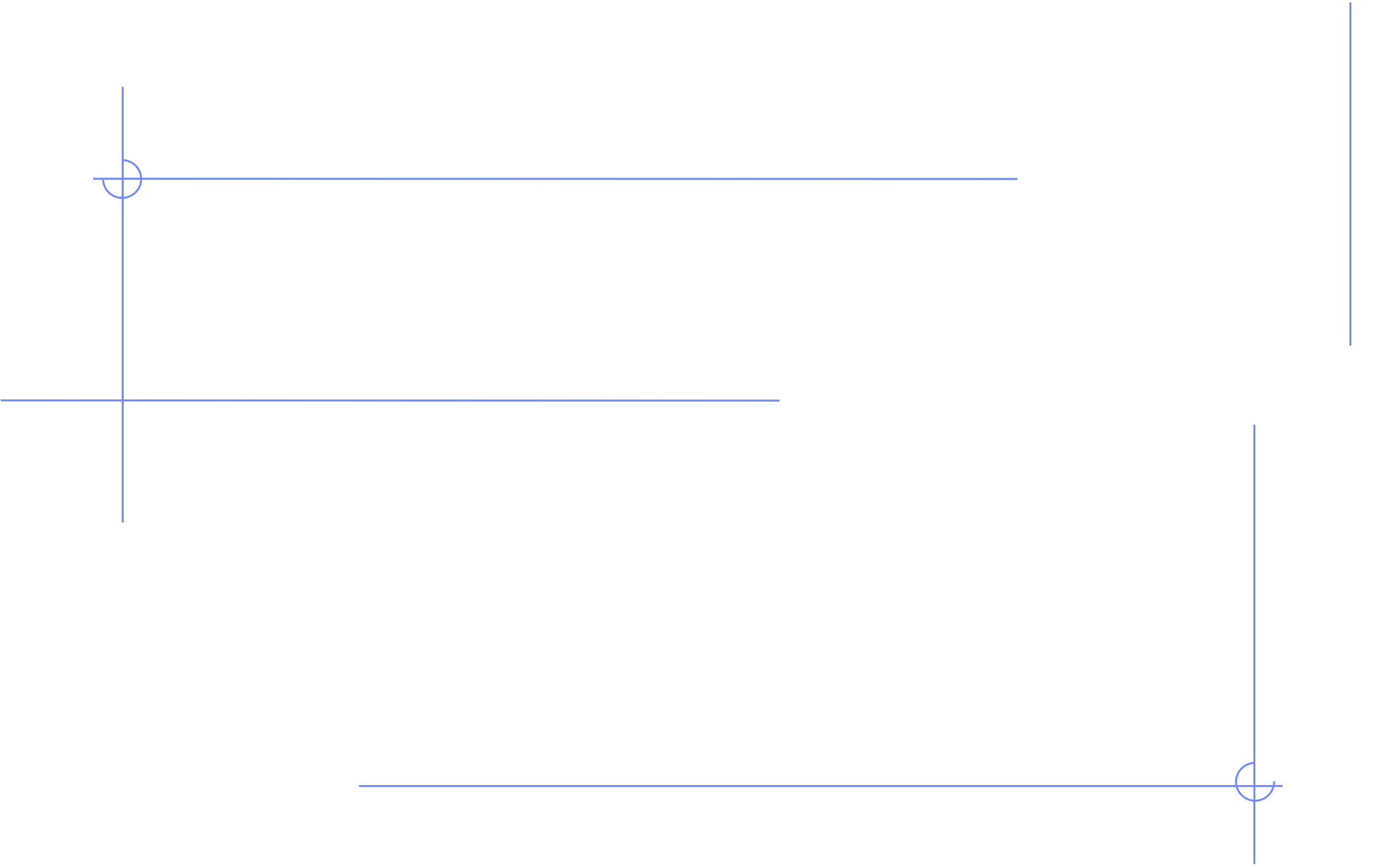
«Προγραμματισμός Ι»

