_2)
    {
        /* ομάδα εντολών που θα εκτελείται συνεχώς
        όσο η συνθήκη_2 παραμένει αληθής. */
    }
    /* ομάδα εντολών που θα εκτελείται συνεχώς όσο η
    συνθήκη_1 παραμένει αληθής. */
}
```

Εσωτερικός
`for` βρόχος

Παραδείγματα (I)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i,j;
    for(i = 0; i < 2; i++)
    {
        printf("One ");
        for(j = 0; j < 2; j+=2)
            printf("Two ");
    }
    return 0;
}
```

Έξοδος: One Two One Two

Παραδείγματα (II)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i,j;
    for(i = 0; i < 2; i++)
    {
        for(j = 0; j < 3; j++)
        {
            printf("Two ");
            if(i+j == 1)
                break;
        }
        printf("One ");
    }
    printf("\nVal1 = %d  Val2 = %d\n",i,j);
    return 0;
}
```

Έξοδος: Two Two One Two One
Val1 = 2 Val2 = 0

Παραδείγματα (III)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i,j,k;

    k = 100;
    for(i = 0; i < 2; i++)
    {
        printf("One ");
        for(j = 0; k; j++)
        {
            printf("Two ");
            k -= 50;
        }
    }
    return 0;
}
```

Έξοδος: One Two Two One

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int i,j;
    for(i = 0; i < 5; i++)
    {
        for(j = 0; j <= i; j++)
            printf("* ");

        printf("\n");
    }
    return 0;
}
```

Έξοδος: *

```

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

Παραδείγματα (V)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάσει τους βαθμούς μίας ομάδας 5 φοιτητών σε 3 διαφορετικά μαθήματα και να εμφανίζει στην οθόνη τον μέσο όρο του κάθε φοιτητή στα 3 μαθήματα και τον συνολικό μέσο όρο της ομάδας σε όλα τα μαθήματα

```
#include <stdio.h>

#define LESSONS      3
#define STUDENTS     5

int main(void)
{
    int i,j;
    float grade,stud_grade,class_grade;

    class_grade = 0;
    for(i = 0; i < STUDENTS; i++)
    {
        printf("***** Student %d\n",i+1);

        stud_grade = 0; /* This variable holds the sum of all
grades in all courses of each student. For each student it is
initialized to zero. */
```

Παραδείγματα (V)

```
for(j = 0; j < LESSONS; j++)
{
    printf("Enter grade for lesson %d: ", j+1);
    scanf("%f", &grade);
    stud_grade += grade;
    class_grade += grade; /* Αυτή η μεταβλητή
    μετράει το συνολικό άθροισμα των βαθμών της ομάδας. */
} /* Τέλος της 2ης for */

printf("Average grade for student %d is %.2f\n",
       i+1, stud_grade / LESSONS);
} /* Τέλος της 1ης for */
printf("\n**** Average class grade is %.2f\n",
       class_grade / (STUDENTS * LESSONS));
return 0;
}
```

Ο βρόχος while

- Η εντολή **while**, όπως και η εντολή **for**, χρησιμοποιείται για τη δημιουργία **επαναληπτικών βρόχων** στη C
- Γενική σύνταξη της εντολής **while**:

```
while (συνθήκη)
{
    /* ομάδα εντολών που θα εκτελείται όσο η
       συνθήκη παραμένει αληθής. */
}
```

Εκτέλεση της εντολής `while`

1. **Γίνεται έλεγχος** της τιμής της συνθήκης (η οποία είναι συνήθως μία σχεσιακή έκφραση)
 - Αν η συνθήκη είναι **ψευδής (false)** τότε ο `while` βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της `while` εντολής
 - Αν η συνθήκη είναι **αληθής (true)** τότε εκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα `{ }` και η τιμή της συνθήκης ελέγχεται πάλι
 - Αν η τιμή της συνθήκης γίνει **ψευδής (false)**, τότε ο `while` βρόχος τερματίζεται
 - Αν όχι, **επανεκτελείται** η ομάδα των εντολών του βρόχου `while`

Η παραπάνω διαδικασία **επαναλαμβάνεται** μέχρι η τιμή της συνθήκης να γίνει ψευδής

Παρατηρήσεις (I)

