

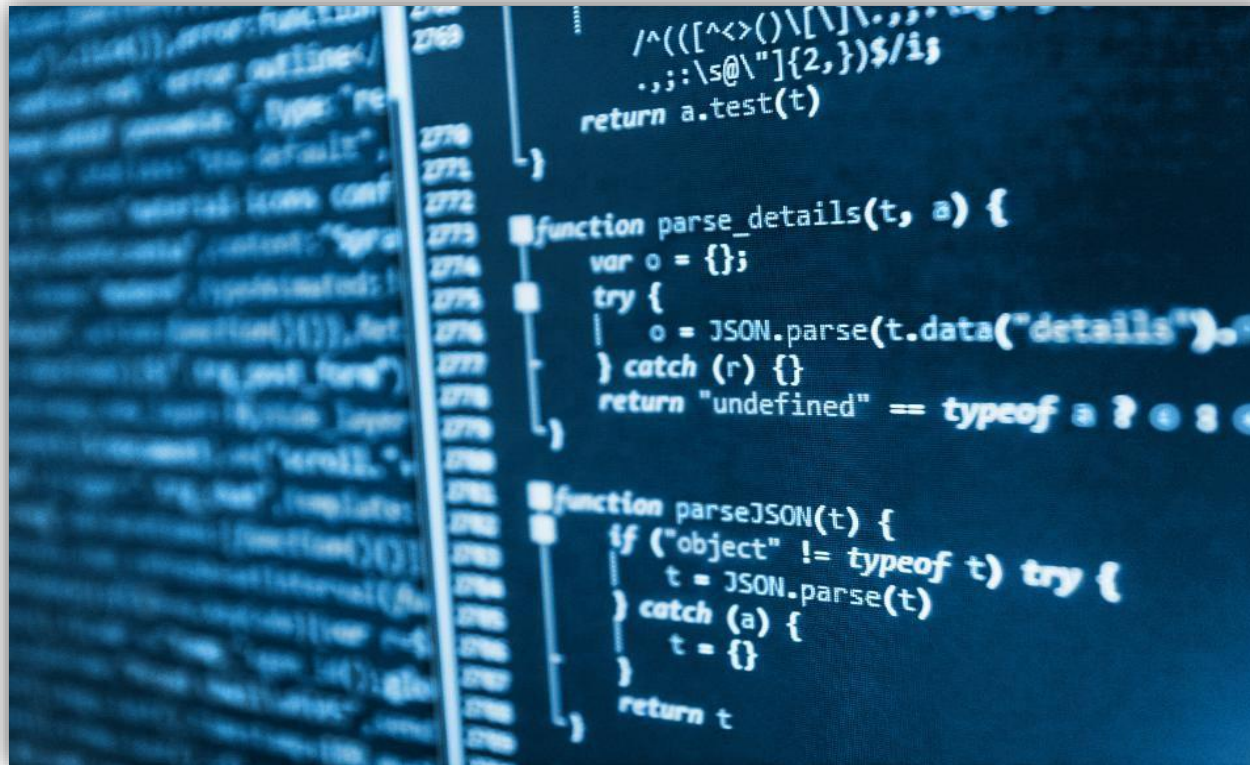
**ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ**



**ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ &
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Ι

Εντολές Επανάληψης



```
.../^\s*([<>()\\[\]{}.,:;@\"']{2,})$/i;
...return a.test(t)

2770
2771
2772
2773 function parse_details(t, a) {
2774   var o = {};
2775   try {
2776     o = JSON.parse(t.data("details"));
2777   } catch (r) {}
2778   return "undefined" == typeof a ? o : a
2779 }
2780
2781 function parseJSON(t) {
2782   if ("object" != typeof t) try {
2783     t = JSON.parse(t)
2784   } catch (a) {
2785     t = {}
2786   }
2787   return t
2788 }
```

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 (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()
{
    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

