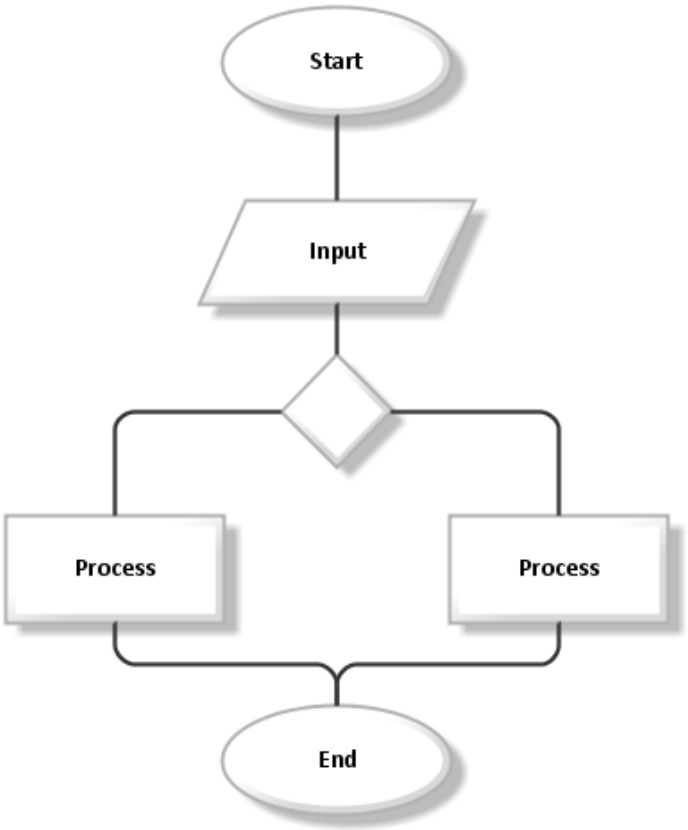
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

Δομημένος Προγραμματισμός με C

Διδακτικές Σημειώσεις για το Μάθημα
Δομημένος Προγραμματισμός



Γουλιάνας Κώστας
Αναπληρωτής Καθηγητής

Θεσσαλονίκη 2019

ΠΡΟΛΟΓΟΣ

Οι Σημειώσεις που ακολουθούν δε φιλοδοξούν να αποτελέσουν ένα ολοκληρωμένο σύγγραμμα. Αποτελούν μια πιο αναλυτική καταγραφή των παραδόσεων στα πλαίσια του μαθήματος του Α' Εξαμήνου, "Δομημένος Προγραμματισμός", του Τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙΠΑΕ. Στόχος των παραδόσεων και των σημειώσεων που ακολουθούν μέσα από πολλά παραδείγματα είναι να μάθουν οι πρωτοετείς φοιτητές τι σημαίνει προγραμματισμός, τι είναι και πώς σχεδιάζεται ένας αλγόριθμος ή ένα λογικό διάγραμμα, τι σημαίνουν και πώς χρησιμοποιούνται οι δομές-εντολές ελέγχου και επανάληψης, τι είναι και πώς χρησιμοποιούνται οι συναρτήσεις, τι είναι παράμετροι και πώς χρησιμοποιούνται. Γίνεται επίσης μια εισαγωγή σε μεθόδους αναζήτησης και ταξινόμησης. Η `C` χρησιμοποιείται σαν εργαλείο κατανόησης των βασικών προγραμματιστικών δομών. Μετά το $7^{ο}$ κεφάλαιο γίνεται μια εισαγωγή στη δομή `struct`, ώστε να είναι εύκολο στους φοιτητές να κατανοήσουν και τις κλάσεις και τα αντικείμενα, τα οποία θα διδαχθούν στο επόμενο εξάμηνα. Στόχος μας είναι οι φοιτητές, μετά το τέλος του εξαμήνου, να είναι σε θέση να σχεδιάζουν μια ολοκληρωμένη εφαρμογή, με τα απαραίτητα πεδία και τις συναρτήσεις επεξεργασίας δεδομένων.

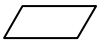
Πίνακας Περιεχομένων

1 ΕΙΣΑΓΩΓΗ – ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Αλγόριθμος

Μια σειρά από σαφή και καθορισμένα βήματα, τα οποία οδηγούν στη λύση ενός προβλήματος, περιγραφή του κάθε βήματος με λόγια και λέξεις-κλειδιά, π.χ. **διάβασε**, **υπολόγισε**, **εμφάνισε**, **αν**, **διαφορετικά**, **για όσο**, **για**.

Διάγραμμα Ροής (Λογικό Διάγραμμα)

Μια σειρά από σαφή και καθορισμένα βήματα, τα οποία οδηγούν στη λύση ενός προβλήματος με ειδικά σχήματα για την κάθε ενέργεια π.χ. **ορθογώνιο**  για την ανάθεση τιμής, **ρόμβος**  για έλεγχο ή επανάληψη, **παραλληλόγραμμο**  για εισαγωγή δεδομένων, ταινία  για εμφάνιση αποτελεσμάτων κ.λ.π..

Πρόγραμμα

Μετατροπή των παραπάνω βημάτων σε **εντολές** που μπορούν να μεταφραστούν από ένα πρόγραμμα στον Ηλεκτρονικό Υπολογιστή (Μεταγλωττιστής ή Διερμηνέας μιας Γλώσσας Προγραμματισμού), με λέξεις-κλειδιά, π.χ. **read**, **print**, **if**, **else**, **for**, **while**.

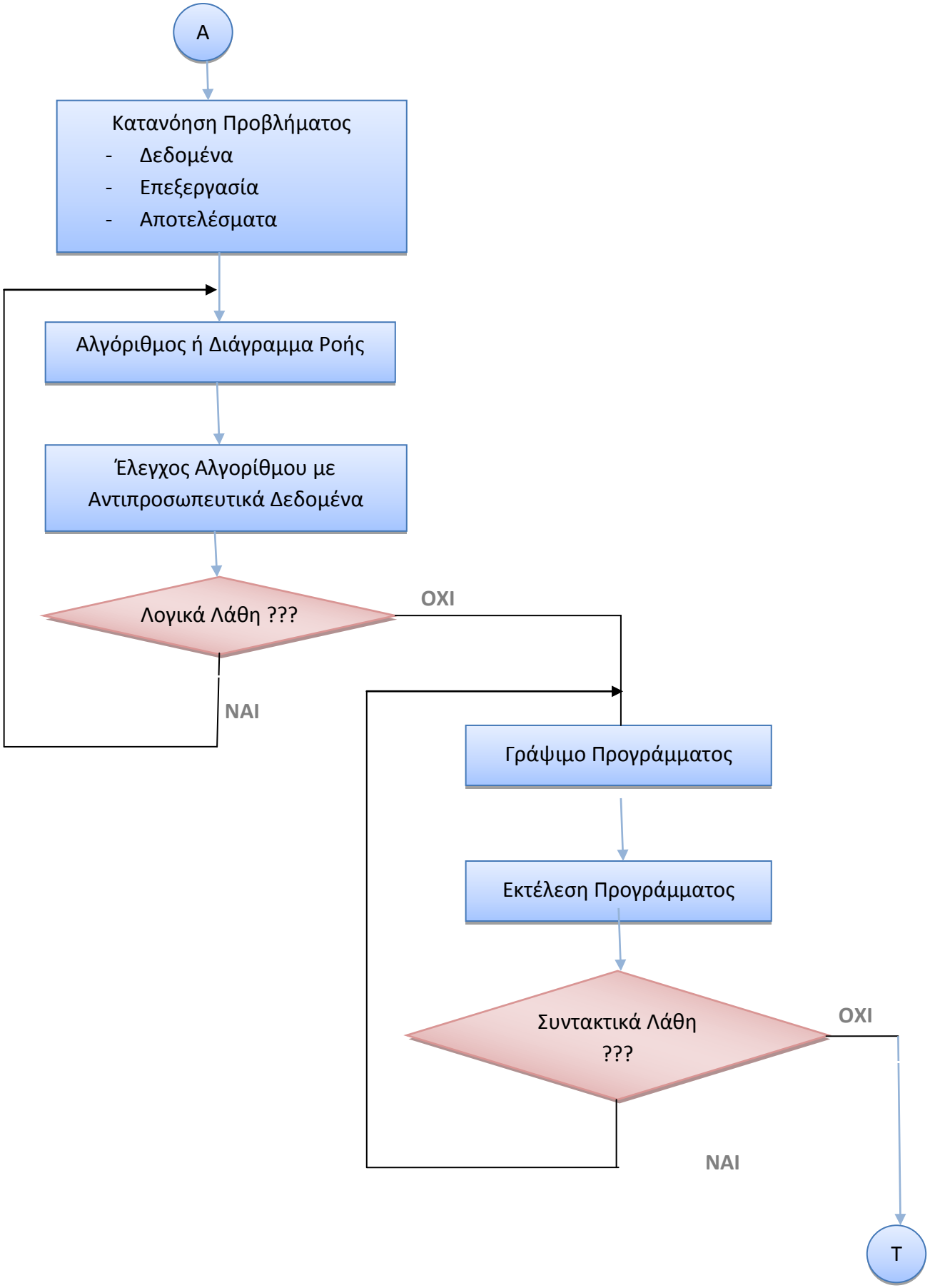
Γλώσσες Προγραμματισμού

Γλώσσα Μηχανής → Συμβολική Γλώσσα (ASSEMBLY) → Γλώσσες Υψηλού Επιπέδου (BASIC, FORTRAN, COBOL, PASCAL, C, Dephi, Visual Basic, C++, Java).

Μεταβλητές

Ονόματα στα αγγλικά, τα οποία αντιπροσωπεύουν θέσεις μνήμης, στις οποίες θα αποθηκεύονται δεδομένα, αριθμοί, χαρακτήρες κ.λ.π.. Συνήθως τα ονόματα που επιλέγονται έχουν σχέση με την ποσότητα που θα αποθηκεύσουν, π.χ. **num** για κάποιον αριθμό, **sum** για άθροισμα, **mo** για το Μέσο Όρο.

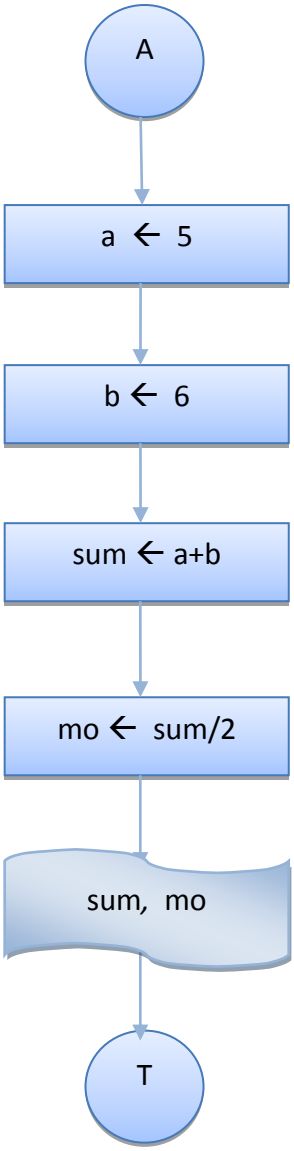
ΔΙΑΓΡΑΜΜΑ ΕΡΓΑΣΙΩΝ ΓΙΑ ΜΕΘΟΔΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ



1. 1 Ένα Απλό Πρόγραμμα

Να γραφεί πρόγραμμα, το οποίο θα δίνει τις τιμές 5 και 6 σε δύο μεταβλητές **a** και **b** και θα υπολογίζει και θα εμφανίζει το άθροισμά τους **sum** και το μέσο όρο **mo**.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Δίνω την τιμή 5 στο a (**a ← 5**)
2. Δίνω την τιμή 6 στο b (**b ← 6**)
3. Βρίσκω το άθροισμα (**sum ← a + b**)
4. Βρίσκω τον μέσο όρο (**mo ← sum/2**)
5. Εμφανίζω τις τιμές των **sum, mo**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/*
Πρόγραμμα που δίνει τις τιμές 5 και 6 σε 2 ακέραιες μεταβλητές i1, i2 και
βρίσκει και εμφανίζει το άθροισμά τους isum και το Μέσο Όρο dmo, ο οποίος
είναι μεταβλητή τύπου double.
*/

        // Δήλωση των ακέραιων μεταβλητών i1, i2
        int i1, i2;

        // Δήλωση της ακέραιας μεταβλητής isum για το άθροισμα
        int isum;

        // Δήλωση της πραγματικής μεταβλητής mo για το Μέσο Όρο
        double dmo;

        // Ανάθεση των τιμών 5 και 6 στις ακέραιες μεταβλητές i1, i2
        i1 = 5;
        i2 = 6;

        // Υπολογισμός του αθροίσματος isum
        isum = i1 + i2;

        // Υπολογισμός του Μέσου Όρου mo
        dmo = isum/2;

        // Εμφάνιση  του αθροίσματος sum και του Μέσου Όρου dmo
        printf ("isum = %d dmo = %lf\n", isum, dmo);
        system("Pause");

}
```

Έξοδος Προγράμματος

```
isum = 11 dmo = 5.000000
Press any key to continue . . .
```

- Μέσος Όρος dmo = 5, γιατί γίνεται ακέραια διαίρεση του 11/2 και αποκόπτεται το δεκαδικό μέρος.

ΠΑΡΑΤΗΡΗΣΕΙΣ

1. **`Η main()`** είναι η βασική συνάρτηση, το κυρίως πρόγραμμα, δεν είναι απαραίτητο να επιστρέφει κάποια τιμή.
2. Οι αγκύλες **`{}`** πηγαίνουν ανά ζεύγη και περικλείουν αυτόνομα κομμάτια κώδικα.
3. Όλες οι εντολές τελειώνουν με το ελληνικό ερωτηματικό "**`;`**".
4. Τα Σχόλια πολλών γραμμών αρχίζουν με το σύμβολο "**`/*`**" και τελειώνουν με το σύμβολο "**`*/`**".
5. Τα Σχόλια μιας γραμμής ή μετά από μια εντολή αρχίζουν με το σύμβολο "**`//`**".
6. Οι μεταβλητές πρέπει να δηλώνουν κάποιο τύπο ανάλογα με την ποσότητα που θα αποθηκεύσουν, π.χ. `int i1, i2` για τους αριθμούς, `double dmo` για το Μέσο Όρο.
7. Τα ονόματα των μεταβλητών μπορούν να αρχίζουν από οποιοδήποτε γράμμα ή τα σύμβολα "**`_`**" "**`$`**", αλλά <u>όχι</u> από ψηφίο. Π.χ. `i1` **<u>και όχι</u>** `1i`.
8. Η δήλωση μιας μεταβλητής περιλαμβάνει τον τύπο των δεδομένων που θα αποθηκεύσει και το όνομά της.

   Παράδειγμα

   ```
   double dmo; // Δήλωση της πραγματικής μεταβλητής dmo για το Μέσο Όρο
   ```

9. Μια δήλωση μπορεί να περιλαμβάνει περισσότερες από μια μεταβλητές.

   Παράδειγμα

   ```
   int i1, i2; // Δήλωση των ακέραιων μεταβλητών i1, i2
   ```

1.2 ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Τα δεδομένα που μπορούν να αποθηκευθούν σε μια μεταβλητή μπορεί να είναι ακέραιοι αριθμοί (π.χ. ο αριθμός `5`), πραγματικοί αριθμοί ή αριθμοί κινητής υποδιαστολής (π.χ. ο αριθμός `5.5`), χαρακτήρες (π.χ. το γράμμα `X`).

Οι **ακέραιες** μεταβλητές ανάλογα με τα `bits` που περιλαμβάνουν και το μέγιστο αριθμό που μπορούν να αποθηκεύσουν φαίνονται στον επόμενο πίνακα :

Τύπος	**bits**	**Ελάχιστη Τιμή**	**Μέγιστη Τιμή**
`short`	16	-32768	32767
`int`	32	-2147483648	2147483647
`unsigned int`	32	0	4294967295
`long int`	32	-2147483648	2147483647

Οι **πραγματικές** (**Κινητής Υποδιαστολής**) μεταβλητές ανάλογα με τα bits που περιλαμβάνουν συνολικά, τα bits της mantissa και το μέγιστο αριθμό που μπορούν να αποθηκεύσουν φαίνονται στον επόμενο πίνακα :

Τύπος	Συνολικά bits	bits mantissa	Ελάχιστη -	Μέγιστη Τιμή
float	32	23	$-3.4*10^{38}$	$3.4*10^{38}$
double	64	52	$-1.7*10^{308}$	$1.7*10^{308}$
long double	80	65	$-1.1*10^{4932}$	$1.1*10^{4932}$

❖ Όλες οι συναρτήσεις δέχονται double παραμέτρους.

Το σύνολο **χαρακτήρων** της C είναι το ASCII. Σε μεταβλητές τύπου char μπορεί να αποθηκευθεί ένας χαρακτήρας ή το αριθμητικό του ισοδύναμο.

```
char ch;     // Δήλωση μεταβλητής που θα αποθηκεύσει χαρακτήρες

ch = 'X';    // Ανάθεση τιμής σε μεταβλητή που αποθηκεύει το
               χαρακτήρα X

ch = 88;      // Ανάθεση τιμής σε μεταβλητή που αποθηκεύει το
                αριθμητικό ισοδύναμο του χαρακτήρα X
```

Οι λογικές (**boolean**) μεταβλητές αποθηκεύουν τις τιμές **true-false**, σαν το αποτέλεσμα κάποιας σύγκρισης. Χρησιμοποιούνται σε εντολές ελέγχου και επανάληψης.

```
bool b; // Δήλωση της λογικής μεταβλητής b
b = true; // Ανάθεση της τιμής true  στη λογική μεταβλητή b
```

ΚΥΡΙΟΛΕΚΤΙΚΕΣ ΣΤΑΘΕΡΕΣ

Πολλές φορές χρειάζεται να χρησιμοποιήσουμε συγκεκριμένους αριθμούς, χαρακτήρες ή ονόματα σε εντολές εκχώρησης, σε εκφράσεις ή σε συγκρίσεις, όπως στο προηγούμενο παράδειγμα που δώσαμε στη μεταβλητή i1 την τιμή 5, τα οποία χαρακτηρίζονται σαν κυριολεκτικές σταθερές και μπορεί να είναι :

- Σταθερές ακεραίων, π.χ. 10, -100. Είναι εξ ορισμού τύπου int. Αν θέλουμε να ορίσουμε σταθερά τύπου long, βάζουμε ένα L ή l στο τέλος, π.χ. long a = 10L.
- Σταθερές κινητής υποδιαστολής, π.χ. 2.35. Είναι εξ ορισμού τύπου double. Για να ορίσουμε σταθερά τύπου float , βάζουμε ένα F ή f στο τέλος, π.χ. float a=2.35F.

ΑΡΧΙΚΟΠΟΙΗΣΗ ΜΕΤΑΒΛΗΤΩΝ

Μαζί με τη δήλωση κάποιας μεταβλητής μπορούμε να εκχωρήσουμε και κάποια αρχική τιμή.

Παράδειγμα

```
int a;   // Δήλωση Μεταβλητής a
a = 10; // Ανάθεση του 10 σαν αρχική τιμή στη μεταβλητή a
```

Το ίδιο αποτέλεσμα έχουμε με την εντολή

```
int a = 10;   // Δήλωση - Ανάθεση του 10 σαν αρχική τιμή στη μεταβλητή a
```

- Αρχικοποίηση μεταβλητών μπορεί να γίνει σε περισσότερες από 1 μεταβλητές.

Παράδειγμα

```
int a = 5, b = 6, sum; // Αρχικοποίηση a, b, δήλωση sum
```

ΔΥΝΑΜΙΚΗ ΑΡΧΙΚΟΠΟΙΗΣΗ

Μπορεί να γίνει δήλωση μιας μεταβλητής στο σημείο του προγράμματος που θέλουμε να **υπολογίσουμε** και να **αποθηκεύσουμε** σε μια μεταβλητή το αποτέλεσμα μιας αριθμητικής έκφρασης.

Παράδειγμα

```
int sum = a + b;
double mo = sum/2;
```

- Προσοχή, η μεταβλητή αν υπάρχει σε `block` (`if`, `while`, `for`) είναι τοπική, έχει εμβέλεια μόνο στο `block`.

ΑΝΑΘΕΣΗ-ΕΚΧΩΡΗΣΗ ΤΙΜΗΣ

Γενικός Τύπος : `<όνομα_μεταβλητής> = <έκφραση>;`

Παράδειγμα

```
a = 10; // Ανάθεση του 10 σαν αρχική τιμή στη με ταβλητή a
```

- Υπάρχει η δυνατότητα στη `C` να έχουμε αλυσίδα εκχωρήσεων.

Παράδειγμα

```
x = y = z = 10; // Το z παίρνει την τιμή 10, την οποια παίρνει το y και μετά το x
```

ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

Οι τελεστές πράξεων που χρησιμοποιούνται στις αριθμητικές εκφράσεις είναι οι παρακάτω :

`+, -, *` : Τελεστές για πρόσθεση, αφαίρεση και πολλαπλασιασμό, όπως και στα μαθηματικά.

`/` : Τελεστής για Διαίρεση

❖ Με ακέραιους γίνεται ακέραια διαίρεση

Παράδειγμα

```
int a = 10;
int b = a/3;        // Αποτέλεσμα της διαίρεσης → b = 3

float a = 10.0f;
float b = a/3;      // Αποτέλεσμα της διαίρεσης → b = 3.333….
```

`%` : Τελεστής για το υπόλοιπο της διαίρεσης 2 αριθμών, ακέραιων ή κινητής υποδιαστολής.

Παράδειγμα

```
int a = 10;
int b = a%3;   //  Αποθηκεύεται στη μεταβλητή b το υπόλοιπο της διαίρεσης a/3 = 1

float a = 10.5f;
float b = a%3; //Αποθηκεύεται στη μεταβλητή b το υπόλοιπο της διαίρεσης a/3 =  1.5
```

1. 3 Είσοδος Δεδομένων – Εμφάνιση Τιμών Μεταβλητών

Η είσοδος δεδομένων απ' το πληκτρολόγιο και η εμφάνιση των αποτελεσμάτων στην οθόνη μπορεί να γίνει με τις συναρτήσεις – εντολές **`scanf`** (scan formatted) και **`printf`** (print formatted) .

1. 3.1 Εμφάνιση Τιμών Μεταβλητών με Προκαθορισμένη Μορφή

- Η γενική σύνταξη της εντολής `printf` είναι :

```
printf ("μηνύματα - προσδιοριστές", <όρισμα-1>, <όρισμα-2>,…, <όρισμα-ν>)΄;
```

❖ Στα διπλά εισαγωγικά περιέχονται **μηνύματα** που θέλουμε να εμφανισθούν, καθώς και **ειδικές οδηγίες** για τη μορφή που θα έχουν τα περιεχόμενα των μεταβλητών τις τιμές των οποίων θα διαβάσουμε ή θα εμφανίσουμε. Οι οδηγίες αυτές ξεκινούν με το χαρακτήρα '**%**', ο οποίος ακολουθείται από κάποιους **προσδιοριστές**, οι οποίοι μπορεί να είναι :

Προσδιοριστής	**Είδος**
`%hd`	Εμφανίζει την τιμή της **short int** μεταβλητής
`%d, %i`	Εμφανίζει την τιμή της **ακέραιας signed int** μεταβλητής
`%ud`	Εμφανίζει την τιμή της **unsigned int** μεταβλητής
`%ld`	Εμφανίζει την τιμή της **long int** μεταβλητής
`%f`	Εμφανίζει την τιμή της μεταβλητής **float** στη μορφή ddd.dddddd
`%lf`	Εμφανίζει την τιμή της μεταβλητής **double** στη μορφή ddd.dddddd
`%e`	Εμφανίζει την τιμή της μεταβλητής **float** ή **double** σε **εκθετική** μορφή 1.dddddd**E**+ddd
`%Lf`	Εμφανίζει την τιμή της μεταβλητής **long double** στη μορφή ddd.dddddd
`%c`	Εμφανίζει την τιμή της μεταβλητής τύπου char
`%s`	Εμφανίζει την τιμή της μεταβλητής τύπου string

Παράδειγμα

Με τις παρακάτω εντολές :

```
short int s1 = 5;
int i1 = 12;
int i2 = 13;
unsigned int ui1 = 14;
float f1 = 0.12345678901234567890;
double d1 = 0.12345678901234567890;
printf("short int s1 = %hd, int i1 = %d, int i2 = %i,
unsigned int ui1 = %u\nfloat f1 = %f, double d1 = %lf d1 =
%e\n", s1, i1, i2, ui1, f1, d1, d1);
```

Θα εμφανιστεί :

```
short int s1 = 5, int i1 = 12, int i2 = 13, unsigned int ui1 = 14
float f1 = 0.123457, double d1 = 0.123457 d1 = 1.234568e-001
```

Παρατηρήσεις

- Τα δεδομένα εκτυπώνονται σε προδιαγεγραμμένη μορφή.
- Οι ακέραιοι αριθμοί καταλαμβάνουν τόσες θέσεις, όσες χρειάζονται.
- Οι πραγματικοί αριθμοί εκτυπώνονται σε δεκαδική ή σε εκθετική μορφή.

- Οι αριθμοί απλής και διπλής ακρίβειας καταλαμβάνουν όσες θέσεις χρειάζονται, με 6 δεκαδικά ψηφία μετά την υποδιαστολή. Αν ο αποθηκευμένος αριθμός έχει περισσότερα από 6 δεκαδικά ψηφία, γίνεται στρογγυλοποίηση σε 6 δεκαδικά ψηφία (π.χ. ο αριθμός 0.12345678901234567890 με 20 δεκαδικά ψηφία εμφανίζεται σαν 0.123457). Αν ο αποθηκευμένος αριθμός δεν χρειάζεται και τις 6 θέσεις οι επί πλέον δεξιές θέσεις συμπληρώνονται με μηδενικά.

- Οι αριθμοί απλής και διπλής ακρίβειας όταν εμφανίζονται σε εκθετική μορφή (κωδικός %e) εμφανίζονται στη μορφή 1.dddddde±<εκθέτης>, όπου τα δεκαδικά ψηφία της mantissa καταλαμβάνουν 6 θέσεις και ο εκθέτης παίρνει τιμές μέχρι το 999.

1. 3.2 Εμφάνιση Τιμών Μεταβλητών με τη Μορφή που επιθυμούμε

Αν θέλουμε να εμφανιστούν τα δεδομένα με τη μορφή που θέλουμε, στον κωδικό της εμφάνισης, μεταξύ του συμβόλου % και του προσδιοριστή, μπορούμε να βάλουμε έναν ή δύο αριθμούς, με τους οποίους καθορίζουμε το μέγιστο και τον αριθμό των δεκαδικών ψηφίων που θέλουμε να καταλαμβάνει ο αριθμός. Έτσι, αν χρησιμοποιήσουμε τις εντολές :

```
short int s1 = 5;
int i1 = 12;
int i2 = 13;
unsigned int ui1 = 14;
float f1 = 0.12345678901234567890;
double d1 = 0.12345678901234567890;
printf("short int s1 = %4hd, int i1 = %5d, int i2 = %6i,
unsigned int ui1 = %3u\nfloat f1 = %.7f, double d1 = %20.16lf
d1 = %.4e\n", s1, i1, i2, ui1, f1, d1, d1);
```

θα εμφανιστεί :

```
short int s1 = ^^^^5, int i1 = ^^^^^12, int i2 = ^^^^^^13,
unsigned int ui1 = ^^^14
float f1 = 0.1234568, double d1 = 0.123457890123457 d1 =
1.2346e-001
```

Παρατηρήσεις

- Τα δεδομένα εκτυπώνονται με τη μορφή που επιθυμεί ο χρήστης.

- Οι ακέραιοι αριθμοί καταλαμβάνουν τόσες θέσεις, όσες και ο αριθμός που υπάρχει μεταξύ % και d. Αν δεν χρειάζονται όλες οι θέσεις, ο αριθμός εμφανίζεται στοιχημένος δεξιά, ενώ οι επί πλέον αριστερές θέσεις συμπληρώνονται με κενά (^^^^).

- Οι αριθμοί απλής και διπλής ακρίβειας καταλαμβάνουν συνολικά με την υποδιαστολή τόσες θέσεις όσο και ο πρώτος αριθμός (20 στο παράδειγμα), με τόσα δεκαδικά ψηφία μετά την υποδιαστολή όσο και ο δεύτερος αριθμός (16 στο παράδειγμα). Αν ο αποθηκευμένος αριθμός έχει περισσότερα δεκαδικά ψηφία, γίνεται

στρογγυλοποίηση. Αν ο αποθηκευμένος αριθμός δεν χρειάζεται τόσα δεκαδικά ψηφία, οι επί πλέον δεξιές θέσεις συμπληρώνονται με μηδενικά, ενώ οι επί πλέον αριστερές θέσεις συμπληρώνονται με κενά.

- Αν θέλουμε να εμφανιστούν τα δεδομένα με αριστερή στοίχιση, στον κωδικό της εμφάνισης, πριν τον ή τους δύο αριθμούς, βάζουμε το '–'. Έτσι, αν χρησιμοποιήσουμε τις εντολές :

```
short int s1 = 5;
int i1 = 12;
int i2 = 13;
unsigned int ui1 = 14;
float f1 = 0.12345678901234567890;
double d1 = 0.12345678901234567890;
printf("short int s1 = %-4hd, int i1 = %-5d, int i2 = %-6i,
unsigned int ui1 = %-3u\nfloat f1 = %-.7f, double d1 = %-
20.16lf d1 = %-.4e\n", s1, i1, i2, ui1, f1, d1, d1);
```

θα εμφανιστεί :

```
short int s1 = 5^^^^, int i1 = 12^^^^^, int i2 = 13^^^^^^,
unsigned int ui1 = 14^^^
float f1 = 0.1234568, double d1 = 0.123457890123457^^^d1 =
1.2346e-001
```

ΧΑΡΑΚΤΗΡΕΣ ΕΛΕΓΧΟΥ

- Στην εντολή `printf` αν υπάρχει το `\n` πριν κλείσουν τα διπλά εισαγωγικά, η επόμενη εκτύπωση θα γίνει στην επόμενη γραμμή. Αυτός ο χαρακτήρας δεν εκτυπώνεται, αλλά χρησιμοποιείται σαν "οδηγός" για το πού θα γίνει η επόμενη εκτύπωση και είναι σαν χαρακτήρας ελέγχου. Οι χαρακτήρες ελέγχου είναι οι παρακάτω :

Χαρακτήρας	Αποτέλεσμα
κανένας	Εκτύπωση στην ίδια γραμμή.
\n	Εκτύπωση στην επόμενη γραμμή.
\t	Εκτύπωση στην ίδια γραμμή μετά ένα tab.

1. 3.3 Εισαγωγή Τιμών Μεταβλητών με Προκαθορισμένη Μορφή

- Η γενική σύνταξη της εντολής `scanf` είναι :

```
scanf ("μηνύματα - προσδιοριστές ", &<όρισμα-1>, &<όρισμα-2>,…, &<όρισμα-ν>)΄;
```

❖ Στα διπλά εισαγωγικά περιέχονται **ειδικές οδηγίες** για τη μορφή που θα έχουν τα περιεχόμενα των μεταβλητών τις τιμές των οποίων θα διαβάσουμε ή θα εμφανίσουμε. Οι οδηγίες αυτές ξεκινούν με το χαρακτήρα '**%**', ο οποίος ακολουθείται από κάποιους **προσδιοριστές**, όπως στην εντολή `printf`.

❖ Πριν από κάθε όρισμα υπάρχει το σύμβολο **&** που σημαίνει ότι περνάμε τη **διεύθυνση** της **μεταβλητής** στην οποία θα αποθηκευθούν τα δεδομένα που διαβάζονται απ' το πληκτρολόγιο.

❖ Στην εκτέλεση του προγράμματος θα πρέπει να δώσουμε τόσες τιμές, όσα και τα ορίσματα στην εντολή `scanf`, χωρισμένες με κενό ή Enter.

❖ Αν θέλουμε να διαβαστούν τα δεδομένα με τη μορφή που θέλουμε, στον κωδικό της εισαγωγής δεδομένων, μεταξύ του συμβόλου `%` και του προσδιοριστή, μπορούμε να βάλουμε έναν ή δύο αριθμούς, με τους οποίους καθορίζουμε το μέγιστο και τον αριθμό των δεκαδικών ψηφίων που θέλουμε να καταλαμβάνει ο αριθμός που θα εισάγουμε.

Παράδειγμα :

```
int a1, b1;
printf("give a1, b1 : ");
scanf("%d %d", &a1, &b1);
```

1. 4 Πρώτη Τροποποίηση Προγράμματος 1.1

Να τροποποιηθεί το πρόγραμμα 1.1, το οποίο θα δίνει τις τιμές `5` και `6` σε δύο μεταβλητές `i1` και `i2` και θα υπολογίζει και θα εμφανίζει το άθροισμά τους `d`**`sum`** και τον Μέσο Όρο `d`**`mo`**, ώστε να μπορεί να εμφανίζει το σωστό αποτέλεσμα (η μεταβλητή `dsum` να δηλωθεί τύπου `double`, ώστε να ΜΗΝ γίνει ακέραια διαίρεση στον Μέσο Όρο).

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main()
{
/*
Πρόγραμμα που δίνει τις τιμές 5 και 6 σε 2 ακέραιες μεταβλητές και βρίσκει
και εμφανίζει το άθροισμά τους και τον Μέσο Όρο. Το άθροισμα και ο Μέσος Όρος
είναι μεταβλητές τύπου double
*/

        // Δήλωση των ακέραιων μεταβλητών i1, i2
        int i1, i2;

        // Δήλωση της πραγματικής μεταβλητής dsum για το άθροισμα
        double dsum;

        // Δήλωση της πραγματικής μεταβλητής dmo για το Μέσο Όρο
        double dmo;

        // Ανάθεση των τιμών 5 και 6 στις ακέραιες μεταβλητές i1, i2
        i1 = 5;
        I2 = 6;

        // Υπολογισμός του αθροίσματος dsum
        dsum = i1 + i2;

        // Υπολογισμός του Μέσου Όρου dmo
        dmo = dsum/2;

        // Εμφάνιση  του αθροίσματος sum και του Μέσου Όρου mo
        printf ("dsum = %lf dmo = %lf\n", dsum,dmo);
        system("Pause");

}
```

Έξοδος Προγράμματος

```
dsum = 11.000000 dmo = 5.500000
Press any key to continue . . .
```

- Το Άθροισμα είναι `11.0` και ο Μέσος Όρος είναι `5.5`

ΤΕΛΕΣΤΕΣ ΣΥΝΤΟΜΩΝ ΕΚΧΩΡΗΣΕΩΝ

Υπάρχει η δυνατότητα στη C να έχουμε σύντομες εκχωρήσεις .