- Η εντολή `while` χρησιμοποιείται συνήθως όταν **δεν γνωρίζουμε τον ακριβή αριθμό** των επαναλήψεων που θέλουμε να εκτελεστεί η ομάδα των εντολών μας
 - ♦ Όταν αντιθέτως **γνωρίζουμε** εκ των προτέρων **τον αριθμό** των επαναλήψεων που επιθυμούμε να εκτελεστούν, τότε συνήθως χρησιμοποιούμε την εντολή `for`
- Όπως και σε προηγούμενες περιπτώσεις (π.χ. εντολές `if-else`, `for`, κτλ), αν η ομάδα εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν



Μην βάζετε το ελληνικό ερωτηματικό ; στο τέλος της `while` εντολής, γιατί το ερωτηματικό θεωρείται ξεχωριστή πρόταση, η οποία σημαίνει ότι δεν υπάρχει ομάδα εντολών για εκτέλεση

Παρατηρήσεις (II)

- Η εντολή `while (x)` είναι ισοδύναμη με την `while (x != 0)`
- Προτείνεται ο δεύτερος τρόπος για να είναι πιο ευανάγνωστο το πρόγραμμα
- Αντίστοιχα, για τον ίδιο ακριβώς λόγο προτείνεται το `while (x == 0)` αντί του `while (!x)`
- Όταν σε μία `while` εντολή η συνθήκη είναι **πάντα αληθής**, τότε αυτός ο `while` βρόχος ονομάζεται **ατέρμονος βρόχος**, γιατί δεν τερματίζεται ποτέ
- Π.χ. ο βρόχος `while (1)` είναι ατέρμονος, γιατί η συνθήκη είναι πάντα αληθής, αφού το 1 είναι διαφορετικό από το 0

Παρατηρήσεις (III)

- Αν η συνθήκη είναι εξ'αρχής ψευδής, τότε δεν θα εκτελεστεί ποτέ το μπλοκ εντολών της **while**
- Π.χ.

```
int a = 10, b = 20;  
while (b < a)  
{  
    printf("%d\n", a);  
    printf("Yes\n");  
}
```

Παρατηρήσεις (IV)

- Οι εντολές `for` και `while` είναι πολύ στενά συνδεδεμένες
- Ουσιαστικά, οι παρακάτω εντολές είναι ισοδύναμες (εκτός αν ο `for` βρόχος περιέχει την εντολή `continue`) :

```
for (expr1; expr2; expr3)
{
    /* εντολές */
}
```

```
expr1;
while (expr2)
{
    /* εντολές */
    expr3;
}
```

Παραδείγματα (I)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει συνεχώς έναν ακέραιο αριθμό και να τον εμφανίζει μέχρι ο χρήστης να εισάγει το μηδέν (0). Το μηδέν (0) που θα εισάγει ο χρήστης να μην εμφανίζεται.

```
#include <stdio.h>
int main(void)
{
    int i = 1; /* Initialize with a nonzero value, just to be sure
that the loop will be executed. */
    while(i != 0)
    {
        printf("Enter number: ");
        scanf("%d", &i);
        if(i != 0)
            printf("Num = %d\n", i);
    }
    return 0;
}
```

Παραδείγματα (II)

- Πόσες φορές εκτελείται ο **while** βρόχος στο παρακάτω πρόγραμμα?

```
#include <stdio.h>
int main(void)
{
    int a = 256, b = 2;

    while(a != b)
    {
        b = b*b;
        a >>= 2;
    }
    return 0;
}
```

Απάντηση: 2 φορές

Παραδείγματα (III)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
    int dig = 0, a = 12345678;
    while(a > 0);
    {
        a /= 10;
        dig++;
    }
    printf("%d\n", dig);
    return 0;
}
```

Απάντηση: Ατέρμονος βρόχος...
(και όχι 8, που πιθανώς απαντήσατε)

Παραδείγματα (IV)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει συνεχώς ακέραιους αριθμούς μέχρι ο χρήστης να εισάγει το 0. Στο τέλος, το πρόγραμμα να εμφανίζει το πλήθος των θετικών και αρνητικών αριθμών που εισήγαγε ο χρήστης. Το μηδέν (0) να μην προσμετράται ούτε στους θετικούς ούτε στους αρνητικούς αριθμούς