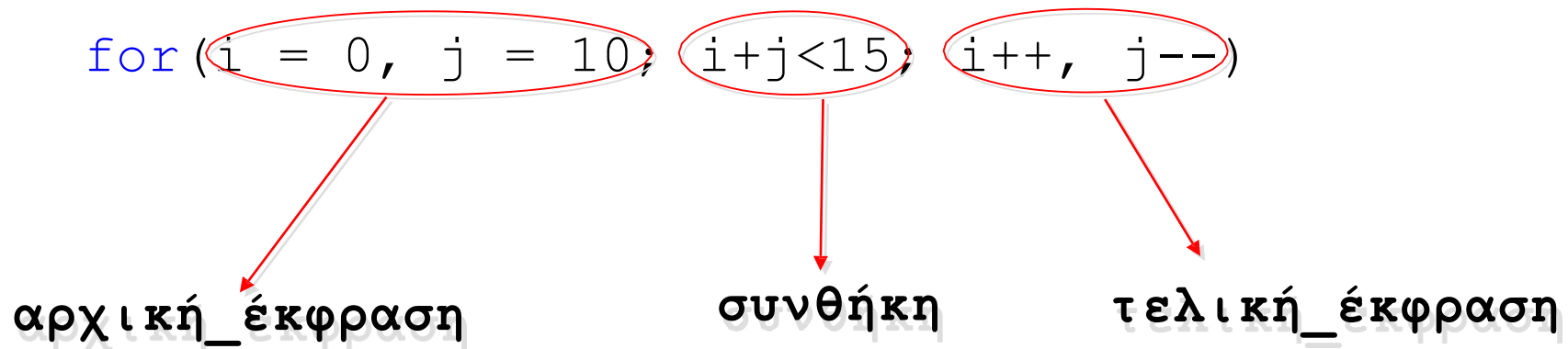
- Συνήθως, `for` βρόχοι με «κενή ομάδα εντολών» χρησιμοποιούνται σαν βρόχοι εισαγωγής χρονικής καθυστέρησης, δηλαδή «για να περάσει η ώρα» μέχρι να γίνει κάποια ενέργεια...

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.:

`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");  
}
```

Μεθοδολογία (I)

ΣΥΝΟΛΙΚΗ ΔΟΜΗ ΑΣΚΗΣΗΣ ΕΠΑΝΑΛΗΨΗΣ

ΣΕΙΡΑ	
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΑΡΧΙΚΟΠΟΙΗΣΗ
1	<i>ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ</i>
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ
4	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ
1	<i>ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ</i>
5	ΔΕΥΤΕΡΕΥΟΥΣΑ ΕΠΕΞΕΡΓΑΣΙΑ
6	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Μεθοδολογία (I)

ΣΥΝΟΛΙΚΗ ΔΟΜΗ ΑΣΚΗΣΗΣ ΕΠΑΝΑΛΗΨΗΣ

ΣΕΙΡΑ	
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΑΡΧΙΚΟΠΟΙΗΣΗ
1	<i>ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ</i>
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ
4	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ
1	<i>ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ</i>
5	ΔΕΥΤΕΡΕΥΟΥΣΑ ΕΠΕΞΕΡΓΑΣΙΑ
6	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Μεθοδολογία (I)

ΣΥΝΟΛΙΚΗ ΔΟΜΗ ΑΣΚΗΣΗΣ ΕΠΑΝΑΛΗΨΗΣ

ΣΕΙΡΑ	
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΑΡΧΙΚΟΠΟΙΗΣΗ
1	<i>ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ</i>
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ
4	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ
1	<i>ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ</i>
5	ΔΕΥΤΕΡΕΥΟΥΣΑ ΕΠΕΞΕΡΓΑΣΙΑ
6	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Μεθοδολογία (I)

ΣΥΝΟΛΙΚΗ ΔΟΜΗ ΑΣΚΗΣΗΣ ΕΠΑΝΑΛΗΨΗΣ

ΣΕΙΡΑ	
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΑΡΧΙΚΟΠΟΙΗΣΗ
1	<i>ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ</i>
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ
4	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ
1	<i>ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ</i>
5	ΔΕΥΤΕΡΕΥΟΥΣΑ ΕΠΕΞΕΡΓΑΣΙΑ
6	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Μεθοδολογία (I)

ΣΥΝΟΛΙΚΗ ΔΟΜΗ ΑΣΚΗΣΗΣ ΕΠΑΝΑΛΗΨΗΣ

ΣΕΙΡΑ	
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΑΡΧΙΚΟΠΟΙΗΣΗ
1	<i>ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ</i>
2	ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ
3	ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ
4	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ
1	<i>ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ</i>
5	ΔΕΥΤΕΡΕΥΟΥΣΑ ΕΠΕΞΕΡΓΑΣΙΑ
6	ΕΞΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Μεθοδολογία (II)

ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ

Είδος επεξεργασίας	Αρχικοποίηση (πριν την επανάληψη)	Υπολογισμός (μέσα στην επανάληψη)	
		Ολικός	Μερικός
Άθροισμα	$s = 0;$	$s = s + x;$ $s += x;$	if (συνθήκη) { $s = s + x;$ }
Πλήθος	$n = 0;$	$n = n + 1;$	if (συνθήκη) { $n = n + 1;$ }
Γινόμενο	$p = 1;$	$p = p * x;$	if (συνθήκη) { $p = p * x;$ }

Μεθοδολογία (II)

ΒΑΣΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ

Είδος επεξεργασίας	Αρχικοποίηση (πριν την επανάληψη)	Υπολογισμός (μέσα στην επανάληψη)	
		Ολικός	Μερικός
Μέγιστο	<i>στην πρώτη τιμή</i> printf("Give value:"); scanf("%f", &x) max = x; <i>αν η μεταβλητή x έχει κάτω όριο k</i> max = k;	<pre>if (x > max) { max = x; }</pre>	<pre>if (συνθήκη) { if (x > max) { max = x; } }</pre>
Ελάχιστο	<i>στην πρώτη τιμή</i> printf("Give value:"); scanf("%f", &x) min = x; <i>αν η μεταβλητή x έχει πάνω όριο k</i> min = k;	<pre>if (x < min) { min = x; }</pre>	<pre>if (συνθήκη) { if (x < min) { min = x; } }</pre>

Μεθοδολογία (II)

ΔΕΥΤΕΡΕΥΟΥΣΑ ΕΠΕΞΕΡΓΑΣΙΑ

Είδος δευτερεύουσας επεξεργασίας	Υπολογισμός
Μέσος όρος	$mo = \text{Άθροισμα} / \text{Πλήθος};$
Ποσοστό	$pos = \text{Μερικό Πλήθος} / \text{Ολικό Πλήθος};$

Ασκήσεις

1. Να γραφεί πρόγραμμα που να εμφανίζει τους αριθμούς από το 1 έως και το 17, καθώς επίσης να υπολογίζει και να εμφανίζει το άθροισμα και το μέσο όρο αυτών των αριθμών.

Solution:[Solution](#)

1. Να γραφεί πρόγραμμα που θα διαβάζει έναν αριθμό N (ακέραιο) και θα υπολογίζει το άθροισμα
$$\Sigma = 1 + 1/2 + 1/3 + 1/4 + \dots + 1/N$$

Στην συνέχεια να εμφανίζεται η τιμή του αθροίσματος.

Solution:[Solution](#)

Ασκήσεις

3. Ένα οικόπεδο θεωρείται «ακριβό», όταν η τιμή πώλησης ανά τετραγωνικό μέτρο είναι μεγαλύτερη των 500€, «φτηνό» όταν η τιμή πώλησης είναι μικρότερη των 250€ και σε οποιαδήποτε άλλη περίπτωση η τιμή θεωρείται «κανονική». Να αναπτύξετε πρόγραμμα που για 5 οικόπεδα:

- να διαβάσει την τιμή πώλησης ολόκληρου του οικοπέδου και τον αριθμό των τετραγωνικών μέτρων του,
- να υπολογίζει την κατηγορία κόστους στην οποία ανήκει και να εμφανίζει το μήνυμα: «ακριβή τιμή» ή «φτηνή τιμή» ή «κανονική τιμή».

Solution: [Solution](#)

Ασκήσεις

4. Σε μία εταιρία εργάζονται 5 υπάλληλοι. Για κάθε υπάλληλο δίνονται: ο μηνιαίος βασικός μισθός και ο αριθμός των παιδιών του. Οι συνολικές αποδοχές του υπολογίζονται ως το άθροισμα του μηνιαίου βασικού μισθού και του οικογενειακού επιδόματός του.
- Το οικογενειακό επίδομα υπολογίζεται ως εξής: 30 ευρώ για κάθε παιδί μέχρι και τρία παιδιά και 40 ευρώ για κάθε παιδί πέραν των τριών (4^ο, 5^ο, 6^ο κ.τ.λ.)
- Να γράψετε πρόγραμμα, το οποίο:
- εισάγει τα κατάλληλα δεδομένα
 - υπολογίζει και εμφανίζει το οικογενειακό επίδομα
 - υπολογίζει και εμφανίζει τις συνολικές αποδοχές του υπαλλήλου

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");
}
```

