

Κεφάλαιο

8

Εντολές αλλαγής ροής και επανάληψης



Εντολές αλλαγής ροής και επανάληψης

Η "επικίνδυνη" εντολή `goto`

Στη C, όπως και στις περισσότερες γλώσσες προγραμματισμού, με την εντολή **goto** μεταφέρουμε τον έλεγχο του προγράμματος σε ένα διαφορετικό του σημείο. Δεδομένου ότι στη C οι προτάσεις ενός προγράμματος δεν αριθμούνται, χρησιμοποιούνται **ετικέτες** (labels) για να "επισημαίνουν" διάφορα σημεία μέσα σε ένα πρόγραμμα.

Μια **ετικέτα** (label) είναι ένα αναγνωριστικό ακολουθούμενο από μία άνω και κάτω τελεία (:), για παράδειγμα:

telos:

Μια ετικέτα μπορεί να βρίσκεται μόνη της σε μια γραμμή του προγράμματος ή να βρίσκεται μπροστά από μια πρόταση.

Η εντολή **goto** μεταφέρει τον έλεγχο σε μια ετικέτα του προγράμματος (αλλά πάντα μέσα στην ίδια συνάρτηση).

Το συντακτικό της `goto` είναι:

goto ετικέτα;

όπου **ετικέτα** είναι η ετικέτα στην οποία θέλουμε να μεταφερθεί ο έλεγχος του προγράμματος.

Το επόμενο πρόγραμμα εμφανίζει συνέχεια αριθμούς ξεκινώντας από το 1 και αυξάνοντάς τους κατά 1.

```
main()
{
    int a;
    a=0;
    pali:
    printf("%d\n", ++a);
    goto pali;
}
```

1
2
3
4

👉 Παρατηρούμε ότι η ετικέτα **pali** δεν ακολουθείται από ερωτηματικό (;). Όταν η ετικέτα βρίσκεται πριν από μία πρόταση, η πρόταση τερματίζεται κανονικά με ερωτηματικό (;). Παρατηρούμε ότι στην **goto** η ετικέτα δεν ακολουθείται από την άνω και κάτω τελεία (**goto pali;**)

Το επόμενο πρόγραμμα εκτελεί την ίδια λειτουργία με το προηγούμενο:

```
main()
{
    int a;
    a=0;
    pali: printf("%d\n", ++a);
    goto pali;
}
```

Η **goto** ως μέθοδος μεταφοράς ελέγχου του προγράμματος είναι ο χειρότερος εχθρός του δομημένου προγραμματισμού. Αφού λοιπόν έγινε κατανοητή η χρήση της **goto**, το επόμενο στάδιο είναι να την ξεχάσουμε. Αν θέλουμε τα προγράμματα μας να είναι δομημένα και ευανάγνωστα δεν πρέπει να τη χρησιμοποιούμε **ποτέ**. Η C διαθέτει εντολές και επαναληπτικές διαδικασίες που κάνουν περιττή τη χρήση της **goto**.

Ο βρόχος while

Η εντολή **while** (ή βρόχος **while**) επιτυγχάνει την επανάληψη μιας πρότασης (απλής ή σύνθετης) ενόσω μια λογική παράσταση είναι αληθής. Το συντακτικό της **while** είναι:

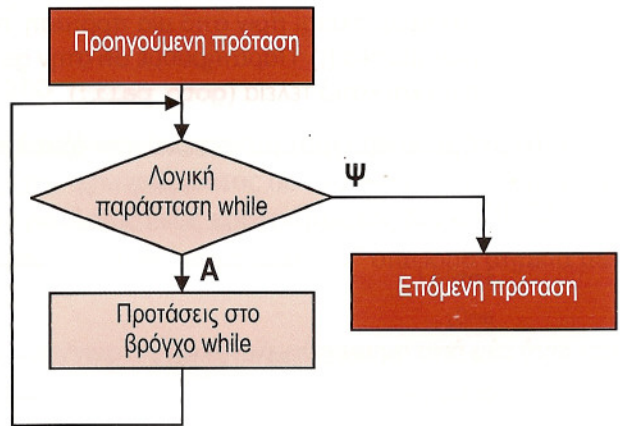
while (λογική.παράσταση) πρόταση; ή

while (λογική.παράσταση)
πρόταση; ή

```
while (λογική.παράσταση)
{
    πρόταση;
    πρόταση;
}
```


Το διπλανό λογικό διάγραμμα δείχνει τη φιλοσοφία της **while**.

Δεδομένου ότι μια λογική παράσταση στη C επιστρέφει αριθμητική τιμή, στη θέση της λογικής παράστασης μπορεί να είναι οποιαδήποτε παράσταση που επιστρέφει αριθμητική τιμή. Αν η τιμή της είναι 0, θεωρείται ψευδής· διαφορετικά θεωρείται αληθής.



Η παρακάτω εντολή **while** εμφανίζει συνέχεια την πρόταση "Η γλώσσα C σε βάθος", δεδομένου ότι η λογική παράσταση έχει συνέχεια τιμή 1 δηλαδή αληθής.

while (1)

```
printf("Η γλώσσα C σε βάθος\n");
```

Το παρακάτω πρόγραμμα εμφανίζει την τιμή της μεταβλητής **a**, ενόσω αυτή είναι μεγαλύτερη του μηδενός. Μετά το τέλος του βρόχου, εμφανίζει τη λέξη "ΤΕΛΟΣ".

```
main()
{
    int a,b;
    a=5;
    while (a>0)
    {
        printf("a=%d\n",a);
        --a;
    }
    printf("ΤΕΛΟΣ\n");
}
```

```
a=5
a=4
a=3
a=2
a=1
ΤΕΛΟΣ
```

Το επόμενο πρόγραμμα περιμένει να πατηθεί το πλήκτρο του διαστήματος (ASCII 32).

```
main()
{
    char ch;
    ch=0;
    while (ch!=32)
    {
        ch=getch();
    }
    printf("επιτέλους το πάτησες\n");
}
```

Ο βρόχος του **while** εκτελείται συνέχεια ενόσω το πλήκτρο που πατιέται δεν είναι το πλήκτρο του διαστήματος. Εκμεταλλευόμενοι τη μαγεία της C, θα μπορούσαμε να γράψουμε το προηγούμενο πρόγραμμα με τον επόμενο τρόπο.

```
main()
{
    char ch;
    ch=0;
    while ((ch=getch()) != 32);
    printf("επιτέλους το πάτησες\n");
}
```

Δεν υπάρχει πρόταση στον βρόχο while.