```
#include <stdio.h>
int main(void)
{
    int i,pos,neg;

    pos = neg = 0;
    while(1)
    {
        printf("Enter number: ");
        scanf("%d",&i);

        if(i == 0)
            break;
        else if(i > 0)
            pos++;
        else
            neg++;
    }
    printf("Pos = %d Neg = %d\n",pos,neg);
    return 0;
}
```

Ο βρόχος do-while

- Η εντολή `do-while`, όπως και οι εντολές `for` και `while`, χρησιμοποιείται για τη δημιουργία επαναληπτικών βρόχων στη C
- Σε αντίθεση με τις εντολές `for` και `while` η συνθήκη στην εντολή `do-while` ελέγχεται αφού εκτελεστούν μία φορά οι εντολές του βρόχου
- Γενική σύνταξη της εντολής `do-while`:

```
do
{
/* ομάδα εντολών που εκτελείται αρχικά μία φορά
και στη συνέχεια κατ' επανάληψη όσο η συνθήκη
παραμένει αληθής. */
}while (συνθήκη) ;
```

while vs. do-while

```
while (not edge) {  
    run();  
}
```

```
do {  
    run();  
} while (not edge);
```



Τα βήματα εκτέλεσης της do-while

1. Εκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα { }
2. **Γίνεται έλεγχος** της τιμής της συνθήκης (η οποία είναι συνήθως μία σχεσιακή έκφραση)
 - Αν η συνθήκη είναι **ψευδής (false)** τότε ο **do-while** βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της **do-while** εντολής
 - Αν η συνθήκη είναι **αληθής (true)** τότε **επανεκτελείται** η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα { }
 - Το βήμα αυτό επαναλαμβάνεται μέχρι η τιμή της συνθήκης να γίνει ψευδής

Παρατηρήσεις (I)

- Ο βρόχος `do-while` χρησιμοποιείται πολύ λιγότερο από τους `for` και `while` βρόχους
- Όποιο πρόβλημα λύνεται με χρήση του βρόχου `do-while` θα μπορούσε να επιλυθεί και με χρήση βρόχων `while` ή `for`



Κάθε `do-while` βρόχος εκτελείται τουλάχιστον μία φορά, ακόμα κι αν η συνθήκη του βρόχου είναι ψευδής



Ο βρόχος `do-while` πρέπει να τελειώνει με το ελληνικό ερωτηματικό (;)

- Συνήθως, οι βρόχοι `do-while` χρησιμοποιούνται όταν θέλουμε να ελέγξουμε τις τιμές εισόδου σε ένα πρόγραμμα

Παρατηρήσεις (II)

- Σε περίπτωση που ο βρόχος `do-while` περιέχει μόνο μία εντολή, τα άγκιστρα μπορούν, και στην `do-while` εντολή, να παραλειφθούν
- Παρόλα αυτά, προτείνεται να χρησιμοποιείτε πάντοτε τα άγκιστρα σε εντολές `do-while`, διότι η παράλειψή τους πιθανότατα να παραπλανήσει τους αναγνώστες του προγράμματός σας (στο «κάτω» κομμάτι της εντολής), κάνοντάς τους να νομίζουν ότι χρησιμοποιείτε `while` εντολή

Π.χ.:

```
do
    printf("%d\n", i);
while(++i <= 10);
```

Παράδειγμα (I)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάσει έναν θετικό ακέραιο αριθμό και με χρήση της εντολής **do-while** να εμφανίζει τη λέξη **This** τόσες φορές στην οθόνη όσο και ο αριθμός που εισήγαγε ο χρήστης.

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

    printf("Enter number: ");
    scanf("%d", &num);
    i = 1;
    do
    {
        printf("This\n");
        i++;
    } while(i <= num);
    return 0;
}
```

- Τροποποιήστε το παραπάνω πρόγραμμα, ώστε να ζητά επίμονα την εισαγωγή θετικού αριθμού σε περίπτωση που ο χρήστης εισάγει μη θετικό ακέραιο (να γίνεται δηλαδή έλεγχος της τιμής εισαγωγής).