4 - *Έλεγχος Προγράμματος - Βρόχοι Επανάληψης*

Δρ. Χαράλαμπος Καρανίκας
Τμήμα Μηχανικών Πληροφορικής



Οι διαφάνειες βασίζονται στο βιβλίο: C: Από τη Θεωρία στην Εφαρμογή Γ. Σ. Τσελίκης – Ν. Δ. Τσελίκας

Έλεγχος Προγράμματος

Η εντολή `if` (Ι)

- Η εντολή **`if`** είναι μία από τις βασικότερες δομές ελέγχου ροής στη C, αλλά και στις περισσότερες γλώσσες προγραμματισμού

- Με την εντολή **`if`** γίνεται δυνατή η επιλεκτική εκτέλεση ενός τμήματος κώδικα, ανάλογα με την τιμή μίας συνθήκης

- Γενική σύνταξη της εντολής **`if`** (στην πιο απλή της μορφή):

```
if(συνθήκη)
{
      ... // ομάδα εντολών
}
```

Η εντολή if (II)

- Αν η συνθήκη είναι αληθής (true), τότε εκτελούνται οι εντολές που περικλείονται στα άγκιστρα { . . . }

```
int x = 3;
if(x != 0)
{
      printf("x isn't zero\n");
}
```

- Αν η συνθήκη δεν είναι αληθής, δηλαδή αν η συνθήκη είναι ψευδής (false), τότε το μπλοκ των εντολών που περικλείεται στα άγκιστρα παρακάμπτεται και συνεπώς δεν εκτελείται

```
int x = -3;
if(x == 0)
{
      printf("x is zero\n");
}
```

Παρατηρήσεις (Ι)

- Αν το μπλοκ εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν

- Π.χ.

```
int x = 3;
if(x > 0)
      printf("x is positive\n");
```

- Αν, βέβαια, το μπλοκ εντολών περιέχει περισσότερες από μία εντολές, τότε τα άγκιστρα είναι απαραίτητα

```
int x = 3;
if(x > 0)
{
      printf("x is positive\n");
      printf("In other words, x is greater than zero\n");
}
```

Παρατηρήσεις (ΙΙ)

ΠΡΟΣΟΧΗ!!!

- Μην βάζετε το ελληνικό ερωτηματικό ; στο τέλος της **if** εντολής, γιατί ουσιαστικά το ερωτηματικό τερματίζει στο σημείο εκείνο την εντολή **if**

- Π.χ. τί εμφανίζει το παρακάτω παράδειγμα ???

```
int x = -3;
if(x > 0);
      printf("x is positive\n");
```

- και τί αυτό ???

```
int x = 3;
if(x > 0);
      printf("x is positive\n");
```

- Στην οθόνη εμφανίζεται το μήνυμα **x is positive** ανεξάρτητα από την τιμή της μεταβλητής **x**

Παρατηρήσεις (III)

□ □□ □ □ □ □ !!!

- Μην συγχέετε τον τελεστή ελέγχου ισότητας **==** (□□□ □□ □□□□) □□ □□□ □□□□□□□□ □□□□ □□□□□ **=** (□□□□ □□□□)

- □ □ □□□□□□□□ □□□□□□□□□ □□□ □□□□□□□□□□□ □□□□□ **x equals 2**, □□ □□□ □ □□□□□□□ □□□□ □□□ □□□□□□□□□□□ **x** □□□□□ **3**

```
int x = 3;
if(x = 2)
      printf("x equals 2\n");
```

- □□□ □□ □□□□□□ «□□ □□□ □□□□□□□□» □□□ □□□□□□□ **if**, η συνθήκη θα έπρεπε να γραφεί ως **if(x == 2)**, □□□□□□□ □□ □□□□ □□□□ □□□□□□□□□□ □□□□

Παρατηρήσεις (IV)

ΠΡΟΣΟΧΗ!!!