Η παράσταση **ch=getch()** περιμένει να πατηθεί ένας χαρακτήρας, τον αναθέτει στη μεταβλητή **ch**, και επιστρέφει ως τιμή την τιμή του **ch**. Η τιμή αυτή με τον τελεστή **!=** ελέγχεται αν είναι διάφορη του 32 (κωδικός του διαστήματος).

☞ Η συγκεκριμένη **while** δεν έχει καμία πρόταση να εκτελέσει. Απλώς ελέγχει συνεχώς τη λογική της παράσταση.

Ο βρόχος do-while

Ο βρόχος **do-while** κάνει παρόμοια δουλειά με τη **while**, με τη διαφορά ότι ελέγχει τη λογική παράσταση στο τέλος και όχι στην αρχή του βρόχου.

Το συντακτικό της **do-while** είναι:

do

πρόταση;

while (λογική.παράσταση);

ή

do

{

πρόταση;

πρόταση;

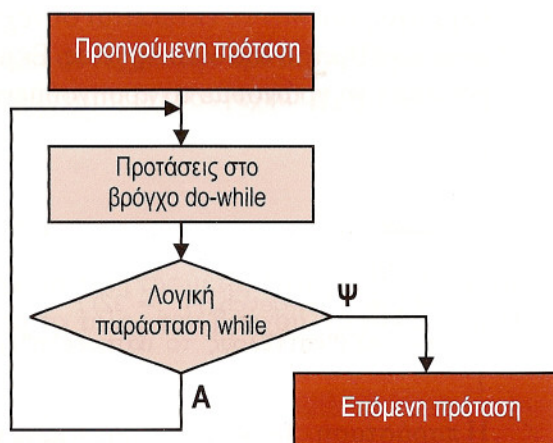
.....

} while (λογική.παράσταση);

☞ Αντίθετα με τη **while**, η **do-while** χρειάζεται ελληνικό ερωτηματικό (;) μετά από τη λογική παράσταση.

Το διπλανό λογικό διάγραμμα δείχνει τη φιλοσοφία της **do-while**.

☞ Παρατηρούμε ότι οι προτάσεις μέσα στο βρόχο του **do-while** θα εκτελεστούν τουλάχιστον μία φορά, ανεξάρτητα από την τιμή της λογικής παράστασης.



Ο βρόχος for

Η εντολή **for** είναι ίσως η εντολή που χρησιμοποιείται πιο συχνά για επαναληπτικές διαδικασίες. Ενώ σε άλλες γλώσσες προγραμματισμού η **for** έχει ένα "αυστηρό" συντακτικό, στη C το συντακτικό της, αν και σαφώς καθορισμένο, της προσφέρει μια μοναδική ευελιξία.

Το συντακτικό της **for** είναι:

for (εκτελέσιμη-πρόταση1;λογική παράσταση;εκτελέσιμη-πρόταση2)
πρόταση-βρόχου;

ή, για σύνθετη πρόταση βρόχου,

```
for (εκτελέσιμη-πρόταση1;λογική παράσταση;εκτελέσιμη-πρόταση2)
{
    πρόταση;
    πρόταση;
    .....
}
```

Η φιλοσοφία λειτουργίας της **for** γίνεται κατανοητή αν μελετήσουμε το διπλανό διάγραμμα ροής. Παρατηρούμε ότι η **πρόταση1** εκτελείται **μία μόνο** φορά στην αρχή του βρόχου.

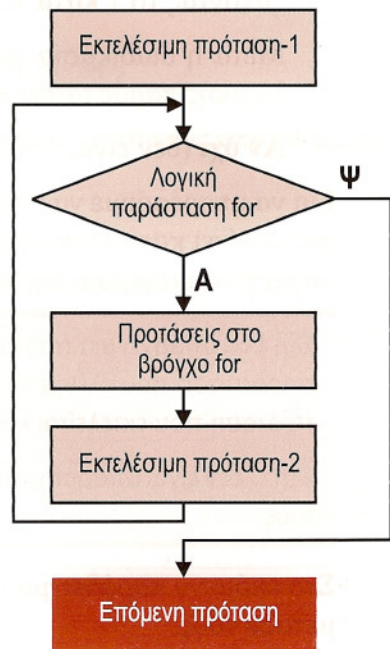
Μετά ελέγχεται η λογική παράσταση. Αν είναι αληθής, εκτελούνται μία φορά οι προτάσεις του βρόχου **for** και αμέσως μετά η **πρόταση2**. Μετά ελέγχεται πάλι η λογική παράσταση κ.ο.κ.

Η εκτέλεση των προτάσεων του βρόχου θα σταματήσει όταν η λογική παράσταση γίνει ψευδής και ο έλεγχος του προγράμματος μεταφέρεται στην επόμενη μετά από τη **for** πρόταση του προγράμματος.

👉 Προσοχή πρέπει να δοθεί στο γεγονός ότι η **πρόταση1**, η **λογική-παράσταση** και η **πρόταση2** βρίσκονται μέσα σε παρενθέσεις και χωρίζονται με ερωτηματικά (;)

Το επόμενο πρόγραμμα εμφανίζει τους αριθμούς από το 1 μέχρι το 8:

```
main()
{
    int i;
    for (i=1; i<=8 ; i=i+1)
```




```
{
    printf("i=%d\n",i);
}
```

- Στην αρχή, το **i** παίρνει τιμή 1 ($i=1$);
- Το πρόγραμμα ελέγχει αν $i \leq 8$.
- **Αν ναι** (είναι ≤ 8), εκτελεί τις εντολές του βρόχου μία φορά και μετά εκτελεί την πρόταση $i=i+1$ αυξάνοντας το **i** κατά 1.