Παράδειγμα

```
x = x + y;     Συντομευμένος τελεστής    x += y;
x = x – y;     Συντομευμένος τελεστής    x –= y;
x = x * y;     Συντομευμένος τελεστής    x *= y;
x = x/y;       Συντομευμένος τελεστής    x /= y;
x = x%y        Συντομευμένος τελεστής    x %= y;
```

ΤΕΛΕΣΤΕΣ ΠΡΟΣΑΥΞΗΣΗΣ-ΠΡΟΜΕΙΩΣΗΣ ++, --

Η C διαθέτει τους παρακάτω τελεστές προσαύξησης ή προμείωσης :

```
x = x + 1;    τελεστές προσαύξησης     x++; ++x;
x = x – 1;    τελεστές προμείωσης      x--; --x;
```

- Αν το ++, -- βρίσκεται **πριν** τη μεταβλητή, **πρώτα** ενημερώνεται η τιμή της μεταβλητής και **μετά** χρησιμοποιείται.

Παράδειγμα

```
x = 10;
y = x++;    // Το y παίρνει την τιμή του x που ήταν 10, το x αυξάνεται ΜΕΤΑ και γίνεται
              11 ( x = 11, y = 10 )

x = 10;
y = ++x;    // ΠΡΩΤΑ αυξάνεται το x και γίνεται 11 και ΜΕΤΑ το y παίρνει την τιμή του x
              που έγινε 11 ( x = 11, y = 11 )
```

ΜΕΤΑΤΡΟΠΗ ΤΥΠΩΝ ΣΕ ΕΚΧΩΡΗΣΕΙΣ

Όταν σε μια αριθμητική έκφραση έχουμε μεταβλητές διαφορετικού τύπου γίνεται **διευρυμένη μετατροπή**, όπου ο τύπος της μεταβλητής στο δεξί μέρος της έκφρασης μετατρέπεται στον τύπο της μεταβλητής στο αριστερό μέρος της έκφρασης. Η μετατροπή γίνεται από το μικρότερο τύπο στο μεγαλύτερο με την παρακάτω σειρά:

`short → int → long → float → double`

Για να γίνει διευρυμένη μετατροπή θα πρέπει ο τύπος προορισμού (αριστερά) να είναι μεγαλύτερος από τους τύπους των μεταβλητών στο δεξί μέρος της έκφρασης. Η διευρυμένη μετατροπή δεν ισχύει για τους τύπους `bool` και `char`.

Εξαίρεση αποτελεί ο τύπος `char`, όπου μπορεί να αποθηκευτεί ένας ακέραιος `int` σε μεταβλητή τύπου `char`.

Παράδειγμα

```
char ch = 88; // Είναι το ίδιο με την εντολή char ch = 'X'
```

Στην περίπτωση που δεν ισχύουν τα παραπάνω ή αν θέλουμε να αποθηκεύσουμε το περιεχόμενο της μεταβλητής κάποιου τύπου σε μεταβλητή διαφορετικού τύπου χρησιμοποιούμε τη **διανομή – casting,** όπου πριν τη **μεταβλητή** ή την **έκφραση** (η οποία πρέπει να είναι μέσα σε **παρενθέσεις**, αν θέλουμε η διανομή να ισχύσει για το αποτέλεσμα της έκφρασης) βάζουμε μέσα σε **παρενθέσεις** τον τύπο στον οποίο θέλουμε να μετατραπεί η τιμή της μεταβλητής ή της έκφρασης. Σ' αυτές τις περιπτώσεις χρειάζεται προσοχή, γιατί μπορεί να μην αποθηκευθεί σωστά η τιμή μιας μεταβλητής σε κάποια άλλη ή να μή χωράει.

Παράδειγμα

```
double d = 10.5;
int i1 = (int)d;          // Αποθηκεύεται στη μεταβλητή i1 το 10, χάνεται το 0.5
int i2 = (int)(d/3);      // Αποθηκεύεται στη μεταβλητή i2 το 3, χάνεται το 0.5
double d1 = (double)i1;   // Αποθηκεύεται στη μεταβλητή d1 το 10.0
```

- Η ακέραια διαίρεση, αν είναι ατελής, χρειάζεται διανομή.

Παράδειγμα

```
int i = 10/3;                  // Αποτέλεσμα i = 3
double d = (double)(10/3);     // Αποτέλεσμα i = 3.0, η διανομή γίνεται στο
                               αποτέλεσμα της ακέραιας διαίρεσης 10/3 = 3
d = (double)10/3;              // Αποτέλεσμα i = 3.333, η διανομή γίνεται
                               στο 10 → μετατροπή σε 10.0/3 = 3.333
```

1. 4 Δεύτερη Τροποποίηση του Προγράμματος 1.1

Να τροποποιηθεί το πρόγραμμα 1.1, το οποίο θα δίνει τις τιμές 5 και 6 σε δύο μεταβλητές **i1** και **i2** και θα υπολογίζει και θα εμφανίζει το άθροισμά τους **isum** και το μέσο όρο **dmo**, ώστε να μπορεί να εμφανίζει το σωστό αποτέλεσμα (**η μεταβλητή isum να δηλωθεί τύπου int, αλλά να γίνει μετατροπή - casting στον υπολογισμό του μέσο όρου, ώστε να ΜΗΝ γίνει ακέραια διαίρεση**).

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/* Πρόγραμμα που δίνει τις τιμές 5 και 6 σε 2 ακέραιες μεταβλητές και βρίσκει
και εμφανίζει το άθροισμά τους και το Μέσο Όρο, ο οποίος είναι μεταβλητή
τύπου double. Η μεταβλητή sum θα δηλωθεί τύπου int, αλλά θα γίνει μετατροπή -
casting στον υπολογισμό του μέσο όρου, ώστε να ΜΗΝ γίνει ακέραια διαίρεση */

        // Δήλωση των ακέραιων μεταβλητών i1, i2
        int i1, i2;

        // Δήλωση της ακέραιας μεταβλητής isum για το άθροισμα
        int isum;

        // Δήλωση της πραγματικής μεταβλητής dmo για το Μέσο Όρο
        double dmo;

        // Ανάθεση των τιμών 5 και 6 στις ακέραιες μεταβλητές i1, i2
        i1 = 5;
        i2 = 6;

        // Υπολογισμός του αθροίσματος isum
        isum = i1 + i2;

          // Υπολογισμός  Μέσου Όρου mo με μετατροπή-casting του isum σε
          double
        dmo = (double)isum/2;

        // Εμφάνιση  του αθροίσματος sum και του Μέσου Όρου mo
        printf ("isum = %d dmo = %lf\n", isum,dmo);
       system("Pause");
}
```

Έξοδος Προγράμματος

```
isum = 11 dmo = 5.500000
Press any key to continue . . .
```

❖ Παρόλο που το άθροισμα είναι ακέραιος (= 11) με τη μετατροπή - casting του αθροίσματος στον υπολογισμό του μέσο όρου (**dmo = (double)isum/2**) ΔΕΝ γίνεται ακέραια διαίρεση του 11/2, αλλά του 11.000000/2, οπότε ΔΕΝ αποκόπτεται το δεκαδικό μέρος και ο Μέσος Όρος = 5.500000.

1.5 Ενσωμάτωση Αρχείων Βιβλιοθηκών

- Η εντολή

```
#include <ονομα_αρχείου_c>
```

ενσωματώνει μόνο κατά τη διάρκεια του χρόνου μεταγλώττισης τα περιεχόμενα του αρχείου <ονομα_αρχείου_c> στο σημείο που έχουμε την εντολή **#include**.

Παραδείγματα

- Οι συναρτήσεις εισόδου και εμφάνισης δεδομένων απαιτούν την ενσωμάτωση του αρχείου που περιέχει τις αντίστοιχες συναρτήσεις βιβλιοθήκης `scanf`, `printf` κ.λ.π., που είναι το `stdio.h`, το οποίο γίνεται με την εντολή :

```
#include <stdio.h>
```

- Η χρήση συναρτήσεων δημιουργίας τυχαίων τιμών ή τα πάγωμα της οθόνης στο τρέξιμο του προγράμματος, το οποίο γίνεται με τη χρήση της εντολής :

```
system("Pause");
```

απαιτούν την ενσωμάτωση του αρχείου που περιέχει τις αντίστοιχες συναρτήσεις βιβλιοθήκης που είναι το `stdlib.h`, και γίνεται με την εντολή :

```
#include <stdlib.h>
```

- Η χρήση μαθηματικών συναρτήσεων με τις οποίες μπορούμε να κάνουμε βασικές πράξεις με εκθετικά, λογαρίθμους, τετραγωνικές ρίζες και τριγωνομετρικές συναρτήσεις, όπως π.χ. τη `sin(x)` για τον υπολογισμό του ημιτόνου του `x`, την `cos(x)` για τον υπολογισμό του συνημιτόνου του `x`, την `sqrt(x)` για τον υπολογισμό της τετραγωνικής ρίζας του `x`, την `exp(x)` για τον υπολογισμό του του e^x, και την `pow(x,y)` για τον υπολογισμό του x^y απαιτούν την ενσωμάτωση του αρχείου που περιέχει τις αντίστοιχες συναρτήσεις βιβλιοθήκης που είναι το `math.h`, και γίνεται με την εντολή :

```
#include <math.h>
```

2 ΕΝΤΟΛΕΣ ΕΛΕΓΧΟΥ

Στα πιο πολλά προγράμματα απαιτούνται να γίνονται κάποιοι έλεγχοι για το αν μπορεί να γίνει μια πράξη (π.χ. αν ο διαιρέτης δεν είναι μηδέν), αν ένας αριθμός ή όνομα υπάρχει σε μια λίστα, αν ένας βαθμός που θα εισαχθεί σε ένα πρόγραμμα είναι μια αποδεκτή τιμή (από 1 μέχρι 10) κ.λ.π. πριν προβούμε στην εκτέλεση κάποιας ή κάποιων εντολών. Για να γίνει ο έλεγχος θα πρέπει να γίνει κάποια σύγκριση, π.χ. ο διαιρέτης δεν είναι ίσος με το μηδέν, ο βαθμός είναι μεταξύ του 0 και του 10 κ.λ.π..

Η πιο απλή μορφή σύγκρισης – εντολής ελέγχου έχει τη μορφή :

```
if (<συνθήκη>)
   εντολή;

if(<συνθήκη>){
   block εντολών;
   }
```

όπου ελέγχεται η συνθήκη αν είναι αληθής. **Αν** ισχύει η συνθήκη, εκτελείται η εντολή ή οι εντολές (οι οποίες θα πρέπει να περικλείονται σε άγγιστρα, αν είναι περισσότερες από μια) μετά το `if`. Αν ΔΕΝ ισχύει η συνθήκη, δε γίνεται τίποτα.

Η συνθήκη μπορεί να περιλαμβάνει μεταβλητές ή αριθμητικές εκφράσεις, ενώ οι συγκρίσεις γίνονται με τους παρακάτω Σχεσιακούς Τελεστές ή Τελεστές Σύγκρισης :

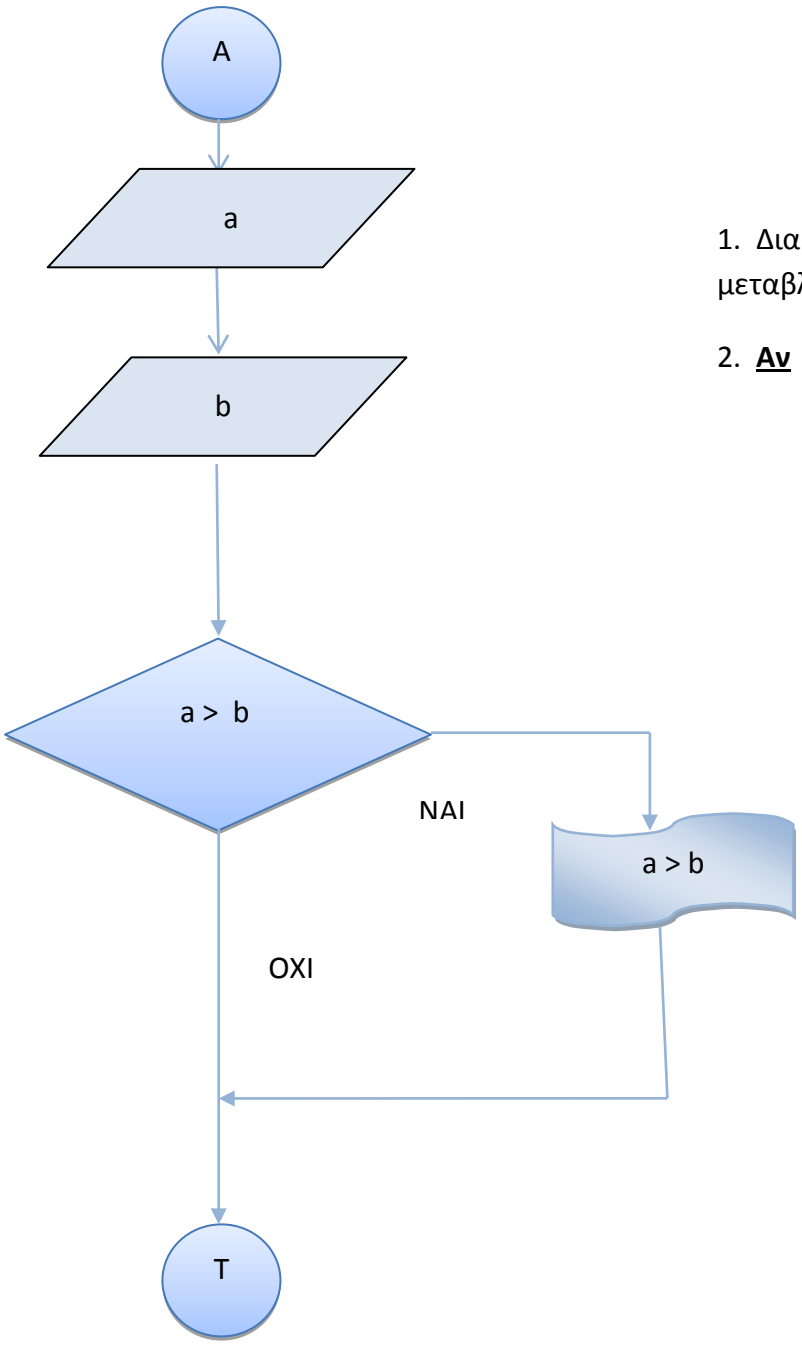
Σύγκριση	**Σύμβολο**	**Σχεσιακός Τελεστής**
Μικρότερο	<	<
Μικρότερο ή Ίσο	≤	<=
Ίσο	=	==
Μεγαλύτερο ή Ίσο	≥	>=
Μεγαλύτερο	>	>
Διάφορο	≠	!=

- Αν η συνθήκη περιέχει **`boolean`** μεταβλητή **ΔΕΝ ΧΡΕΙΑΖΕΤΑΙ** να ελεγχθεί αν η τιμή της είναι **`true`** ή **`false`**. Αντί **`του if (b == true)`** μπορούμε να γράψουμε **`if(b)`.**

2. 1 Ένα Απλό Πρόγραμμα με την Εντολή Ελέγχου if

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα διαβάζει 2 ακέραιες τιμές στις μεταβλητές **a** και **b** και θα ελέγχει αν ο αριθμός **a** είναι μεγαλύτερος του **b,** οπότε σ' αυτή την περίπτωση θα εμφανίζει τη σχέση τους, **a > b.**

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω 2 ακέραιες τιμές στις μεταβλητές **a** και **b**
2. **Αν** το a > b **τότε**

   Εμφανίζω το **a > b**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/*
Δίνονται 2 ακέραιοι αριθμοί a, b. Να ελεγχθεί αν ο αριθμός a είναι
μεγαλύτερος του b, οπότε σ' αυτή την περίπτωση θα εμφανίζει τη σχέση
τους, a  > b
*/
        // Δήλωση των ακέραιων μεταβλητών a, b
        int a, b;

        // Διάβασμα 2 ακεραίων a, b
          printf ("Give two integer numbers : ");
          scanf("%d %d", &a, &b);

        // Έλεγχος αν ο a είναι μεγαλύτερος του b, εμφάνιση της σχέσης
        if (a > b)
              printf("a = %d > b = %d\n", a, b );
       system("Pause");
}
```

Έξοδος Προγράμματος

```
Give two integer numbers : 6 5
a = 6 > b = 5
Press any key to continue . . .
```

Μια άλλη μορφή σύγκρισης – εντολής ελέγχου έχει τη μορφή :

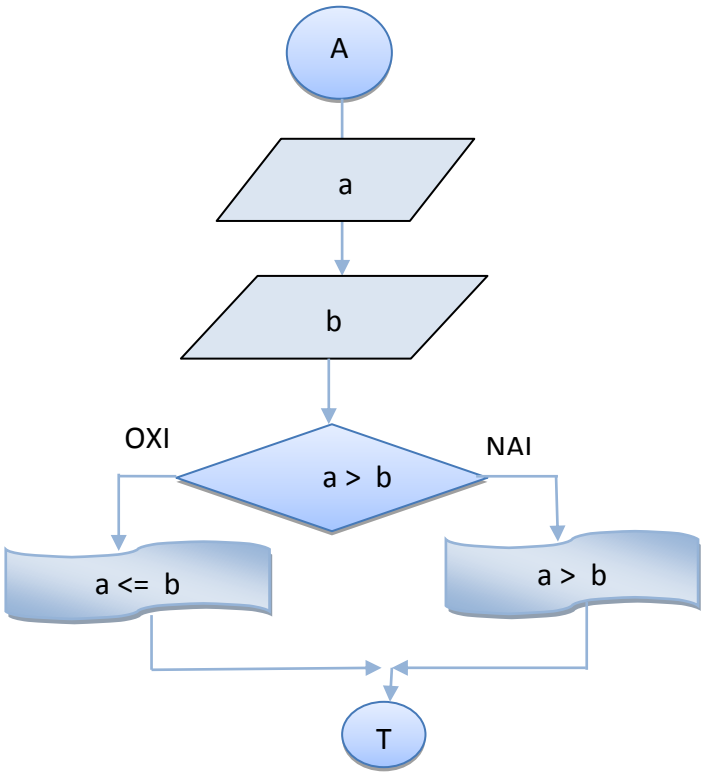
```
if (<συνθήκη>)
  εντολή-1; ή >){ block εντολών-1; }
else
  εντολή-2; ή >){ block εντολών-2; }
```

όπου ελέγχεται η συνθήκη αν είναι αληθής. **Αν ισχύει** η συνθήκη, εκτελείται η εντολή-1 ή οι εντολές-1, μετά το `if`. Αν **ΔΕΝ** ισχύει η συνθήκη, εκτελείται η εντολή-2 ή οι εντολές-2, μετά το `else`.

2. 2 Τροποποίηση Προγράμματος 2.1 με την Εντολή Ελέγχου if - else

Να τροποποιηθεί το Πρόγραμμα 2.1, ώστε να διαβάζει 2 ακέραιους αριθμούς **a** και **b** και να ελέγχει αν ο αριθμός **a** είναι μεγαλύτερος του **b,** οπότε σ' αυτή την περίπτωση θα εμφανίζει τη σχέση τους, **a > b**. Αν δεν ισχύει η συνθήκη, θα εμφανίζει **a <= b.**

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω 2 ακέραιους αριθμούς **a** και **b**
2. **Αν** το a > b **τότε**

   Εμφανίζω το **a** > **b**

   Διαφορετικά

   Εμφανίζω το **a** <= **b**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/*  Δίνονται 2 ακέραιοι αριθμοί a, b. Να ελεγχθεί αν ο αριθμός a είναι
μεγαλύτερος του b, οπότε σ' αυτή την περίπτωση θα εμφανίζει τη σχέση
τους, a  > b. Αν δεν ισχύει η συνθήκη, θα εμφανίζει a <= b */
        int a, b; // Δήλωση των ακέραιων μεταβλητών a, b

        // Διάβασμα 2 ακεραίων a, b
        printf ("Give two integer numbers : ");
       scanf("%d %d", &a, &b);

        // Έλεγχος αν ο a είναι μεγαλύτερος του b, εμφάνιση της σχέσης
        if (a > b)
              printf("a = %d > b = %d\n", a, b );
        else
              printf("a = %d <= b = %d\n", a, b );
       system("Pause");
}
```

Έξοδος Προγράμματος

```
Give two integer numbers : 6 5
a = 6 > b = 5
Give two integer numbers : 5 6
a = 5 <= b = 6
Press any key to continue . . .
```

Όταν σε έναν έλεγχο υπάρχουν **περισσότερα από δύο ενδεχόμενα** μπορούμε να χρησιμοποιήσουμε τη σκάλα `if – else – if`. Σ' αυτήν την περίπτωση υπάρχουν περισσότερες συνθήκες να ελεγχθούν και εκτελούνται οι αντίστοιχες εντολές.

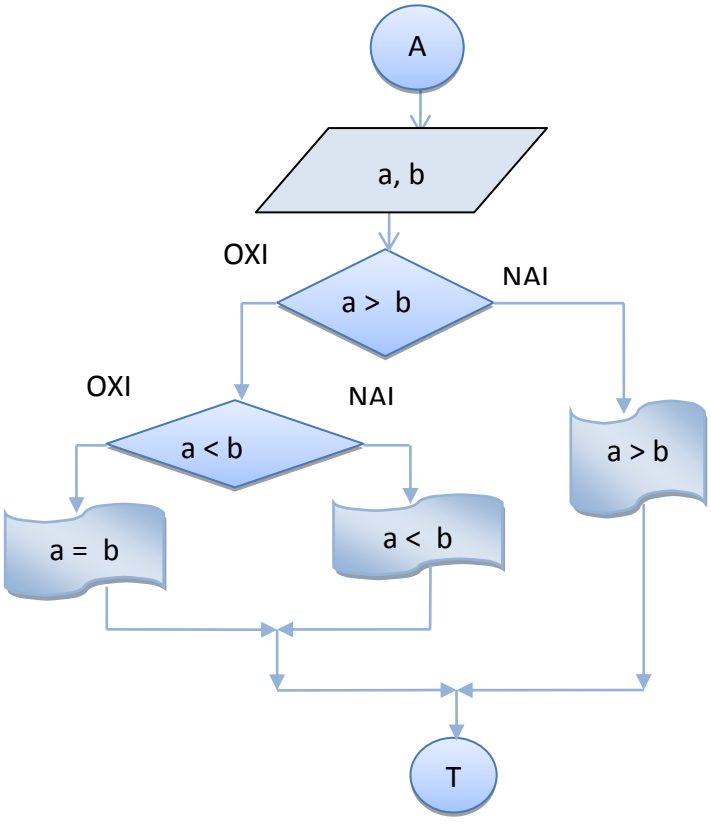
```
if (<συνθήκη-1>)
  εντολή-1; ή >){ block εντολών-1; }
else if (<συνθήκη-2>)
  εντολή-2; ή >){ block εντολών-2; }
else
  εντολή-3; ή >){ block εντολών-3; }
```

όπου ελέγχεται η συνθήκη-1. **Αν** ισχύει η συνθήκη-1, εκτελείται η εντολή-1 ή οι εντολές-1, μετά το `if`. **Αν** ΔΕΝ ισχύει η συνθήκη-1, ελέγχεται η συνθήκη-2. Αν ισχύει, εκτελείται η εντολή-2 ή οι εντολές-2, μετά το `else if`. **Αν** ΔΕΝ ισχύει η συνθήκη-2, εκτελείται η εντολή-3 ή οι εντολές-3, μετά το `else`.

2.3 Τροποποίηση Προγράμματος 2.1 με την Εντολή Ελέγχου if – else -if

Να τροποποιηθεί το Πρόγραμμα 2.1, ώστε να διαβάζει 2 ακέραιους αριθμούς **a** και **b** και να ελέγχει αν ο αριθμός **a** είναι μεγαλύτερος του **b,** οπότε σ' αυτή την περίπτωση θα εμφανίζει τη σχέση τους, **a > b**. Διαφορετικά, θα ελέγχει αν ο αριθμός **a** είναι μικρότερος του **b** και θα εμφανίζει **a < b.** Αν δεν ισχύει ούτε αυτό, θα εμφανίζει **a = b.**

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω 2 ακέραιους αριθμούς **a** και **b**
2. **Αν** το a > b **τότε**

   Εμφανίζω το **a > b**

   Διαφορετικά Αν το a < b

   Εμφανίζω το **a < b**

   Διαφορετικά

   Εμφανίζω το **a = b**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/*  Δίνονται 2 ακέραιοι αριθμοί a, b. Να ελεγχθεί αν ο αριθμός a είναι
μεγαλύτερος του b, οπότε σ' αυτή την περίπτωση θα εμφανίζει τη σχέση τους, a
> b. Αν δεν ισχύει η συνθήκη, θα εμφανίζει a < b ή  a = b */
        int a, b; // Δήλωση των ακέραιων μεταβλητών a, b

        // Διάβασμα 2 ακεραίων a, b
        printf ("Give two integer numbers : ");
       scanf("%d %d", &a, &b);

        // Έλεγχος αν ο a είναι μεγαλύτερος του b, εμφάνιση της σχέσης
        if (a > b)
              printf("a = %d > b = %d\n", a, b );
        else
              if (a < b)
                    printf("a = %d < b = %d\n", a, b );
              else
                    printf("a = %d = b = %d\n", a, b );
       system("Pause");
}
```

Έξοδος Προγράμματος

```
Give two integer numbers : 6 5
a = 6 > b = 5
Press any key to continue . . .

Give two integer numbers : 5 6
a = 5 < b = 6
Press any key to continue . . .

Give two integer numbers : 5 5
a = 5 = b = 5
Press any key to continue . . .
```

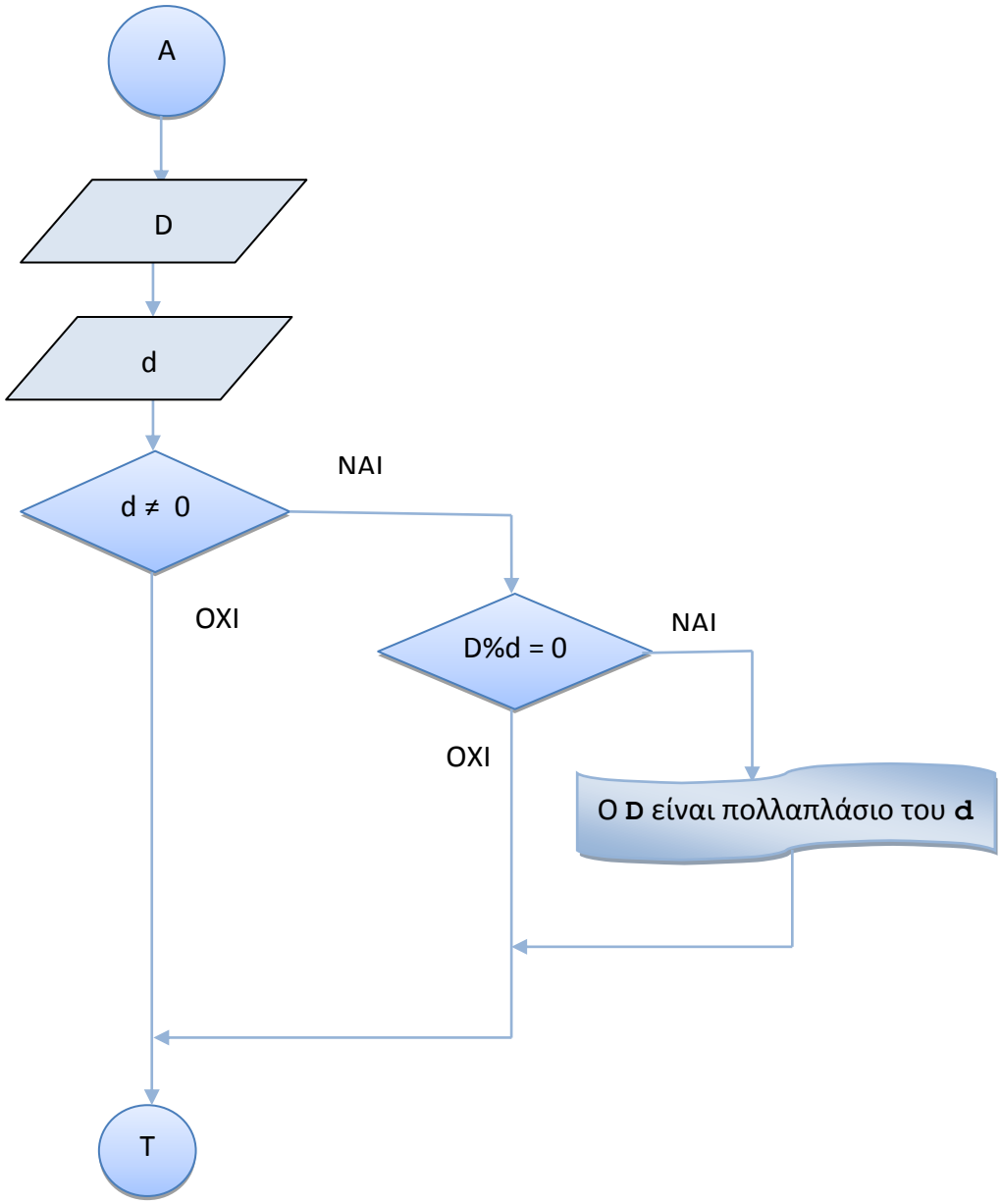
Παρατήρηση

- Μπορεί στο τμήμα του `if` να υπάρχει και δεύτερο `if`, δηλαδή να ελεγχθεί και κάποια άλλη συνθήκη, όπως φαίνεται στο επόμενο παράδειγμα :

2.4 Πρόγραμμα με την Εντολή Ελέγχου if – if

Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο θα διαβάζει δύο ακέραιους αριθμούς, το **D** μεταξύ του 10 και 20 και το **d** μεταξύ του 0 και 5. Αν ο αριθμός **d** είναι διάφορος του μηδενός, θα ελέγχει αν ο αριθμός **d** διαιρεί ΑΚΡΙΒΩΣ τον αριθμό **D**, οπότε και θα εμφανίζει το μήνυμα "Ο **D** είναι πολλαπλάσιο του **d** ".