- Η □□□□ □□□□ □□□ μη μηδενικής τιμής σε μία μεταβλητή ισοδυναμεί με αληθή συνθήκη, ενώ η εκχώρηση μηδενικής τιμής ισοδυναμεί με ψευδή συνθήκη

- Π.χ. τι εμφανίζει το παρακάτω κομμάτι κώδικα;

```
int x = -3;
if(x = -2)
      printf("x equals -2\n");
```

- Και τι αυτό;

```
int x = 0;
if(x = 0)
      printf("x equals zero\n");
```

Παρατηρήσεις (V)

- Η έκφραση:

  `if(x)` είναι ισοδύναμη με `if(x != 0)`

- Η έκφραση:

  `if(!x)` είναι ισοδύναμη με `if(x == 0)`

- Η εντολή `if` μπορεί προαιρετικά να συμπληρώνεται με την εντολή `else`, όπως θα δούμε στη συνέχεια

Η εντολή if...else (Ι)

- Όταν θέλουμε να προσδιορίσουμε μία ομάδα εντολών που θα εκτελεστεί όταν μία συνθήκη είναι αληθής (true) και μία άλλη ομάδα εντολών που θα εκτελεστεί όταν η συνθήκη αυτή είναι ψευδής (false), τότε χρησιμοποιούμε την εντολή ελέγχου **if...else**

- Γενική σύνταξη της εντολής **if...else**:

```
if(συνθήκη)
{
     ... // ομάδα εντολών Α
}
else
{
     ... // ομάδα εντολών Β
}
```

Η εντολή if...else (II)

- Όταν η **συνθήκη** είναι αληθής (true), [illegible] ομάδα εντολών A (δηλ. οι εντολές που περιέχονται ανάμεσα στα άγκιστρα του **if**), ενώ όταν η συνθήκη είναι ψευδής (false), τότε εκτελείται η ομάδα εντολών B (δηλ. οι εντολές που περιέχονται ανάμεσα στα άγκιστρα του **else**)

- [illegible]

```
int x = -3;
if(x > 0)
{
      printf("x is positive\n");
}
else
{
      printf("x is negative or zero\n");
}
```

Παρατηρήσεις

- Θυμηθείτε ότι στην περίπτωση της εντολής **if**, αν η ομάδα εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν.

- Το ίδιο ισχύει και στην περίπτωση της εντολής **if...else**

- Δηλαδή, το προηγούμενο παράδειγμα θα μπορούσε να γραφεί και ως εξής:

```
int x = -3;
if(x > 0)
      printf("x is positive\n");
else
      printf("x is negative or zero\n");
```

- Αν, βέβαια, κάποια από τις ομάδες εντολών περιέχει περισσότερες από μία εντολές, τότε τα άγκιστρα είναι απαραίτητα στο συγκεκριμένο μπλοκ

Ένθετες if εντολές (Ι)

- Στη γενικότερη περίπτωση, τα μπλοκ εντολών των **if** και **else** εντολών επιτρέπεται να περιέχουν και άλλες **if** και **else** εντολές, οι οποίες με τη σειρά τους μπορεί να περιέχουν και άλλες, κ.ο.κ.

- Όταν υπάρχει μία **if** εντολή μέσα σε μία άλλη, τότε αυτή η **if** εντολή ονομάζεται **ένθετη** ή **φωλιασμένη** (**nested**)

- Παράδειγμα με δύο ένθετες **if** εντολές

```
#include <stdio.h>
int main()
{
        int a = 10,b = 20,c = 30;

        if(a > 5)
        {
                if(b == 20)
                        printf("1\n");

                if(c == 40)
                        printf("2\n");
                else
                        printf("3\n");
        }
        else
                printf("4\n");

        return 0;
}
```

Ένθετες if εντολές (ΙΙ)

- Στην περίπτωση που ένα πρόγραμμα περιέχει ένθετες **if** εντολές, ο κανόνας είναι ότι κάθε **else** εντολή συνδέεται με την αμέσως προηγούμενη **if** εντολή που υπάρχει στην ίδια ομάδα εντολών (δηλ. ανάμεσα στα ίδια άγκιστρα), αρκεί αυτή να μη σχετίζεται με άλλη **else** εντολή

```
#include <stdio.h>
int main()
{
        int a = 5;
      ? if(a != 5) X
            ? if(a-2 > 5) √
                    printf("One\n");
        else
              printf("Two\n");
        return 0;
}
```