Μετά η διαδικασία συνεχίζεται με τον έλεγχο της λογικής παράστασης και πάλι κ.ο.κ.

i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8

- **Αν όχι** (δεν είναι ≤ 8), ο βρόχος τερματίζεται.

Για να μπορέσουμε να εκμεταλλευτούμε την ευελιξία που προσφέρει η C πρέπει να γίνει κατανοητή η φιλοσοφία της **for** στη C. Ας συνοψίσουμε με απλά λόγια το συντακτικό της πριν προχωρήσουμε στα επόμενα παραδείγματα.

Στη **for** υπάρχει μια πρόταση που εκτελείται μόνο μία φορά στην αρχή, μία λογική παράσταση που ελέγχεται κάθε φορά πριν από την εκτέλεση του βρόχου και μία πρόταση που εκτελείται κάθε φορά μετά από την εκτέλεση του βρόχου.

Στη C δεν είναι απαραίτητο αυτές οι τρεις προτάσεις να έχουν κάποια σχέση μεταξύ τους.

Στο επόμενο παράδειγμα οι τρεις προτάσεις της **for** δεν έχουν κάποια σχέση μεταξύ τους:

```
for (a=2;b>4;c++)
{
    .....
    .....
}
```

- Αποθηκεύει στο **a** το 2.
- Ελέγχει αν το $b > 4$ και, αν είναι, εκτελεί τις εντολές του βρόχου.

- Κάθε φορά, μετά από την εκτέλεση του βρόχου αυξάνει το `c` κατά 1 (`c++`).
- Η εκτέλεση του βρόχου σταματάει όταν το `b` σταματήσει να είναι μεγαλύτερο από το 4.

Ο επόμενος βρόχος for δημιουργεί απλώς μια καθυστέρηση:

```
for (i=1; i<=1000; i++);
```

 Παρατηρούμε ότι δεν υπάρχουν προτάσεις στο σώμα του βρόχου.

Ο επόμενος βρόχος απλώς περιμένει να πατηθεί το A:

```
for (ch='*'; ch!='A'; ch=getch());
```

 Ούτε εδώ υπάρχουν προτάσεις στο σώμα του βρόχου.

Ο επόμενος βρόχος θα εμφανίσει στην οθόνη τους αριθμούς από το 1 μέχρι το 1000:

```
for (a=1; a<=1000; printf("%d\n", a++));
```

 Και πάλι δεν υπάρχουν προτάσεις στο σώμα του βρόχου.

Μετά το τέλος ενός βρόχου `for`, η μεταβλητή του βρόχου έχει τιμή την τελευταία της τιμή, για την οποία **δεν εκτελέστηκε** ο βρόχος, π.χ

```
for (i=1; i<=20; i=i+6)
{
    προτάσεις βρόχου;
}
printf("i=%d\n", i);
```

Μόλις τελειώσει ο βρόχος, το `i` θα έχει την τιμή 25 που είναι η επόμενη τιμή του `i` μετά το 19 για την οποία **δεν** έγινε επανάληψη του βρόχου.

Ένθετοι βρόχοι for

Ο βρόχος `for` μπορεί να περιέχει οποιαδήποτε πρόταση, επομένως και έναν άλλο βρόχο `for`.

Στην περίπτωση του επόμενου κώδικα, ο εσωτερικός βρόχος θα εκτελεστεί πέντε (5) φορές λόγω του εξωτερικού `for`. Έτσι η πρόταση `printf("%d %d\n", i, j);` θα εκτελεστεί δεκαπέντε φορές (5 x 3), ενώ η `printf("-----\n");` πέντε φορές επειδή βρίσκεται μόνο μέσα στον εξωτερικό βρόχο.

```
for (i=1;i<=5;i++)
{
    for (j=1;j<=3;j++)
    {
        printf("%d %d\n",i,j);
    }
    printf( "-----\n");
}
```

1	1
1	2
1	3

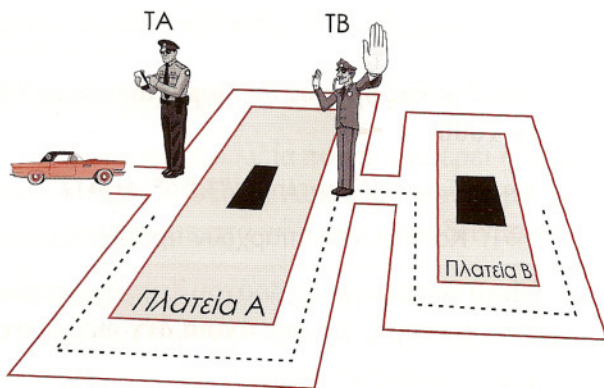
2	1
2	2
2	3

5	1
5	2
5	3

Ένας εποπτικός τρόπος για να γίνει κατανοητή η λειτουργία των ένθετων **for** φαίνεται στο επόμενο παράδειγμα:

Υποθέτουμε οι έχουμε δύο πλατείες, την πλατεία Α και την πλατεία Β. Στην είσοδο κάθε πλατείας υπάρχει ένας τροχονόμος (ΤΑ,ΤΒ).

Ο τροχονόμος ΤΑ υποχρεώνει τα αυτοκίνητα που μπαίνουν στην πλατεία Α να κάνουν τον γύρο της πέντε φορές ενώ ο τροχονόμος ΤΒ υποχρεώνει τα αυτοκίνητα που διέρχονται από την είσοδο της πλατείας Β να κάνουν τον γύρο της τρεις φορές.