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν ακέραιο `D` μεταξύ του `10` και `20`
2. Διαβάζω έναν ακέραιο `d` μεταξύ του `0` και `5`
3. **Αν** ο αριθμός **d** είναι διάφορος του μηδενός (`d ≠ 0`)
   Αν ο αριθμός **d** διαιρεί ΑΚΡΙΒΩΣ τον αριθμό **D**
   Εμφανίζω το μήνυμα "Ο **D** είναι πολλαπλάσιο του **d** "

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/*
Πρόγραμμα, το οποίο  διαβάζει δύο ακέραιους αριθμούς, το D  μεταξύ του 10 και
20 και το  d  μεταξύ του 0 και 5. Αν ο αριθμός d  είναι διάφορος του μηδενός,
θα ελέγχει αν ο αριθμός d  διαιρεί ΑΚΡΙΒΩΣ τον αριθμό D, οπότε και θα
εμφανίζει το μήνυμα "Ο D είναι πολλαπλάσιο του d ".
*/
        // Διάβασμα ακεραίου D
        printf ("Give an integer number between 10 and 20 : ");
        scanf("%d, &D);

        // Διάβασμα ακεραίου d
        printf ("Give an integer number between 0 and 5 : ");
        scanf("%d, &d);

        // Έλεγχος αν ο d είναι διάφορος του 0
        if (d != 0)
          // Έλεγχος αν ο αριθμός d διαιρεί ΑΚΡΙΒΩΣ τον αριθμό D
          if (D % d == 0)
               printf("D = %d pollaplasio tou d = %d\n", D, d );   }
}
```

Έξοδος Προγράμματος

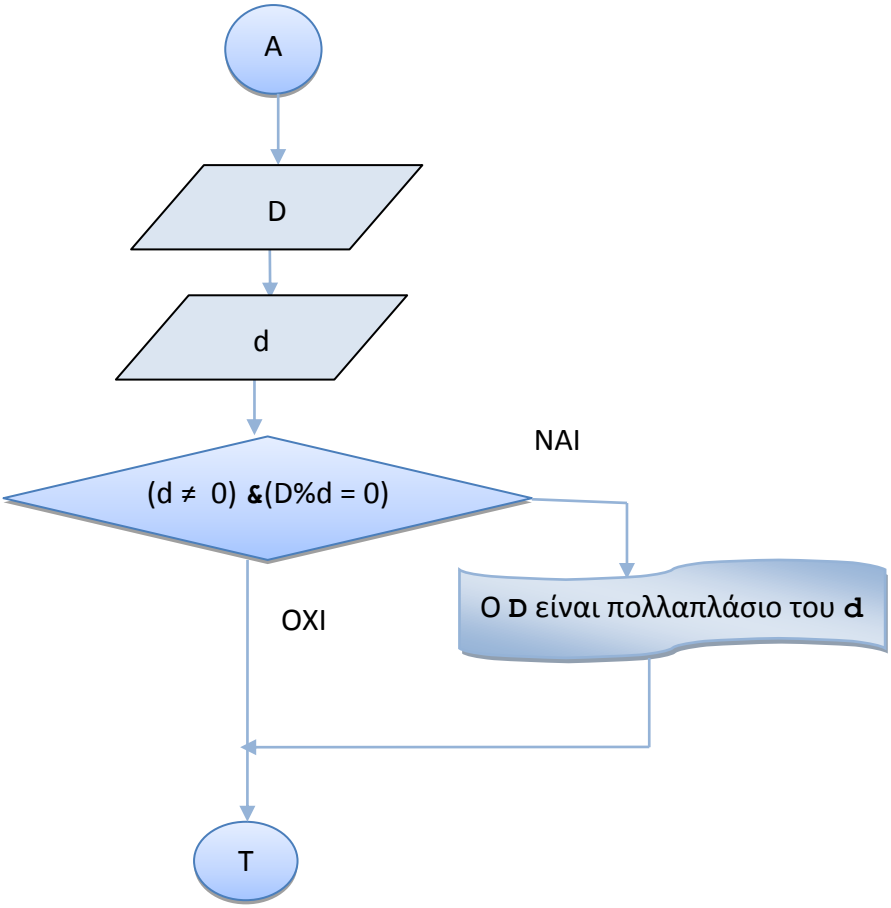
```
Give an integer number between 10 and 20 : 15
Give an integer number between 0 and 5 : 5
D = 15 pollaplasio tou d = 5
Press any key to continue . . .
```

- Όταν έχουμε ένα **if** χωρίς **else** αμέσως μετά από ένα άλλο **if**, μπορούμε αντί των δύο **if** να χρησιμοποιήσουμε ένα, αλλά με **διπλή συνθήκη**. Οι δύο συνθήκες μπορούν να συνδεθούν με το **&**, ώστε η συνολική συνθήκη να είναι αληθής, αν είναι και οι δύο επί μέρους συνθήκες αληθείς. Στην προκείμενη περίπτωση, θα μπορούσαμε να ελέγξουμε με μια εντολή **if** αν ισχύουν και οι δύο συνθήκες, δηλαδή ο αριθμός **d** είναι διάφορος του μηδενός και ο αριθμός **d** διαιρεί ΑΚΡΙΒΩΣ τον αριθμό **D**.

2.4.1 Το Πρόγραμμα 2.4 με Διπλή Συνθήκη στην Εντολή Ελέγχου if

Να τροποποιηθεί το Πρόγραμμα 2.4, ώστε να διαβάζει δύο ακέραιους αριθμούς, το **D** μεταξύ του 10 και 20 και το **d** μεταξύ του 0 και 5. Αν (ο αριθμός **d** είναι διάφορος του μηδενός) και (ο αριθμός **d** διαιρεί ΑΚΡΙΒΩΣ τον αριθμό **D**), θα εμφανίζει το μήνυμα "Ο **D** είναι πολλαπλάσιο του **d** ".

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

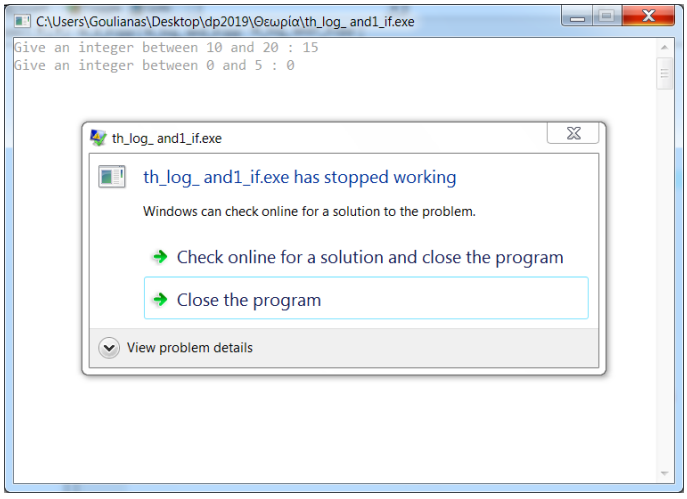
1. Διαβάζω έναν ακέραιο `D` μεταξύ του `10` και `20`
2. Διαβάζω έναν ακέραιο `d` μεταξύ του `0` και `5`
3. **Αν** (ο **d** είναι διάφορος του μηδενός - d ≠ 0) **&** **(**ο **d** διαιρεί ΑΚΡΙΒΩΣ τον **D)**
   Εμφανίζω το μήνυμα "Ο **D** είναι πολλαπλάσιο του **d** "

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/* Πρόγραμμα, το οποίο  διαβάζει δύο ακέραιους αριθμούς, το D  μεταξύ του 10
και 20 και το  d  μεταξύ του 0 και 5. Αν ο αριθμός d  είναι διάφορος του
μηδενός και ο αριθμός d  διαιρεί ΑΚΡΙΒΩΣ τον αριθμό D, θα εμφανίζει το μήνυμα
"Ο D είναι πολλαπλάσιο του d ". */
        // Διάβασμα ακεραίου D
        printf ("Give an integer number between 10 and 20 : ");
       scanf("%d, &D);

        // Διάβασμα ακεραίου d
        printf ("Give an integer number between 0 and 5 : ");
       scanf("%d, &d);

        // Έλεγχος αν ο d είναι διάφορος του 0 & ο  d διαιρεί ΑΚΡΙΒΩΣ τον D
        if (d != 0 & D % d == 0)
              printf("D = %d pollaplasio tou d = %d\n", D, d );    }
}
```

Έξοδος Προγράμματος

```
Give an integer number between 10 and 20 : 15
Give an integer number between 0 and 5 : 5
D = 15 pollaplasio tou d = 5
Press any key to continue . . .

Give an integer number between 10 and 20 : 15
Give an integer number between 0 and 5 : 0
```



- Στο προηγούμενο πρόγραμμα χρησιμοποιήσαμε το Λογικό Τελεστή **&** για να συνδέσουμε τις δύο συνθήκες, ώστε να ισχύουν ταυτόχρονα και οι δύο, αλλά δημιούργησε ένα **σφάλμα εκτέλεσης – run time error** (διαίρεση με το μηδέν) για την τιμή **d = 0**, γιατί ελέγχονται και οι δύο συνθήκες `(d != 0 )` και `( D % d == 0)`, οπότε κάνει τη διαίρεση `D / d` για να βρεί το υπόλοιπο. Για να αποφύγουμε τέτοια προβλήματα, μπορούμε να χρησιμοποιήσουμε το Βραχυκυκλωμένο Λογικό Τελεστή **&&**, ο οποίος σε αντίθεση με τον Τελεστή **&** **ΔΕΝ ελέγχει τη δεύτερη συνθήκη, αν η πρώτη συνθήκη είναι ψευδής** (ο έλεγχος της δεύτερης συνθήκης πραγματικά δε χρειάζεται αν η πρώτη συνθήκη είναι ψευδής, αφού θα πρέπει να είναι αληθείς ΚΑΙ ΟΙ ΔΥΟ συνθήκες).

2.4.2 Το Πρόγραμμα 2.4 με Διπλή Συνθήκη στην Εντολή Ελέγχου if και το Βραχυκυκλωμένο Λογικό Τελεστή &&

Να τροποποιηθεί το Πρόγραμμα 2.4, ώστε να διαβάζει δύο ακέραιους αριθμούς, το **D** μεταξύ του 10 και 20 και το **d** μεταξύ του 0 και 5. Αν (ο αριθμός **d** είναι διάφορος του μηδενός) και (ο αριθμός **d** διαιρεί ΑΚΡΙΒΩΣ τον αριθμό **D**), θα εμφανίζει το μήνυμα "Ο **D** είναι πολλαπλάσιο του **d** ". Να χρησιμοποιηθεί στη διπλή συνθήκη ο Βραχυκυκλωμένος Λογικός Τελεστής **&&**.

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/* Πρόγραμμα, το οποίο  διαβάζει δύο ακέραιους αριθμούς, το D  μεταξύ του 10
και 20 και το  d  μεταξύ του 0 και 5. Αν ο αριθμός d  είναι διάφορος του
μηδενός και ο αριθμός d  διαιρεί ΑΚΡΙΒΩΣ τον αριθμό D, θα εμφανίζει το μήνυμα
"Ο D είναι πολλαπλάσιο του d ". */
        // Διάβασμα ακεραίου D
        printf ("Give an integer number between 10 and 20 : ");
        scanf("%d, &D);

        // Διάβασμα ακεραίου d
        printf ("Give an integer number between 0 and 5 : ");
        scanf("%d, &d);

        // Έλεγχος αν ο d είναι διάφορος του 0 & ο  d διαιρεί ΑΚΡΙΒΩΣ τον D
        if (d != 0 && D % d == 0)
              printf("D = %d pollaplasio tou d = %d\n", D, d );   }
}
```

Έξοδος Προγράμματος

```
Give an integer number between 10 and 20 : 15
Give an integer number between 0 and 5 : 0
Press any key to continue . . .
```

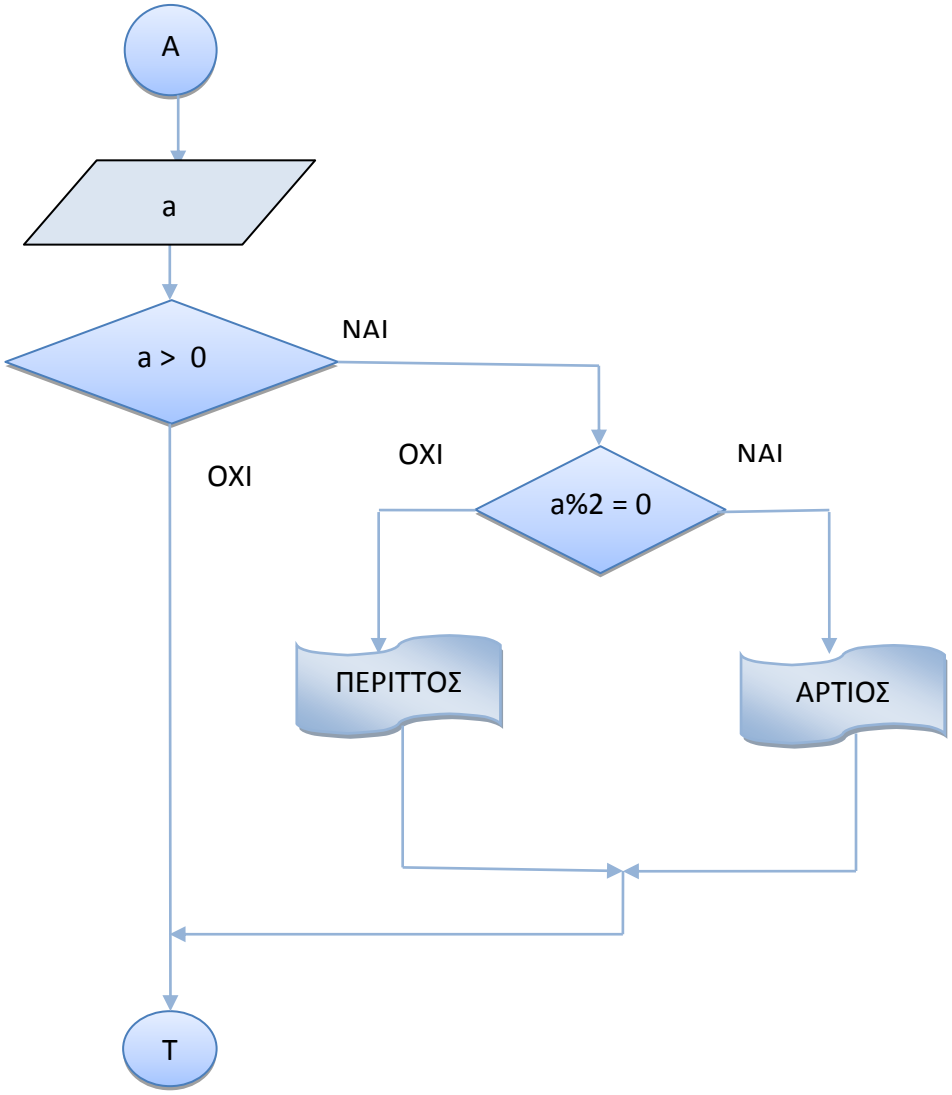
Παρατήρηση

Μπορεί στο τμήμα ενός if να υπάρχει ένα πλήρες if, όπως φαίνεται στο επόμενο παράδειγμα :

2.4.3 Πρόγραμμα με την Εντολή Ελέγχου if – if – else

Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο θα **διαβάζει** έναν ακέραιο αριθμό **a** μεταξύ του −5 και 5. Αν ο αριθμός είναι **θετικός**, θα ελέγχει αν ο αριθμός **a** είναι **άρτιος** ή **περιττός** και θα εμφανίζει την τιμή του με αντίστοιχο μήνυμα.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν ακέραιο **a** μεταξύ του **−5** και **5**
2. **Αν** ο αριθμός **a** είναι θετικός (**a > 0**)
   Αν Το υπόλοιπο της διαίρεσης του **a** δια του **2** είναι 0 (**a % 2 = 0**)
   Εμφανίζω το μήνυμα **a** ΑΡΤΙΟΣ
   Διαφορετικά
   Εμφανίζω το μήνυμα **a** ΠΕΡΙΤΤΟΣ

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
 /*  Το πρόγραμμα διαβάζει έναν ακέραιο αριθμό μεταξύ του -5 και 5.
 Αν ο αριθμός είναι θετικός,  ελέγχεται αν είναι άρτιος ή περιττός και
 εμφανίζεται το αντίστοιχο μήνυμα  */
     // Διάβασμα ακέραιου αριθμού a μεταξύ του -5 και5
     printf ("Give an integer a between -5 and 5 : ");
     scanf("%d", &a);

     // Έλεγχος αν ο a είναι μεγαλύτερος του 0
     if (a > 0)
       // Έλεγχος αν ο a είναι άρτιος ή περιττός, εμφάνιση μηνύματος
       if (a %2 == 0)
           printf("a = %d artios\n", a );
       else
          printf("a = %d perittos\n", a );
}
```

Έξοδος Προγράμματος

```
Give an integer number between -5 and 5 : 4
a = 4 artios
Press any key to continue . . .

Give an integer number between -5 and 5 : 3
a = 3 perittos
Press any key to continue . . .

Give an integer number between -5 and 5 : 0
Press any key to continue . . .
```

Άσκηση 2.1 : Να τροποποιηθεί ο Αλγόριθμος 2.4.3, ώστε να βρίσκει αν ένας θετικός ακέραιος αριθμός είναι άρτιος ή περιττός με τη **χρήση διπλής συνθήκης**.

Λογικοί Τελεστές

Οι Τελεστές που συνδέουν δύο ή περισσότερες συνθήκες μεταξύ τους είναι οι παρακάτω :

Σύνδεση	**Λογικός Τελεστής**
ΚΑΙ (AND)	&
Ή (OR)	|
Αποκλειστικό Ή (XOR)	^
ΟΧΙ	!
Βραχυκυκλωμένο ΚΑΙ (Short Cirquit AND)	&&
Βραχυκυκλωμένο Ή (Short Cirquit OR)	||

Η διαφορά ανάμεσα στο βραχυκυκλωμένο τελεστή **&&** και τον απλό **&** είναι ότι ο βραχυκυκλωμένος τελεστής **&&** σε αντίθεση με τον Τελεστή **&** **ΔΕΝ** ελέγχει τη δεύτερη συνθήκη, **αν** η πρώτη συνθήκη είναι **ψευδής**, ενώ η διαφορά ανάμεσα στο βραχυκυκλωμένο τελεστή **||** και τον απλό **|** είναι ότι ο βραχυκυκλωμένος τελεστής **||** σε αντίθεση με τον Τελεστή **|** **ΔΕΝ** ελέγχει τη δεύτερη συνθήκη, αν η πρώτη συνθήκη είναι **αληθής**.

Ο Τριαδικός Τελεστής ?

Μπορούμε, αντί να χρησιμοποιήσουμε την εντολή με **if**, **else**, να χρησιμοποιήσουμε τον τριαδικό τελεστή **?**, ο οποίος έχει το ίδιο αποτέλεσμα. Ας δούμε ένα παράδειγμα υπολογισμού της απόλυτης τιμής ενός ακέραιου αριθμού :

Παράδειγμα

```
printf ("Give an integer a between -5 and 5 : ");
scanf("%d", &a);
if ( a < 0 )
  absa = -n;
else
  absa = n;
printf("absa = %d\n", absa );
system("Pause");
```

Οι παραπάνω εντολές θα μπορούσαν να γραφούν :

```
printf ("Give an integer a between -5 and 5 : ");
scanf("%d", &a);
absa = a < 0 ? -a:a;
printf("absa = %d\n", absa );
system("Pause");
```

όπου ελέγχεται η συνθήκη `a < 0` και αν είναι αληθής, το `absa = -a,` διαφορετικά, το `absa = a`, δηλαδή ότι κάνει και το `if-else`.

2.5 Η Σκάλα if – else - if

Για να ελέγξουμε την περίπτωση που έχουμε περισσότερα από τρία ενδεχόμενα, χρησιμοποιούμε τη σκάλα `if else if`, όπου κάθε `else` αντιστοιχεί στο κοντινότερο `if`:

```
if (<συνθήκη-1>)
  εντολή-1; ή >){ block εντολών-1; }
else if (<συνθήκη-2>)
  εντολή-2; ή >){ block εντολών-2; }
else if (<συνθήκη-3>)
  εντολή-3; ή >){ block εντολών-3; }
…
else
  εντολή-n; ή >){ block εντολών-n; }
```

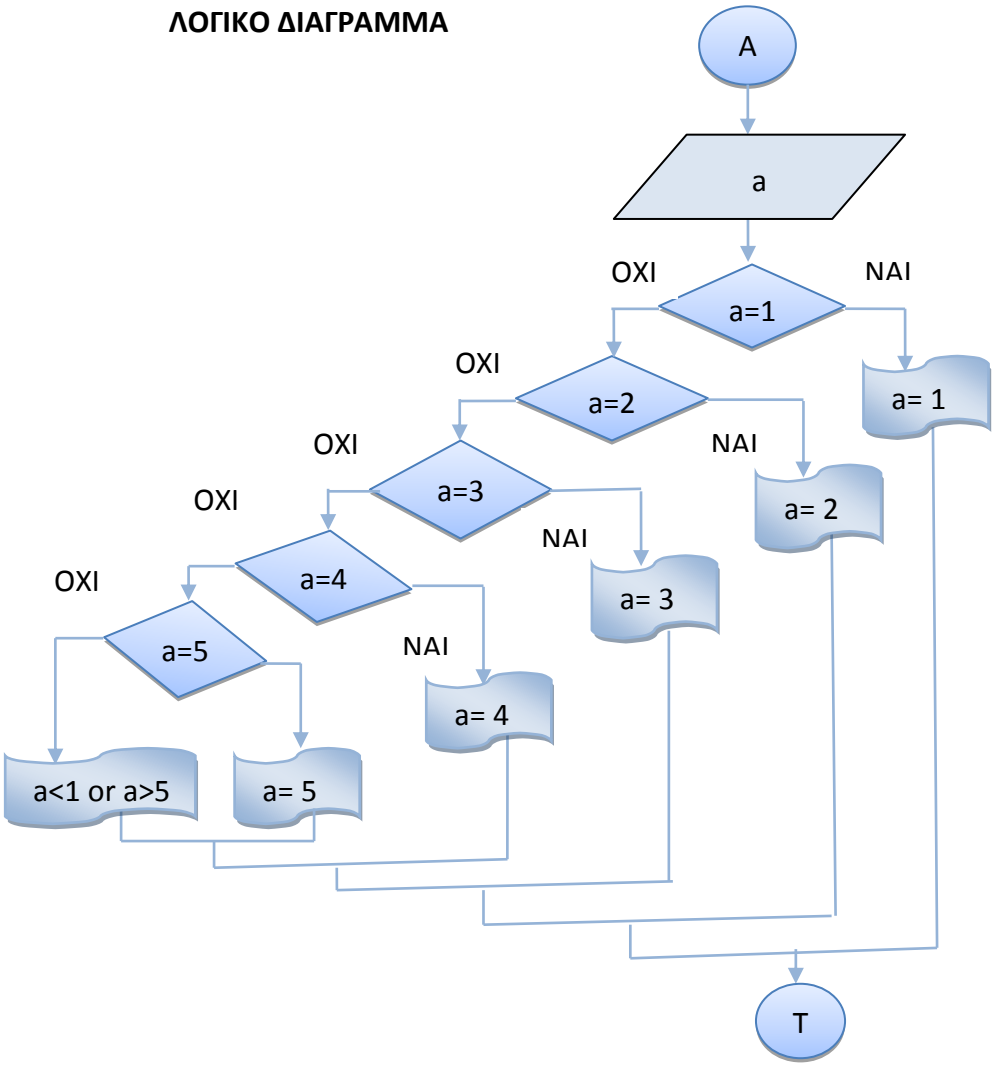
2.5.1 Πρόγραμμα με την Εντολή Ελέγχου if – else if –else if … else

Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο θα διαβάζει έναν ακέραιο αριθμό **a** μεταξύ του 0 και 10 και θα ελέγχει αν ο αριθμός **a** είναι το 1, το 2, το 3, το 4 ή το 5. Σε κάθε περίπτωση θα εμφανίζει την τιμή του ή το μήνυμα αριθμός < 1 ή > 5.

ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν ακέραιο μεταξύ 1-5
2. **Αν** (a = 1)
   Εμφάνισε "a = 1"
   Διαφορετικά, **Αν** (a = 2)
   Εμφάνισε "a = 2"
   Διαφορετικά **Αν** (a = 3)
   Εμφάνισε "a = 3"
   Διαφορετικά **Αν** (a = 4)
   Εμφάνισε "a = 4"
   Διαφορετικά
   Εμφάνισε "a = 5"
   Διαφορετικά
   Εμφάνισε "a <1 or a > 5"

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
main() {
/* Πρόγραμμα, το οποίο  διαβάζει έναν ακέραιο αριθμό a μεταξύ του 0 και 10,
ελέγχει αν ο αριθμός a είναι το 1, το 2, το 3, το 4 ή το 5 και σε κάθε
περίπτωση θα εμφανίζει την τιμή του. */
     int a;  // Δήλωση a

     // Διάβασμα a
     printf ("Give an integer a between 1 and 5 : ");
     scanf("%d", &a);

     // Έλεγχος, εμφάνιση της τιμής του a
     if (a == 1)
       printf("a = %d = 1\n", a );
     else if (a == 2)
       printf("a = %d = 2\n", a );
     else if (a == 3)
       printf("a = %d = 3\n", a );
     else if (a == 4)
       printf("a = %d = 4\n", a );
     else
       printf("a = %d = 5\n", a );
     else
       printf("a = %d < 1 or a = %d > 5\n", a, a );
     system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer a between 1 and 5 : 3
a = 3 = 3
Press any key to continue . . .

Give an integer a between 1 and 5 : 7
a = 7 < 1 or a = 7 > 5
Press any key to continue . . .
```

2.6 Η Εντολή Ελέγχου switch

Στο παράδειγμα 2.5.1 χρειάστηκε να χρησιμοποιήσουμε **4** εντολές **if** για να ελέγξουμε αν η τιμή του **a** είναι το **1**, το **2**, το **3**, το **4** ή το **5**. Σ' αυτές τις περιπτώσεις, μπορούμε να χρησιμοποιήσουμε την εντολή **switch**, η οποία επιλέγει ανάμεσα σε διάφορες περιπτώσεις. Η γενική της μορφή είναι :

```
switch ( <έκφραση> )
     {
        case <σταθερά_1> :     εντολές_1;
                               break;
        case <σταθερά_2> :     εντολές_2;
                               break;
        ...                    ...
        ...                    ...
        case <σταθερά_n> :     εντολές_n;
                               break;
        default :              εντολές_n+1
      }
```

όπου η <έκφραση> μπορεί να είναι μεταβλητή ή αριθμητική έκφραση τύπου **char, byte, short, int,** ενώ οι <σταθερά_1>, <σταθερά_2>, ..., <σταθερά_n> είναι κυριολεκτικές σταθερές του τύπου της έκφρασης.

- Η επιλογή default είναι προαιρετική και θα εκτελεστούν οι εντολές της, αν η έκφραση <έκφραση> δεν πάρει καμιά απ' τις τιμές που εμφανίζονται στις γραμμές case.

- Ανάλογα με την τιμή που θα έχει η <έκφραση> θα εκτελεστούν οι αντίστοιχες εντολές της κάθε περίπτωσης.

- Η εντολή break είναι προαιρετική, αλλά, αν δεν υπάρχει, μετά την εκτέλεση των εντολών κάποιας περίπτωσης, θα εκτελεστούν και οι εντολές των υπόλοιπων περιπτώσεων.

- Μπορεί να υπάρχει εντολή switch σε κάποια περίπτωση (case) μιας άλλης εντολής switch και οι περιπτώσεις της να παίρνουν ίδιες τιμές με τις περιπτώσεις της εξωτερικής εντολής switch.

- Μπορεί να υπάρχουν κενές περιπτώσεις χωρίς τιμές. Αυτές εντάσσονται στην πρώτη επόμενη περίπτωση που έχει εντολές. Π.χ. αν θέλω να κάνω κάτι για τις περιπτώσεις 1, 2, 3, αφήνω κενές τις περιπτώσεις 1,2 και γράφω τις εντολές στην περίπτωση 3.

2.6.1 Τροποποίηση του Προγράμματος 2.5.1 με την Εντολή Ελέγχου switch

Να γραφεί Πρόγραμμα, το οποίο θα δημιουργεί έναν τυχαίο ακέραιο αριθμό **a** μεταξύ του **0** και 10, θα εμφανίζει την τιμή του και θα ελέγχει με τη χρήση της εντολής **switch** αν ο αριθμός **a** είναι το **1**, το **2**, το **3**, το **4** ή το **5**. Σε κάθε περίπτωση θα εμφανίζει την τιμή του ή το μήνυμα αριθμός < 1 ή > 5.

ΠΡΟΓΡΑΜΜΑ

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

/* Πρόγραμμα, το οποίο  διαβάζει έναν ακέραιο αριθμό a μεταξύ του 0 και 10,
ελέγχει αν ο αριθμός a είναι το 1, το 2, το 3, το 4 ή το 5 και σε κάθε
περίπτωση θα εμφανίζει την τιμή του. */
     int a;  // Δήλωση a

     // Διάβασμα a
     printf ("Give an integer a between 0 and 10 : ");
     scanf("%d", &a);

    switch ( a )
     {
        case 1 :  printf("a = %d = 1\n", a );
                  break;
        case 2 :  printf("a = %d = 2\n", a );
                  break;
        case 3 :  printf("a = %d = 3\n", a );
                  break;
        case 4 :  printf("a = %d = 4\n", a );
                  break;
        case 5 :  printf("a = %d = 5\n", a );
                  break;
        default : printf("a = %d < 1 or a = %d > 5\n", a, a );
                  break;
     }
}
```

Έξοδος Προγράμματος

```
Give an integer a between 1 and 5 : 3
a = 3 = 3
Press any key to continue . . .

Give an integer a between 1 and 5 : 7
a = 7 < 1 or a = 7 > 5
Press any key to continue . . .
```

2.5.2 Διάβασμα Χαρακτήρα

Να γραφεί Πρόγραμμα, το οποίο θα διαβάζει απ' το πληκτρολόγιο ένα χαρακτήρα **ch**, θα εμφανίζει την τιμή του και θα ελέγχει με τη χρήση της εντολής **switch** αν ο χαρακτήρας **ch** είναι το **'a'**, το **'b'**, το **'c'**, το **'d'** ή το **'e'**. Σε κάθε περίπτωση θα εμφανίζει την τιμή του.

