Η ΓΛΩΣΣΑ C

Μάθημα 5:

Δομές Επανάληψης

Δημήτρης Ψούνης

www.psounis.gr

Περιεχόμενα Μαθήματος

Α. Πίνακες

1. Εισαγωγή στους Πίνακες

1. Μονοδιάστατοι Πίνακες

- Ένας **πίνακας** είναι μια σειρά από μεταβλητές ίδιου τύπου αποθηκευμένες στην μνήμη.
- Για παράδειγμα ένας πίνακας 10 ακεραίων είναι 10 ακέραιες μεταβλητές αποθηκευμένες στην σειρα (η μία μετά την άλλη) στην μνήμη.
  - Ένας πίνακας θα δηλώνεται (στο τμήμα δήλωσης μεταβλητών) με εντόλη της μορφής:

```
Τύπος_Δεδομένων ΟΝΟΜΑ_ΠΙΝΑΚΑ[ΠΛΗΘΟΣ];
```

  - Όπου:
    - Τύπος_Δεδομένων: Ο τύπος δεδομένων των μεταβλητών του πίνακα.
    - ΟΝΟΜΑ_ΠΙΝΑΚΑ: Το όνομα που επιλέγουμε εμείς για τον πίνακα.
    - ΠΛΗΘΟΣ (προσέξτε ότι είναι μέσα σε αγκύλες): Πόσες μεταβλητές θα περιέχει ο πίνακας

Α. Πίνακες

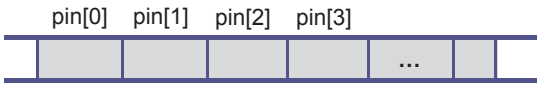
1. Εισαγωγή στους Πίνακες

1. Μονοδιάστατοι Πίνακες

- Για παράδειγμα με την δήλωση:

```
int pin[4];
```

  - Δηλώνουμε έναν πίνακα με 4 ακέραιες μεταβλητές.
  - Με την εντολή δήλωσης, δεσμεύεται στην μνήμη χώρος για τον πίνακα και είναι σημαντικό ότι οι θέσεις αυτές είναι σε μια σειρά.
- Τα ονόματα των 4μεταβλητών που κατασκευάσαμε είναι:
  - pin[0], pin[1], pin[2], pin[3]
    - Παρατηρούμε ότι η αρίθμηση ξεκινά από το 0 έως και ένα λιγότερο από τον αριθμό που δηλώσαμε.
    - Άρα μετά την εντολή δήλωσης, έχουν δεσμευτεί 4 **διαδοχικές** (προσοχή!) θέσεις στην μνήμη για να αποθηκεύσουν τιμές σε αυτές τις μεταβλητές:



Α. Πίνακες
1. Εισαγωγή στους Πίνακες
2. Παράδειγμα

- Μετά τη δήλωση του πίνακα είναι σαν να έχουμε 4 μεταβλητές που μπορούμε να χρησιμοποιήσουμε στο πρόγραμμά μας.
- Δείτε για παράδειγμα το εξής απλό πρόγραμμα:
- Οι μεταβλητές διαχειρίζονται όπως οι τυπικές ακέραιες μεταβλητές

```
/* array.c: Aplo programma me pinaka */
#include <stdio.h>

int main()
{
   int pin[3];
   int sum;

   pin[0]=1;
   pin[1]=3;
   pin[2]=4;
   sum=pin[0]+pin[1]+pin[2];
   printf("\n%d+%d+%d=%d",pin[0],pin[1],pin[2],sum);
}
```

- Προσοχή, να μην γράψουμε αριθμό για όριο του πίνακα που είναι εκτός του επιτρεπτού. Π.χ. Αν έχουμε δήλωση int pin[4]; και γράψουμε για όνομα μεταβλητής pin[5] θα έχουμε σφάλμα όταν θα εκτελέσουμε το πρόγραμμα!
- Περισσότερα για τους πίνακες θα δούμε σε επόμενο μάθημα, όπου θα δούμε και πολυδιάστατους πίνακες (διδιάστατους, τριδιάστατους κ.λπ.)
- Εδώ κάνουμε μια απλή αναφορά στους πίνακες για να τους χρησιμοποιήσουμε με τη δομή επανάληψης στις ασκήσεις. Πολύ περισσότερα σε επόμενο μάθημα.