Ας υποθέσουμε ότι ένα αυτοκίνητο μπαίνει στην πλατεία Α και περνάει από τον τροχονόμο ΤΑ ο οποίος σημειώνει πόσες φορές πέρασε το αυτοκίνητο (την πρώτη φορά σημειώνει τον αριθμό 1), μετά το αυτοκίνητο περνάει από την είσοδο της πλατείας Β όπου ο τροχονόμος ΤΒ το υποχρεώνει να κάνει το γύρο της πλατείας Β τρεις φορές. Βγαίνοντας από την πλατεία Β (αφού έχει κάνει τους τρεις γύρους) ξαναπερνά από τον ΤΑ (που σημειώνει τον αριθμό 2, δηλαδή ότι πέρασε για δεύτερη φορά) και ξαναμπαίνει στην πλατεία Β όπου ο ΤΒ τον υποχρεώνει να κάνει πάλι τρεις γύρους κ.ο.κ. Αυτό θα συνεχιστεί μέχρι να περάσει πέντε φορές από τον ΤΑ οπότε θα μπορεί να βγει από την πλατεία Α και να συνεχίσει το δρόμο του. Το αυτοκίνητο συνολικά θα έχει κάνει πέντε

φορές τον γύρο της πλατείας Α και δεκαπέντε φορές (3x5) τον γύρο της πλατείας Β.

Το τμήμα κώδικα που ακολουθεί εξηγεί τη χρήση των ένθετων βρόχων **for**.

👉 Αν κάποια πρόταση περικλείεται από πολλούς βρόχους, τότε θα εκτελεστεί τόσες φορές όσο το γινόμενο των εκτελέσεων κάθε βρόχου ξεχωριστά.

Π.χ. η πρόταση `printf("%d\n", l)` του βρόχου 4 θα εκτελεστεί 160 ($=5*8*4$) φορές επειδή περικλείεται από το βρόχο 1 (θα γίνει 5 φορές), το βρόχο 3 (θα γίνει 8 φορές), και το βρόχο 4 (θα γίνει 4 φορές).

<pre>printf("Αρχή"); for (i=1; i<=5; i++) { ❶ printf("*****"); for (j=1; j<=3; j++) ❷ printf("%d\n", j); for (k=1; k<=8; k++) { ❸ printf("Νίκος"); for (l=1; l<=4; l++) ❹ printf("%d\n", l); } printf("----"); }</pre>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">Εκτελείται μόνο μία φορά</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">Εκτελείται πέντε (5) φορές διότι βρίσκεται μόνο μέσα στον βρόχο 1.</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">Εκτελείται δεκαπέντε (15= 3x5) φορές διότι βρίσκεται μέσα στους βρόχους 1 και 2.</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">Εκτελείται σαράντα (40) φορές διότι βρίσκεται μέσα στους βρόχους 1 και 3 ($5*8=40$).</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">Εκτελείται 160 φορές διότι βρίσκεται στους βρόχους 1, 3, και 4 ($5*8*4=160$).</div> <div style="border: 1px solid black; padding: 5px;">Εκτελείται πέντε (5) φορές διότι βρίσκεται μόνο μέσα στον βρόχο 1.</div>
--	---

Η τελευταία πρόταση `printf("----")` θα εκτελεστεί πέντε φορές διότι βρίσκεται στο βρόχο 1.

Σκεφτείτε ποιες θα είναι οι τιμές των μεταβλητών **i**, **j**, **k**, και **l** μετά το τέλος του προηγούμενου κώδικα.

Ο τελεστής "κόμμα" (,)

Το κόμμα (,) ως τελεστής συνδέει πολλές παραστάσεις μαζί. Όλες οι παραστάσεις εκτελούνται αλλά το αποτέλεσμα ολόκληρης της πρότασης είναι το αποτέλεσμα **μόνο** της τελευταίας παράστασης, π.χ.

```
x = (y=3, y+1);
```

το **x** θα πάρει την τιμή 4.

Πρώτα θα εκτελεστεί η **y=3** και το **y** θα πάρει την τιμή 3, μετά θα εκτελεστεί η **y+1** και το αποτέλεσμα όλης της παράστασης (**y=3, y+1**) θα είναι η τιμή του **y+1** δηλαδή 4. Έτσι, το **x** θα πάρει τελικά την τιμή 4.

Στο επόμενο πρόγραμμα:

```
main()
{
    int y,x;
    char ch;
    x = (ch=getch(), ++ch, y=ch);
}
```

ας εξετάσουμε την πρόταση **x=(ch=getch(), ++ch, y=ch);**

Το πρόγραμμα:

- Θα περιμένει να πληκτρολογηθεί ένας χαρακτήρας, του οποίου ο κωδικός θα αποθηκευτεί στο **ch**. Έστω ότι πληκτρολογείται το 'A' οπότε στο **ch** θα καταχωριστεί το 65.
- Θα αυξήσει το περιεχόμενο του **ch** κατά 1, άρα θα γίνει 66.
- Θα βάλει στο **y** το 66 και όλη η πρόταση θα επιστρέψει ως τιμή την τιμή της τελευταίας παράστασης (**y=ch**), δηλαδή το 66.
- Στο **x** θα καταχωριστεί το 66.

Με απλά λόγια, η φιλοσοφία του τελεστή κόμμα (,) σε μια πρόταση είναι:

Κάνε αυτό και εκείνο και το άλλο, και επίστρεψε ως τιμή την τιμή της τελευταίας παράστασης.

Τα επόμενο παράδειγμα είναι χαρακτηριστικό για τη χρήση του τελεστή κόμμα (,):

```
for (i=1, j=9; i!=j; ++i, --j)
{
    printf("%d, %d\n", i, j);
}
printf("ΤΕΛΟΣ");
```

1,9
2,8
3,7
4,6
ΤΕΛΟΣ

Το πρόγραμμα θα εμφανίσει τα αποτελέσματα που βλέπετε στο πλαίσιο.