Παράδειγμα (II)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει έναν θετικό ακέραιο $N > 3$ και στη συνέχεια να υπολογίζει το αποτέλεσμα της παρακάτω μαθηματικής σχέσης. Το πρόγραμμα θα πρέπει να αναγκάζει τον χρήστη να εισάγει ακέραιο μεγαλύτερο του 3.

$$\Pi = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots - \frac{1}{N}$$

```
#include <stdio.h>
int main(void)
{
    int i, num;
    double a, val;

    do
    {
        printf("Enter number > 3: ");
        scanf("%d", &num);
    } while(num <= 3);

    val = 0;
    a = 1;
    for(i = 1; i <= num; i++)
    {
        val += a/i;
        a = -a;
    }
    printf("Val = %e\n", val);
    return 0;
}
```

Η εντολή goto

- Η εντολή `goto` χρησιμοποιείται με σκοπό να μεταφέρει την εκτέλεση του προγράμματος σε κάποια άλλη εντολή μέσα στην ίδια συνάρτηση, με την προϋπόθεση ότι η εντολή έχει μία ετικέτα
- Γενική σύνταξη της εντολής `goto`:

```
goto label;
```

- Όταν εκτελείται η εντολή `goto` η εκτέλεση του προγράμματος μεταβαίνει άμεσα στην εντολή που ακολουθεί τη θέση που έχει δηλωθεί με το όνομα `label`
- Η θέση με το όνομα `label` πρέπει να είναι **μοναδική** μέσα στη συνάρτηση και προσδιορίζεται γράφοντας **το όνομά της και άνω κάτω τελεία** :

Παράδειγμα

- Αν ο χρήστης εισάγει την τιμή -1 η εκτέλεση του προγράμματος μεταβαίνει στη θέση **START** και ο **for** βρόχος εκτελείται πάλι από την αρχή

```
#include <stdio.h>
int main(void)
{
    int i, num;
START:
    for(i = 0; i < 5; i++)
    {
        printf("Enter number: ");
        scanf("%d", &num);
        if(num == -1)
            goto START;
    }
    return 0;
}
```

Παρατηρήσεις (I)

- Γενικά, δεν προτείνεται η χρήση της εντολής `goto`, γιατί η μετάβαση της εκτέλεσης του προγράμματος από ένα σημείο σε κάποιο άλλο και μετά σε κάποιο άλλο, κ.ο.κ, οδηγεί σε δυσνόητο κώδικα που δεν είναι καλά οργανωμένος και άρα δύσκολα ελέγχεται
- Πολλοί μάλιστα είναι εντελώς αντίθετοι στη χρήση της `goto`, υποστηρίζοντας ότι δεν έχει καμία θέση μέσα σε ένα καλά δομημένο πρόγραμμα
- Ωστόσο, υπάρχουν περιπτώσεις που η `goto` μπορεί να φανεί χρήσιμη, όπως για την αποφυγή επανάληψης του ίδιου κώδικα με την ομαδοποίηση κοινών εντολών. Π.χ.

```
if(error_1)
    goto ERROR;

...
if(error_2)
    goto ERROR;

...
ERROR:
... // Εντολές διαχείρισης ανεπιθύμητων καταστάσεων.
```

Παρατηρήσεις (II)

- Σαν ένα ακόμα παράδειγμα, επειδή η **break** τερματίζει μόνο το **switch** ή τον βρόχο στον οποίο περιέχεται, η **goto** μπορεί να φανεί χρήσιμη για την άμεση έξοδο από ένα ένθετο **switch**, ή από ένα **switch** μέσα σε ένα βρόχο, ή από έναν ένθετο βρόχο. Π.χ.

```
for(i = 0; i < 10; i++)
    for(j = 0; j < 20; j++)
        for(k = 0; k < 30; k++)
        {
            if(συνθήκη)
                goto NEXT;
        }
NEXT:
...
```

- Αν και γενικά πρέπει να αποφεύγετε την χρήση της, να θυμάστε ότι κάποιες φορές μπορεί να φανεί χρήσιμη και να οδηγήσει σε απλούστερο και πιο ευανάγνωστο κώδικα. Άλλωστε, και οι εντολές **break**, **continue** και **return** (που χρησιμοποιούνται από τους «κατακριτές» της **goto**) δεν είναι τίποτα άλλο από παραλλαγές της **goto**