- Όταν γίνεται χρήση ένθετων εντολών **if** προτείνεται η χρήση των αγκίστρων, για να είναι πιο ξεκάθαρη η σχέση μεταξύ των εντολών **else** και **if** (ιδιαίτερα στην περίπτωση που στο πρόγραμμά σας χρησιμοποιείτε μεγάλο αριθμό από **if** και **else** εντολές)

Ένθετες if εντολές (III)

- Στο διπλανό πρόγραμμα, η εντολή `else printf("3\n");` αντιστοιχεί στην πλησιέστερη `if` εντολή, που είναι η `if(c == 40)`

- Όμως, η τελική εντολή `else printf("4\n");` δεν αντιστοιχίζεται με την πλησιέστερη `if` εντολή, που είναι η `if(b == 20)`, γιατί δεν ανήκουν στο ίδιο μπλοκ

- Η εντολή αυτή συνδέεται με την εντολή `if(a > 5)`

- Άρα, η ποια είναι η έξοδος του προγράμματος ???

```
#include <stdio.h>
int main()
{
        int a = 10,b = 20,c = 30;

        if(a > 5)
        {
                if(b == 20)
                        printf("1\n");

                if(c == 40)
                        printf("2\n");
                else
                        printf("3\n");
        }
        else
                printf("4\n");

        return 0;
}
```

Έξοδος: 1 3

Προτεινόμενη σύνταξη ένθετων `if` εντολών

- Μία πολύ συνηθισμένη χρήση των ένθετων εντολών `if` στηρίζεται στην ακόλουθη σύνταξη:

```
if(συνθήκη_A)
{
      ... /* ομάδα εντολών A */
}
else if(συνθήκη_B)
{
      ... /* ομάδα εντολών B */
}
else if(συνθήκη_C)
{
      ... /* ομάδα εντολών C */
}
.
.
.
else
{
      ... /* ομάδα εντολών N */
}
... /* επόμενες εντολές του προγράμματος. */
```

- Βάσει αυτής της σύνταξης, όταν βρεθεί μία συνθήκη που να είναι αληθής, τότε εκτελείται η ομάδα εντολών που σχετίζεται με αυτή και οι υπόλοιπες `else if` συνθήκες αγνοούνται
- Δηλαδή, η εκτέλεση του κώδικα συνεχίζει με την πρώτη εντολή που υπάρχει μετά την τελευταία `else` εντολή

Παράδειγμα

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

        printf("Enter number: ");
        scanf("%d",&a);

        if(a == 1)
                printf("One\n");
        else if(a == 2)
                printf("Two\n");
        else
                printf("Something else\n");

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

Παρατηρήσεις

- Σημειώστε ότι η τελική **else** εντολή δεν είναι υποχρεωτικό να υπάρχει
- Αν δεν υπάρχει, και καμία συνθήκη δεν είναι αληθής, τότε – πολύ απλά – το πρόγραμμα δεν κάνει τίποτα
- Ποια θα ήταν η έξοδος του προηγούμενου παραδείγματος αν δεν υπήρχε η τελική **else** εντολή (βλ. δίπλα) ενώ ο χρήστης εισήγαγε την τιμή 3 ???

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

        printf("Enter number: ");
        scanf("%d",&a);

        if(a == 1)
                printf("One\n");
        else if(a == 2)
                printf("Two\n");

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

Έξοδος: **End**

Ο τελεστής ?: (Ι)

- Ο τελεστής ?: επιτρέπει την εκτέλεση μίας από δύο ενέργειες, σύμφωνα με την τιμή μίας έκφρασης και η σύνταξή του είναι:

```
exp1 ? exp2 : exp3;
```

- Σε μία εντολή με τον τελεστή ?: αν η έκφραση **exp1** είναι αληθής, τότε θα εκτελεστεί η έκφραση που ακολουθεί το ερωτηματικό ? (δηλαδή η **exp2**), αλλιώς θα εκτελεστεί η έκφραση που ακολουθεί την άνω-κάτω τελεία : (δηλαδή η **exp3**)

- Π.χ.

```
#include <stdio.h>
int main()
{
        int b = 20;
        (b > 10) ? printf("One\n") : printf("Two\n");
        return 0;
}
```