Η φράση `i=1, j=9` θα εκτελεστεί μία φορά και θα βάλει στη μεταβλητή `i` το 1 και στην `j` το 9.

Πριν από την εκτέλεση του βρόχου, το πρόγραμμα θα ελέγξει αν το `i` είναι διάφορο του `j`. Αν είναι, θα εκτελέσει τις εντολές του βρόχου και θα εμφανίσει τις τιμές των `i` και `j`.

Μετά από κάθε εκτέλεση του βρόχου, θα εκτελείται η φράση `++i, --j`, η οποία αυξάνει το `i` κατά 1 και μειώνει το `j` κατά 1.

Ο βρόχος θα σταματήσει να εκτελείται όταν το `i` γίνει ίσο με το `j`.

Η εντολή `break`

Η εντολή `break` έχει δύο χρήσεις:

- Τερματίζει μια πρόταση `case` στην εντολή `switch`.
- Εξαναγκάζει άμεσο τερματισμό ενός βρόχου `while`, `do-while`, ή `for`.

Στην παράγραφο αυτή θα αναφερθούμε στη δεύτερη χρήση της `break` αφού στον πρώτο τρόπο χρήσης της αναφερθήκαμε στην εντολή `switch`.

Τα επόμενα παραδείγματα δείχνουν παραστατικά τη λειτουργία της `break`. Αναφέρεται επίσης και ένα ισοδύναμο παράδειγμα, που χρησιμοποιεί την εντολή `goto` για την έξοδο από τον βρόχο. Η "επιστράτευση" της `goto` έγινε για να γίνει πιο κατανοητή η λειτουργία της `break` και όχι επειδή προτείνεται ως εναλλακτική λύση. Να υπενθυμίσουμε ότι η εντολή `goto` δεν πρέπει ποτέ να χρησιμοποιείται για την μεταφορά ελέγχου του προγράμματος σε ένα διαφορετικό του σημείο.

```
while (1)
```

```
{
    ch=getch();
    if (ch==27) break;
    putch(ch);
}
```

```
printf("Τέλος βρόχου while\n");
```

```
do
```

```
{
    ch=getch();
    if (ch==27) break;
    putch(ch);
}
```

```
while (1);
```

```
printf("Τέλος βρόχου do-while\n");
```

```
for (i=1;i<=100;++i)
```

```
{
    ch=getch();
    if (ch==27) break;
    putch(ch);
}
```

```
printf("Τέλος βρόχου for\n");
```

Μόλις πατηθεί το πλήκτρο Esc (ASCII 27) ο βρόχος σταματάει και εκτελείται η αμέσως επόμενη πρόταση

```
while(1)
```

```
{
    ch=getch();
    if (ch==27) goto edo;
    putch(ch);
}
```

```
edo:
```

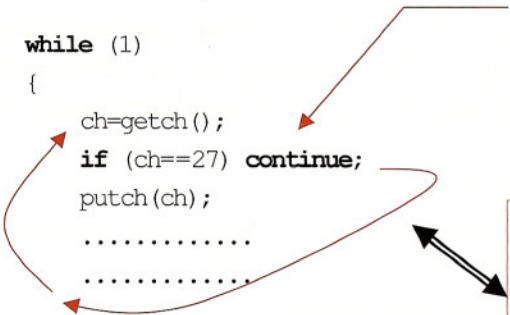
```
printf("Τέλος βρόχου while\n");
```


Η εντολή `continue`

Η εντολή `continue` έχει μια παρόμοια λειτουργία με την `break`. Η διαφορά της είναι ότι **δεν** τερματίζει τον βρόχο (όπως η `break`), αλλά επιβάλλει την εκτέλεση της επόμενης επανάληψης παραλείποντας τις ενδιάμεσες εντολές.

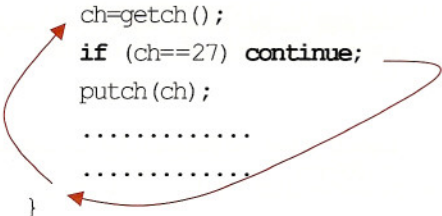
Τα επόμενα παραδείγματα δείχνουν παραστατικά τη λειτουργία της. Και εδώ αναφέρεται ένα ισοδύναμο παράδειγμα που χρησιμοποιεί την εντολή `goto` για την επανάληψη του βρόχου.

```
while (1)
{
    ch=getch();
    if (ch==27) continue;
    putchar(ch);
    .....
    .....
}
printf("Τέλος βρόχου while\n");
```

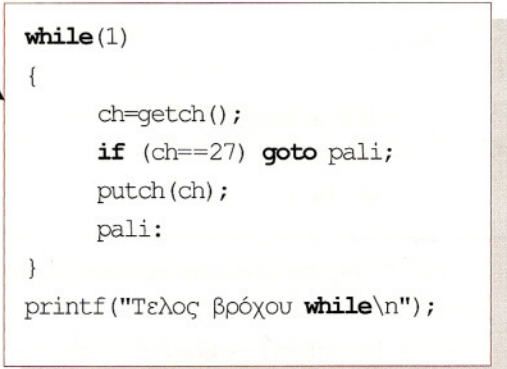


Η εντολή `continue` μεταφέρει τον έλεγχο του προγράμματος αμέσως μετά την τελευταία πρόταση του βρόχου αλλά πριν από το τέλος του (δεξιά άγκιστρο). Έτσι γίνεται κανονικά (αν πρέπει να γίνει) η επόμενη επανάληψη του βρόχου και απλώς παραλείπεται η εκτέλεση των προτάσεων από την εντολή `continue` μέχρι το τέλος του βρόχου.

```
do
{
    ch=getch();
    if (ch==27) continue;
    putchar(ch);
    .....
    .....
}
while (1);
printf("Τέλος βρόχου do-while\n");
```