ΠΡΟΓΡΑΜΜΑ

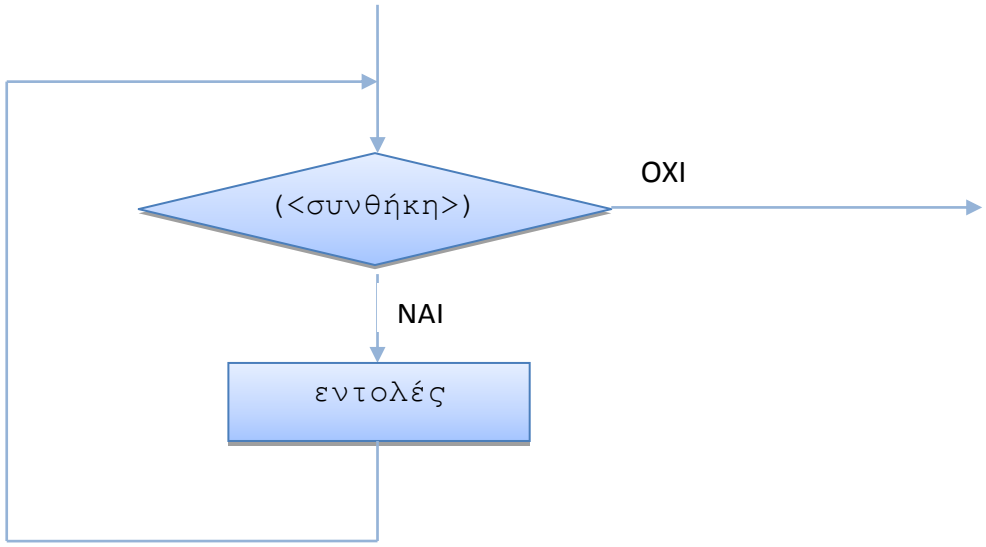
```
#include <stdio.h>
#include <stdlib.h>
main() {

/* Πρόγραμμα, το οποίο  διαβάζει απ' το πληκτρολόγιο έναν χαρακτήρα ch και θα
ελέγχει με τη χρήση της εντολής switch αν ο χαρακτήρας ch είναι το 'a', το
'b', το 'c', το 'd' ή το 'e' και σε κάθε περίπτωση θα εμφανίζει την τιμή του
ή το μήνυμα ch  not in ['a':'e'] */
     char ch;  // Δήλωση ch

     // Διάβασμα απ' το πληκτρολόγιο ενός χαρακτήρα ch
     printf ("Give a character between 'a' and 'e' : ");
     scanf("%c", &ch);

    // Έλεγχος, εμφάνιση της τιμής του ch
    switch ( ch )
     {
        case 'a' :printf("ch = %c = 'a'\n", ch );
                  break;
        case 'b' :printf("ch = %c = 'b'\n", ch );
                  break;
        case 'c' :printf("ch = %c = 'c'\n", ch );
                  break;
        case 'd' :printf("ch = %c = 'd'\n", ch );
                  break;
        case 'e' :printf("ch = %c = 'e'\n", ch );
                  break;
        default : printf("ch = %c not in ['a': 'e']\n", ch );
                  break;
     }
}
```

Έξοδος Προγράμματος

```
Give a character between 'a' and 'e' : e
ch = e = 'e'
Press any key to continue . . .

Give a character between 'a' and 'e' : f
ch = f not in ['a': 'e']
Press any key to continue . . .
```

3 ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ (while, do…while)

Στα πιο πολλά προγράμματα απαιτείται κάποια ή κάποιες εντολές να εκτελούνται πολλές φορές για όσο ισχύει κάποια **συνθήκη**. Ο αριθμός των επαναλήψεων μπορεί να είναι άγνωστος ή γνωστός. Στα λογικά διαγράμματα η επανάληψη αυτή των εντολών συμβολίζεται με μια **ανακύκλωση** του ελέγχου ροής του προγράμματος. Όπως και στις εντολές ελέγχου, θα πρέπει να γίνει κάποια **σύγκριση**. Η **εντολή επανάληψης** (σε μορφή διαγράμματος ροής, αλγορίθμου και C) στην περίπτωση που ο αριθμός των επαναλήψεων είναι **άγνωστος**, έχει τη μορφή :




```
Για όσο (<ισχύει κάποια συνθήκη>)
   εκτέλεση εντολής ή block εντολών

ή

while (<συνθήκη>)
   εντολή; ή { block εντολών; }
```

όπου ελέγχεται η συνθήκη, αν είναι αληθής. **Αν** ισχύει η συνθήκη, εκτελείται η εντολή ή οι εντολές (οι οποίες θα πρέπει να περικλείονται σε άγγιστρα, αν είναι περισσότερες από μια) μετά το `while`. Η εκτέλεση των εντολών **τερματίζεται**, όταν πάψει να ισχύει η συνθήκη. Για να γίνει αυτό απαιτείται να υπάρχει μέσα στο `block` εντολών του `while` κάποια εντολή ή εντολές που **αλλάζουν** την τιμή κάποιας ή κάποιων μεταβλητών που περιλαμβάνονται στη συνθήκη.

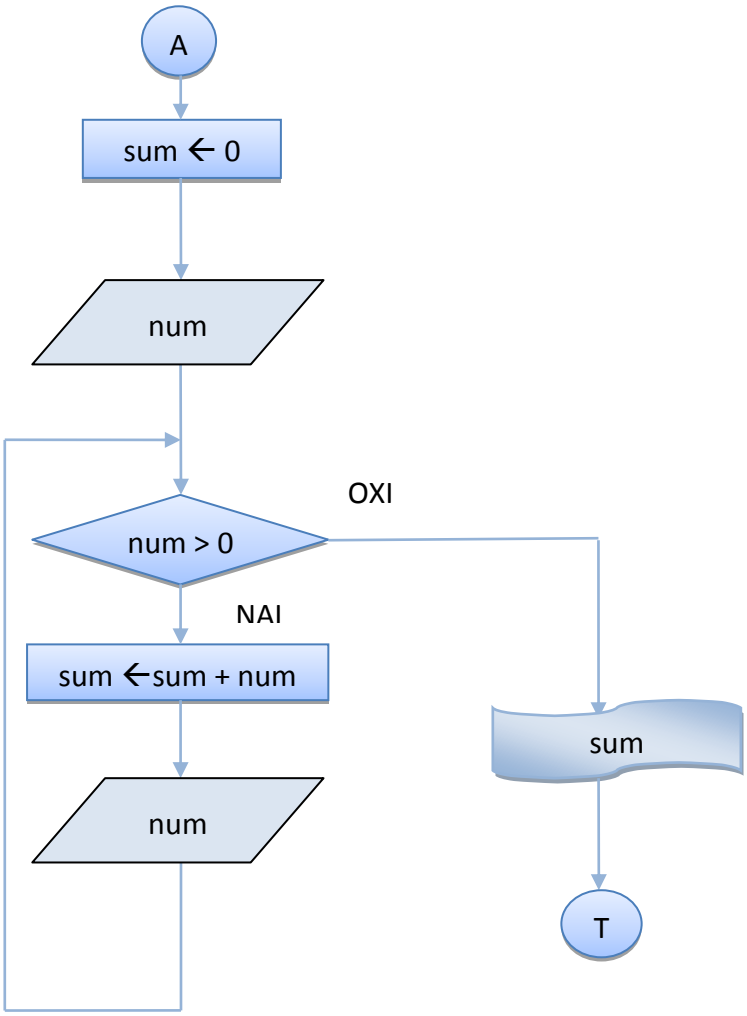
Παρατηρήσεις

- Ανάλογα με τις τιμές που έχουν οι μεταβλητές που περιλαμβάνονται στη συνθήκη πριν το `while`, οι εντολές μπορεί να μην εκτελεστούν **καμιά** φορά, αν η **συνθήκη** είναι εξ αρχής **ψευδής**.

3. 1 Ένα Απλό Πρόγραμμα με την Εντολή Επανάληψης while

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο διαβάζει ακέραιους αριθμούς, οι οποίοι θα αποθηκεύονται στη μεταβλητή **num** και τους προσθέτει σε έναν αθροιστή **sum**, για όσο οι αριθμοί είναι θετικοί. Όταν εισαχθεί κάποιος μη θετικός αριθμός, εμφανίζει την τιμή του αθροιστή **sum** και τερματίζει.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Δίνω αρχική τιμή το μηδέν στον αθροιστή `sum` (`sum` = 0)
2. Διαβάζω έναν ακέραιο αριθμό `num`
3. **Για όσο** ο αριθμός `num` είναι > 0 (`num > 0`)
    i. Προσθέτω το `num` στο `sum` (`sum` ← `sum + num`)
    ii. Διαβάζω έναν νέο ακέραιο αριθμό `num`
4. Εμφανίζω την τιμή του αθροιστή `sum`

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

/* Πρόγραμμα, το οποίο διαβάζει ακέραιους αριθμούς, οι οποίοι θα
αποθηκεύονται στη μεταβλητή num και τους προσθέτει σε έναν αθροιστή sum, για
όσο οι αριθμοί είναι θετικοί.  Όταν δοθεί κάποιος μη θετικός αριθμός,
εμφανίζει την τιμή του αθροιστή sum και τερματίζει. */
main()
{
        int num;
        int sum = 0;

        // Διάβασμα ακέραιου αριθμού
        printf ("Give an integer number num > 0 : ");
        scanf("%d", &num);

        // Για όσο ο αριθμός είναι θετικός
        while ( num > 0)
        {
            // Πρόσθεση του αριθμού στον αθροιστή sum
            sum = sum + num;

            // // Διάβασμα νέου ακέραιου αριθμού
            printf ("Give an integer number num > 0 : ");
            scanf("%d", &num);
        }

        // Εμφάνιση αθροίσματος sum
        printf ("sum = %d\n ", sum);
        system("Pause");
}
```

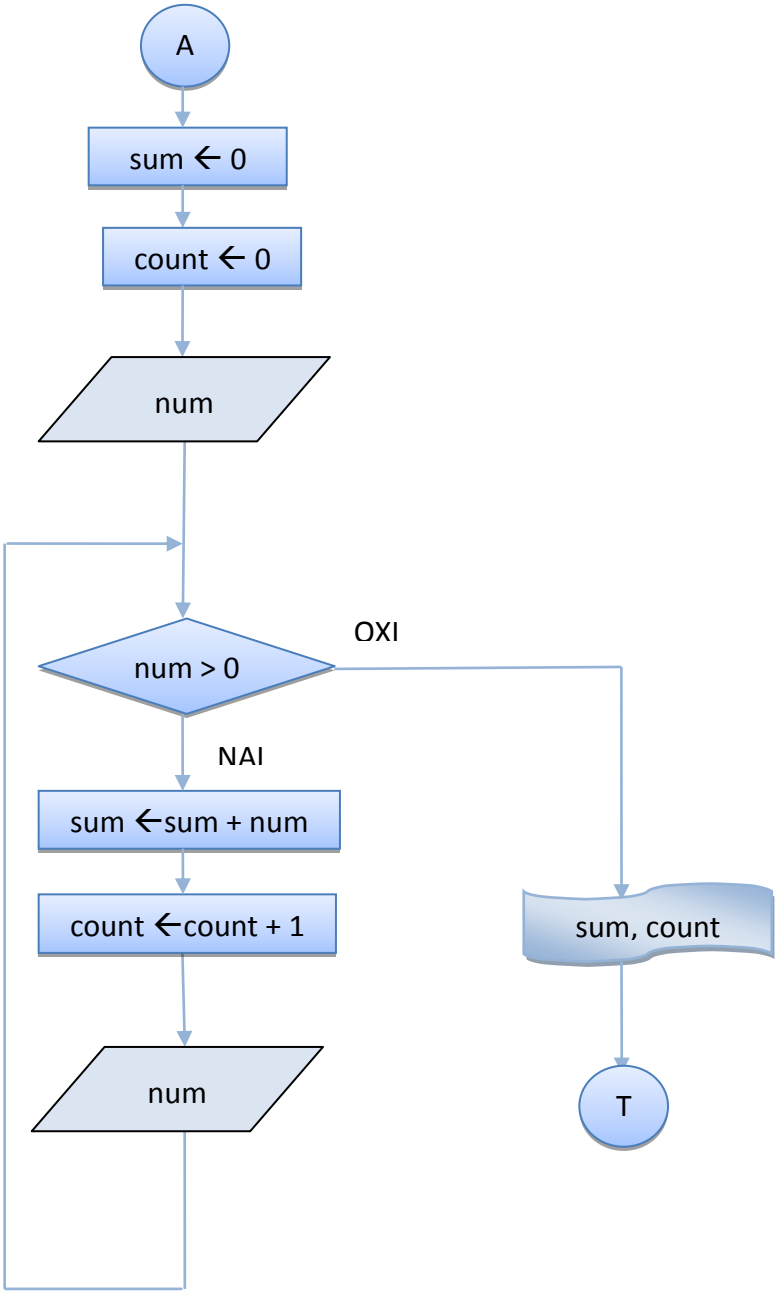
Έξοδος Προγράμματος

```
Give an integer number num > 0 : 5
Give an integer number num > 0 : 7
Give an integer number num > 0 : 0
sum = 12
Press any key to continue . . .
```

3. 2 Τροποποίηση Προγράμματος 3.1 για τον Υπολογισμό και του Πλήθους των Αριθμών με την Εντολή Επανάληψης while

Να τροποποιηθεί το Πρόγραμμα 3.1, ώστε να διαβάζει ακέραιους αριθμούς, οι οποίοι θα αποθηκεύονται στη μεταβλητή **num**, θα τους προσθέτει σε έναν αθροιστή **sum**, για όσο οι αριθμοί είναι θετικοί **και θα τους μετράει**. Όταν δημιουργηθεί κάποιος μη θετικός αριθμός, θα εμφανίζει **την τιμή του μετρητή** και του αθροιστή **sum** και θα τερματίζει.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Δίνω αρχική τιμή το μηδέν στον αθροιστή `sum` (`sum` = 0)
2. **Δίνω αρχική τιμή το μηδέν στο μετρητή `count` (`count` = 0)**
5. Διαβάζω έναν ακέραιο αριθμό `num`
6. **Για όσο** ο αριθμός `num` είναι > 0 (`num > 0`)
   i. Προσθέτω το `num` στο `sum` (`sum` ← `sum + num`)
   ii. Αυξάνω την τιμή του μετρητή κατά 1 (`count` ← `count + 1`)
   iii. Διαβάζω έναν νέο ακέραιο αριθμό `num`
3. Εμφανίζω την τιμή του αθροιστή `sum` **και του μετρητή `count`**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

/* Πρόγραμμα, το οποίο διαβάζει ακέραιους αριθμούς, οι οποίοι θα
αποθηκεύονται στη μεταβλητή num και τους προσθέτει σε έναν αθροιστή sum και
τους μετράει, για όσο οι αριθμοί είναι θετικοί.  Όταν δοθεί κάποιος μη
θετικός αριθμός, εμφανίζει την τιμή του αθροιστή sum και του μετρητή count
και τερματίζει.
*/
main()
{
        int num;
        int sum = 0;
        int count = 0;

        // Διάβασμα ακέραιου αριθμού
        printf ("Give an integer number num > 0 : ");
        scanf("%d", &num);

        // Για όσο ο αριθμός είναι θετικός
        while ( num > 0)
        {
            // Πρόσθεση του αριθμού στον αθροιστή sum
            sum = sum + num;

            // Αύξηση του μετρητή count κατά 1
            count++;

        // Διιάβασμα νέου ακέραιου αριθμού
        printf ("Give an integer number num > 0 : ");
        scanf("%d", &num);
        }
        // Εμφάνιση αθροίσματος sum και μετρητή count
        printf ("sum = %d count = %d\n ", sum, count);
        system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer number num > 0 : 5
Give an integer number num > 0 : 7
Give an integer number num > 0 : 0
sum = 12 count = 2
Press any key to continue . . .
```

Άσκηση 3.1

Να τροποποποιηθεί ο αλγόριθμος 3.2, ώστε να βρίσκει - σε κάθε περίπτωση - και να εμφανίζει και το **Μέσο Όρο**.

Παρατηρήσεις

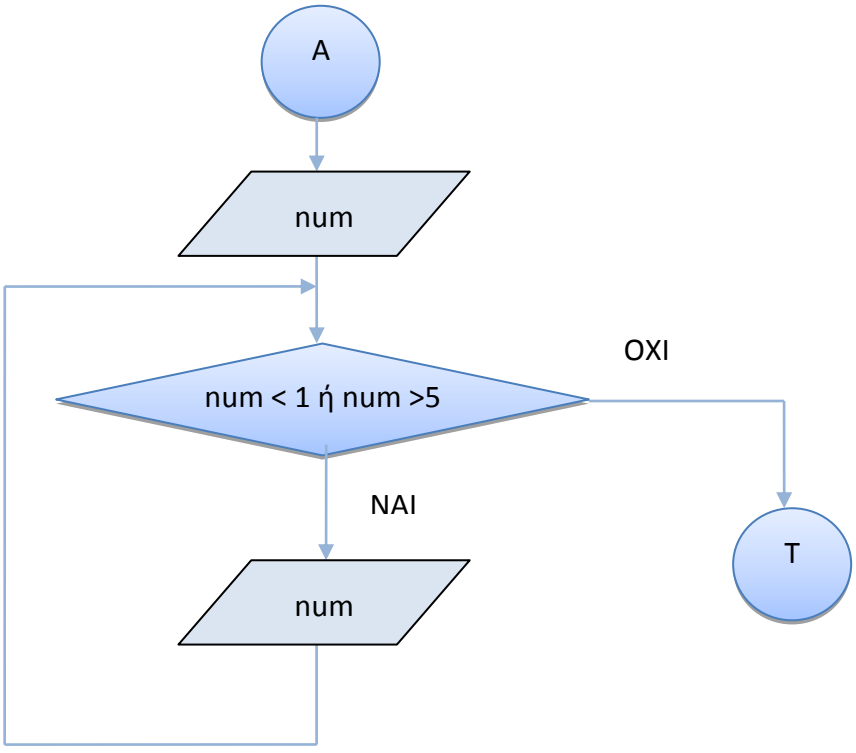
- Η δημιουργία του τυχαίου αριθμού και η εμφάνιση της τιμής του επαναλαβάνονται δύο φορές. **Πριν** την εντολή `while`, γιατί η μεταβλητή `num` που κάνει τον έλεγχο στη συνθήκη πρέπει να έχει κάποια τιμή για να γίνει η πρώτη σύγκριση, αλλά και **μέσα** στο σώμα του `while`, γιατί μας χρειάζεται ο νέος αριθμός. Στην περίπτωση που ο πρώτος αριθμός είναι το μηδέν, η συνθήκη του `while` είναι εξ αρχής ψευδής, οπότε δεν εκτελούνται ποτέ οι εντολές που υπάρχουν στο σώμα του `while`.

- Σε πολλές εφαρμογές θα χρειαστεί να εμφανίσουμε ένα `menu` επιλογών και να διαβάσουμε κάποια τιμή για την επιλογή, ώστε να εκτελέσουμε και τις αντίστοιχες εντολές. Στο επόμενο παράδειγμα υλοποιούμε κάτι αντίστοιχο, δίνοντας έναν αριθμό μεταξύ του `1` και `5`.

3. 3 Ένα Απλό Πρόγραμμα για Δημιουργία Επιλογής με while

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο διαβάζει ακέραιους αριθμούς, οι οποίοι θα αποθηκεύονται στη μεταβλητή **num**. Για όσο οι αριθμοί είναι μικρότεροι του `1` ή μεγαλύτεροι του `5`, το πρόγραμμα διαβάζει νέο αριθμό. Το πρόγραμμα τερματίζει, όταν ο αριθμός που θα δοθεί είναι μεταξύ του `1` και του `5`.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν ακέραιο αριθμό num
2. **Για όσο** ο αριθμός num είναι < 1 ή > 5 (num < 1 ή num > 5)
   a. Διαβάζω έναν νέο ακέραιο αριθμό num

ΠΡΟΓΡΑΜΜΑ

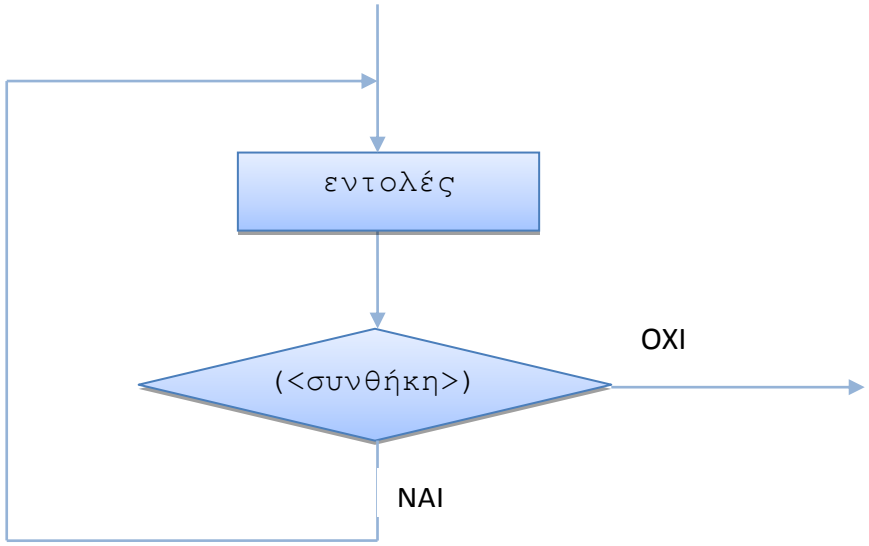
```
#include <stdio.h>
#include <stdlib.h>
/* Πρόγραμμα, το οποίο διαβάζει ακέραιους αριθμούς, οι οποίοι θα αποθηκεύονται στη μεταβλητή num. Για όσο οι αριθμοί είναι μικρότεροι του 1 ή μεγαλύτεροι του 5, το πρόγραμμα διαβάζει νέο αριθμό.  Το πρόγραμμα τερματίζει, όταν ο αριθμός που θα δοθεί είναι μεταξύ του 1 και του 5. */
main()
{
        int num;
        // Διάβασμα ακέραιου αριθμού
        printf ("Give an integer number num between 1 and 5 : ");
        scanf("%d", &num);

        // Για όσο ο αριθμός είναι εκτός των ορίων ( 1-5) που θέλουμε
        while (( num < 1) || ( num > 5))
          {
          // Διάβασμα νέου ακέραιου αριθμού
          printf ("Give an integer number num between 1 and 5  : ");
          scanf("%d", &num);
          }

        system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer number num between 1 and 5  : -1
Give an integer number num between 1 and 5  : 7
Give an integer number num between 1 and 5  : 3
Press any key to continue . . .

Give an integer number num between 1 and 5  : 2
Press any key to continue . . .
```

❖ Στο προηγούμενο πρόγραμμα χρειαζόμαστε **οπωσδήποτε** έναν αριθμό μεταξύ του 1 και 5. Σ' αυτή την περίπτωση, θα μπορούσαμε να αποφύγουμε την επανάληψη των εντολών Δημιουργίας και Εμφάνισης της τιμής του τυχαίου αριθμού num, χρησιμοποιώντας μια άλλη μορφή της εντολής while, τη do while, στην οποία ο έλεγχος τη συνθήκης γίνεται **ΜΕΤΑ** την εκτέλεση των εντολών και όχι **ΠΡΙΝ**, όπως γίνεται με την εντολή while.

Η **σύνταξη** (σε μορφή διαγράμματος ροής, αλγορίθμου και C) της εντολής `do while` είναι:




```
Κάνε τα παρακάτω
   εντολές;
Για όσο (<η συνθήκη ισχύει>
```

ή

```
do {
       εντολή; ή εντολές;
  }
while ( <συνθήκη>);
```

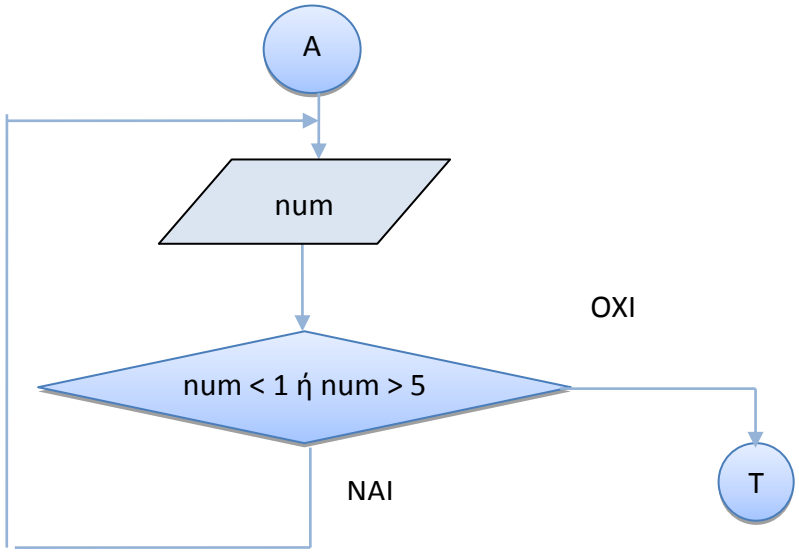
Παρατήρηση

Επειδή με το `while` τελειώνει η εντολή, χρειάζεται στο τέλος ερωτηματικό.