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 ???

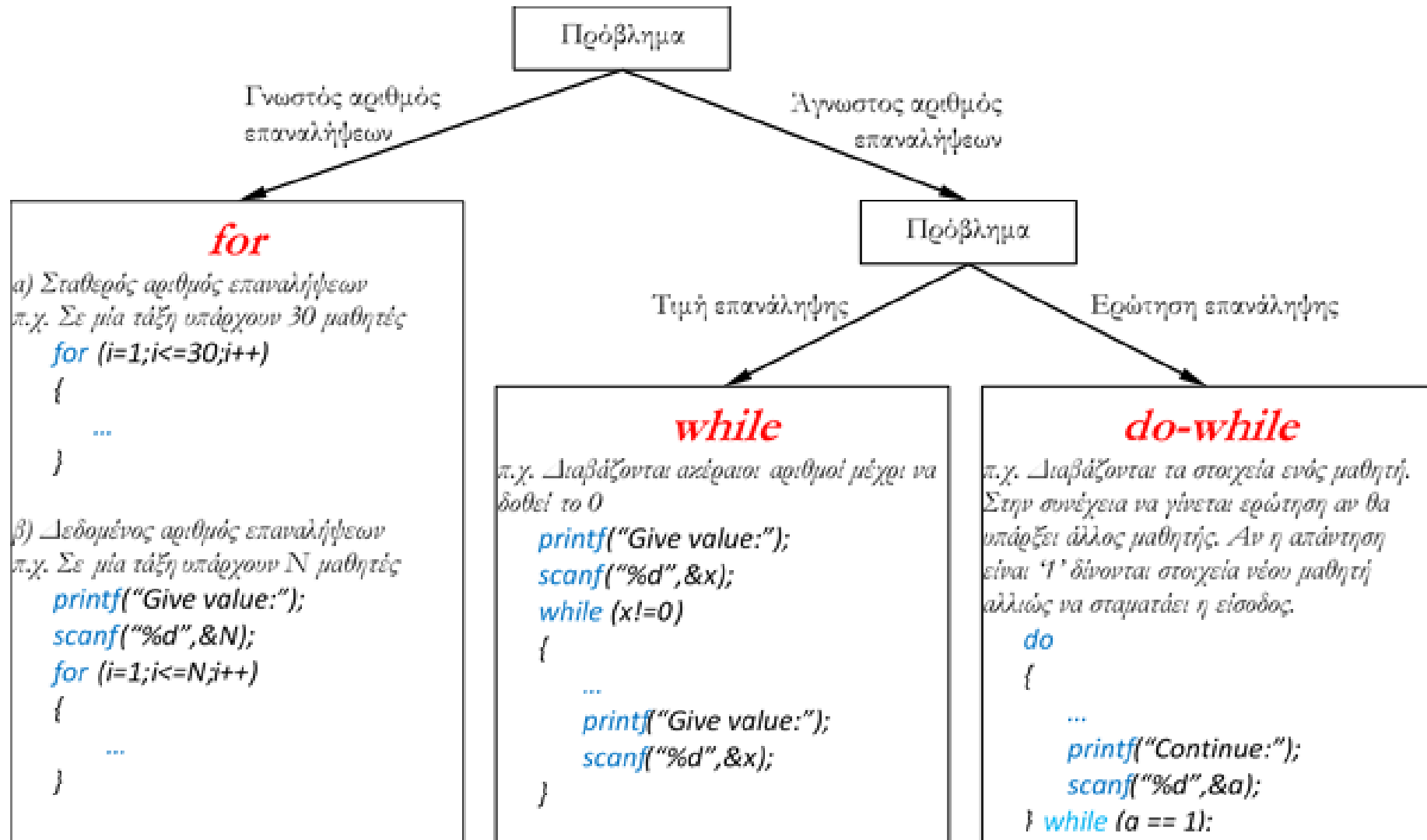
```
#include <stdio.h>
int main()
{
    int i = 5;
    do
    {
        printf("text\n");
        i += 5;
    } while(i > 10);

    return 0;
}
```