Β. Δομές Επανάληψης
1. Γενικά

- Η εντολή επανάληψης είναι η σημαντικότερη δομή σε ένα πρόγραμμα.
  - ...διότι μας επιτρέπει να εκτελέσουμε ένα τμήμα κώδικα πολλές φορές, το οποίο είναι το κύριο χαρακτηριστικό του προγραμματισμού.
- Στη γλώσσα C που μαθαίνουμε, υπάρχουν τρεις τρόποι για να κάνουμε επανάληψη της εκτέλεσης ενός τμήματος κώδικα:
  - Η εντολή **for (…; … ; …) { … }** στην οποία ρητά αναφέρουμε πόσες φορές θέλουμε να εκτελεστεί ένα τμήμα κώδικα.
  - Η εντολή **while(…) { … }**
  - Η εντολή **do{…} while(…);**
- Θα αναλύσουμε τους τρεις τρόπους επανάληψης και πότε χρησιμοποιούμε τον καθένα

Β. Δομές Επανάληψης
2. Η δομή for
1. Συντακτικό της for

- Το συντακτικό της εντολής for είναι:

```
for( αρχική; συνθήκη; βήμα)
    εντολή;
```

```
for( αρχική; συνθήκη; βήμα)
{
    (εντολές)
}
```

- Δείτε ότι αν τρέχουμε μία εντολή τότε δεν είμαστε αναγκασμένοι να χρησιμοποιήσουμε άγκιστρα, αλλιώς αν τρέχουμε παραπάνω από μία εντολή πρέπει να βάλουμε τις εντολές σε άγκιστρα (όπως στην if, ακολουθεί την for ένα μπλοκ κώδικα).
- Με το παραπάνω συντακτικό γίνονται οι εξής ενέργειες:
  - 1. Τρέχει η αρχική: είναι μια εντολή που τρέχει μία μόνο φορά στην αρχή. Αναφέρεται και σαν εντολή αρχικοποίησης
  - 2. Ελέγχεται η συνθήκη: συνήθως είναι μια σύγκριση τιμών.
    - 2.1 Αν η συνθήκη είναι ψευδής, τότε βγαίνουμε από την for και πάμε στην αμέσως επόμενη εντολή.
    - 2.2. Αλλιώς αν η συνθήκη είναι αληθής, εκτελείται η εντολή αύξηση και μεταβαίνουμε και πάλι στο βήμα 2.

Β. Δομές Επανάληψης
2. Η δομή for
2. Διάγραμμα Ροής Προγράμματος

- Και το διάγραμμα ροής προγράμματος:

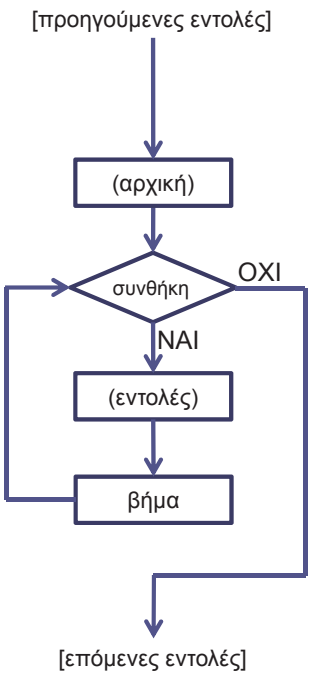
```
[προηγούμενες εντολές]
for( αρχική; συνθήκη; βήμα)
    εντολή;

[επόμενες εντολές]
```

```
[προηγούμενες εντολές]

for( αρχική; συνθήκη; βήμα)
{
    (εντολές)
}

[επόμενες εντολές]
```



Β. Δομές Επανάληψης
2. Η δομή for
3. Παραδείγματα Εκτέλεσης

- Παράδειγμα 1:
  - Τι κάνει το ακόλουθο τμήμα κώδικα;

```
for (i=1; i<=5; i++)
   printf("\nKalimera");
```

  - Απάντηση:
    - Εκτελείται η εντολή i=1.
    - Ελέγχεται αν `i<=5`. Είναι αληθές (`1<=5`), άρα τυπώνεται «Καλημέρα» και εκτελείται η αύξηση `i++`, άρα το i γίνεται 2.
    - Ελέγχεται αν `i<=5`. Είναι αληθές (`2<=5`), άρα τυπώνεται «Καλημέρα» και εκτελείται η αύξηση `i++`, άρα το i γίνεται 3.
    - Ελέγχεται αν `i<=5`. Είναι αληθές (`3<=5`), άρα τυπώνεται «Καλημέρα» και εκτελείται η αύξηση `i++`, άρα το i γίνεται 4.
    - Ελέγχεται αν `i<=5`. Είναι αληθές (`4<=5`), άρα τυπώνεται «Καλημέρα» και εκτελείται η αύξηση `i++`, άρα το i γίνεται 5.
    - Ελέγχεται αν `i<=5`. Είναι αληθές (`5<=5`), άρα τυπώνεται «Καλημέρα» και εκτελείται η αύξηση `i++`, άρα το i γίνεται 6.
    - Ελέγχεται αν `i<=5`. Είναι ψευδές (`6<=5`), άρα μεταβαίνουμε στην αμέσως επόμενη εντολή μετά την for.
  - Άρα το τμήμα κώδικα τυπώνει 5 φορές στην οθόνη τη λέξη ΚΑΛΗΜΕΡΑ