3. 4 Τροποποίηση του Προγράμματος 3.3 για τη Δημιουργία Επιλογής με την εντολή do while

Να τροποποιηθεί το πρόγραμμα 3.3, ώστε να διαβάζει ακέραιους αριθμούς, οι οποίοι θα αποθηκεύονται στη μεταβλητή **num**. Για όσο οι αριθμοί είναι μικρότεροι του 1 ή μεγαλύτεροι του 5, το πρόγραμμα θα διαβάζει νέο αριθμό. Το πρόγραμμα θα τερματίζει, όταν ο αριθμός που θα δοθεί είναι μεταξύ του 1 και του 5. **Η υλοποίηση να γίνει με την εντολή do while.**

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

Κάνε τα παρακάτω

1. Διαβάζω έναν ακέραιο αριθμό num

Για όσο ο αριθμός num είναι < 1 ή > 5 (num < 1 ή num > 5)

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/* Πρόγραμμα, το οποίο διαβάζει ακέραιους αριθμούς, οι οποίοι θα αποθηκεύονται στη μεταβλητή num. Για όσο οι αριθμοί είναι μικρότεροι του 1 ή μεγαλύτεροι του 5, το πρόγραμμα διαβάζει νέο αριθμό.  Το πρόγραμμα τερματίζει, όταν ο αριθμός που θα δοθεί είναι μεταξύ του 1 και του 5. */
main()
{
        int num;

        do
        {
         // Διάβασμα ακέραιου αριθμού
         printf ("Give an integer number num between 1 and 5 : ");
         scanf("%d", &num);
        }
        // Για όσο ο αριθμός είναι εκτός των ορίων ( 1-5) που θέλουμε
        while (( num < 1) || ( num > 5));
        system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer number num between 1 and 5  : 3
Press any key to continue . . .
```

Παρατήρηση

- Οι εντολές μετά το do στην εντολή do while εκτελούνται **ΤΟΥΛΑΧΙΣΤΟΝ** μία φορά.

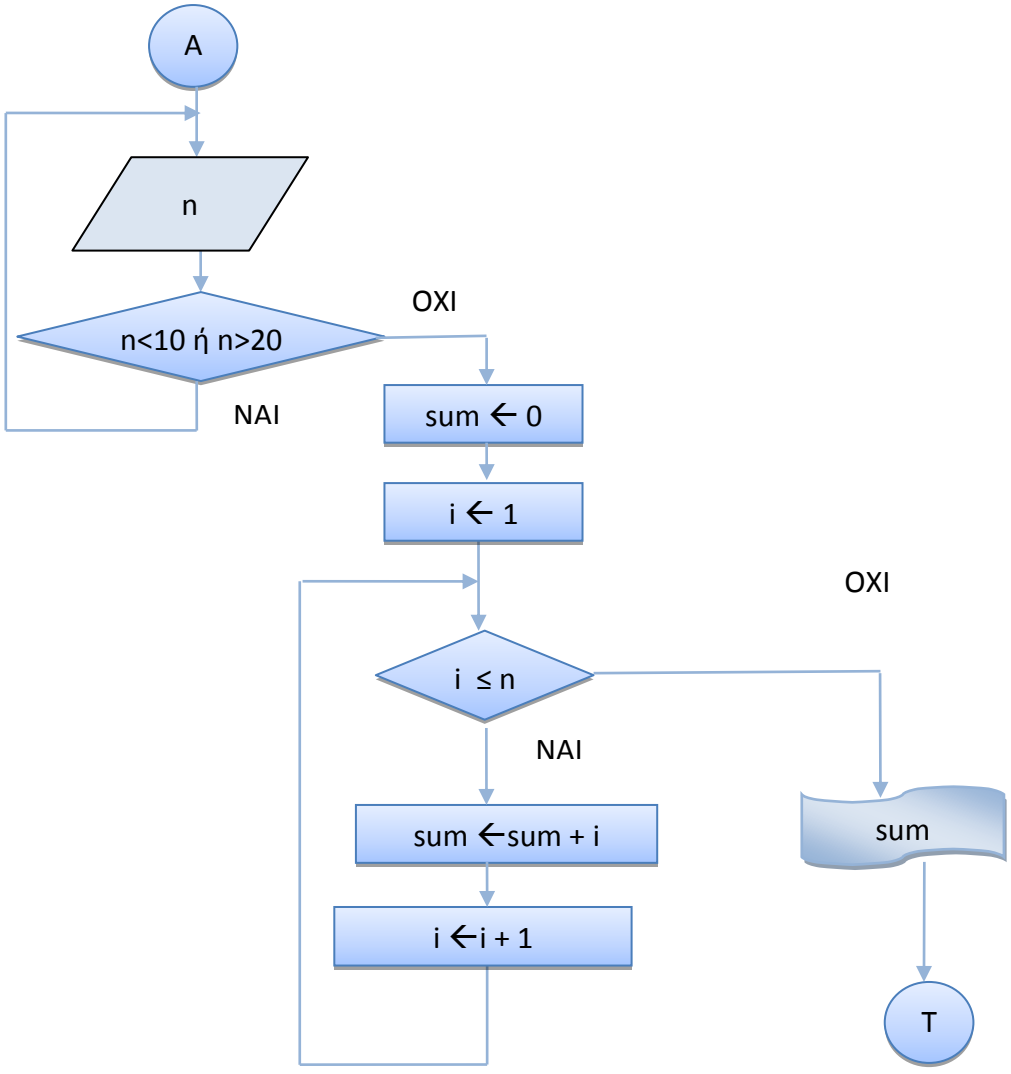
4 ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ - for

Υπάρχουν προβλήματα, στα οποία ο αριθμός των επαναλήψεων κάποιων εντολών είναι **γνωστός** εκ των προτέρων, όπως στο επόμενο παράδειγμα :

4. 1 Πρόγραμμα για τον Υπολογισμό του Αθροίσματος 1+2+… + n με την **Εντολή Επανάληψης while**

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα διαβάζει έναν ακέραιο αριθμό μεταξύ του 10 και 20, ο οποίος θα αποθηκεύεται στη μεταβλητή **n** και θα υπολογίζει και θα εμφανίζει το άθροισμα 1 + 2 + ... + **n**.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν ακέραιο αριθμό `n` στο `[10, 20]`
2. Δίνω αρχική τιμή το μηδέν στον αθροιστή `sum` (`sum` ← 0)
3. Δίνω αρχική τιμή το `1` στον μετρητή `i` (`i` ← `1`)
4. **Για όσο** ο μετρητής `i` είναι ≤ n (`i` ≤ n)
   a. Προσθέτω το `i` στο `sum` (`sum` ← `sum + i`)
   b. Αυξάνω την τιμή του μετρητή κατά `1` (`i` ← `i + 1`)
5. Εμφανίζω την τιμή του αθροιστή `sum`

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/*Πρόγραμμα, το οποίο  διαβάζει έναν ακέραιο αριθμό μεταξύ του  10 και 20, ο
οποίος θα αποθηκεύεται στη μεταβλητή n,  και  υπολογίζει και  εμφανίζει το
άθροισμα 1 + 2 + ... + n.*/
main()
{
        int i, n, sum;

        do
        {
         // Διάβασμα ακέραιου αριθμού
         printf ("Give an integer number n between 10 and 20 : ");
         scanf("%d", &n);
        }
        while (( num < 10) || ( num > 20));

        // Αρχικές τιμές στο άθροισμα και τον μετρητή
        sum = 0;
        i = 1;

        // Για όσο ο μετρητής δεν ξεπέρασε το n
        while (i <= n)
        {
          // Πρόσθεση του μετρητή i στον αθροιστή sum
          sum += i;

          // Αύξηση του μετρητή i κατά 1
          i++;
        }

       // Εμφάνιση αθροίσματος sum
       printf ("sum = %d \n", sum);
       system("Pause");
}
```

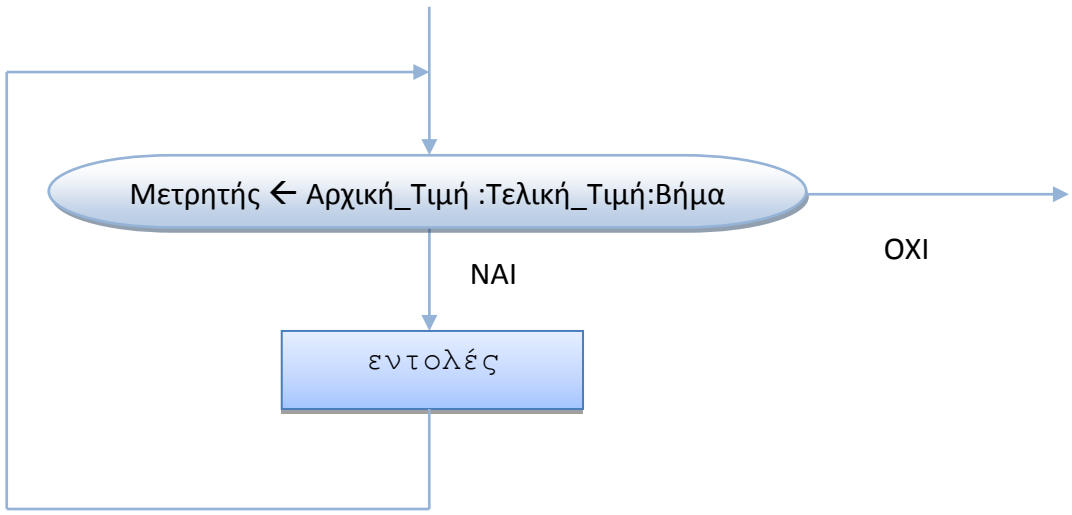
Έξοδος Προγράμματος

```
Give an integer number n between 10 and 20 : 12
sum = 78
Press any key to continue . . .
```

ΑΣΚΗΣΗ 4.1 : Να τροποποιηθεί ο προηγούμενος αλγόριθμος ώστε να υπολογίζει/ εμφανίζει και το **Μέσο Όρο** των αριθμών `1 + 2 + ... + n` με **`do`**...**`while`**.

4. 2 Η Εντολή Επανάληψης for

Στα προβλήματα με γνωστό αριθμό επαναλήψεων, οι εντολές ανάθεσης **αρχικής τιμής** στο μετρητή, ελέγχου της **συνθήκης**, αν **ο μετρητής ξεπέρασε την τελική τιμή** και **ενημέρωσης της τιμής του μετρητή** μέσα στο σώμα της επανάληψης μπορούν να συμπεριληφθούν σε μια εντολή, την εντολή `for`, της οποίας η σύνταξη (σε μορφή διαγράμματος ροής, αλγορίθμου και `C`) είναι :



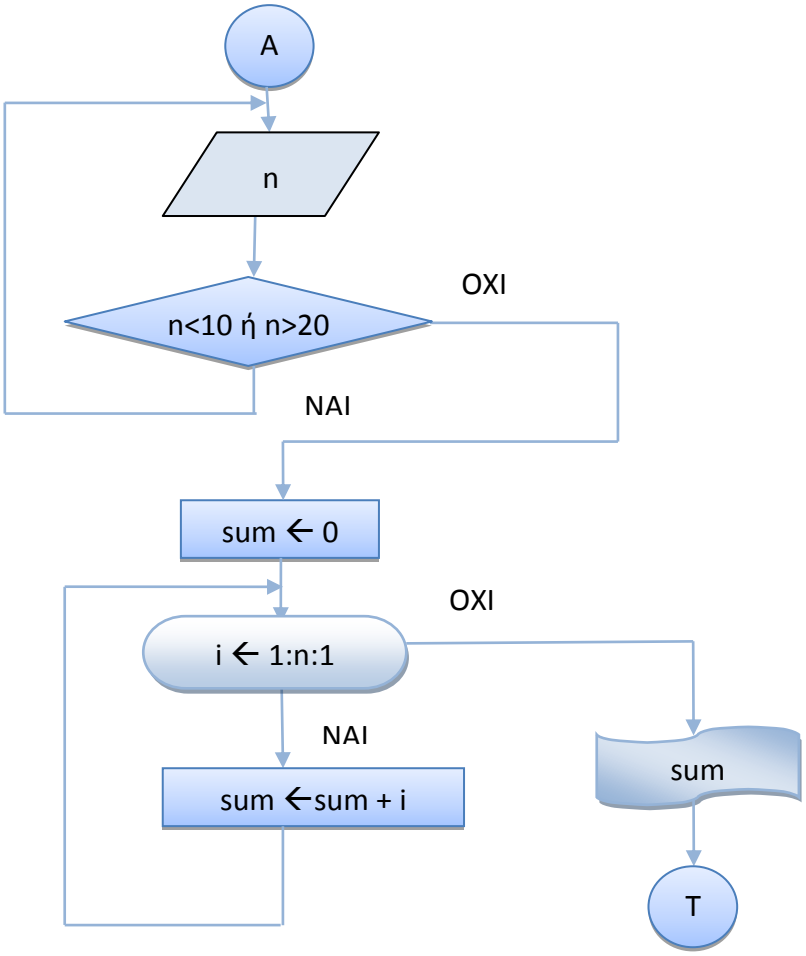
`Για` <μετρητής>=<αρχική τιμή>:<τελική τιμή>:<βήμα>
`εντολές;`

```
for (<αρχική τιμή μετρητή>; <ο μετρητής ξεπέρασε την τελική τιμή?>; <ενημέρωση τιμής μετρητή>)
{
    εντολές;
}
```

4.2.1 Πρόγραμμα για τον Υπολογισμό του Αθροίσματος 1+2+…+ n με την Εντολή Επανάληψης for

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα διαβάζει έναν ακέραιο αριθμό μεταξύ του `10` και `20`, ο οποίος θα αποθηκεύεται στη μεταβλητή **n** και θα υπολογίζει και θα εμφανίζει το άθροισμα `1 + 2 + ... + n` **με την εντολή επανάληψης for**.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν ακέραιο αριθμό `n` στο `[10, 20]`
2. Δίνω αρχική τιμή το μηδέν στον αθροιστή `sum` (`sum` ← 0)
3. **Για** τις τιμές του μετρητή `i` από το `1` μέχρι και το `n` με βήμα `1`
   Προσθέτω το `i` στο `sum` (`sum ← sum + i`)
4. Εμφανίζω την τιμή του αθροιστή `sum`

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/*Πρόγραμμα, το οποίο  διαβάζει έναν ακέραιο αριθμό μεταξύ του  10 και 20, ο
οποίος αποθηκεύεται στη μεταβλητή n  και υπολογίζει και εμφανίζει το άθροισμα
1 + 2 + ... + n με την  εντολή for.*/
main()
{
        int i, n, sum;

        do
        {
         // Διάβασμα ακέραιου αριθμού
         printf ("Give an integer number n between 10 and 20 : ");
         scanf("%d", &n);
        }
        while (( num < 10) || ( num > 20));


        // Αρχική τιμή 0 στο άθροισμα
        sum = 0;

        // Για την τιμή του μετρητή i από το 1 μέχρι και το n
        for ( i = 1;i <= n;i++ )
          // Πρόσθεση του αριθμού στον αθροιστή sum
          sum += i;

        // Εμφάνιση αθροίσματος sum
       printf ("sum = %d \n", sum);
       system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer number n between 10 and 20 : 12
sum = 78
Press any key to continue . . .
```

ΑΣΚΗΣΗ 4.2 : Να τροποποιηθεί ο προηγούμενος αλγόριθμος ώστε να υπολογίζει και να εμφανίζει και το **Μέσο Όρο** των αριθμών `1 + 2 + ... + n`.

ΑΣΚΗΣΗ 4.3 : Να τροποποιηθεί ο προηγούμενος αλγόριθμος ώστε να υπολογίζει και να εμφανίζει το **Άθροισμα** και το **Μέσο Όρο** των αριθμών `n+(n-1)+…+2+1` (ο μετρητής θα αρχίσει από το `n` και θα **μειώνεται** κατά `1`, μέχρι και το `1`).

Παρατηρήσεις

Η εντολή `for` δεν είναι απαραίτητο να περιέχει και την <αρχική τιμή στο μετρητή> τον έλεγχο αν <ο μετρητής ξεπέρασε την τελική τιμή> και την <ενημέρωση της τιμής του μετρητή>.

Παράδειγμα

Από την προηγούμενη πλήρη εντολή επανάληψης `for`

```
for (i = 1;i <= n;i++)
  sum += i;
```

μπορεί να λείπουν κάποια απ' αυτά ή όλα, αρκεί να υπάρχουν τα αντίστοιχα ερωτηματικά.

Π.χ. θα μπορούσε να λείπει :

- Η <αρχική τιμή στο μετρητή>, οπότε θα έχουμε :

```
i = 1;
for (;i <= n;i++)
  sum += i;
```

- Η <ενημέρωση της τιμής του μετρητή>, οπότε θα έχουμε:

```
for (i = 1;i <= n;){
  sum += i;
  i++;
}
```

- Η <αρχική τιμή στο μετρητή> και η <ενημέρωση της τιμής του μετρητή>, οπότε θα έχουμε :

```
i = 1;
for (;i <= n;){
  sum += i;
  i++;
}
```

- Το σώμα της εντολής επανάληψης . Π.χ.

```
for (i = 1;i <= n; sum += i++);
```

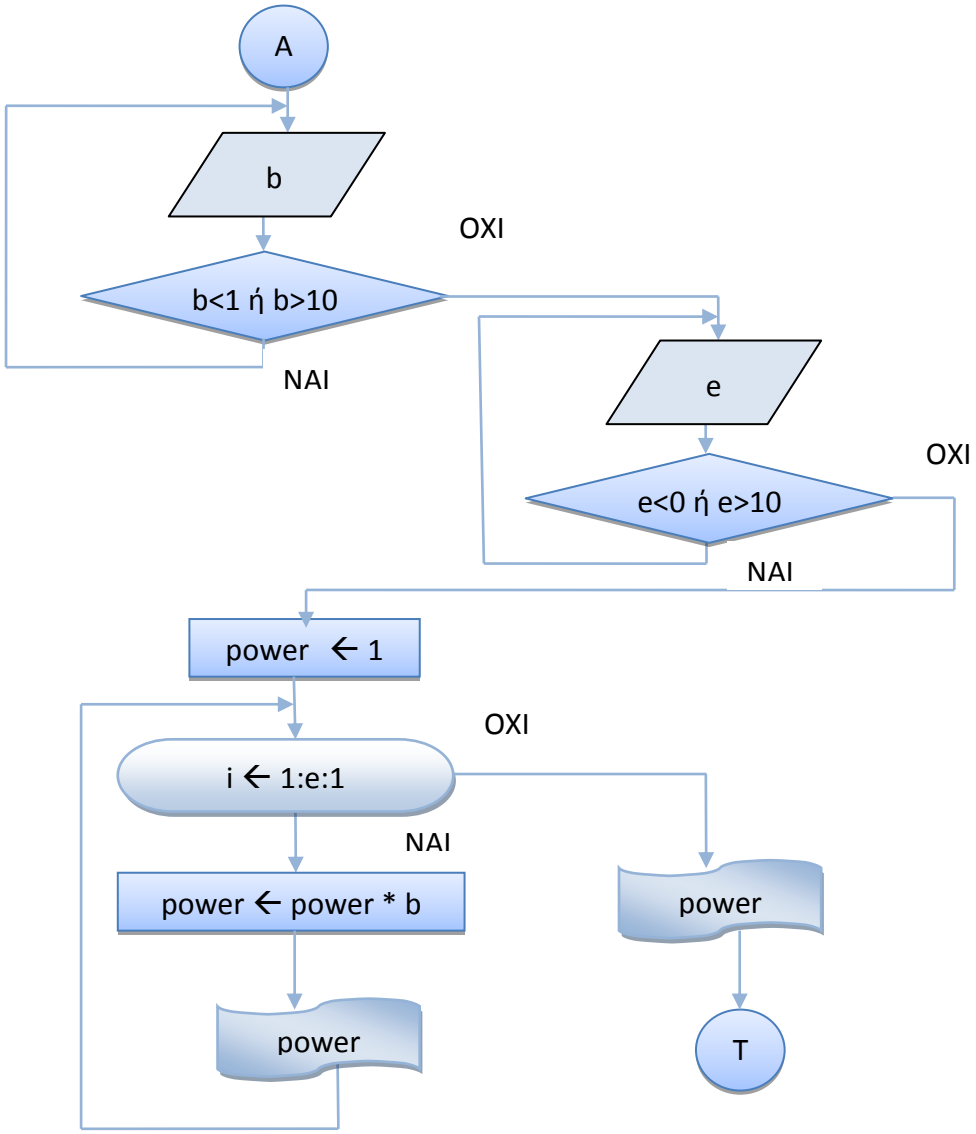
4.2.2 Πρόγραμμα για τον Υπολογισμό του x^y με την Εντολή Επανάληψης for

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα προσομοιώνει την συνάρτηση `pow( Βάση, Εκθέτης )` με **Βάση** έναν **ακέραιο** αριθμό και **εκθέτη** έναν **ακέραιο** αριθμό. Διαβάζει έναν **ακέραιο** αριθμό στο `1-10` για τη Βάση και έναν ακέραιο αριθμό στο `0-10` για τον Εκθέτη. Για να υπολογίσει τη δύναμη `Βάση^Εκθέτη`, δίνει την τιμή `1` σαν αρχική τιμή στη δύναμη και με την εντολή `for` πολλαπλασιάζει τη δύναμη με τη Βάση, όσες φορές είναι η ακέραια τιμή του εκθέτη και εμφανίζει κάθε φορά την τιμή της δύναμης.

ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν **ακέραιο** αριθμό `b` στο `[1, 10]`
2. Διαβάζω έναν **ακέραιο** αριθμό `e` στο `[0, 10]`
3. Δίνω αρχική τιμή στη `Δύναμη power = 1`
4. **Για** τις τιμές του μετρητή `i` από το `1` μέχρι και τον εκθέτη `e`
   Πολλαπλασιάζω τη Δύναμη με τη Βάση (`power ← power * b`)
   Εμφανίζω την τιμή της Δύναμης `power`
5. Εμφανίζω την τελική τιμή της Δύναμης `power`

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/*Πρόγραμμα, το οποίο προσομοιώνει τη μέθοδο pow( Βάση, Εκθέτης ) με Βάση
έναν ακέραιο αριθμό και εκθέτη έναν ακέραιο αριθμό. Διαβάζει έναν ακέραιο
αριθμό στο 1-10 για τη Βάση και έναν ακέραιο αριθμό στο 0-10 για τον Εκθέτη.
Για να υπολογίσει τη δύναμη Βάση^Εκθέτη, δίνει την τιμή 1 σαν αρχική τιμή στη
δύναμη και με την εντολή for πολλαπλασιάζει τη δύναμη με τη Βάση, όσες φορές
είναι η ακέραια τιμή του εκθέτη και εμφανίζει κάθε φορά την τιμή της
δύναμης.*/
main()
{
        int b, e, i, power;

        do
        {
         // Διάβασμα ακέραιου αριθμού για τη βάση
         printf ("Give an integer number b (bash) between 1 and 10 : ");
         scanf("%d", &b);
        }
        while (b < 1 || b > 10);
       do
        {
        printf ("Give an integer number e (ektheths) between 0 and 10 : ");
        scanf("%d", &e);
        }
       while (e < 0 || e > 10);

       // Αρχική Τιμή στη Δύναμη = 1
       power = 1;

       // Για τόσες φορές όσες η τιμή του εκθέτη
       for ( i = 1; i <= e; i++ )
       {
           // Υπολογισμός Επόμενης Τιμής της Δύναμης
           power = power * b;

           // Εμφάνιση της νέας Τιμής της Δύναμης
           printf ("power # %d = %d \n", i, power);       }
       }
       // Εμφάνιση της Τελικής Τιμής της Δύναμης
       printf ("last power = %d \n", power);
       system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer number b (bash) between 1 and 10 : 5
Give an integer number e (ektheths) between 0 and 10 : 3
power # 1 = 5
power # 2 = 25
power # 3 = 125
last power = 125
Press any key to continue . . .


Give an integer number b (bash) between 1 and 10 : 1
Give an integer number e (ektheths) between 0 and 10 : 3
power # 1 = 1
power # 2 = 1
power # 3 = 1
last power = 1
Press any key to continue . . .
```

```
Give an integer number b (bash) between 1 and 10 : 3
Give an integer number e (ektheths) between 0 and 10 : 0
last power = 1
Press any key to continue . . .
```

ΑΣΚΗΣΗ 4.4 : Να τροποποιηθεί ο προηγούμενος αλγόριθμος ώστε να ελέγχει και την περίπτωση που **η βάση είναι 0 ή 1** και να εμφανίζει την τιμή της, **χωρίς** να χρειαστεί να κάνει καμιά επανάληψη.

4. 3 Εμφωλευμένες Εντολές Επανάληψης for-while, break, continue

Στα επόμενα παραδείγματα εξετάζεται η χρήση της εντολής επανάληψης `for` μέσα στο σώμα μιας εντολής επανάληψης `while`, η χρήση της εντολής επανάληψης `for` μέσα στο σώμα μιας άλλης εντολής επανάληψης `for`, και η χρήση των εντολών `break` και `continue`.

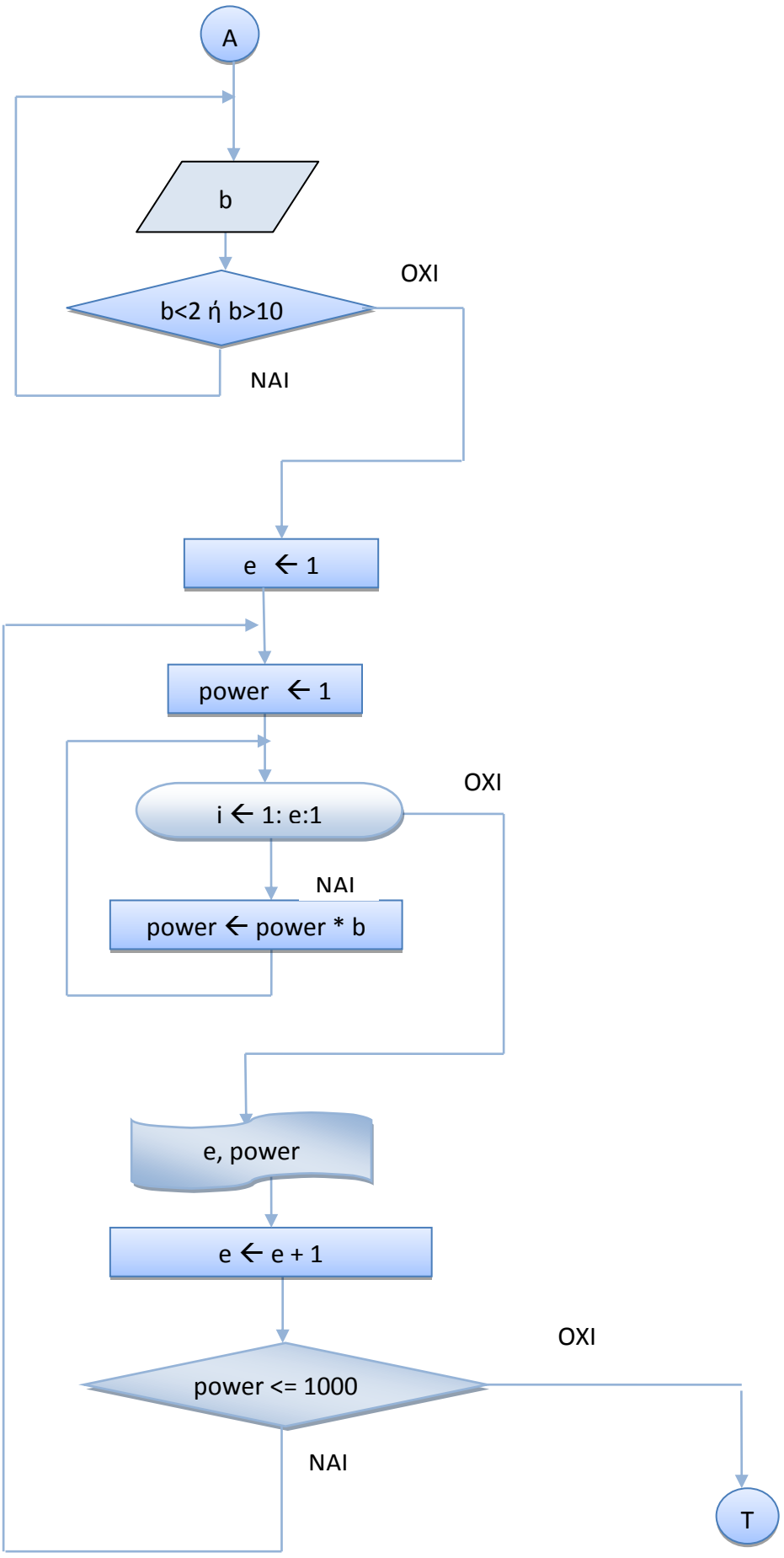
4.3.1 Εμφωλευμένοι Βρόχοι (for – while)

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα υπολογίζει και θα εμφανίζει όλες τις **Δυνάμεις** ενός ακέραιου αριθμού στο [2,10] μέχρι που η Δύναμη να ξεπεράσει το **1000** χρησιμοποιώντας τις Εντολές Επανάληψης **`for`** για τον υπολογισμό της κάθε δύναμης και **`do while`** για τον έλεγχο του τερματισμού.

ΑΛΓΟΡΙΘΜΟΣ

1. Διαβάζω έναν **ακέραιο** αριθμό `b` στο `[2, 10]`
2. Αρχική Τιμή στον Εκθέτη `e` ← `1`
3. **Κάνε** τα παρακάτω
   Δίνω αρχική τιμή στη `Δύναμη power` ← `1`
   Για τις τιμές του μετρητή `i` από το `1` μέχρι και τον εκθέτη `e`
   Πολλαπλασιάζω τη Δύναμη με τη Βάση (`power` ← `power * b`)
   Εμφανίζω την τιμή της Δύναμης και του εκθέτη
   Αυξάνω την τιμή του εκθέτη κατά 1
   Για όσο η Δύναμη είναι μικρότερη ή ίση του `1000`

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/*
Πρόγραμμα, το οποίο Υπολογίζει και εμφανίζει όλες τις Δυνάμεις ενός ακέραιου
αριθμού στο [2,10] μέχρι που η Δύναμη να ξεπεράσει το 1000 χρησιμοποιώντας
τις Εντολές Επανάληψης for για τον υπολογισμό της κάθε δύναμης και do while
για τον έλεγχο του τερματισμού.
*/
main()
{
        int b, e, i, power;

        do
        {
         // Διάβασμα ακέραιου αριθμού για τη βάση
         printf ("Give an integer number b (bash) between 2 and 10 : ");
         scanf("%d", &b);
        }
        while (b < 2 || b > 10);
       // Αρχική Τιμή στον Εκθέτη = 1
       e = 1;

       do {
         // Αρχική Τιμή στη Δύναμη = 1
         power = 1;

         // Για τόσες φορές όσες η τιμή του εκθέτη
         for ( i = 1; i <= e; i++) {
           // Υπολογισμός Επόμενης Τιμής της Δύναμης
           power = power * b;
         }
         // Εμφάνιση του εκθέτη και της νέας Τιμής της Δύναμης
         printf ("power # %d = %d \n", e, power);
         // Αϋξηση του Εκθέτη κατά 1
         e += 1;
       }
       while (power <= 1000);
}
```

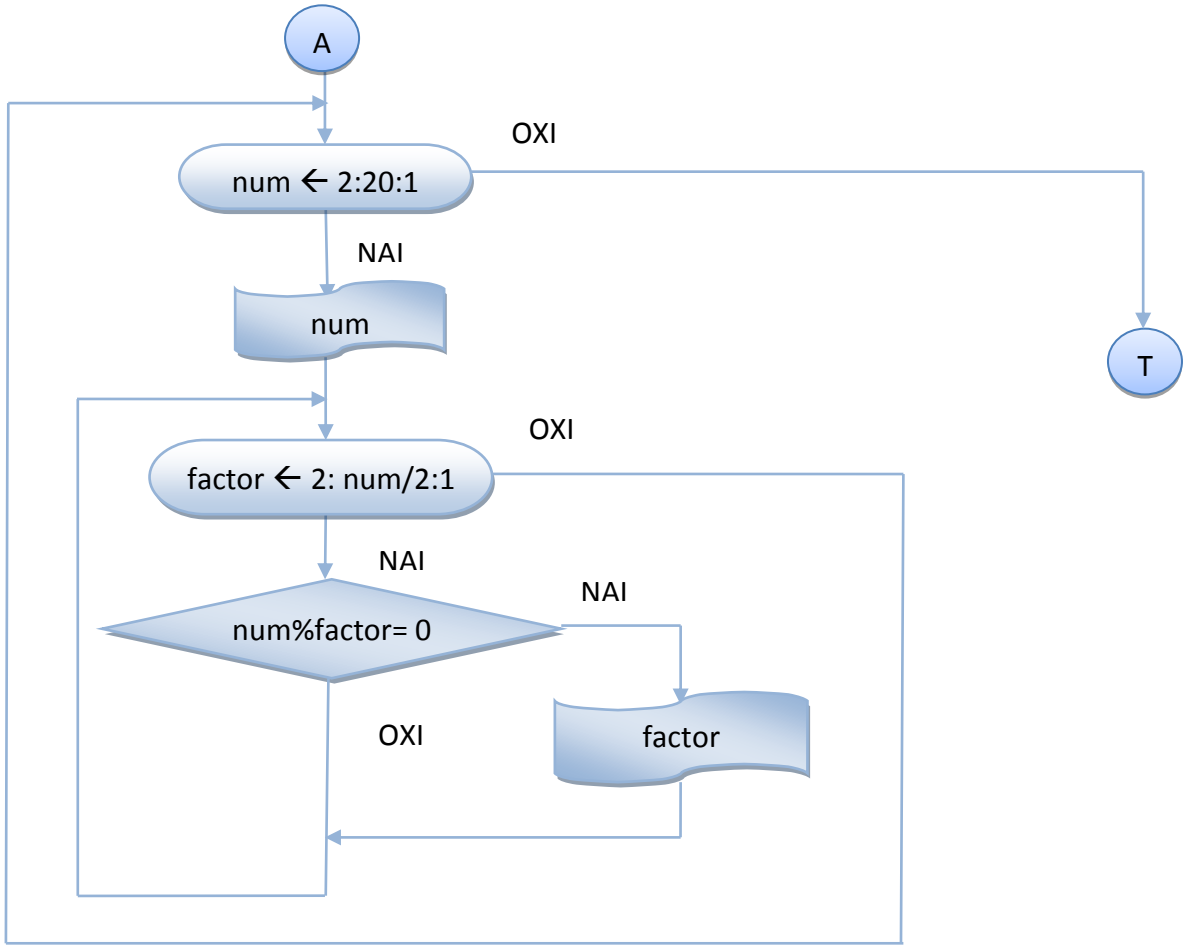
Έξοδος Προγράμματος