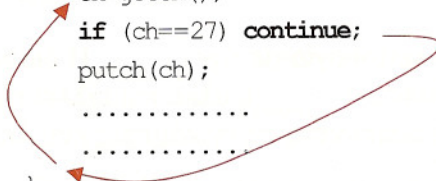


```
while(1)
{
    ch=getch();
    if (ch==27) goto pali;
    putchar(ch);
    pali:
}
printf("Τέλος βρόχου while\n");
```



```
for (i=1;i<=100;++i)
{
```

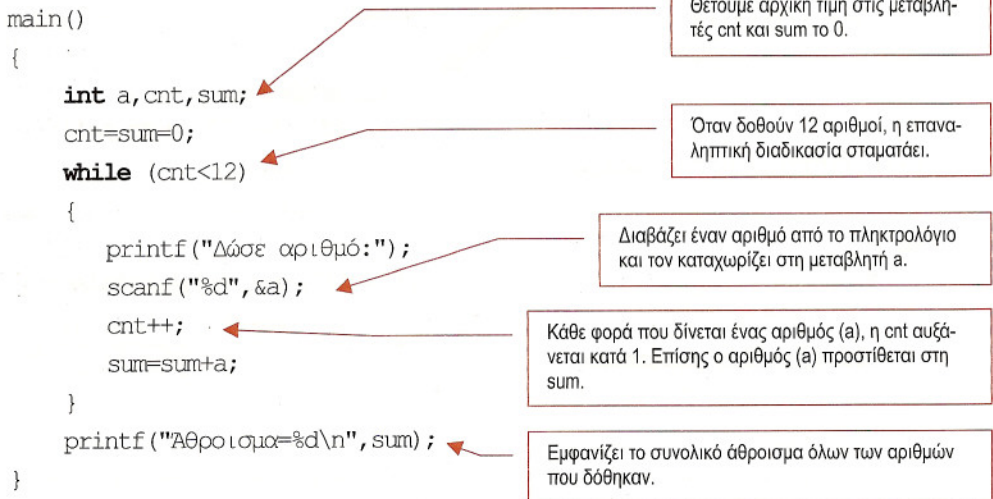
```
    ch=getch();  
    if (ch==27) continue;  
    putchar(ch);  
    .....  
    .....  
}  
printf("Τέλος βρόχου for\n");
```



Καταμέτρηση και άθροιση σε επαναλαμβανόμενες διαδικασίες

Το παρακάτω πρόγραμμα διαβάζει έναν-έναν δώδεκα αριθμούς που δίνονται από το πληκτρολόγιο και εμφανίζει το άθροισμά τους. Για την καταμέτρηση των αριθμών χρησιμοποιείται η μεταβλητή `cnt` και για το άθροισμά τους η μεταβλητή `sum`.

```
main()  
{  
    int a,cnt,sum;  
    cnt=sum=0;  
    while (cnt<12)  
    {  
        printf("Δώσε αριθμό:");  
        scanf("%d",&a);  
        cnt++;  
        sum=sum+a;  
    }  
    printf("Άθροισμα=%d\n",sum);  
}
```



Θέτουμε αρχική τιμή στις μεταβλητές `cnt` και `sum` το 0.

Όταν δοθούν 12 αριθμοί, η επαναληπτική διαδικασία σταματάει.

Διαβάζει έναν αριθμό από το πληκτρολόγιο και τον καταχωρίζει στη μεταβλητή `a`.

Κάθε φορά που δίνεται ένας αριθμός (`a`), η `cnt` αυξάνεται κατά 1. Επίσης ο αριθμός (`a`) προστίθεται στη `sum`.

Εμφανίζει το συνολικό άθροισμα όλων των αριθμών που δόθηκαν.

Η συνάρτηση `scanf()` διαβάζει έναν αριθμό από το πληκτρολόγιο και τον καταχωρίζει στη μεταβλητή `a`.

Ας υποθέσουμε ότι δίνουμε με τη σειρά τους αριθμούς 3,4,78,9 ... που βλέπουμε και στην πρώτη στήλη του διπλανού πίνακα.

Κάθε φορά που εκτελείται η πρόταση `cnt++`, αυξάνεται η τιμή της μεταβλητής `cnt` κατά 1. Έτσι, την πρώτη φορά που θα διαβάσει τον πρώτο αριθμό η `cnt` θα γίνει 1, όταν διαβάσει το 4 θα γίνει 2, με το 78 θα γίνει 3 κ.ο.κ. Όταν διαβάσει τον τελευταίο αριθμό, η `cnt` θα γίνει 12. Με λίγα λόγια, χρησιμοποιούμε τη μεταβλητή `cnt` για να μετρήσουμε πόσους αριθμούς διαβάσαμε μέχρι στιγμής.

a	cnt	sum
3	1	3
4	2	7
78	3	85
9	4	94
12	5	106
34	6	140
45	7	185
6	8	191
78	9	269
99	10	278
2	11	290
3	12	293

Με την πρόταση `sum=sum+a` κάθε φορά που διαβάζει το πρόγραμμα έναν αριθμό, τον προσθέτει στο μέχρι στιγμής περιεχόμενο της `sum`. Την πρώτη φορά (όταν η `a` περιέχει το 3) η `sum` θα γίνει 3, όταν το πρόγραμμα διαβάσει το 4 η `sum` θα γίνει 7 (3 που είχε από πριν + 4), όταν διαβάσει το 78 η `sum` θα γίνει 85 (7+78) κ.ο.κ. Όταν θα έχει διαβάσει τον τελευταίο αριθμό, η `sum` θα είναι 293 το οποίο είναι το άθροισμα όλων των αριθμών που δώσαμε μέχρι στιγμής. Με λίγα λόγια χρησιμοποιούμε τη μεταβλητή `sum` για να μετρήσουμε το άθροισμα των αριθμών που δώσαμε μέχρι στιγμής.