Β. Δομές Επανάληψης
2. Η δομή for
3. Παραδείγματα Εκτέλεσης

- Συνήθως στις εφαρμογές χρησιμοποιείται η τιμή της μεταβλητής που τροποποιείται στην αύξηση. Είναι σημαντικό ότι σε κάθε επανάληψη η τιμή της μεταβλητής είναι διαφορετική

- Παράδειγμα 2:
  - Τι κάνει το ακόλουθο τμήμα κώδικα;

```
for (i=1; i<=3; i++)
   printf("\n%d",i);
```

  - Απάντηση:
    - Εκτελείται η εντολή i=1.
    - Ελέγχεται αν `i<=3`. Είναι αληθές (`1<=3`), άρα τυπώνεται «`1`» και εκτελείται η αύξηση `i++`, άρα το `i` γίνεται 2.
    - Ελέγχεται αν `i<=3`. Είναι αληθές (`2<=3`), άρα τυπώνεται «`2`» και εκτελείται η αύξηση `i++`, άρα το `i` γίνεται 3.
    - Ελέγχεται αν `i<=3`. Είναι αληθές (`3<=3`), άρα τυπώνεται «`3`» και εκτελείται η αύξηση `i++`, άρα το `i` γίνεται 4.
    - Ελέγχεται αν `i<=3`. Είναι ψευδές (`4<=3`), άρα μεταβαίνουμε στην αμέσως επόμενη εντολή μετά την for.
  - Και το πρόγραμμα τερματίζει

Β. Δομές Επανάληψης
2. Η δομή for
3. Παραδείγματα Εκτέλεσης

- Συνήθως, χρησιμοποιώντας την τιμή της μεταβλητής κάνουμε ακόμη και περίπλοκους υπολογισμούς.

- Παράδειγμα 3:
  - Τι κάνει το ακόλουθο τμήμα κώδικα (οι j,x είναι ακέραιες μεταβλητες);

```
for (j=0; j<2; j++)
{
   x=j*j-1;
   printf("\n%d",x);
}
```

  - Απάντηση: Οι εντολές που περικλείονται ανάμεσα στα άγκιστρα εκτελούνται για τις διαδοχικές τιμές της μεταβλητής. Άρα:
    - Εκτελείται η εντολή `j=0`.
    - Ελέγχεται αν `j<2`. Είναι αληθές (`0<2`), άρα υπολογίζεται `x=0*0-1=-1` άρα τυπώνεται «`-1`» και εκτελείται η αύξηση `j++`, άρα το `j` γίνεται 1.
    - Ελέγχεται αν `j<2`. Είναι αληθές (`1<2`), άρα υπολογίζεται `x=1*1-1=0` άρα τυπώνεται «`0`» και εκτελείται η αύξηση `j++`, άρα το `j` γίνεται 2.
    - Ελέγχεται αν `j<2`. Είναι ψευδές (`2<2`), άρα μεταβαίνουμε στην επόμενη εντολή μετά τη for.

Β. Δομές Επανάληψης
2. Η δομή for
3. Παραδείγματα Εκτέλεσης

- Ας δούμε και ένα παράδειγμα που η μεταβλητή μειώνεται αντί να αυξάνεται.

- Παράδειγμα 4:
  - Τι κάνει το ακόλουθο τμήμα κώδικα (οι j,x είναι ακέραιες μεταβλητές);

```
for (j=2; j>=0; j--)
{
   x=j*j-1;
   printf("\n%d",x);
}
```

  - Απάντηση: Οι εντολές που περικλείονται ανάμεσα στα άγκιστρα εκτελούνται για τις διαδοχικές τιμές της μεταβλητής. Άρα:
    - Εκτελείται η εντολή `j=2`.
    - Ελέγχεται αν `j>=0`. Είναι αληθές (`2>=0`), άρα υπολογίζεται `x=2*2-1=3` άρα τυπώνεται «`3`» και εκτελείται η μείωση `j--`, άρα το `j` γίνεται 1.
    - Ελέγχεται αν `j>=0`. Είναι αληθές (`1>=0`), άρα υπολογίζεται `x=1*1-1=0` άρα τυπώνεται «`0`» και εκτελείται η μείωση `j--`, άρα το `j` γίνεται 0.
    - Ελέγχεται αν `j>=0`. Είναι αληθές (`0>=0`), άρα υπολογίζεται `x=0*0-1=0` άρα τυπώνεται «`-1`» και εκτελείται η μείωση `j--`, άρα το `j` γίνεται `-1`.
    - Ελέγχεται αν `j>=0`. Είναι ψευδές `(-1>=0)`, άρα μεταβαίνουμε στην επόμενη εντολή μετά τη for.