```
Give an integer number b (bash) between 2 and 10 : 2
power # 1 = 2
power # 2 = 4
power # 3 = 8
power # 4 = 16
power # 5 = 32
power # 6 = 64
power # 7 = 128
power # 8 = 256
power # 9 = 512
power # 10 = 1024
Press any key to continue . . .
```

4.3.2 Εμφωλευμένοι Βρόχοι (for – for)

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα υπολογίζει όλους τους **παράγοντες** των αριθμών από το `2` μέχρι το `20` (για τον κάθε αριθμό θα βρίσκει και θα εμφανίζει τους **διαιρέτες** του **εκτός** της μονάδας και του ίδιου του αριθμού) χρησιμοποιώντας **δύο** Εντολές Επανάληψης **`for`**.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. **Για** τις τιμές του μετρητή `num` από το `2` μέχρι και το `20`
   Εμφάνιση της τιμής του μετρητή `num`
   Για τις τιμές του `factor` από το `2` μέχρι και το `num/2`
   Αν ο αριθμός `num` διαιρείται ακριβώς με το `factor`
   Εμφανίζω την τιμή του παράγοντα `factor`

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/*Πρόγραμμα, το οποίο υπολογίζει όλους τους παράγοντες των αριθμών από το 2
μέχρι το 20 ( για τον κάθε αριθμό θα βρίσκει και θα εμφανίζει  τους διαιρέτες
του εκτός της μονάδας και του ίδιου του αριθμού ) χρησιμοποιώντας δύο Εντολές
Επανάληψης for.*/
main()
{
        int num, factor;

        // Για τις τιμές του μετρητή num από το 2 μέχρι και το 20
        for ( num = 2;num <= 20;num++ ) {

            // Εμφάνιση της τιμής του μετρητή num
            printf ("Factors of number %d = ", num);
            // Για τις τιμές του factor από το 2 μέχρι και το num/2
            for ( factor = 2;factor <= num/2;factor++ )

                // Αν ο αριθμός num διαιρείται ακριβώς με το factor
                if ( num % factor == 0)

                    // Εμφάνιση της τιμής του παράγοντα factor
                    printf ("%d ", factor);

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

Έξοδος Προγράμματος

```
Factors of 2 :
Factors of 3 :
Factors of 4 : 2
Factors of 5 :
Factors of 6 : 2  3
Factors of 7 :
Factors of 8 : 2  4
Factors of 9 : 3
Factors of 10 : 2  5
Factors of 11 :
Factors of 12 : 2  3  4  6
Factors of 13 :
Factors of 14 : 2  7
Factors of 15 : 3  5
Factors of 16 : 2  4  8
Factors of 17 :
Factors of 18 : 2  3  6  9
Factors of 19 :
Factors of 20 : 2  4  5  10
Press any key to continue . . .
```

Άσκηση 4.5 : Να τροποποιηθεί το προηγούμενο πρόγραμμα, ώστε να εμφανίζει **μόνο** τους αριθμούς που έχουν παράγοντες, **ΟΧΙ** και τους πρώτους.

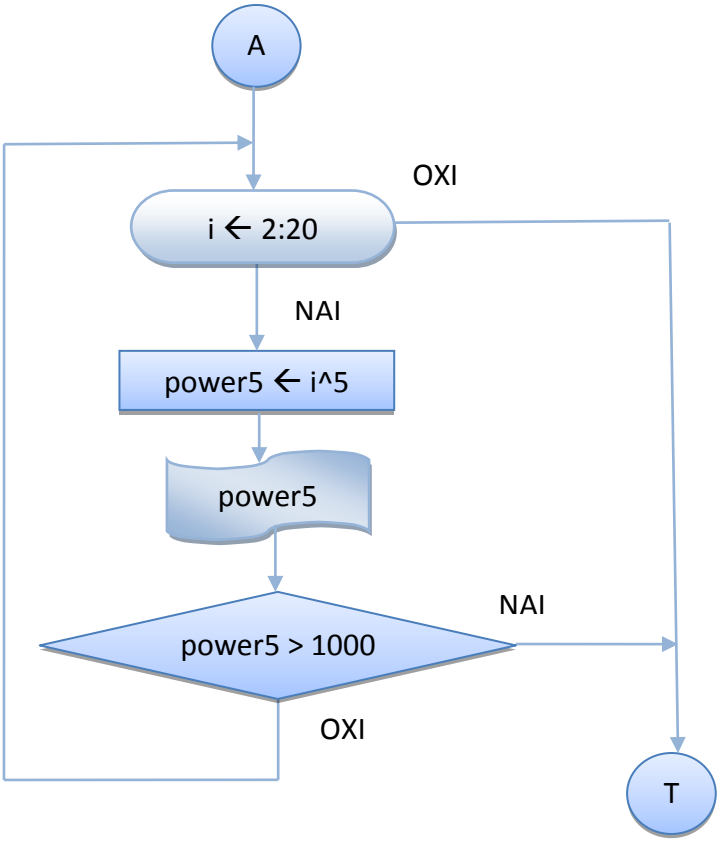
4.3.3 Η Εντολή break

Η εντολή `break` χρησιμοποιείται για τη διακοπή οποιουδήποτε βρόχου επανάληψης, ακόμη κι αν εξακολουθεί να ισχύει η συνθήκη. Εκτός από την έξοδο από κάθε περίπτωση (`case`) της εντολής επιλογής (`switch`) και την έξοδο από τον ατέρμονο βρόχο επανάληψης `for (;;)` μπορεί να χρησιμοποιηθεί για την διακοπή οποιουδήποτε βρόχου επανάληψης, όπως φαίνεται στο επόμενο παράδειγμα :

Παράδειγμα

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα υπολογίζει με τη χρήση της συνάρτησης `pow()` την **πέμπτη** δύναμη των ακέραιων αριθμών από το `2` μέχρι και το `20` και θα την εμφανίζει. Το πρόγραμμα θα τερματίζει με την εντολή `break`, αν η τιμή της πέμπτης δύναμης κάποιου από τους αριθμούς `2-20` είναι μεγαλύτερη του `1000`.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

Για τους αριθμούς `i = 2:20`
Υπολόγισε την Πέμπτη δύναμη `power5` του αριθμού `i`
Εμφάνισε την Πέμπτη δύναμη `power5` του αριθμού `i`
Αν η `power5` είναι μεγαλύτερη του `1000`
`Διακοπή Βρόχου Επανάληψης`

ΠΡΟΓΡΑΜΜΑ

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
    /*
    Πρόγραμμα το οποίο υπολογίζει με τη χρήση της συνάρτησης pow() την
    πέμπτη δύναμη των ακέραιων αριθμών από το 2 μέχρι και το 20 και  την
    εμφανίζει. Το πρόγραμμα  τερματίζει με την εντολή break, αν  η τιμή
    της πέμπτης δύναμης κάποιου από τους αριθμούς 2-20 είναι μεγαλύτερη
    του 1000
     */
 main()
{
        int i;
        int power5;
        // Για τους αριθμούς από το 2 μέχρι και το 20
        for ( i=2;i<=20;i++ ) {
          // Υπολογισμός i^5
          power5 = (int)pow(i,5);
          // Εμφάνιση i^5
          printf ("power of %d = %d\n", i, power5 );
          // Έλεγχος - Διακοπή, αν η δύναμη ξεπέρασε το 1000
          if (power5 > 1000 ) break;
        }
       system("Pause");
}
```

Έξοδος Προγράμματος

```
2^5 = 32
3^5 = 243
4^5 = 1024
Press any key to continue . . .
```

Άσκηση 4.6 : Να τροποποιηθεί ο προηγούμενος Αλγόριθμος/πρόγραμμα ώστε να ΜΗΝ εμφανίζει την τιμή του `power5`, αν είναι μεγαλύτερη του `1000`.

Άσκηση 4.7 : Να τροποποιηθεί ο προηγούμενος Αλγόριθμος/πρόγραμμα ώστε να κάνει το ίδιο **ΧΩΡΙΣ** τη χρήση των εντολών `for` και `break`.

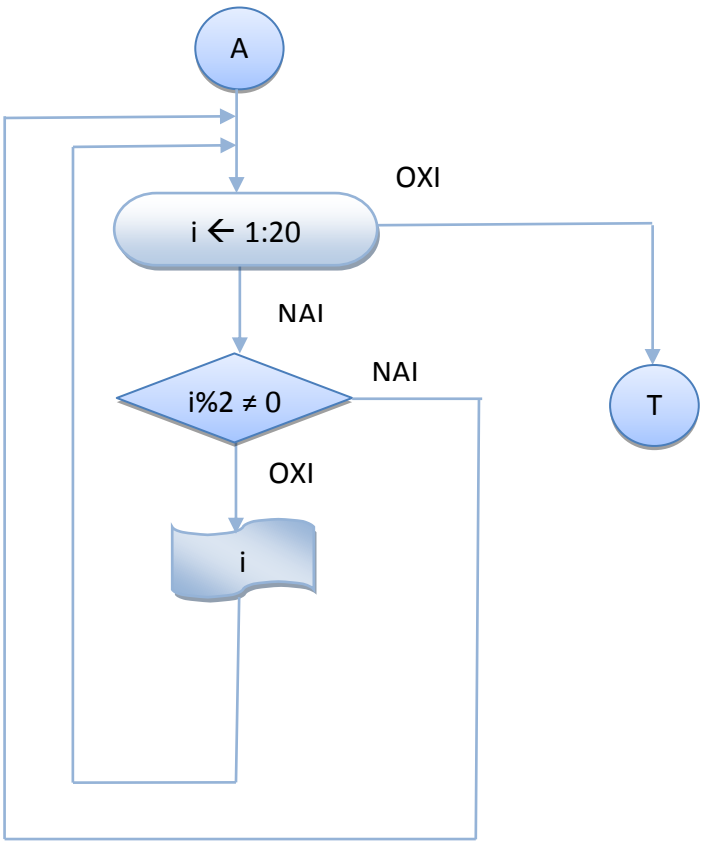
4.3.4 Η Εντολή continue

Η εντολή `continue` είναι το συμπλήρωμα της εντολής `break` και χρησιμοποιείται για να στείλει τον έλεγχο στη συνθήκη του οποιουδήποτε βρόχου επανάληψης, όπως φαίνεται στο επόμενο παράδειγμα :

Παράδειγμα

Να γραφεί Αλγόριθμος/πρόγραμμα, το οποίο θα βρίσκει και θα εμφανίζει όλους τους άρτιους ακέραιους αριθμούς από το 1 ως το 20 με τη χρήση της εντολής `continue`.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

Για τους αριθμούς `i = 1:20`
    Αν ο αριθμός `i` **δεν** διαιρείται ακριβώς με το `2`
        Συνέχισε με τον επόμενο αριθμό
    Εμφάνισε τον αριθμό `i`

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
    /*
     * Πρόγραμμα, το οποίο βρίσκει και  εμφανίζει όλους τους άρτιους
     * ακέραιους αριθμούς από το 1 ως το 20 με τη χρήση της εντολής
     * continue.
     */
main()
{
        int i;
        // Για τους αριθμούς από το 1 μέχρι και το 200
        for ( i=1;i<=20;i++ ) {
          // Έλεγχος αν ο αριθμός είναι άρτιος
          if (i%2 != 0) continue;
          // Εμφάνιση i
          printf ("%d ", i );
        }
       printf ("\n" );
       system("Pause");
}
```

Έξοδος Προγράμματος

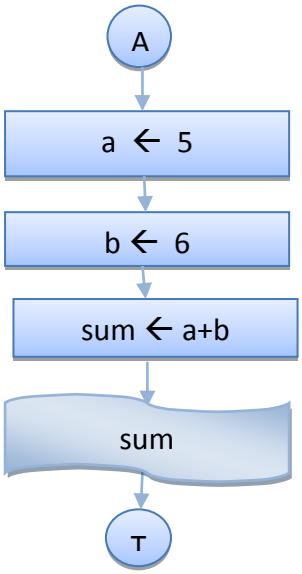
```
2 4 6 8 10 12 14 16 18 20
Press any key to continue . . .
```

Άσκηση 4.8 : Να τροποποιηθεί ο προηγούμενος Αλγόριθμος/πρόγραμμα ώστε να κάνει το ίδιο **ΧΩΡΙΣ** τη χρήση της εντολής `continue`.

5 ΣΥΝΑΡΤΗΣΕΙΣ - ΠΑΡΑΜΕΤΡΟΙ

Να γραφεί πρόγραμμα, το οποίο θα δίνει τις τιμές 5 και 6 σε δύο μεταβλητές **a** και **b** και θα υπολογίζει και θα εμφανίζει το άθροισμά τους **sum**.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



ΑΛΓΟΡΙΘΜΟΣ

1. Δίνω την τιμή 5 στο a (**a ← 5**)
2. Δίνω την τιμή 6 στο b (**b ← 6**)
3. Βρίσκω το άθροισμα (**sum ← a+b**)
4. Εμφανίζω την τιμή του **sum**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>
/* Πρόγραμμα που δίνει τις τιμές 5 και 6 σε 2 ακέραιες μεταβλητές και
βρίσκει και εμφανίζει το άθροισμά τους*/
main()
{
        // Δήλωση-Ανάθεση των τιμών 5, 6 στις ακέραιες μεταβλητές a, b
        int a = 5, b = 6;
        // Δήλωση - Υπολογισμός του αθροίσματος sum
        int sum = a + b;
        // Εμφάνιση  του αθροίσματος sum
        printf("sum = %d\n", sum);
        system("Pause");
}
```

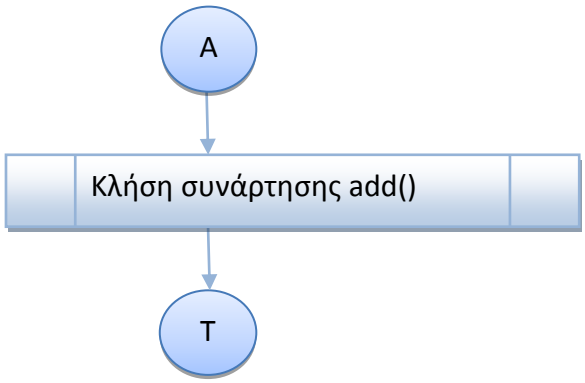
Έξοδος Προγράμματος

```
sum = 11
Press any key to continue . . .
```

5. 1 ΣΥΝΑΡΤΗΣΕΙΣ

Θα μπορούσαμε να καλέσουμε μια συνάρτηση `add()`, η οποία να δίνει τις τιμές στις μεταβλητές **a , b** και να υπολογίζει και να εμφανίζει το άθροισμα **sum** και η συνάρτηση `main()` απλώς να την καλεί. Σ' αυτή την περίπτωση το λογικό διάγραμμα και ο αλγόριθμος της `main()` και της συνάρτησης `add()` θα γινόταν :

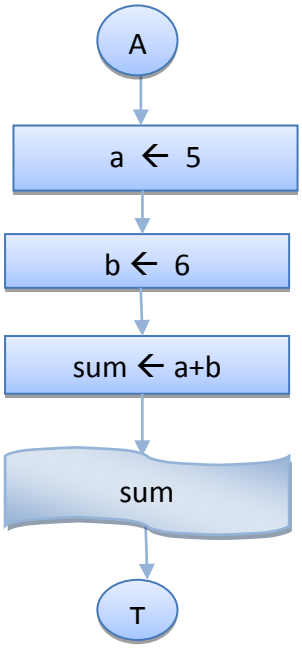
ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ main()



ΑΛΓΟΡΙΘΜΟΣ main()

1. Κλήση συνάρτησης `add()`

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ ΣΥΝΑΡΤΗΣΗΣ `add()`



ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΑΡΤΗΣΗΣ `add()`

1. Δίνω την τιμή `5` στο `a` (**a ← 5**)
2. Δίνω την τιμή `6` στο `b` (**b ← 6**)
3. Βρίσκω το άθροισμα **sum** (**sum ← a+b**)
4. Εμφανίζω την τιμή του **sum**

Οι **Συναρτήσεις** είναι αυτοτελή τμήματα κώδικα που εκτελούν κάποιες εργασίες και περιέχουν ότι και η συνάρτηση `main()`, **δηλώσεις** μεταβλητών και **εντολές**. Μια μορφή συνάρτησης έχει την παρακάτω σύνταξη :

```
void <όνομα_συνάρτησης>()
{
      εντολές;
}
```

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

// Κώδικας Συνάρτησης add()
void add()
{
        // Δήλωση-Ανάθεση των τιμών 5, 6 στις ακέραιες μεταβλητές a, b
        int a = 5, b = 6;
        // Δήλωση - Υπολογισμός του αθροίσματος sum
        int sum = a + b;
        // Εμφάνιση  του αθροίσματος sum
        printf("sum = %d\n", sum);
}


/* Πρόγραμμα που καλεί τη συνάρτηση add(), η οποία δίνει τις τιμές 5 και
6 σε 2 ακέραιες μεταβλητές και βρίσκει και εμφανίζει το άθροισμά τους*/

main()
{
    // Κλήση Συνάρτησης add()
    add();
    system("Pause");
}
```

Έξοδος Προγράμματος

```
sum = 11
Press any key to continue . . .
```

Παρατηρήσεις

- Η συνάρτηση `add()` δεν παίρνει **καμία** πληροφορία από την `main()`.
- Δεν στέλνει **καμιά** πληροφορία στην `main()`, γι' αυτό και δηλώνεται τύπου `void`.
- Οι μεταβλητές της συνάρτησης λέγονται **τοπικές μεταβλητές**, δημιουργούνται όταν καλείται η συνάρτηση, **δεν είναι ορατές στην καλούσα συνάρτηση** και παύουν να υπάρχουν, όταν ολοκληρωθεί η εκτέλεση των εντολών της συνάρτησης.

- Ο κώδικας της συνάρτησης `add()` βρίσκεται στο ίδιο αρχείο με την συνάρτηση `main()` και η κλήση της γίνεται χρησιμοποιώντας απλώς το όνομά της. Μπορεί να γραφεί πριν τον κώδικα της `main()`, ή και μετά τον κώδικα της `main()`, αρκεί η υπογραφή της (η δήλωση της συνάρτησης) να γραφεί πριν τη `main()` :

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

// Δήλωση της συνάρτησης add()
void add();

main()
/* Πρόγραμμα που καλεί τη συνάρτηση add(), η οποία δίνει τις τιμές 5 και
6 σε 2 ακέραιες μεταβλητές και βρίσκει και εμφανίζει το άθροισμά τους*/
{
    // Κλήση Συνάρτησης add()
    add();
    system("Pause");
}

// Κώδικας Συνάρτησης add()
void add()
{
        // Δήλωση-Ανάθεση των τιμών 5, 6 στις ακέραιες μεταβλητές a, b
        int a = 5, b = 6;
        // Δήλωση - Υπολογισμός του αθροίσματος sum
        int sum = a + b;
        // Εμφάνιση  του αθροίσματος sum
        printf("sum = %d\n", sum);        }
}
```

Έξοδος Προγράμματος

```
sum = 11
Press any key to continue . . .
```

5.1.1 Συνάρτηση που Δέχεται Παραμέτρους από τη main()

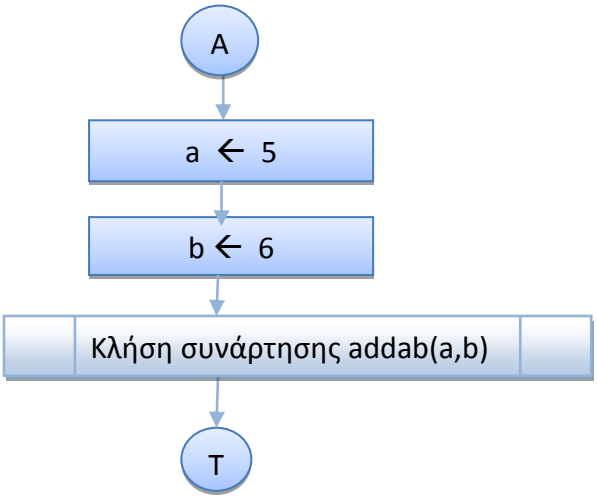
Η προηγούμενη συνάρτηση κάνει μια πολύ συγκεκριμμένη δουλειά. Υπολογίζει το άθροισμα των μεταβλητών **a** και **b** για τις συγκεκριμμένες τιμές 5 και 6. Αν θέλουμε να υπολογίζει και να εμφανίζει το άθροισμα οποιονδήποτε 2 αριθμών **a** και **b** που θα παίρνουν τιμές στην `main()` θα πρέπει να περάσουμε τις τιμές των μεταβλητών **a** και

b σαν **παραμέτρους** στην συνάρτηση. Αυτό γίνεται στην δήλωση της συνάρτησης, οπότε η γενική της σύνταξη θα είναι :

```
void <όνομα_συνάρτησης>(<λίστα_παραμέτρων>)
{
      εντολές;
}
```

όπου η <λίστα_παραμέτρων> περιλαμβάνει δηλώσεις μεταβλητών χωρισμένες με κόμμα, μέσω των οποίων **περνάμε** στην συνάρτηση τις **τιμές** της συνάρτησης που την καλεί.

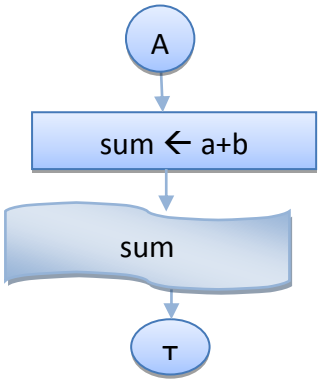
ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ main()



ΑΛΓΟΡΙΘΜΟΣ main()

1. Δίνω την τιμή 5 στο a (**a ← 5**)
2. Δίνω την τιμή 6 στο b (**b ← 6**)
3. Κλήση συνάρτησης `addab(a, b)`

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ ΣΥΝΑΡΤΗΣΗΣ `addab(a, b)`



ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΑΡΤΗΣΗΣ
`addab(a,b)`

1. Βρίσκω το άθροισμα (**sum ← a+b**)
2. Εμφανίζω την τιμή του **sum**

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

// Δήλωση της συνάρτησης addab()
void addab(int a, int b)
{
        // Δήλωση - Υπολογισμός του αθροίσματος sum
        int sum = a + b;
        // Εμφάνιση  του αθροίσματος sum
        printf("sum = %d\n", sum);        }
}

main()
{
/* Πρόγραμμα που καλεί την συνάρτηση addab(), η οποία δέχεται τις τιμές
5 και 6 σε 2 ακέραιες μεταβλητές και βρίσκει και εμφανίζει το άθροισμά
τους */
    // Δήλωση-Ανάθεση των τιμών 5, 6 στις ακέραιες μεταβλητές a, b
    int a = 5, b = 6;

    // Κλήση συνάρτησης addab()
    addab(a, b);
    system("Pause");
}
```

Έξοδος Προγράμματος

```
sum = 11
Press any key to continue . . .
```

Παρατηρήσεις

- Οι πληροφορίες περνάνε στην συνάρτηση μέσω των μεταβλητών `a` και `b` στη δήλωση.
- Οι μεταβλητές `a` και `b` στη δήλωση της συνάρτησης λέγονται **παράμετροι** και μπορούν να έχουν **διαφορετικά** ονόματα από τις μεταβλητές της `main()`).
- Μπορεί να χρησιμοποιηθεί **οποιοδήποτε όνομα** για τις παραμέτρους, αρκεί να χρησιμοποιούνται τα ίδια ονόματα και μέσα στην συνάρτηση.
- Πρέπει να συμφωνεί ο **αριθμός**, η **σειρά** και ο **τύπος** των **ορισμάτων** (μεταβλητών της καλούσας συνάρτησης) και των **παραμέτρων** της συνάρτησης.
- Στην κλήση της συνάρτησης **δεν χρειάζονται οι τύποι** των μεταβλητών- ορισμάτων.
- Στην κλήση της συνάρτησης τα ονόματα των παραμέτρων αντικαθίστανται από τα ονόματα των μεταβλητών της καλούσας συνάρτησης. Οι τιμές των μεταβλητών (ορίσματα) περνάνε στην συνάρτηση.
- Η συνάρτηση **τελειώνει** όταν εκτελεστούν όλες οι εντολές ή αν υπάρχει η εντολή `return`.

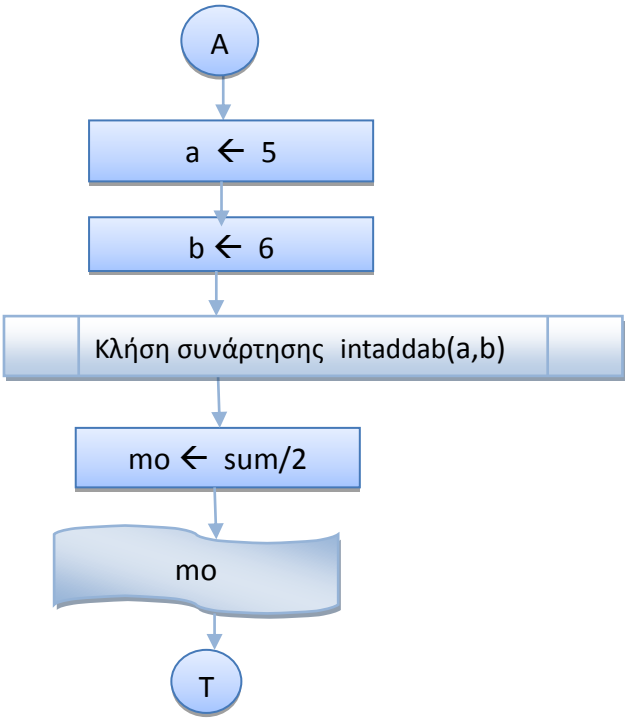
- Μπορεί να μην χρησιμοποιηθεί η τοπική μεταβλητή `sum`. Σ' αυτή την περίπτωση, ο κώδικας της συνάρτησης θα γίνει :

```
void addab(int a, int b) {
    printf("sum = %d\n", a + b);}
```

5.1.2 Συνάρτηση που Επιστρέφει και το Άθροισμα στη `main()`

Να γραφεί πρόγραμμα, το οποίο θα δίνει τις τιμές `5` και `6` σε δύο μεταβλητές `a` και `b` και θα καλεί τη συνάρτηση `intaddab()`, η οποία υπολογίζει και **επιστρέφει** στη `main()` την τιμή του αθροίσματος **`sum`**, την οποία και θα εμφανίζει. Μετά το πρόγραμμα θα υπολογίζει και θα εμφανίζει και το μέσο όρο **`mo`**.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



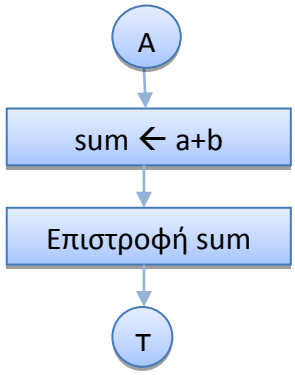
ΑΛΓΟΡΙΘΜΟΣ `main()`

1. Δίνω την τιμή `5` στο `a` (**`a ← 5`**)
2. Δίνω την τιμή `6` στο `b` (**`b ← 6`**)
3. Κλήση συνάρτησης `intaddab(a,b)`
4. Υπολογισμός Μέσου Όρου (**`mo ← sum/2`**)
5. Εμφάνιση Μέσου Όρου **`mo`**

ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΑΡΤΗΣΗΣ
`intaddab(a,b)`

1. Βρίσκω το άθροισμα (**sum ← a+b**)
2. **Επιστρέφω** το άθροισμα **sum**

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ ΣΥΝΑΡΤΗΣΗΣ `intaddab(a, b)`



Για να μπορέσει μια συνάρτηση να **επιστρέψει** μια τιμή στην καλούσα συνάρτηση, θα πρέπει :

- Να δηλωθεί στην **επικεφαλίδα** της συνάρτησης ο **τύπος** της τιμής που θα επιστραφεί.
- Να υπάρχει **μέσα** στον κώδικα της συνάρτησης η εντολή **`return`** με την οποία θα επιστραφεί αυτή η τιμή.

Η σύνταξη της δήλωσης της συνάρτησης θα έχει τη μορφή :

```
<τύπος_επιστροφής> <όνομα_συνάρτησης> (<λίστα_παραμέτρων>)
    {
    εντολές;
    return <έκφραση_τύπου_επιστροφής>;
    }
```

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

// Δήλωση της συνάρτησης intaddab()
int intaddab(int a, int b)
{
        // Δήλωση - Υπολογισμός του αθροίσματος sum
        int sum = a + b;
        // Επιστροφή Τιμής Αθροίσματος sum
        return sum;
}

main()
{
/* Πρόγραμμα το οποίο δίνει τις τιμές 5 και 6 σε δύο μεταβλητές a και b και
καλεί τη συνάρτηση add(), η οποία υπολογίζει και επιστρέφει στη main()την
τιμή του αθροίσματος sum, την οποία και εμφανίζει. Μετά το πρόγραμμα
υπολογίζει και εμφανίζει και το μέσο όρο mo. */
    // Δήλωση-Ανάθεση των τιμών 5, 6 στις ακέραιες μεταβλητές a, b
    int a = 5, b = 6;

    // Κλήση συνάρτησης intaddab()
    int sum = intaddab(a, b);

    // Δήλωση - Υπολογισμός του Μέσου Όρου mo
    double mo = (double) sum/2;

    // Εμφάνιση του αθροίσματος sum και του Μέσου Όρου mo
    printf("sum = %d mo = %lf\n", sum, mo);
    system("Pause"); }
```

Έξοδος Προγράμματος

```
sum = 11 mo = 5.500000
Press any key to continue . . .
```

Παρατηρήσεις

- Όταν επιστρέφει η τιμή μιας μεταβλητής από κάποια συνάρτηση πρέπει να δηλώνεται ο **τύπος** της.
- Ο **τύπος της** συνάρτησης πρέπει να είναι ίδιος με τον **τύπο** της **μεταβλητής** που επιστρέφει.
- Η **κλήση** της συνάρτησης χρησιμοποιείται στην καλούσα συνάρτηση σαν μια οποιαδήποτε μεταβλητή σε εντολές εκχώρησης, ελέγχου, εμφάνισης κ.λ.π..
- Η συνάρτηση επιστρέφει **μόνο μια τιμή** για **μεταβλητές**, εκφράσεις **απλού τύπου**.
- Μπορεί να υπάρχουν στη συνάρτηση **περισσότερες** από μία εντολές **`return`**.
- Μπορεί να μη χρησιμοποιηθεί η τοπική μεταβλητή `sum` και να επιστρέψει η έκφραση υπολογισμού του. Σ' αυτή την περίπτωση ο κώδικας της συνάρτησης θα είναι :

```
int intaddab(int a, int b)
{
    // Υπολογισμός - Επιστροφή Τιμής Αθροίσματος (a + b)
    return (a + b);
}
```

- Μπορεί να μην χρησιμοποιηθεί η μεταβλητή `sum` στη `main()` και να χρησιμοποιηθεί το όνομα της συνάρτησης με τις παραμέτρους. Αντί των εντολών :

```
// Κλήση Συνάρτησης  intaddab()
int sum = intaddab(a, b);
// Δήλωση - Υπολογισμός του Μέσου Όρου mo
double mo = (double) sum/2;
```

θα μπορούσαμε να γράψουμε :

```
// Δήλωση - Υπολογισμός του Μέσου Όρου mo - Κλήση συνάρτησης intaddab()
double mo = (double) (intaddab(a, b))/2;
printf("sum = %d mo = %lf\n", intaddab(a, b), mo);
```

- Ο τύπος επιστροφής μιας συνάρτησης μπορεί να είναι οποιοσδήποτε από τους απλούς τύπους, ακέραιος (`short, int, long`), κινητής υποδιαστολής (`float, double`), χαρακτήρας (`char`), ή λογικός (`boolean`) που επιστρέφει τις τιμές `true` ή `false`, όπως στο επόμενο παράδειγμα :

5.1.2 Συνάρτηση που Επιστρέφει Αποτέλεσμα Τύπου boolean

Να γραφεί πρόγραμμα, το οποίο θα διαβάζει έναν ακέραιο **a** στο [1, 10], θα καλεί μια συνάρτηση, η οποία θα βρίσκει αν ο αριθμός είναι ΑΡΤΙΟΣ ή ΠΕΡΙΤΤΟΣ και θα εμφανίζει το κατάλληλο μήνυμα.