Όταν θέλουμε να **μετρήσουμε** πόσες φορές έγινε ένα συμβάν χρησιμοποιούμε μια μεταβλητή στην οποία δίνουμε αρχική τιμή 0 (π.χ. `cnt=0`) και κάθε φορά που γίνεται το συμβάν που θέλουμε να μετρήσουμε την αυξάνουμε κατά 1 (`cnt++`). Τη θέση αυτή την ονομάζουμε **μετρητή**.

Όταν θέλουμε να **αθροίσουμε** τις διαφορετικές τιμές που παίρνει μια μεταβλητή (π.χ. η `a`) χρησιμοποιούμε μια θέση μνήμης (την οποία καλούμε **αθροιστή**) στην οποία δίνουμε αρχική τιμή 0 (π.χ. `sum=0`) και κάθε φορά που η μεταβλητή παίρνει μια νέα τιμή αυξάνουμε τον αθροιστή κατά το περιεχόμενο της μεταβλητής (π.χ. `sum=sum+a`).

Παραδείγματα

- Π.1** Το επόμενο πρόγραμμα ζητάει συνέχεια ζεύγη αριθμών και υπολογίζει το μέσο όρο τους. Σταματάει όταν και οι δύο αριθμοί που δοθούν είναι 0.

```
main()
{
    int a,b;
    float mo;
    do
    {
        scanf("%d %d", &a, &b);
        mo=(a+b)/2.0;
        printf("MO=%f\n", mo);
    } while (a!=0 && b!=0);
}
```

- Π.2** Το επόμενο πρόγραμμα εμφανίζει την προπαίδεια.

```
main()
{
    int a,b;
    for(a=1;a<=9;a++)
        for(b=1;b<=10;b++)
            printf("%d x %d = %d\n", a,b,a*b);
}
```

```
1 x 1 = 1
1 x 2 = 2
.....
9 x 9 = 81
9 x 10 = 90
```

- Π.3** Το επόμενο πρόγραμμα ζητάει συνέχεια χαρακτήρες έναν-έναν και μετράει τα αστεράκια (*) που δίνουμε. Σταματάει μόλις δώσουμε το τρίτο αστεράκι (*).

```
main()
{
    int a=0;
    char ch;
    while(1)
```

```
{
    ch=getch();
    if(ch=='*') a++;
    if(a==3) break;
}
```

Π.4 Το επόμενο πρόγραμμα εμφανίζει την τελική τιμή του **k**, η οποία είναι 31.

```
main()
{
    int a,b,k=4;
    for(a=1;a<=9;a=a+3)
    {
        --k;
        for(b=1;b<=5;b++)
        {
            k=k+2;
        }
    }
    printf("k=%d\n",k);
}
```

Η εντολή αυτή θα εκτελεστεί 3 φορές (για a 1,4 και 7) και κάθε φορά αφαιρεί 1 από το k.

Η εντολή αυτή θα εκτελεστεί 15 (3x5) φορές (3 λόγω του εξωτερικού for και 5 λόγω του εσωτερικού for), αυξάνοντας κάθε φορά το k κατά 2.

Τελικά το k θα έχει τιμή 31 (4-3+30). 4 αρχική τιμή, μείον 3 λόγω της --k, συν 30 λόγω της k=k+2 που θα εκτελεστεί 15 φορές.

Π.5 Το επόμενο πρόγραμμα ζητάει έναν ακέραιο θετικό αριθμό και εμφανίζει ένα-ένα τα ψηφία του αντίστροφα. Έτσι, αν δώσουμε π.χ. το 5234 εμφανίζει το 4325.

```
main()
{
    int ar,y,p;
    printf("Δώσε αριθμό:");
    scanf("%d",&ar);
    do
    {
        y=ar % 10;
```

Ο αλγόριθμος που ακολουθούμε για το συγκεκριμένο πρόβλημα είναι ο ακόλουθος: Παίρνουμε το υπόλοιπο και το πηλίκο του αριθμού δια 10. Το υπόλοιπο αποτελεί το πρώτο ψηφίο. Αντικαθιστούμε τον αριθμό με το πηλίκο που βρήκαμε και συνεχίζουμε τη διαδικασία μέχρι να βρούμε πηλίκο 0.

```
        p=ar/10;  
        printf("%d",y);  
        ar=p;  
    } while (p!=0);  
}
```

Ανασκόπηση Κεφαλαίου 8

- Η εντολή **goto**, αν και υποστηρίζεται από τη C για τη μεταφορά της ροής του προγράμματος σε ένα διαφορετικό του σημείο, θα πρέπει να αποφεύγεται γιατί οδηγεί σε δυσνόητο και μη δομημένο κώδικα.
- Η εντολή **while** χρησιμοποιείται για την επαναλαμβανόμενη εκτέλεση προτάσεων ενόσω μια λογική συνθήκη είναι αληθής. Ο έλεγχος της συνθήκης γίνεται κάθε φορά πριν την εκτέλεση των προτάσεων.
- Η εντολή **do-while** χρησιμοποιείται για την επαναλαμβανόμενη εκτέλεση προτάσεων ενόσω μια συνθήκη είναι αληθής. Ο έλεγχος της συνθήκης γίνεται κάθε φορά μετά από την εκτέλεση των προτάσεων, οπότε οι προτάσεις είναι βέβαιο ότι θα εκτελεστούν τουλάχιστον μία φορά.
- Η εντολή **for** χρησιμοποιείται επίσης για την επαναλαμβανόμενη εκτέλεση προτάσεων. Το πλήθος των επαναλήψεων στη **for** συνήθως εξαρτάται από την τιμή μιας μεταβλητής.
- Η εντολή **break** τερματίζει ένα βρόχο **while**, **do-while** ή **for**.
- Η εντολή **continue** επιβάλλει την εκτέλεση της επόμενης επανάληψης σε ένα βρόχο **while**, **do-while** ή **for**, παρακάμπτοντας τις ενδιάμεσες προτάσεις.