Β. Δομές Επανάληψης
2. Η δομή for
3. Παραδείγματα Εκτέλεσης

- Μπορούμε να έχουμε και οποιοδήποτε βήμα αύξησης επιθυμούμε με το κατάλληλο συντακτικό.
- Παράδειγμα 5:
  - Τι κάνει το ακόλουθο τμήμα κώδικα (η `i` είναι ακέραια μεταβλητή);

```
for (i=0; i<=6; i+=2)
   printf("\n%d",i*i);
```

  - Απάντηση:
    - Εκτελείται η εντολή `i=0`.
    - Ελέγχεται αν `i<=6`. Είναι αληθές `(0<=6)`, υπολογίζεται `0*0=0` τυπώνεται «0» και εκτελείται η αύξηση `i+=2`, άρα το i γίνεται `2`.
    - Ελέγχεται αν `i<=6`. Είναι αληθές `(2<=6)`, υπολογίζεται `2*2=4` τυπώνεται «4» και εκτελείται η αύξηση `i+=2`, άρα το i γίνεται `4`.
    - Ελέγχεται αν `i<=6`. Είναι αληθές `(4<=6)`, υπολογίζεται `4*4=16` τυπώνεται «16» και εκτελείται η αύξηση `i+=2`, άρα το i γίνεται `16`.
    - Ελέγχεται αν `i<=6`. Είναι αληθές `(6<=6)`, υπολογίζεται `6*6=36` τυπώνεται «36» και εκτελείται η αύξηση `i+=2`, άρα το i γίνεται `8`.
    - Ελέγχεται αν `i<=6`. Είναι ψευδές `(8<=6)`, άρα μεταβαίνουμε στην επόμενη εντολή μετά τη for.

Β. Δομές Επανάληψης
2. Η δομή for
3. Παραδείγματα Εκτέλεσης

- Αν και ο συνηθισμένος τρόπος χρήσης της for, είναι κάποιος από αυτούς που είδαμε, έχουμε επιπλέον δικαίωματα:
  - Να βάλουμε παραπάνω απο μία αρχικές εντολές (χωρισμένες με κόμμα)
  - Να βάλουμε παραπάνω από μια εντολές αύξησης (χωρισμένες με κόμμα)
  - Να έχουμε πιο σύνθετες συνθήκες (π.χ. και με λογικό έλεγχο.)
- Εκτελέστε το παρακάτω πρόγραμμα:

```
/* complex_for.c: Deixnei to oloklиromeno sintaktiko tis for */
#include <stdio.h>

main()
{
   int i,j;

   for (i=0,j=0; i<5 && j<5; i++,j+=2)
      printf("\ni=%d,j=%d: ",i,j);

}
```


Β. Δομές Επανάληψης
3. Η δομή do..while
1. Συντακτικό της do..while

- Το συντακτικό της εντολής do…while (επανέλαβε…όσο) είναι:

```
do
{
   (Εντολές)
}
while (Συνθήκη);
```

- Με την παρατήρηση ότι εδώ δεν μπορούμε να παραλείψουμε τα άγκιστρα!.

- Με το παραπάνω συντακτικό γίνονται οι εξής ενέργειες:
  - 1. Εκτελούνται οι εντολές
  - 2. Ελέγχεται η συνθήκη: (την οποία έχουμε συντάξει με λογικούς και σχεσιακούς τελεστές)
    - 2.1 Αν η απάντηση είναι ΑΛΗΘΗΣ (ισχύει η συνθήκη), τότε ξαναρχίζουμε από την αρχή (πρώτη εντολή μετά από το do)
    - 2.2 Αν η απάντηση είναι ΨΕΥΔΗΣ (δεν ισχύει η συνθήκη), τότε βγαίνουμε από την επανάληψη και τρέχουμε την αμέσως επόμενη εντολή μετά τη while.

Β. Δομές Επανάληψης
3. Η δομή do..while
2. Διάγραμμα Ροής Προγράμματος

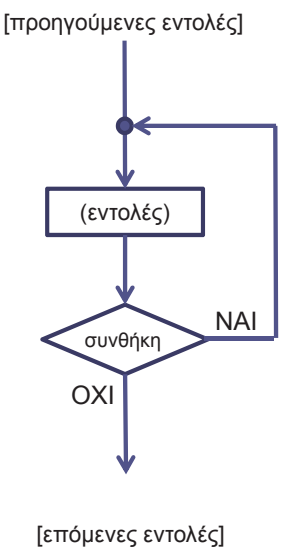
- Και το διάγραμμα ροής προγράμματος:

```
[προηγούμενες εντολές]

do
{
   (Εντολές)
}
while (Συνθήκη);

[επόμενες εντολές]
```