- Ο τελεστής ?: χρησιμοποιείται συνήθως για να υποκαταστήσει την εντολή **if**, όταν αυτή έχει απλή μορφή

Ο τελεστής ?: (ΙΙ)

- Η τιμή μίας έκφρασης με τον τελεστή `?:` είναι ίση με την τιμή της έκφρασης που εκτελείται **τελευταία**

- Ποια είναι η τιμή της μεταβλητής **max** στην παρακάτω έκφραση ;

```
max = (a > b)? a : b;
```

- Η παραπάνω έκφραση είναι ισοδύναμη με:

```
if(a > b)
        max = a;
else
        max = b;
```

Ο τελεστής ?: (ΙΙΙ)

- Η έκφραση μετά την την άνω-κάτω τελεία : (δηλαδή η exp3) μπορεί να αντικατασταθεί από άλλη έκφραση που χρησιμοποιεί τον τελεστή ?:

- Π.χ.

```
k = exp1 ? exp2 : add1 ? add2 : add3;
```

Η παραπάνω έκφραση είναι ισοδύναμη με:

```
if(exp1)
        k = exp2;
else if(add1)
        k = add2;
else
        k = add3;
```

Η εντολή switch (I)

- Η εντολή ελέγχου **switch** χρησιμοποιείται εναλλακτικά έναντι της **if-else-if** δομής, όταν επιθυμούμε να ελέγξουμε μία έκφραση για όλες τις δυνατές τιμές που αυτή η έκφραση μπορεί να πάρει και να χειριστούμε την κάθε περίπτωση με διαφορετικό τρόπο

- Γενική σύνταξη της εντολής **switch**:

```
switch(έκφραση)
{
      case σταθερά_1:
      /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
      έκφρασης είναι ίση με τη σταθερά_1. */
      break;

      case σταθερά_2:
      /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
      έκφρασης είναι ίση με τη σταθερά_2. */
      break;
      ...
      case σταθερά_n:
      /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
      έκφρασης είναι ίση με τη σταθερά_n. */
      break;

      default:
      /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
      έκφρασης δεν είναι ίση με καμία από τις προηγούμενες
      σταθερές. */
      break;
}
```

Η εντολή switch (II)

- Η `έκφραση` που ελέγχεται πρέπει να είναι ακέραιη μεταβλητή ή έκφραση
- Οι τιμές των `σταθερά_1`, `σταθερά_2`, ... , `σταθερά_n` πρέπει και αυτές να είναι ακέραιες σταθερές με διαφορετικές τιμές μεταξύ των
- Τα «βήματα» κατά την εκτέλεση της εντολής **`switch`**:

1. Η τιμή της έκφρασης συγκρίνεται διαδοχικά με κάθε μία από τις `σταθερά_1`, `σταθερά_2`, ..., `σταθερά_n`
   - Αν βρεθεί μία ίδια τιμή, τότε εκτελούνται οι εντολές που ακολουθούν το αντίστοιχο **`case`** και στη συνέχεια γίνεται τερματισμός της εντολής **`switch`** μέσω της εντολής **`break`** (λεπτομέρειες για την εντολή **`break`** σε επόμενο slide...)
   - Αν δεν βρεθεί ίδια τιμή, τότε εκτελούνται οι εντολές που ακολουθούν το **`default`** και στη συνέχεια γίνεται τερματισμός της εντολής **`switch`** μέσω της εντολής **`break`**
2. Και στις δύο περιπτώσεις, η εκτέλεση του κώδικα συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της **`switch`** εντολής

Παράδειγμα

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

	printf("Enter number: ");
	scanf("%d",&a);

	switch(a)
	{
		case 1:
			printf("One\n");
		break;

		case 2:
			printf("Two\n");
		break;

		default:
			printf("Something else\n");
		break;
	}

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

Παρατηρήσεις (I)

- Η ύπαρξη της **default** περίπτωσης στην εντολή **switch** δεν είναι υποχρεωτική (όπως δεν ήταν υποχρεωτική και η ύπαρξη της εντολής **else** στην εντολή **if**)