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

bool isArtios(int a)
{
        bool artios = false;  // Αρχική τιμή false
        if ( a%2 == 0) // Ο a διαιρείται ακριβώς με το 2 - άρτιος
          artios = true; // είναι ΑΡΤΙΟΣ
        return artios; // επιστροφή Boolean τιμής

}

main()
{
/* Πρόγραμμα, το οποίο θα διαβάζει έναν ακέραιο a στο [1, 10], καλεί μια
συνάρτηση, η οποία  βρίσκει αν ο αριθμός είναι ΑΡΤΙΟΣ ή ΠΕΡΙΤΤΟΣ  και
εμφανίζει το κατάλληλο μήνυμα */
    int a;
    do
    {
      printf("Give an integer in [1, 10] : ");
      scanf("%d", &a);
    }
    while (a<1 || a>10);
    if (isArtios(a)) // Είναι άρτιος ???
      printf("a = %d artios\n", a);
    else
      printf("a = %d perittos\n", a);
    system("Pause");
}
```

Έξοδος Προγράμματος

```
Give an integer in [1, 10] : 4
a = 4 artios
Press any key to continue . . .
```

Παρατήρηση

Δεν χρειάζεται να ελέγξουμε αν η τιμή που επιστρέφει η συνάρτηση **`isArtios(a)`** είναι `true` ή `false`, δηλαδή η εντολή **`if (isArtios(a))`** να γίνει :

`if (isArtios(a)) == true)`

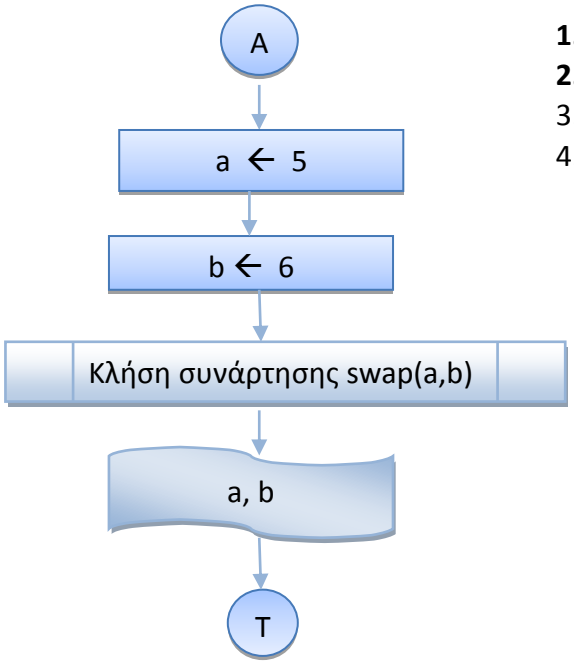
5.3 Πέρασμα Παραμέτρων σε Συναρτήσεις

Οι παράμετροι των βασικών τύπων περνάνε στην συνάρτηση με **τιμή**, δηλαδή στην συνάρτηση είναι διαθέσιμο ένα **αντίγραφο** της τιμής της μεταβλητής και **ΟΧΙ** η **διεύθυνσή** της, οπότε **δεν αλλάζει το περιεχόμενο** που είχαν πριν, ακόμα και αν αλλάζει η τιμή τους μέσα στην συνάρτηση. Αυτό μπορεί να γίνει κατανοητό με το επόμενο παράδειγμα :

5.3.1 Παράμετροι με Τιμή – Ανταλλαγή των Τιμών Δυο Μεταβλητών

Να γραφεί πρόγραμμα που να καλεί μια συνάρτηση η οποία **ανταλλάσει** τις τιμές 2 ακέραιων αριθμών.

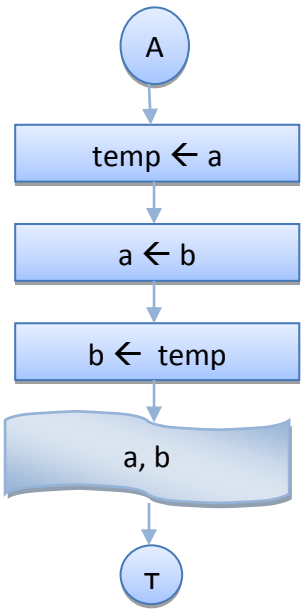
ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ main()



ΑΛΓΟΡΙΘΜΟΣ main()

1. Δίνω την τιμή `5` στο `a` (**`a ← 5`**)
2. Δίνω την τιμή `6` στο `b` (**`b ← 6`**)
3. Κλήση συνάρτησης `swap(a, b)`
4. Εμφάνιση `a, b`

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ ΣΥΝΑΡΤΗΣΗΣ `swap(a, b)`



ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΑΡΤΗΣΗΣ `swap(a,b)`

1. Αποθηκεύω το **`a`** στο **`temp`** (**`temp←a`**)
2. Αποθηκεύω το **`b`** στο **`a`** (**`a ← b`**)
3. Αποθηκεύω το **`temp`** στο **`b`** (**`temp ← b`**)
4. Εμφάνιση `a, b`

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

/* Πρόγραμμα το οποίο δίνει τις τιμές 5 και 6 σε δύο μεταβλητές a και b και
καλεί την συνάρτηση swap(), η οποία ανταλλάσσει τις τιμές των a και b, τις
οποίες και εμφανίζει. */

void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("In swap : a = %d b = %d\n", a, b);
}

main()
{
    int a = 5, b = 6;
    printf("In main before swap : a = %d b = %d\n", a, b);
    swap(a, b);
    printf("In main after swap : a = %d b = %d\n", a, b);
    system("Pause");
}
```

Έξοδος Προγράμματος

```
In main before swap : a = 5 b = 6
In swap : a = 6 b = 5
In main after swap : a = 5 b = 6
Press any key to continue . . .
```

Παρατηρήσεις

- Τα ορίσματα (οι μεταβλητές `a, b` της `main()`) περνάνε στην συνάρτηση **με τιμή**, δηλαδή ένα αντίγραφο του περιεχομένου των μεταβλητών `a, b` **και <u>όχι</u> οι διευθύνσεις** των μεταβλητών.
- Με τις αλλαγές στην συνάρτηση το περιεχόμενο των μεταβλητών `a, b` της `main()` **δεν επηρεάζεται.**
- Μετά την κλήση της συνάρτησης οι μεταβλητές `a, b` περιέχουν **τις ίδιες τιμές** που είχαν και **πριν** την κλήση της συνάρτησης.
- Για να αλλάξει το περιεχόμενο των μεταβλητών `a, b` θα πρέπει να περάσουμε σαν παραμέτρους στην κλήση τις **διευθύνσεις** τους και να χρησιμοποιήσουμε στη δήλωση της συνάρτησης **δείκτες** (**pointers**).

6 ΠΙΝΑΚΕΣ

Πίνακας είναι μια διάταξη στοιχείων που το καθένα τους έχει κάποια συγκεκριμένη **θέση**. Καταλαμβάνει μια περιοχή μνήμης με το λογικό όνομα του πίνακα, ενώ κάθε στοιχείο του ξεχωρίζει με κάποιο **δείκτη**. Στη C η αρίθμηση των δεικτών ξεκινάει από το 0 μέχρι το μέγεθος του πίνακα−1.

Παράδειγμα ενός πίνακα 5 θέσεων με περιεχόμενα τους αριθμούς 1−5.

1	2	3	4	5
Θέση 0	Θέση 1	Θέση 2	Θέση 3	Θέση 4

Δήλωση Πίνακα

Η Δήλωση ενός Πίνακα στη C γίνεται με τη δήλωση του **τύπου** των στοιχείων που θα αποθηκεύσει, το **όνομά** του και το **μέγεθός** του.

Παράδειγμα

```
int pin1[5];
```

όπου δηλώνουμε τον πίνακα p1, ο οποίος θα αποθηκεύσει 5 ακέραιους αριθμούς.

Το μέγεθος του πίνακα μπορεί να είναι η τιμή μιας μεταβλητής. Οι επόμενες εντολές έχουν το ίδιο αποτέλεσμα :

```
int n = 5;
int pin2[n];
```

Ένας πίνακας μπορεί να δημιουργηθεί με **δυναμική αρχικοποίηση**, όπου οι τιμές του περικλείονται σε άγκιστρα και χωρίζονται με κόμμα.

Παράδειγμα

```
int pin3[] = {1, 2, 3, 4, 5};
```

όπου δηλώνουμε τον πίνακα pin2, ο οποίος θα αποθηκεύσει τους ακέραιους αριθμούς από το 1 μέχρι το 5. **Σαν στοιχεία μέσα στα άγγιστρα, εκτός από σταθερούς αριθμούς, μπορεί να είναι ονόματα μεταβλητών, εκφράσεις ή κλήσεις συναρτήσεων.**

Η πρόσβαση σε κάποιο στοιχείο του πίνακα γίνεται με το όνομά του και τη θέση του (δείκτης) μέσα σε αγκύλες. Π.χ. το pin1[3] αναφέρεται στο 4^{ο} στοιχείο του pin1.

6.1 Παράδειγμα Δημιουργίας – Γεμίσματος 2 Πινάκων με Τιμές απ' το Πληκτρολόγιο και Δυναμική Αρχικοποίηση

- Να γραφεί Αλγόριθμος/Πρόγραμμα το οποίο γεμίζει έναν πίνακα με **δυναμική αρχικοποίηση** και δίνει τις τιμές από το 1 μέχρι το 5, δηλώνει έναν άλλον πίνακα n=5 θέσεων, τον οποίο γεμίζει με τιμές απ' το πληκτρολόγιο και εμφανίζει τα περιεχόμενα των 2 πινάκων.

Αλγόριθμος

1. Δήλωση - Αρχικοποίηση πίνακα pin1
2. **Για** κάθε θέση i = 0 μέχρι (n – 1)
   Εκχωρούμε μια τιμή απ' το πληκτρολόγιο στο στοιχείο pin2[i]
3. **Για** κάθε θέση i = 0 μέχρι 4
   Εμφανίζουμε το στοιχείο pin1[i]
4. **Για** κάθε θέση i = 0 μέχρι (n – 1)
   Εμφανίζουμε το στοιχείο pin2[i]

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

main()

/* Το πρόγραμμα δημιουργεί 2 πίνακες, τον πρώτο με δυναμική αρχικοποίηση και
τον γεμίζει με τις ακέραιες τιμές 1-5, τον δεύτερο τον  γεμίζει με ακέραιες
τιμές που δίνει ο χρήστης και εμφανίζει τα στοιχεία των 2 πινάκων */
{
    int i;

    // Εκχώρηση ακέραιας τιμής στο μέγεθος του πίνακα 2
    int n = 4;

   // Δήλωση πίνακα 2
    int pin2[n];

    // Δήλωση - Αρχικοποίηση πίνακα 1
    int pin1[] = {1, 2, 3, 4, 5};

    // Γέμισμα πίνακα 2 με ακέραιες τιμές
    for (i=0;i<=n-1;i++){
        printf("Give pin2[%d] : ", i);
        scanf("%d", &pin2[i]); }

    // Εμφάνιση στοιχείων πίνακα 1
    printf("pin1 = ");
    for (i=0;i<=4;i++)
       printf("%d ", pin1[i]);
    printf("\n");

   // Εμφάνιση στοιχείων πίνακα 2
    for (i=0;i<=n-1;i++)
     printf("pin2[%d] = %d\n", i, pin2[i]);

    system("Pause");
}
```

Έξοδος Προγράμματος :

```
pin1 = 1 2 3 4 5
pin2[0] = 6
pin2[1] = 7
pin2[2] = 8
pin2[3] = 9
Press any key to continue . . .
```

6.2 Οι Πίνακες σαν Παράμετροι σε Συναρτήσεις

Σε αντίθεση με της μεταβλητές απλού τύπου, οι πίνακες περνάνε **με αναφορά** σαν παράμετροι στις συναρτήσεις, οπότε μπορεί να αλλάξει το περιεχόμενό τους. Θα μπορούσαμε π.χ. να αλλάξουμε τα περιεχόμενα των θέσεων ενός πίνακα `p[i], p[j]` περνώντας σαν παραμέτρους σε μια συνάρτηση `swapPiPj()` το όνομα του πίνακα και τους δείκτες των στοιχείων που θα αλλάξουν περιεχόμενο, όπως φαίνεται στο επόμενο πρόγραμμα.

ΠΡΟΓΡΑΜΜΑ

```
#include <stdio.h>
#include <stdlib.h>

/* Πρόγραμμα το οποίο δίνει τις τιμές 1, 2, 3, 4, 5 σε έναν πίνακα με
δυναμική αρχικοποίηση, εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης
showP(), διαβάζει τις τιμές 2 δεικτών i και j στο 0-4 και καλεί την συνάρτηση
swapPiPj(), με την οποία γίνεται ανταλλαγή των περιεχομένων του πίνακα p[i],
p[j] και εμφανίζει τα στοιχεία του πίνακα μετά την ανταλλαγή με την κλήση της
συνάρτησης showP(). */

void swapPiPj(int i, int j, int p[])
{
    int temp;
    temp = p[i];
    p[i] = p[j];
    p[j] = temp;
}


void showP(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}
```

```
main()
{
    int i, j;
    int p[] = {1, 2, 3, 4, 5};
    printf("p = ");
    showpin( 5,  p);
    printf("Give i, j in 0-4 : ");
    scanf("%d %d",&i, &j);
    swapPiPj(i, j, p);
    printf("after swap p[%d], p[%d]  p = ", i, j);
    showP( 5,  p);
    system("Pause");
}
```

Έξοδος Προγράμματος

```
p = 1 2 3 4 5
after swap p[2], p[3]  p = 1 2 4 3 5
Press any key to continue . . .
```

6.2.1 Παράδειγμα Δημιουργίας – Γεμίσματος 2 Πινάκων με Τιμές απ' το Πληκτρολόγιο και Δυναμική Αρχικοποίηση – Χρήση Συναρτήσεων για Γέμισμα - Εμφάνιση

- Να γραφεί Αλγόριθμος/Πρόγραμμμα το οποίο γεμίζει έναν πίνακα με **δυναμική αρχικοποίηση** και δίνει τις τιμές από το 1 μέχρι το 5, δηλώνει έναν άλλον πίνακα n=4 θέσεων, τον οποίο γεμίζει με **τιμές** απ' το πληκτρολόγιο καλώντας τη συνάρτηση **`fillPin()`** και εμφανίζει τα περιεχόμενα των 2 πινάκων καλώντας τη συνάρτηση **`showPin()`**.

Αλγόριθμος

1. Δήλωση - Αρχικοποίηση πίνακα 1
2. Δήλωση – Κλήση συνάρτησης **`fillPin()`** για το γέμισμα του πίνακα 2 με τιμές απ' το πληκτρολόγιο
3. Εμφάνιση στοιχείων πίνακα 1 - Κλήση συνάρτησης **`showPin()`**
4. Εμφάνιση στοιχείων πίνακα 2 - Κλήση συνάρτησης **`showPin()`**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

void fillPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
    {
     printf("Give p[%d] : ", i);
     scanf("%d", &p[i]);
    }
}

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}


main()

/* Το πρόγραμμα δημιουργεί 2 πίνακες, τον πρώτο με δυναμική αρχικοποίηση και
τον γεμίζει με τις ακέραιες τιμές 1-5, τον δεύτερο τον  γεμίζει με ακέραιες
τιμές που δίνει ο χρήστης με την κλήση της συνάρτησης fillPin() και εμφανίζει
τα στοιχεία των 2 πινάκων με την κλήση της συνάρτησης showPin()  */
{
    int i;

    // Εκχώρηση ακέραιας τιμής στο μέγεθος του πίνακα 2
    int n = 4;

   // Δήλωση πίνακα 2
    int pin2[n];

    // Δήλωση - Αρχικοποίηση πίνακα 1
    int pin1[] = {1, 2, 3, 4, 5};

    // Γέμισμα πίνακα 2 με την κλήση της συνάρτησης fillpin()
    fillPin(n, pin2);

    // Εμφάνιση στοιχείων πίνακα 1
    printf("pin1 = ");
    showpin( 5,  pin1);

   // Εμφάνιση στοιχείων πίνακα 2
    printf("pin2 = ");
    showPin( n,  pin2);

    system("Pause");
}
```

Έξοδος Προγράμματος :

```
pin1 = 1 2 3 4 5
pin2 = 6 7 8 9
Press any key to continue . . .
```

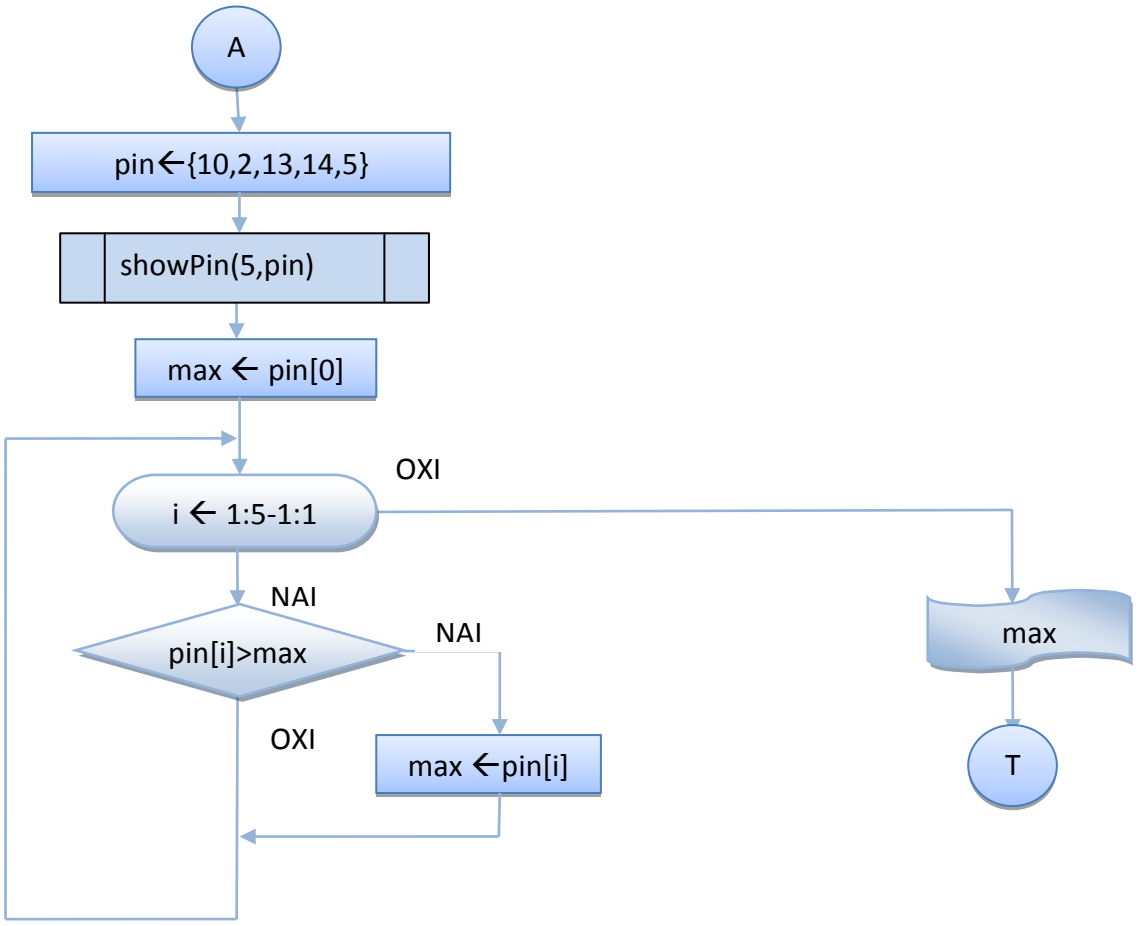
6.3 Εύρεση Στοιχείου με τη Μέγιστη ή Ελάχιστη Τιμή σε έναν Πίνακα

Σε πολλά προβλήματα, χρειάζεται να βρούμε το στοιχείο με τη μεγαλύτερη ή μικρότερη τιμή σε έναν πίνακα. Ο πιο συνηθισμένος τρόπος είναι να ελέγξουμε **όλα** τα στοιχεία του πίνακα και να κρατάμε κάθε φορά το μεγαλύτερο, όπως φαίνεται στο επόμενο παράδειγμα :

6.3.1 Εύρεση στοιχείου με τη Μέγιστη Τιμή σε έναν Πίνακα

- Να γραφεί Πρόγραμμα που να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων με δυναμική αρχικοποίηση, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης showPin(), να βρίσκει το στοιχείο με τη μεγαλύτερη τιμή και να το εμφανίζει.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



Αλγόριθμος

1. Δήλωση - Αρχικοποίηση πίνακα `pin[]` με `n=5` θέσεις
2. Εμφάνιση στοιχείων πίνακα `pin` – κλήση `showPin()`
3. Θέτουμε σαν μέγιστο στοιχείο `max` το πρώτο στοιχείο `pin[0]` (αφού δεν υπάρχει άλλο μέχρι στιγμής να συγκριθεί)
4. **Για** τα υπόλοιπα στοιχεία `pin[i]`, για `i = 1` μέχρι `4` :
   Αν το στοιχείο `pin[i]` είναι μεγαλύτερο από το `max`
   Θέτουμε σαν μέγιστο στοιχείο `max` το `pin[i]`
5. Εμφάνιση της τιμής του μεγίστου στοιχείου `max`

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}

main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με δυναμική
αρχικοποίηση, εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης showPin(),
βρίσκει το στοιχείο με τη μεγαλύτερη τιμή και το Εμφανίζει */
{
        int i, max;

        // Δήλωση - Αρχικοποίηση πίνακα
        int pin[] = {10,2,13,14,5};

        // Εμφάνιση στοιχείων πίνακα
        printf("pin = ");
        showPin( 5,  pin);

        // Εύρεση μεγίστου στοιχείου του πίνακα
        max = pin[0];
        for (i = 1;i <= 5-1;i++)
           if (pin[i] > max ) {
             max = pin[i];

        // Εμφάνιση μεγίστου στοιχείου του πίνακα
        printf("max pin[i] = %d\n", max);

        system("Pause"); }

}
```

Έξοδος Προγράμματος :

```
pin = 10 2 13 14 5
max pin[i] = 14
Press any key to continue . . .
```

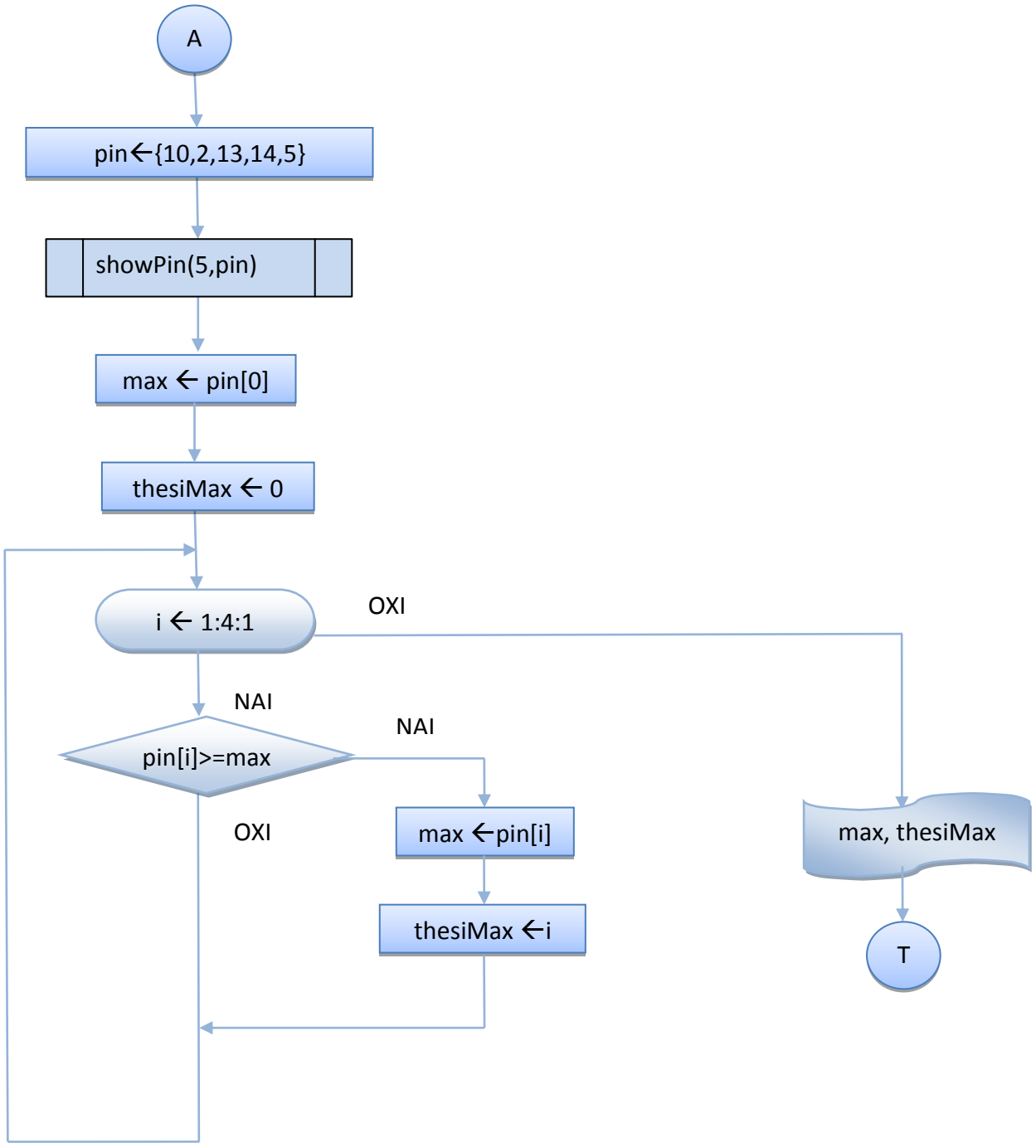
Παρατήρηση :

Πολλές φορές μας ενδιαφέρει να βρούμε όχι μόνο το στοιχείο με τη μεγαλύτερη τιμή, αλλά και τη **θέση** αυτού του στοιχείου στον πίνακα. Έχοντας τη θέση, μπορούμε να βρούμε και το αντίστοιχο στοιχείο. Αυτό φαίνεται στο επόμενο παράδειγμα :

6.3.2 Εύρεση Στοιχείου με τη Μέγιστη Τιμή σε έναν Πίνακα και της Θέσης του στον Πίνακα

- Να γραφεί Πρόγραμμα που να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων με δυναμική αρχικοποίηση, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης `showPin()`, να βρίσκει το στοιχείο με τη μεγαλύτερη τιμή και τη θέση του στον πίνακα και να εμφανίζει το στοιχείο με τη μεγαλύτερη τιμή και **τη θέση του** στον πίνακα

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ



Αλγόριθμος

1. Δήλωση - Αρχικοποίηση πίνακα `pin[]` με `n=5` θέσεις
2. Εμφάνιση στοιχείων πίνακα `pin` – κλήση `showPin()`
3. Θέτουμε σαν μέγιστο στοιχείο `max` το πρώτο στοιχείο `pin[0]` (αφού δεν υπάρχει άλλο μέχρι στιγμής να συγκριθεί)
4. **Θέτουμε σαν θέση μεγίστου στοιχείου `thesiMax` το 0**
5. **Για** τα υπόλοιπα στοιχεία `pin[i]`, για `i = 1` μέχρι `4` :
   Αν το στοιχείο `pin[i]` είναι μεγαλύτερο από το `max`
   Θέτουμε σαν μέγιστο στοιχείο `max` την τιμή του `pin[i]`
   Θέτουμε την τιμή του `i` στο `thesiMax`
6. Εμφάνιση της τιμής του μεγίστου στοιχείου `max` και **της θέσης του `thesiMax`**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}

main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με δυναμική
αρχικοποίηση, εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης ShowPin(),
Βρίσκει το στοιχείο με τη μεγαλύτερη τιμή και τη θέση του στον πίνακα και
Εμφανίζει το στοιχείο με τη μεγαλύτερη τιμή και τη θέση του στον πίνακα */
{
        int i, max;

        // Δήλωση - Αρχικοποίηση πίνακα
        int pin[] = {10,2,13,14,5};

        // Εμφάνιση στοιχείων πίνακα
        printf("pin = ");
        showPin( 5,  pin);

        // Εύρεση μεγίστου στοιχείου του πίνακα και της θέσης του στον πίνακα
        max = pin[0];
        int thesiMax = 0;
        for (i = 1;i <= 5-1;i++)
           if (pin[i] >= max )
             {
              max = pin[i];
              thesiMax = i;
             }

        // Εμφάνιση μεγίστου στοιχείου του πίνακα και της θέσης του στον πίνακα
        printf("max = %d thesiMax = %d\n", max, thesiMax );

        system("Pause");
}
```

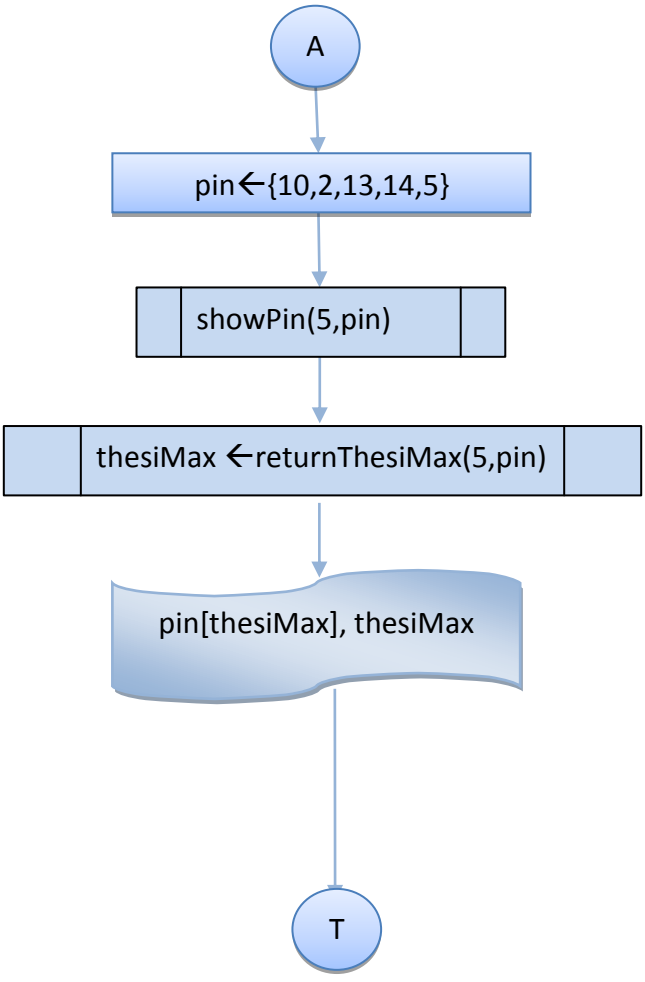
Έξοδος Προγράμματος :

```
pin = 10 2 13 14 5
max = 14 thesiMax = 3
Press any key to continue . . .
```