- Παρατηρήστε ότι η συνθήκη ελέγχεται αφού έχουν εκτελεστεί οι εντολές



Β. Δομές Επανάληψης
3. Η δομή do…while
3. Παραδείγματα Εκτέλεσης

- Μπορούμε να χρησιμοποιήσουμε την do..while με αντίστοιχο τρόπο με την for.
- Παράδειγμα 1:
  - Τι κάνει το ακόλουθο τμήμα κώδικα (η `i` είναι ακέραια μεταβλητή);

```
i=0;
do{
   i=i+1;
   printf("\n%d",i);
} while(i<3);
```

  - Αρχικά `i=0`
    - Γίνεται επανάληψη:
      - Τίθεται `i=0+1=1` και τυπώνεται «1»
      - Γίνεται έλεγχος συνθήκης. `1<3` είναι ΑΛΗΘΗΣ, άρα θα επαναλαβουμε
    - Γίνεται επανάληψη:
      - Τίθεται `i=1+1=2` και τυπώνεται «2»
      - Γίνεται έλεγχος συνθήκης. `2<3` είναι ΑΛΗΘΗΣ, άρα θα επαναλαβουμε
    - Γίνεται επανάληψη:
      - Τίθεται `i=2+1=3` και τυπώνεται «3»
      - Γίνεται έλεγχος συνθήκης. `3<3` είναι ΨΕΥΔΗΣ, άρα δεν θα επαναλαβουμε και πηγαίνουμε στην επόμενη εντολή του κώδικα.

Β. Δομές Επανάληψης
3. Η δομή do…while
3. Παραδείγματα Εκτέλεσης

- Και εδώ θα κάνουμε ένα από τα πιο συχνά λάθη! Θα πέσουμε σε ατέρμονα βρόχο!
- Παράδειγμα 2: Τι κάνει το ακόλουθο τμήμα κώδικα (η `i` είναι ακέραια μεταβλητή);

```
i=5;
do{
   i=i+1;
   printf("\n%d",i);
} while(i>3);
```

  - Αρχικά `i=5`
    - Γίνεται επανάληψη:
      - Τίθεται `i=5+1=6` και εκτυπώνεται «6»
      - Γίνεται έλεγχος συνθήκης. `6>3` είναι ΑΛΗΘΗΣ, άρα θα επαναλαβουμε
    - Γίνεται επανάληψη:
      - Τίθεται `i=6+1=7` και εκτυπώνεται «7»
      - Γίνεται έλεγχος συνθήκης. `7>3` είναι ΑΛΗΘΗΣ, άρα θα επαναλαβουμε....
  - ..... Θα τερματίσει ποτέ το πρόγραμμα?
  - Η απάντηση είναι ΟΧΙ! Σε κάθε επανάληψη το i θα αυξάνεται, άρα ποτέ δεν θα γίνει <= 3!

- **Ατέρμων βρόχος** (infinite loop) είναι μία (επανάληψη που δεν ολοκληρώνεται ποτέ)
- Αποτελεί από τα συχνότερα προγραμματιστικά λάθη!

Β. Δομές Επανάληψης
3. Η δομή do…while
3. Παραδείγματα Εκτέλεσης

Παρατήρηση:
- Η σύνταξη της δομής είναι αρκετά απλή, αλλά θα πρέπει εμείς, ως προγραμματιστές να συντάξουμε σωστά τις υπόλοιπες εντολές. Συγκεκριμένα:
  - Πρέπει να αρχικοποιήσουμε σωστά την μεταβλητή που θα έχουμε στην εντολή συνθήκης. Έτσι πριν την do θα πρέπει να αρχικοποιήσουμε την μεταβλητή που θα χρησιμοποιήσουμε (**εντολή αρχικοποίησης**)
  - Πρέπει η μεταβλητή που έχουμε στην εντολή αρχικοποίησης να επηρεάζεται στο σώμα της επανάληψης (**εντολή αύξησης μεταβλητής**)

- Σχηματικά:

```
i=0;                  <- Εντολή Αρχικοποίησης (την γράφουμε εμείς)
do{
   (εντολή ή εντολές)
   i=i+1;             <- Εντολή αύξησης μεταβλητής (την γράφουμε εμείς)
} while (i<=3);       <- Συνθήκη
```


Β. Δομές Επανάληψης
3. Η δομή do…while
4. Αμυντικός Προγραμματισμός

- **Αν ξέρουμε πόσες φορές θέλουμε να τρέξει η επανάληψη, είναι προτιμότερο να χρησιμοποιήσουμε for**, γιατί στο συντακτικό της γράφουμε τις εντολές αρχικοποίησης και αύξησης, και έτσι ελαχιστοποιείται η πιθανότητα λάθους.