- Σε περίπτωση που δεν υπάρχει η **default** περίπτωση και η τιμή της έκφρασης δεν είναι ίση με κάποια από τις τιμές των `σταθερά_1`, `σταθερά_2`, ..., `σταθερά_n`, τότε γίνεται τερματισμός της εντολής **switch**, χωρίς να γίνει κάποια άλλη ενέργεια

- Δηλαδή, η ροή του προγράμματος συνεχίζει με την εκτέλεση της πρώτης εντολής μετά το **switch**

Παρατηρήσεις (ΙΙ)

- Αν τα μπλοκ εντολών που αντιστοιχούν σε δύο ή περισσότερες **case** περιπτώσεις είναι κοινά, τότε μπορεί να γίνει συνένωση των αντίστοιχων **case**

- Π.χ. αν τα μπλοκ εντολών για τις περιπτώσεις των σταθερά_1, σταθερά_2 και σταθερά_3 είναι κοινά, τότε τα αντίστοιχα **case** συνενώνονται ως εξής (έχουν, όπως βλέπουμε, κοινή **break**)

```
case σταθερά_1:
case σταθερά_2:
case σταθερά_3:
/* μπλοκ εντολών που θα εκτελεστεί αν η τιμή της έκφρασης
είναι ίση με σταθερά_1 ή σταθερά_2 ή σταθερά_3. */
break;
```

Παρατηρήσεις (III)

- Κάθε **switch** εντολή μπορεί να γραφτεί ισοδύναμα με χρήση πολλαπλών εντολών **if-else-if**

- ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΤΗΣ **switch** έναντι της **if**:
  1. Η εντολή **switch** διαφέρει από την εντολή **if** στο ότι η **switch** κάνει έλεγχο μόνο για ισότητα (δηλαδή, για τιμές της έκφρασης που να είναι ίσες με σταθερές **case**), ενώ η συνθήκη σε μία **if** εντολή μπορεί να είναι οποιουδήποτε τύπου
  2. Οι τιμές της **έκφρασης** της **switch** και των συγκρινόμενων σταθερών πρέπει υποχρεωτικά να είναι ακέραιες

Παράδειγμα

Ποια είναι η έξοδος του προγράμματος,

αν ο χρήστης πληκτρολογήσει:

Α) 2

Β) 1

Γ) 0

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

	printf("Enter number: ");
	scanf("%d",&a);

	switch(a)
	{
		case 1:
			printf("One\n");

		case 2:
			printf("Two\n");
		break;

		default:
			printf("Something else\n");
		break;
	}
	printf("End\n");
	return 0;
}
```

Έξοδος:

Α)
```
Two
End
```

Β)
```
One
Two
End
```

Γ)
```
Something else
End
```

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()
{
        int a;

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

Έξοδος:

```
0
1
2
3
4
```

Παρατηρήσεις (Ι)

- 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()
{
        int a;

        for(a = 0; a < 5; a++)
                printf("%d\n",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

- Π.χ. η εντολή:

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

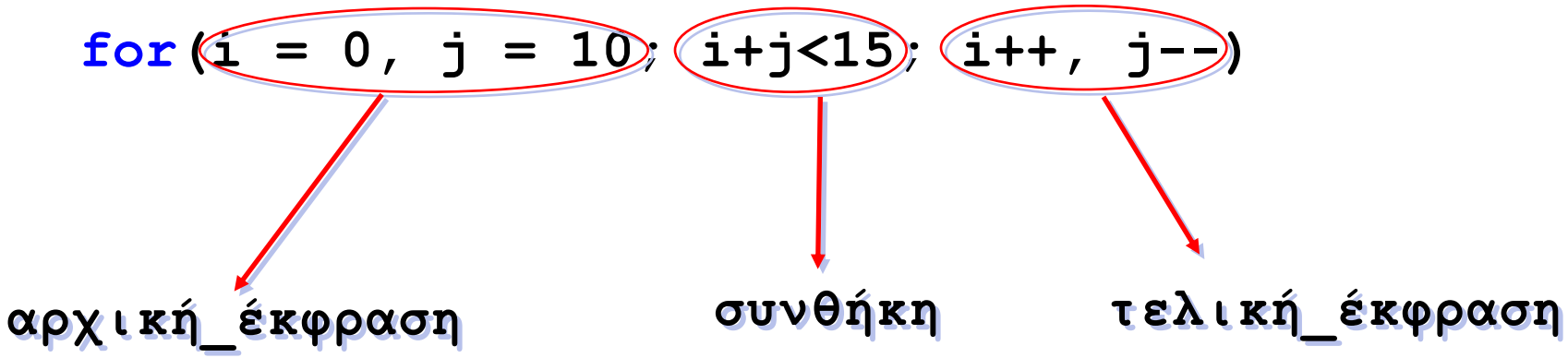
αυξάνει την τιμή του **a** χίλιες φορές και δεν κάνει τίποτα άλλο

- Συνήθως, **for** βρόχοι με «κενή ομάδα εντολών» χρησιμοποιούνται σαν βρόχοι εισαγωγής χρονικής καθυστέρησης, δηλαδή «για να περάσει η ώρα» μέχρι να γίνει κάποια ενέργεια...

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.:



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");
}
```