Ασκήσεις Κεφαλαίου 8

- 8.1** Να γραφεί πρόγραμμα το οποίο να υπολογίζει το άθροισμα των αριθμών από το 1 μέχρι το 1000. ★★
- 8.2** Να γραφεί πρόγραμμα το οποίο να διαβάζει συνέχεια χαρακτήρες από το πληκτρολόγιο. Όταν πατηθεί το Esc, να σταματάει και να εμφανίζει το πλήθος των ελληνικών και το πλήθος των λατινικών χαρακτήρων που πληκτρολογήθηκαν. ★★
- 8.3** Τι κάνει το επόμενο πρόγραμμα (το 27 είναι ο κωδικός του πλήκτρου <Esc>); ★★

```
main()
{
    char ch;
    int a, fl=0;
    ch=1;
    a=0;
    while(ch!=27)
    {
        ch=getch();
        if(ch=='*') fl=1;
        if(fl==1) ++a;
    }
    printf("%d χαρακτήρες\n",a);
}
```

- 8.4** Τι κάνει το επόμενο πρόγραμμα; Ποιο το νόημα του περιεχομένου των μεταβλητών num1 και num2; ★★

```
main()
{
    int a,num,num1,num2;
    num1=num2=0;
    for (a=1;a<=100;++a)
```

```
{
    scanf("%d", &num);
    switch(num % 2)
    {
        case 0:
            ++num2;
            break;
        case 1:
            ++num1;
            break;
    }
}
printf("num1=%d\n num2=%d\n", num1, num2);
}
```

8.5 Να γραφεί πρόγραμμα το οποίο να κάνει τα εξής: ★★ ★

- Να διαβάζει συνέχεια χαρακτήρες από το πληκτρολόγιο.
- Να σταματάει μόλις πληκτρολογηθούν δυο αστεράκια (*).
- Να εμφανίζει πόσοι χαρακτήρες μεσολάβησαν μεταξύ του πρώτου και του δεύτερου αστεριού.

Για παράδειγμα. αν δοθούν οι χαρακτήρες `q w e r * s d f g e r *` θα σταματήσει (επειδή πληκτρολογήθηκαν δύο αστεράκια) και θα εμφανίσει τον αριθμό 6 (διότι μεταξύ τους μεσολαβούν έξι χαρακτήρες, οι `s d f g e r`).

8.6 Τι θα εμφανίσει το επόμενο πρόγραμμα; ★★

```
main()
{
    int i, j;
    for (i=1; i<=10; ++i)
    {
        for (j=1; j<i; ++j)
            printf("%d\n", j);
    }
}
```

8.7 Να γραφεί πρόγραμμα το οποίο θα εμφανίζει στην οθόνη όλους τους χαρακτήρες από το χαρακτήρα με κωδικό 32 μέχρι το χαρακτήρα με κωδικό 255. Τι μετατροπή πρέπει να γίνει στο πρόγραμμα αυτό ώστε να εμφανίζει σε κάθε γραμμή της οθόνης 30 χαρακτήρες; ★ ★ ★

8.8 Να γραφεί πρόγραμμα το οποίο θα υπολογίζει το άθροισμα των κλασμάτων $1/1 + 1/2 + 1/3 + 1/4 + 1/5 + \dots + 1/100$. ★ ★

8.9 Να γραφεί πρόγραμμα το οποίο θα ζητάει έναν ακέραιο θετικό αριθμό και θα εμφανίζει τον αντίστοιχο δυαδικό του αριθμό. Για παράδειγμα, αν του δώσουμε το 5234 να εμφανίσει τον 1010001110010. ★ ★ ★

8.10 Τι θα εμφανιστεί από το επόμενο πρόγραμμα; ★ ★

```
main()
{
    int i,j,k=4;
    for (i=1;i<=10;i=i+2)
    {
        k++;
        for (j=1;j<5;++j)
            k=k+2;
    }
    printf("k=%d\n",k);
}
```

8.11 Τι θα εμφανιστεί από το επόμενο πρόγραμμα; ★ ★

```
main()
{
    int i,j=6,k=4;
    i=(k=k+2,j=k*10);
    printf("i=%d j=%d k=%d\n",i,j,k);
}
```


- 8.12** Τι κάνει το επόμενο πρόγραμμα; Πότε θα σταματήσει; Ποιος θα είναι ο τελευταίος αριθμός που θα δούμε; Ποιους αριθμούς δεν θα δούμε ποτέ;

★

```
main()
{
    int a;
    do
    {
        a=rand();
        if(a>=100) continue;
        printf("%d\n",a);
    } while(a!=0);
}
```

- 8.13** Να γραφεί πρόγραμμα το οποίο θα ζητάει έναν ακέραιο θετικό αριθμό και θα εμφανίζει το άθροισμα των ψηφίων του. Για παράδειγμα, αν του δώσουμε το 5234, να εμφανίσει τον αριθμό 14 ($5+2+3+4$). ★★★

- 8.14** Να γραφεί πρόγραμμα το οποίο να εμφανίζει όλους τους αριθμούς από το 1 μέχρι το 100. Οι αριθμοί να εμφανίζονται ανά δεκάδες σε κάθε γραμμή της οθόνης. Η πρώτη γραμμή, π.χ., από το 1 μέχρι το 10, η δεύτερη από το 11 μέχρι το 20 κ.ο.κ. ★

- 8.15** Ποια από τα επόμενα αληθεύουν: ★

- ☐ Ο βρόχος **do-while** εκτελείται τουλάχιστον μία φορά.
- ☐ Στην εντολή **for** τα τρία τμήματά της πρέπει να σχετίζονται μεταξύ τους.
- ☐ Η εντολή **goto** οδηγεί σε μη δομημένα προγράμματα.
- ☐ Η πρόταση **while(1)** έχει αποτέλεσμα τη συνεχή εκτέλεση ενός βρόχου.
- ☐ Ο τελεστής κόμμα (,) επιστρέφει την τιμή της πρώτης παράστασης.