- Ωστόσο **η δομή do..while** είναι πολύ χρήσιμη **όταν θέλουμε να διαβάσουμε μεταβλητές που να έχουν συγκεκριμένες τιμές.**
  - Π.χ. Αν θέλουμε να διαβάσουμε έναν ακέραιο μεταξύ 1 και 100 και πρέπει να αποφύγουμε ο χρήστης να εισάγει μία λανθασμένη τιμή τότε εφαρμόζουμε **αμυντικό προγραμματισμό** και κάνουμε έλεγχο αν η τιμή που εισήγαγε ο χρήστης είναι σωστή.
  - Έτσι τον αναγκάζουμε να επαναπληκτρολογήσει μέχρι να βάλει την σωστή τιμή.
  - Ο αμυντικός προγραμματισμός είναι καλή προγραμματιστική τακτική σε προγράμματα που ζητάμε είσοδο από τον χρήστη.

- Στην επόμενη διαφάνεια φαίνεται πως θα κάνουμε αμυντικό προγραμματισμό για να διαβάσουμε έναν αριθμό από 1 εώς 100.

Β. Δομές Επανάληψης
3. Η δομή do…while
4. Αμυντικός Προγραμματισμός

```
/* defensive.c: Amintikos Programmatismos gia to diavasma enos akeraioy
*/

#include <stdio.h>

main()
{
   int i;

   do {
      printf("Dwste enan akeraio apo 1 ews 100: ");
      scanf("%d",&i);
   } while (i<1 || i>100);

   printf("Eisagate arithmo mesa sta oria 1 ews 100: %d",i);

}
```


Β. Δομές Επανάληψης
4. Η δομή while
1. Συντακτικό της while

- Το συντακτικό της δομής while (επανέλαβε) είναι ίδια με την do..while με την μόνη διαφορά ότι ο έλεγχος γίνεται στην αρχή της επανάληψης και όχι στο τέλος της επανάληψης :

```
while(Συνθήκη)
{
   (Εντολή ή Εντολές)
}
```

- Και εδώ αν έχουμε μόνο μία εντολή μπορούμε να παραλείψουμε τα άγκιστρα!
- Με το παραπάνω συντακτικό γίνονται οι εξής ενέργειες:
  - 1. Ελέγχεται η συνθήκη: (την οποία έχουμε συντάξει με λογικούς και σχεσιακούς τελεστές)
    - 1.1 Αν η απάντηση είναι ΑΛΗΘΗΣ (ισχύει η συνθήκη), τότε ξαναρχίζουμε από την αρχή (πρώτη εντολή μετά από το do)
    - 1.2 Αν η απάντηση είναι ΨΕΥΔΗΣ (δεν ισχύει η συνθήκη), τότε βγαίνουμε από την επανάληψη και τρέχουμε την αμέσως επόμενη εντολή μετά τη while.

Β. Δομές Επανάληψης
4. Η δομή while
2. Διάγραμμα Ροής Προγράμματος

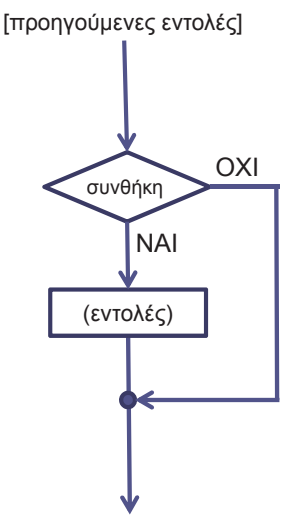
- Και το διάγραμμα ροής προγράμματος:

```
[προηγούμενες εντολές]

while(Συνθήκη)
{
   (Εντολές)
}


[επόμενες εντολές]
```

- Παρατηρήστε ότι η συνθήκη ελέγχεται πριν εκτελεστούν οι εντολές



Β. Δομές Επανάληψης
4. Η δομή while
3. Παραδείγματα Εκτέλεσης

- Ένα πρώτο παράδειγμα, για την αντιστοιχία με τις άλλες δομές επανάληψης
- Παράδειγμα 1:
  - Τι κάνει το ακόλουθο τμήμα κώδικα (οι `k,l` είναι ακέραιες μεταβλητές);

```
k=5;
while(k<8)
{
   l=2*k+1;
   k=k+1;
   printf("%d",l);
}
```

  - Απάντηση:
    - Αρχικοποιείται το `k` με 5
    - Γίνεται ο έλεγχος συνθήκης (`5<8`). Είναι ΑΛΗΘΕΣ, άρα γίνονται τα βηματα `l=2*5+1=11` και `k=5+1=6`. Τυπώνεται «11».
    - Γίνεται ο έλεγχος συνθήκης (`6<8`). Είναι ΑΛΗΘΕΣ, άρα γίνονται τα βηματα `l=2*6+1=13` και `k=6+1=7`. Τυπώνεται «13».
    - Γίνεται ο έλεγχος συνθήκης (`7<8`). Είναι ΑΛΗΘΕΣ, άρα γίνονται τα βηματα `l=2*7+1=15` και `k=7+1=8`. Τυπώνεται «15».
    - Γίνεται ο έλεγχος συνθήκης (`8<8`). Είναι ΨΕΥΔΕΣ, άρα τερματίζει η επανάληψη.