Παραδείγματα (Ι)

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

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

Παραδείγματα (ΙΙ)

- 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()
{
	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 (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

        for(i = 12; i > 2; i-=5)
                printf("%d ",i);
        return 0;
}
```

Έξοδος: **12 7**

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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()
{
        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 εντολή **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()
{
        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()
{
        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

Εσωτερικός 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. */
}
```

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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 (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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 (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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()
{
        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 διαβάζει 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()
{
      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;  /* Αυτή η μεταβλητή μετράει το
άθροισμα των βαθμών του κάθε φοιτητή σε όλα τα μαθήματα.
Μηδενίζεται για κάθε φοιτητή.*/
```

(συνεχίζεται...→)

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 (Ι)

- 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 (ΙΙ)

- 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");
}
```

Παραδείγματα (Ι)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει συνεχώς έναν ακέραιο αριθμό και να τον εμφανίζει μέχρι ο χρήστης να εισάγει το 0

```
#include <stdio.h>
int main()
{
	int i = 1;
	while(i != 0)
	{
		printf("Enter number: ");
		scanf("%d",&i);

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

Παραδείγματα (ΙΙ)

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

```
#include <stdio.h>
int main()
{
        int a = 256,b = 4;

        while(a != b)
                b = b*b;

        return 0;
}
```

Απάντηση: 2 φορές

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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 να μην προσμετράται ούτε στους θετικούς ούτε στους αρνητικούς αριθμούς

```
#include <stdio.h>
int main()
{
	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*

- 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);
```

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

- 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 (;)

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        int i = 5;
        do
        {
                printf("text\n");
                i += 5;
        } while(i > 10);

        return 0;
}
```

Έξοδος: `text`

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 location;
```

- Όταν εκτελείται η εντολή goto η εκτέλεση του προγράμματος μεταβαίνει άμεσα στην εντολή που ακολουθεί τη θέση που έχει δηλωθεί με το όνομα `location`

- Η θέση με το όνομα `location` 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()
{
        int i,num;

START:
        for(i = 0; i < 5; i++)
        {
                printf("Enter number: ");
                scanf("%d",&num);
                if(num == -1)
                        goto START;
        }
        return 0;
}
```

Παρατηρήσεις (Ι)

- 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 αρκετά προγράμματα χιλιάδων γραμμών, χωρίς καμία εντολή `goto`, τα οποία δεν έχουν καμία σχέση με δομημένο προγραμματισμό

Παρατηρήσεις (II)

- Ίσως, ορισμένες φορές, η χρήση της **goto** να οδηγεί και σε απλούστερο κώδικα (π.χ. στην άμεση έξοδο από έναν «αρκετά ένθετο» **for** βρόχο, όπως τον παρακάτω)

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

- Εξάλλου, και οι εντολές **break**, **continue** και **return** (που χρησιμοποιούνται από τους «κατακριτές» της **goto**) δεν είναι τίποτα άλλο από παραλλαγές της **goto**