Έξοδος: text

Μεθοδολογία (III)

ΕΝΔΕΙΚΤΙΚΗ ΕΠΙΛΟΓΗ ΕΝΤΟΛΗΣ ΕΠΑΝΑΛΗΨΗΣ



Ασκήσεις

5. Κατασκευάστε μια ταμειακή μηχανή η οποία υπολογίζει το πληρωτέο σύνολο ενός πελάτη. Να αναπτύξετε πρόγραμμα:
- Να εκτυπώνει το μήνυμα «ΔΩΣΕ ΤΙΜΗ ΠΡΟΪΟΝΤΟΣ» και να διαβάζει την τιμή του προϊόντος.
 - Να επαναλαμβάνει τη διαδικασία μέχρι να δοθεί τιμή προϊόντος 0.
 - Να εμφανίζει την έκφραση «ΠΛΗΡΩΤΕΟ ΠΟΣΟ» και το ποσό που θα πρέπει να πληρώσει ο πελάτης.

Solution:[Solution](#)

Ασκήσεις

6. Να γραφεί πρόγραμμα το οποίο θα διαβάσει βάση και ύψος ενός τριγώνου (πραγματικοί αριθμοί), θα υπολογίζει το εμβαδόν του και θα εμφανίζει το μήνυμα:

το τρίγωνο με ύψος ... και βάση ... έχει εμβαδόν ...

Στη συνέχεια θα ρωτάει το χρήστη αν θέλει να υπολογίσει το εμβαδόν κάποιου άλλου τριγώνου, και αν πάρει σαν απάντηση “Υ” θα επαναλαμβάνει την διαδικασία αλλιώς θα τερματίζεται.

Solution: [Solution](#)

Ασκήσεις

7. Να γραφεί πρόγραμμα το οποίο θα διαβάσει διάφορους ακέραιους αριθμούς και η εισαγωγή δεδομένων να σταματάει με τον αριθμό 0. Στη συνέχεια να υπολογίζει και να εμφανίζει το μέσο όρο των θετικών και το μέσο όρο των αρνητικών αριθμών.

Solution: [Solution](#)

Ένθετοι επαναληπτικοί 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)
{
    Εσωτερικός for βρόχος
    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 (I)

- 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 (II)

- 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 (III)

- 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: *

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

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

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