Β. Δομές Επανάληψης

5. Συμπεράσματα

1. Προτεινόμενη Χρήση των Δομών Επανάληψης

- Βλέπουμε ότι και οι 3 εντολές επανάληψης με παρόμοιο τρόπο κάνουν τις ίδιες ενέργειες.
  - Η πιο συνηθισμένη δομή είναι η **for** και την χρησιμοποιούμε όταν ξέρουμε ποιες τιμές θα πάρει η μεταβλητή.
  - Αν δεν ξέρουμε ακριβώς ποιες τιμές θα πάρει ή μεταβλητή ή πόσες φορές πρέπει να γίνει η επανάληψη, τότε χρησιμοποιούμε την δομή **while**.
  - Η δομή **do…while** χρησιμοποιείται πιο σπάνια, κυρίως για αμυντικό προγραμματισμό σε διάβασμα μεταβλητών.

Β. Δομές Επανάληψης

5. Συμπεράσματα

2. Προσομοίωση της for από την while και την do..while

- Ενδιαφέρον επίσης έχει ότι η εντολή for μπορεί να προσομοιωθεί από τις άλλες δύο ως ακολούθως:

```
for (i=1; i<=10; i++)
{
   (Εντολή ή εντολές)
}
```

- Με εντολή while

```
i=1;
while(i<=10)
{
   (Εντολή ή εντολές)
   i=i+1;
}
```

- Με εντολή do…while

```
i=1;
do{
   (Εντολή ή εντολές)
   i=i+1
}while(I<=10);
```


Γ. Ασκήσεις

Εφαρμογή 1 (Άθροισμα και Γινόμενο Αριθμών)

- (α)Τι κάνει το ακόλουθο πρόγραμμα;
- (β) Πως μπορούμε να τροποποιήσουμε το πρόγραμμά μας, έτσι ώστε να προστίθενται 10 αριθμοί αντί για 3.
- (γ) Γράψτε το πρόγραμμα ask1_ginomeno.c που θα υπολογίζει το γινόμενο των 3 αριθμών που διαβάζει από την είσοδο.

```
/* ask1.c */
#include <stdio.h>

main()
{
   int i,sum,x;
   sum=0;

   for(i=1; i<=3; i++)
   {
      printf("\nEisagete ton %d-o arithmo: ",i);
      scanf("%d",&x);
      sum=sum+x;
   }
   printf("\n\n%d",sum);
}
```


Γ. Ασκήσεις

Εφαρμογή 2 (Εμφωλιασμένοι Βρόχοι)

- Μεταγλωττίστε, εκτελέστε και μελετήστε το παρακάτω πρόγραμμα:

```
/* ask2.c */
#include <stdio.h>

main()
{
   int i,j;

   for(i=1; i<=4; i++)
   {
      for (j=1; j<=4; j++)
      {
         printf("\n%d+%d=%d",i,j,i+j);
      }
   }
}
```

- **Σημείωση:** Επειδή έχουμε επανάληψη μέσα στην επανάληψη, η παραπάνω δομή χαρακτηρίζεται προγραμματιστικά **«εμφωλιασμένοι βρόχοι»** (**nested loops**)

Γ. Ασκήσεις
Εφαρμογή 3 (Εμφωλιασμένοι Βρόχοι)

- (α) Χωρίς να εκτελέσετε το πρόγραμμα, υπολογίστε πόσα X θα εκτυπώσει το ακόλουθο πρόγραμμα και με ποια μορφή;
- (β) Εκτελέστε το πρόγραμμα και τροποποιήστε κατά βούληση το παραλληλόγραμμο που εκτυπώνεται.
- (γ) Μετατρέψτε το πρόγραμμα σε ένα ισοδύναμο που χρησιμοποιεί την εντολή while, αντί για την εντολή for.

```
/* ask3.c */

#include <stdio.h>

main()
{
   int M,N;
   int i,j;

   M=10;
   N=20;

   for(i=0; i<=M; i++)
   {
      for (j=0; j<=N; j++)
      {
         printf("X");
      }
      printf("\n");
   }
}
```


Γ. Ασκήσεις
Εφαρμογή 4 (Εμφωλιασμένοι Βρόχοι)

- (α) Χωρίς να εκτελέσετε το πρόγραμμα, υπολογίστε πόσα X θα εκτυπώσει το ακόλουθο προγραμμα και με ποια μορφή;
- (β) Εκτελέστε το πρόγραμμα και τροποποιήστε κατά βούληση το “τρίγωνο” που εκτυπώνεται.
- (γ) Μετατρέψτε το πρόγραμμα σε ένα ισοδύναμο που χρησιμοποιεί την εντολή do…while, αντί για την εντολή for.

```
/* ask4.c */

#include <stdio.h>

main()
{
   int N;
   int i,j;

   N=10;

   for(i=0; i<=N; i++)
   {
      for (j=i; j<=N; j++)
      {
         printf("X");
      }
      printf("\n");
   }
}
```


Γ. Ασκήσεις
Εφαρμογή 5 (Άθροισμα Αριθμών με Πίνακα)

- Κατασκευάστε πρόγραμμα που:
  - Προτρέπει τον χρήστη να εισάγει 10 ακέραιους αριθμούς και τους αποθηκεύει σε έναν μονοδιάστατο πίνακα 10 θέσεων.
  - Έπειτα υπολογίζει το άθροισμα τους και το τυπώνει στην οθόνη.

Γ. Ασκήσεις
Εφαρμογή 6 (Γινόμενο Αριθμών με Πίνακα)

- Κατασκευάστε πρόγραμμα που:
  - Προτρέπει τον χρήστη να εισάγει 5 άκεραιους αριθμούς και τους αποθηκεύει σε έναν μονοδιάστατο πίνακα 5 θέσεων. Επίσης να εφαρμόζεται αμυντικός προγραμματισμός, έτσι ώστε κάθε ακέραιος που εισάγει ο χρήστης να έχει τιμή από 1 εώς 8.
  - Έπειτα υπολογίζει το γινόμενό τους και το τυπώνει στην οθόνη.

Γ. Ασκήσεις
Εφαρμογή 7 (Ελάχιστος από Ν αριθμούς)

- Κατασκευάστε πρόγραμμα που:
  - Προτρέπει τον χρήστη να εισάγει έναν αριθμό Ν. Ο αριθμός Ν να είναι μεταξύ 1 και 20 με εφαρμογή αμυντικού προγραμματισμού.
  - Έπειτα να διαβάζει από την είσοδο και να εισάγει Ν αριθμούς σε έναν μονοδιάστατο πίνακα.
  - Τέλος, να βρίσκει και να τυπώνει τον ελάχιστο από τους Ν αριθμούς.

- **Υπόδειξη:** Όταν δεν ξέρουμε εκ των προτέρων το μέγεθος του πίνακα που θα χρησιμοποιήσουμε, δεσμεύουμε προκαταβολικά τον μέγιστο χώρο που μπορεί το πρόγραμμα να χρησιμοποιήσει.
  - Για παράδειγμα σε αυτήν την άσκηση, θα πρέπει να κατασκευάσετε έναν πίνακα 20 θέσεων, παρόλο που το πρόγραμμα θα χρησιμοποιήσει τόσες θέσεις, όσο το Ν που θα εισάγει ο χρήστης.
  - Σε επόμενο μάθημα θα μάθουμε πως δεσμεύεται ο χώρος στην μνήμη δυναμικά. Δηλαδή κατά τον χρόνο εκτέλεσης να δεσμεύεται ο χώρος μνήμης που απαιτείται.

Γ. Ασκήσεις
Εφαρμογή 8 (Μέσος όρος Ν αριθμών)

- Κατασκευάστε πρόγραμμα που:
  - Προτρέπει τον χρήστη να εισάγει έναν αριθμό Ν. Ο αριθμός Ν να είναι μεταξύ 1 και 20 με εφαρμογή αμυντικού προγραμματισμού.
  - Έπειτα να διαβάζει από την είσοδο και να εισάγει Ν αριθμούς σε έναν μονοδιάστατο πίνακα.
  - Τέλος, να βρίσκει και να τυπώνει τον μέσο όρο των Ν αριθμών.

- **Υπόδειξη:** Όταν δεν ξέρουμε εκ των προτέρων το μέγεθος του πίνακα που θα χρησιμοποιήσουμε, δεσμεύουμε προκαταβολικά τον μέγιστο χώρο που μπορεί το πρόγραμμα να χρησιμοποιήσει.
  - Για παράδειγμα σε αυτήν την άσκηση, θα πρέπει να κατασκευάσετε έναν πίνακα 20 θέσεων, παρόλο που το πρόγραμμα θα χρησιμοποιήσει τόσες θέσεις, όσο το Ν που θα εισάγει ο χρήστης.
  - Σε επόμενο μάθημα θα μάθουμε πως δεσμεύεται ο χώρος στην μνήμη δυναμικά. Δηλαδή κατά τον χρόνο εκτέλεσης να δεσμεύεται ο χώρος μνήμης που απαιτείται.