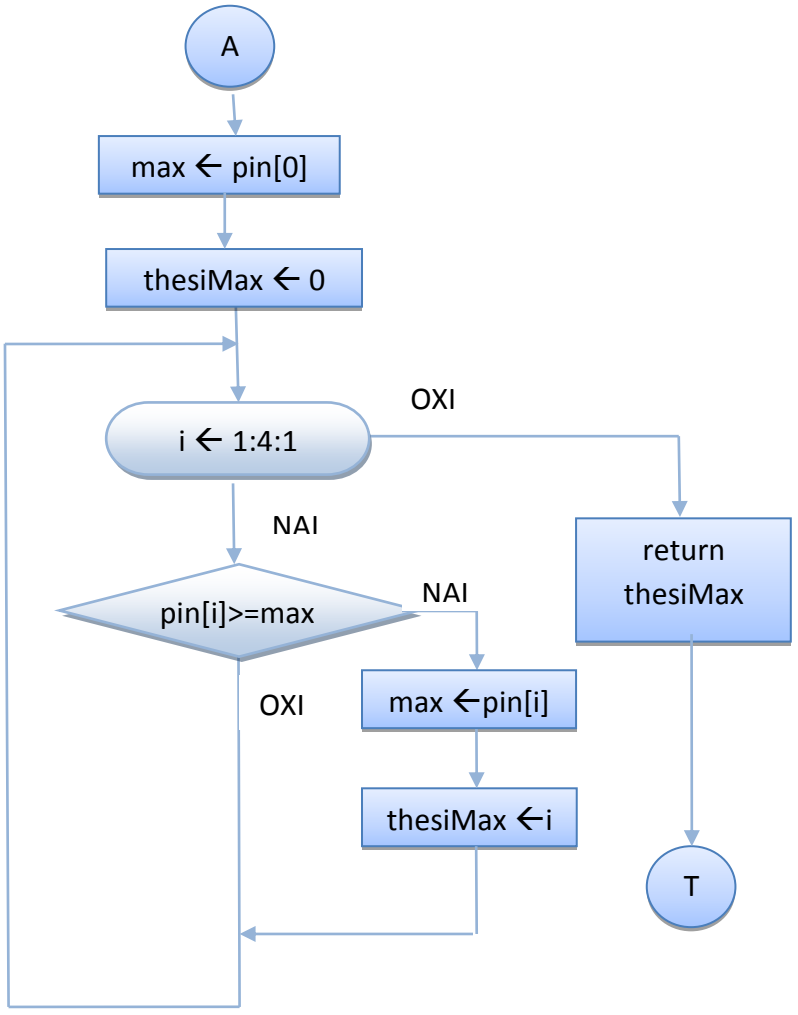
6.3.3 Εύρεση Στοιχείου με τη Μέγιστη Τιμή σε έναν Πίνακα και της Θέσης του στον Πίνακα με την Κλήση της Συνάρτησης returnThesiMax()

- Να γραφεί Πρόγραμμα που να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων με δυναμική αρχικοποίηση, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης `showPin()`, να βρίσκει το στοιχείο με τη μεγαλύτερη τιμή και τη θέση του στον πίνακα, την οποία θα επιστρέφει με την κλήση της συνάρτησης `returnThesiMax()` και να εμφανίζει το στοιχείο με τη μεγαλύτερη τιμή και **τη θέση του** στον πίνακα

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `returnThesiMax()`



Αλγόριθμος `returnThesiMax()`

1. Θέτουμε σαν μέγιστο στοιχείο `max` το πρώτο στοιχείο `pin[0]` (αφού δεν υπάρχει άλλο μέχρι στιγμής να συγκριθεί)
2. **Θέτουμε σαν θέση μεγίστου στοιχείου `thesiMax` το 0**
3. **Για** τα υπόλοιπα στοιχεία `pin[i]`, για `i = 1` μέχρι `4` :
   Αν το στοιχείο `pin[i]` είναι μεγαλύτερο ή ίσο με το `max`
   Θέτουμε σαν μέγιστο στοιχείο `max` την τιμή του `pin[i]`
   Θέτουμε την τιμή του `i` στο `thesiMax`
4. Επιστρέφουμε την τιμή της μεταβλητής **`thesiMax`**

Αλγόριθμος `main()`

1. Δήλωση - Αρχικοποίηση πίνακα `pin[]` με `5` θέσεις
2. Εμφάνιση στοιχείων πίνακα `pin` – κλήση `showPin()`
3. Εύρεση θέσης μεγίστου στοιχείου στον πίνακα `pin` – κλήση `returnThesiMax()`
4. Εμφάνιση της τιμής του μεγίστου στοιχείου `pin[thesiMax]` και της θέσης του `thesiMax`

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>


int returnThesiMax(int n, int pin[])
{
   int i, max, thesiMax;
   max = pin[0];
   thesiMax = 0;
   for (i = 1;i <= n-1;i++)
      if (pin[i] > max )
      {
        max = pin[i];
        thesiMax = i;
      }
   return thesiMax;
}

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}
```

```
main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με δυναμική
αρχικοποίηση, εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης showPin(),
βρίσκει το στοιχείο με τη μεγαλύτερη τιμή και τη θέση του στον πίνακα και
εμφανίζει το στοιχείο με τη μεγαλύτερη τιμή και τη θέση του στον πίνακα με την
κλήση της συνάρτησης returnThesiMax()  */
{
        int i, thesiMax;

        // Δήλωση - Αρχικοποίηση πίνακα
        int pin[] = {10,2,13,14,5};

        // Εμφάνιση στοιχείων πίνακα
        printf("pin = ");
        showPin( 5,  pin);

        // Εύρεση θέσης μεγίστου στοιχείου του πίνακα κλήση returnThesiMax()
        int thesiMax = returnThesiMax( 5,  pin);

        // Εμφάνιση μεγίστου στοιχείου του πίνακα και της θέσης του στον πίνακα
        printf("max = %d thesiMax = %d\n", pin[thesiMax], thesiMax );

        system("Pause");
}
```

Έξοδος Προγράμματος :

```
pin = 10 2 13 14 5
max = 14 thesiMax = 3
Press any key to continue . . .
```

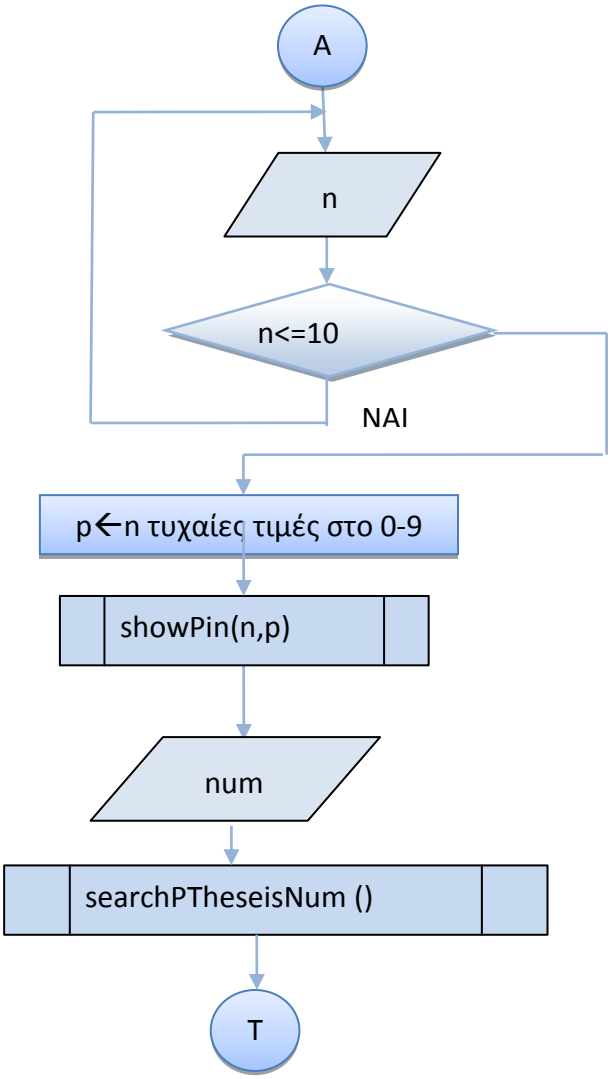
6.4 Εύρεση Στοιχείου σε έναν Πίνακα - Σειριακή Αναζήτηση

Σε πολλά προβλήματα, χρειάζεται να βρούμε αν υπάρχει κάποιο στοιχείο σε έναν πίνακα διατρέχοντας όλα τα στοιχεία του πίνακα για να βρούμε όλες τις θέσεις, στις οποίες βρίσκεται ή να σταματήσουμε μόλις το βρούμε για πρώτη φορά.

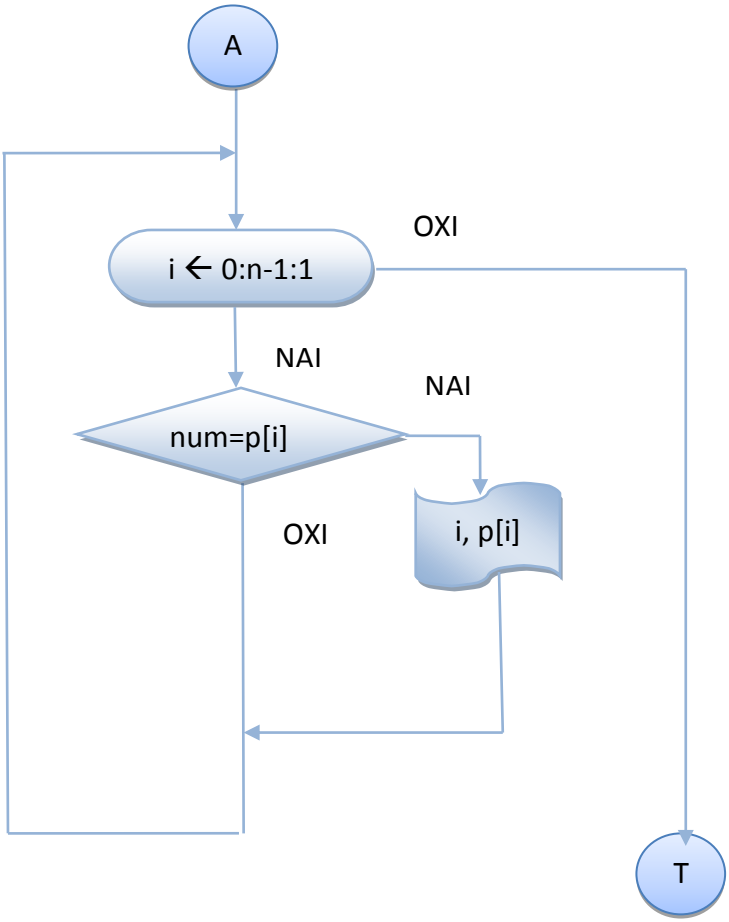
6.4.1 Εύρεση Όλων των Εμφανίσεων ενός Στοιχείου σε έναν Πίνακα

- Να γραφεί Πρόγραμμα που να διαβάζει έναν ακέραιο αριθμό `n>10`, να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων `n` θέσεων με τυχαίες τιμές στο διάστημα `[0,9]`, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης `showPin()`, να διαβάζει έναν ακέραιο αριθμό `num` και να καλεί τη συνάρτηση `searchPTheseisNum()`, η οποία θα διατρέχει όλα τα στοιχεία του πίνακα, ώστε να βρει και να εμφανίσει τις θέσεις στον πίνακα, στις οποίες υπάρχει ο αριθμός `num` και τις αντίστοιχες τιμές των στοιχείων του πίνακα.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ searchPTheseisNum **()**



Αλγόριθμος searchPTheseisNum **()**

1. **Για** τα στοιχεία `p[i]`, για `i = 0` μέχρι `n-1` :
   Αν το στοιχείο `p[i]` είναι ίσο με το `num`
   Εμφανίζουμε τη θέση `i` και το στοιχείο `p[i]`

Αλγόριθμος `main()`

1. **Κάνε**
   Διάβασε τιμή για το `n`
   Για όσο `(n<=10)`
2. Γέμισμα θέσεων πίνακα `p[]` με τυχαίες τιμές στο `[0,9]`
3. Εμφάνιση στοιχείων πίνακα `p` – κλήση `showPin()`
4. Διάβασε τιμή για το `num`
5. Εύρεση στοιχείων του πίνακα `p` ίδια με το `num` - κλήση `searchPTheseisNum()`

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

void searchPTheseisNum(int num, int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
     if ( num == p[i] )
        printf("Found num = %d in position %d, p[%d] = %d\n", num, i, i, p[i]);
}

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}

main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με τυχαίους
ακέραιους στο [0,9], εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης
showPin(), διαβάζει έναν ακέραιο αριθμό num και με την κλήση της συνάρτησης
searchPTheseisNum() ψάχνει όλα τα στοιχεία του πίνακα p να βρει σε ποιες θέσεις
υπάρχει ο αριθμός num και Εμφανίζει το στοιχείο και τη θέση του στον πίνακα */
{
    int i, num, n;
    // Διάβασμα n > 10
    do
    {
       printf("Give an integer n > 10 : ");
       scanf("%d", &n);
    }
    while ( n <= 10 );
    // Δήλωση πίνακα
    int p[n];
    // Γέμισμα πίνακα με τυχαίους ακέραιους στο [0,9]
    for (i = 0;i <= n-1;i++)
      p[i] = rand() % 10;
    // Εμφάνιση στοιχείων Πίνακα
    showPin( n,  p);
    // Διάβασμα num
    printf("Give an integer num : ");
    scanf("%d", &num);
    // Κλήση της συνάρτησης searchPTheseisNum()
    searchPTheseisNum( num,  n,  p);
    system("Pause");
}
```

Έξοδος Προγράμματος :

```
Give an integer n > 10 : 15
p = 1 7 4 0 9 4 6 6 2 4 5 5 1 7 1
Give an integer num : 4
Found num = 4 in position 2, p[2] = 4
Found num = 4 in position 5, p[5] = 4
Found num = 4 in position 9, p[9] = 4
Press any key to continue . . .
```

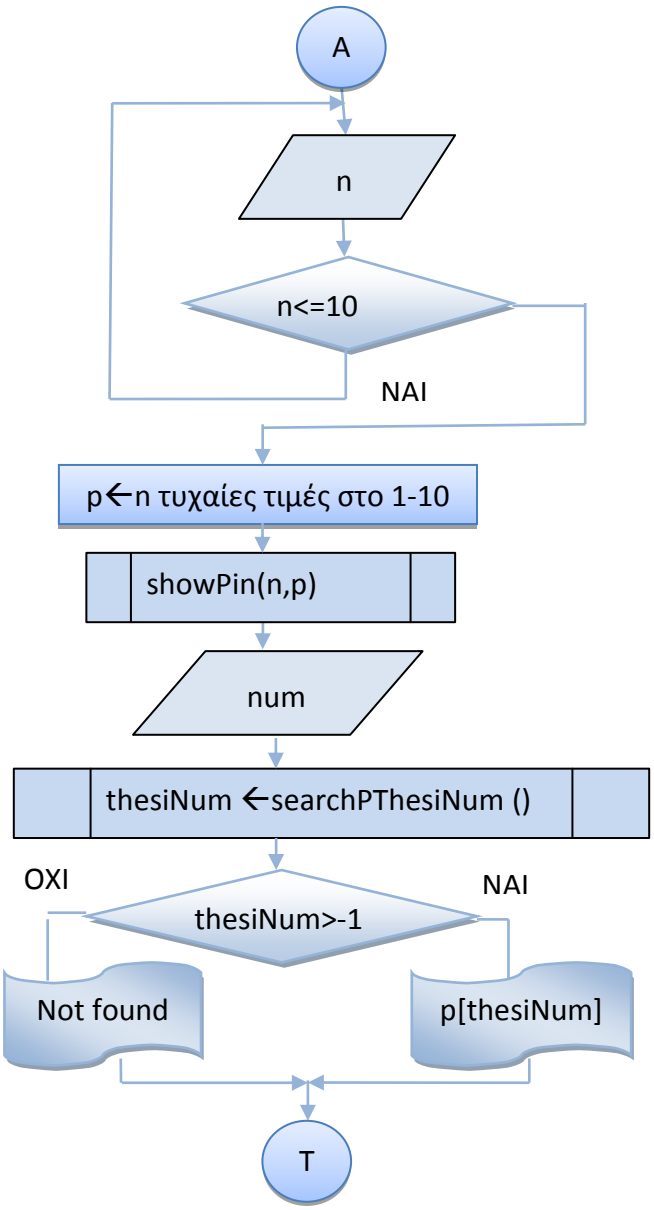
Παρατήρηση

Η συνάρτηση `rand()` επιστρέφει μια ακέραια τιμή μεταξύ του `0` και του `RAND_MAX = 32767`. Χρησιμοποιώντας το `rand() % 10` την προβάλλουμε στο διάστημα `[0,9]`. Αν θέλουμε να πάρουμε τιμές στο διάστημα `[a,b]` χρησιμοποιούμε το `(rand() % (b - a + 1)) + a`.

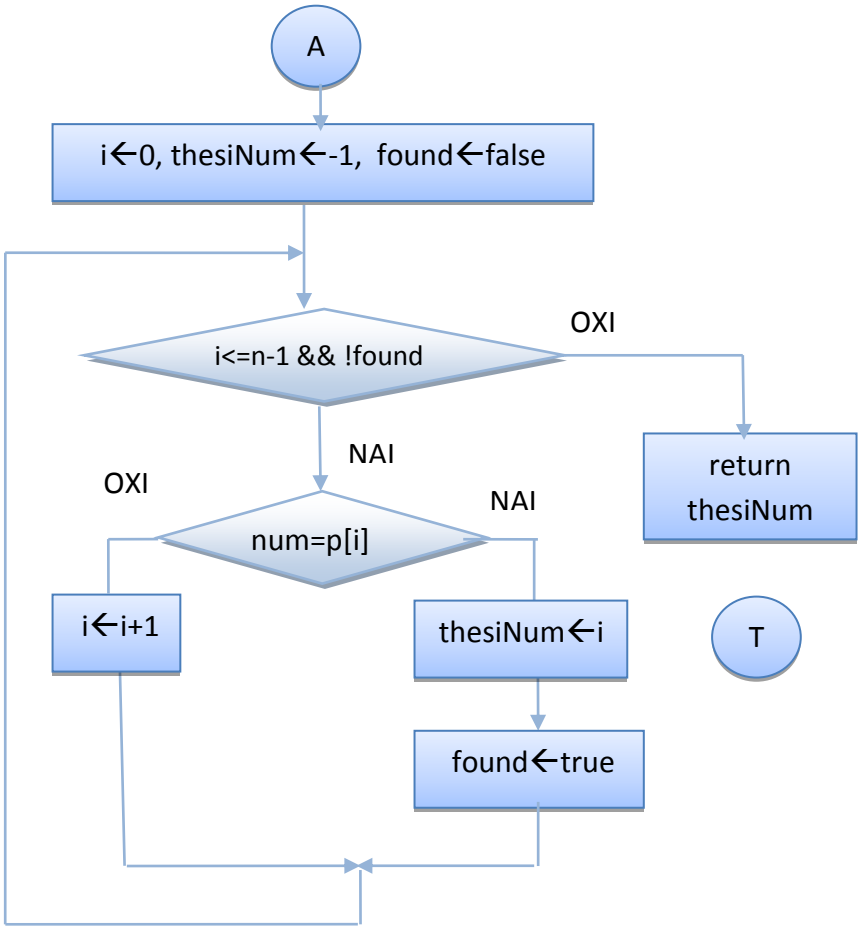
6.4.2 Εύρεση της Πρώτης Εμφάνισης ενός Στοιχείου σε έναν Πίνακα, Επιστροφή της Θέσης

- Να γραφεί Πρόγραμμα που να διαβάζει έναν ακέραιο αριθμό `n>10`, να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων `n` θέσεων με τυχαίες τιμές στο διάστημα `[1,10]`, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης `showPin()`, να διαβάζει έναν ακέραιο αριθμό `num` και να καλεί τη συνάρτηση `searchPThesiNum()`, η οποία θα διατρέχει όλα τα στοιχεία του πίνακα, ώστε να βρει και να επιστρέψει τη θέση του πίνακα, στην οποία βρέθηκε ο αριθμός `num`. Αν δεν βρεθεί ο αριθμός `num`, επιστρέφει το `-1`.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ searchPThesiNum()



Αλγόριθμος searchPThesiNum **()**

1. i←0
2. thesiNum←-1
3. found←false
4. **Για Όσο** (i <= n-1 και found = false)
   Αν το στοιχείο p[i] είναι ίσο με το num
   Κρατάμε στο thesiNum τη θέση i
   Κάνουμε τη found true
   Αλλιώς
   Αυξάνουμε το μετρητή i κατά 1 (i←i+1)

Αλγόριθμος main()

1. **Διάβασε** τιμή για το n
2. **Για όσο** (n<=10)
3. Γέμισμα θέσεων πίνακα p[] με τυχαίες τιμές στο [1,10]
4. Εμφάνιση στοιχείων πίνακα p – κλήση showPin()
5. Διάβασε τιμή για το num
6. Κλήση searchPThesiNum(), αποθήκευση στο thesiNum της θέσης (από 0 μέχρι n-1) του στοιχείου, αν βρέθηκε ίδιο με το num , του -1, αν δεν βρέθηκε στον πίνακα p.
7. **Αν** βρέθηκε (thesiNum > -1)
   Εμφανίζουμε τη θέση thesiNum και το στοιχείο p[thesiNum]
   Αλλιώς
   Εμφανίζουμε το μήνυμα “not Found”

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

int searchPThesiNum(int num, int n, int p[])
{
   int i=0;
   int thesiNum = -1;
   bool found = false;
   while ( i<=n-1 && !found)
     if ( num == p[i] )
      {
       thesiNum = i;
       found = true;
      }
     else
       i++;
   return thesiNum;
}

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");  }
```

```
main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με τυχαίους
ακέραιους στο [1,10], εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης
showPin(), διαβάζει έναν ακέραιο αριθμό num και με την κλήση της συνάρτησης
searchPThesiNum() ψάχνει όλα τα στοιχεία του πίνακα p να βρει την πρώτη θέση
στον πίνακα, στην οποία υπάρχει ο αριθμός num και την επιστρέφει */
{
    int i, num, n;

    // Διάβασμα n > 10
    do
    {
       printf("Give an integer n > 10 : ");
       scanf("%d", &n);
    }
    while ( n <= 10 );

    // Δήλωση πίνακα
    int p[n];

    // Γέμισμα πίνακα με τυχαίους ακέραιους στο [1,10]
    for (i = 0;i <= n-1;i++)
      p[i] = (rand() % (10 - 1 + 1)) + 1;

    // Εμφάνιση στοιχείων Πίνακα
    showPin( n,  p);

    // Διάβασμα num
    printf("Give an integer num : ");
    scanf("%d", &num);

    // Κλήση της συνάρτησης searchPThesiNum()
    thesiNum = searchPThesiNum( num,  n,  p);

    if ( thesiNum > -1 )
      printf("Found num = %d in position %d, p[%d] = %d\n", num, thesiNum,
      thesiNum, p[thesiNum]);
    else
     printf("num = %d not Found\n", num);
 system("Pause");
}
```

Έξοδος Προγράμματος :

```
Give an integer n > 10 : 15
p = 1 7 4 0 9 4 6 6 2 4 5 5 1 7 1
Give an integer num : 4
Found num = 4 in position 2, p[2] = 4
Press any key to continue . . .

Give an integer n > 10 : 15
p = 1 7 4 0 9 4 6 6 2 4 5 5 1 7 1
Give an integer num : 14
num = 14 not Found
Press any key to continue . . .
```

Παρατήρηση:

Η `boolean` μεταβλητή `found` χρειάζεται σε τέτοιου είδους προβλήματα, για να σημειώσουμε ότι κάποια συνθήκη ικανοποιήθηκε (π.χ. βρέθηκε το στοιχείο), αλλά στην περίπτωσή μας δεν χρειάζεται, γιατί η τιμή της τοπικής μεταβλητής `thesiNum` μας δείχνει και το αν βρέθηκε ή όχι. Αφού ξεκινάει με την τιμή `-1`, αν η τιμή αυτή αλλάξει, αυτό σημαίνει ότι το στοιχείο μας βρέθηκε στον πίνακα. Θα μπορούσαμε λοιπόν να παραλείψουμε τη `boolean` μεταβλητή `found` και να χρησιμοποιήσουμε στο `while` τον έλεγχο της τιμής της μεταβλητής `thesiNum`.

Αλγόριθμος searchPThesiNum **( )**

1. `i←0`
2. `thesiNum←-1`
3. **Για Όσο** (`i <= n-1` και `thesiNum == -1`)
   Αν το στοιχείο `p[i]` είναι ίσο με το `num`
   Κρατάμε στο `thesiNum` τη θέση `i`
   Αλλιώς
   Αυξάνουμε το μετρητή i κατά `1 (i←i+1)`

```
int searchPThesiNum(int num, int n, int p[])
{
   int i=0;
   int thesiNum = -1;
   while ( i<=n-1 && thesiNum == -1)
     if ( num == p[i] )
       thesiNum = i;
    else
       i++;
   return thesiNum;
}
```

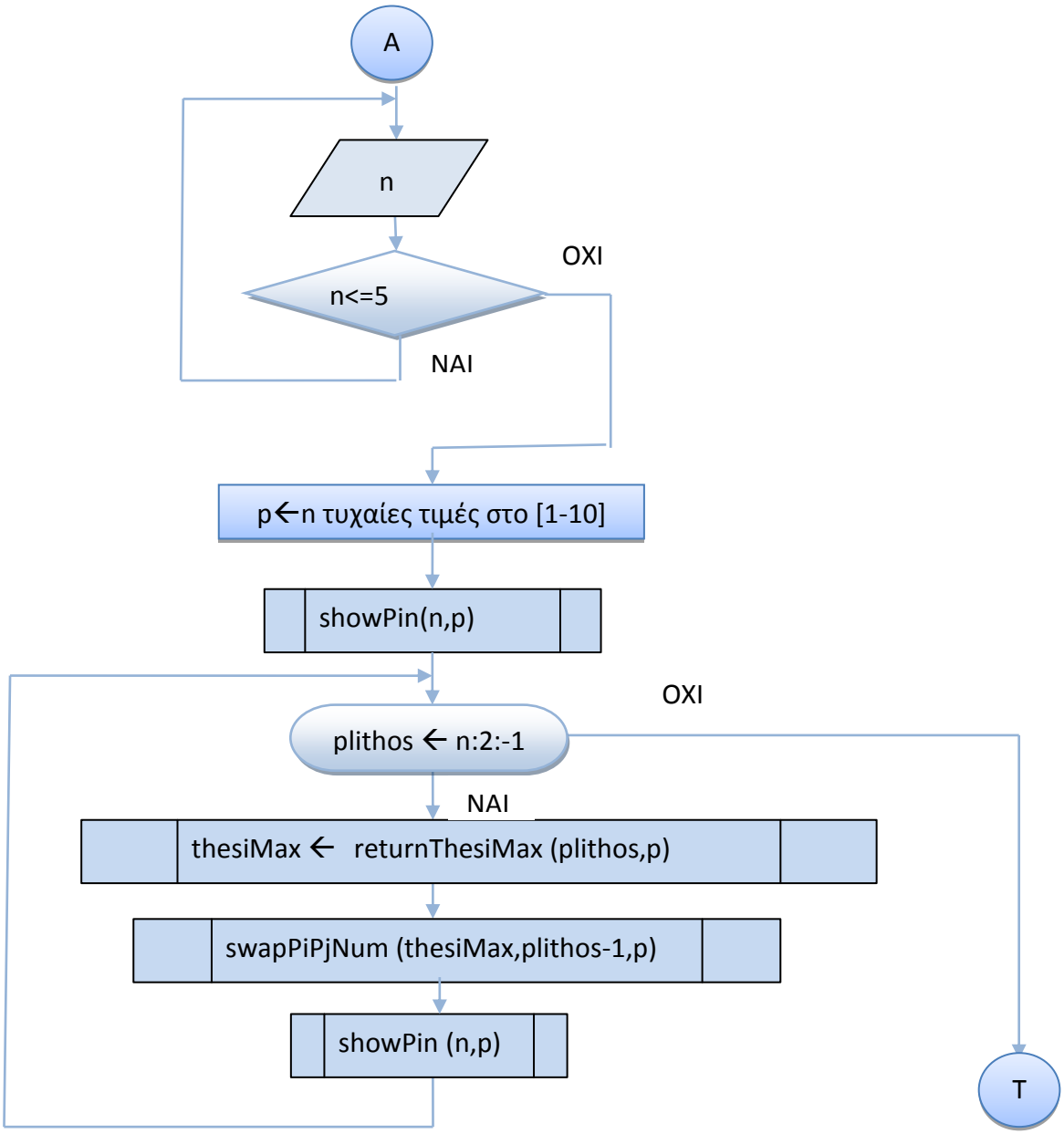
6.5 Ταξινόμηση των Στοιχείων σε Έναν Πίνακα

Πολλές φορές υπάρχει ανάγκη να ταξινομηθούν τα στοιχεία ενός πίνακα σε αύξουσα ή φθίνουσα διάταξη. Υπάρχουν πολλοί τρόποι να γίνει αυτό. Θα εξετάσουμε απλές μεθόδους με τη χρήση των συναρτήσεων που έχουν διδαχθεί.

6.5.1 Ταξινόμηση των Στοιχείων Ενός Πίνακα με την Εύρεση του Μεγίστου Κάθε Φορά Στοιχείου

- Να γραφεί Πρόγραμμα που να διαβάζει έναν ακέραιο αριθμό `n>5`, να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων `n` θέσεων με τυχαίες τιμές στο διάστημα `[1,10]`, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης `showPin()` και να καλεί `n-1` φορές τη συνάρτηση `returnThesiMax()`, η οποία θα βρίσκει το μέγιστο κάθε φορά στοιχείο, το οποίο με την κλήση της συνάρτησης `swapPiPj()`, θα το ανταλλάσσει με το στοιχείο της τελευταίας, προ-τελευταίας κ.λ.π. θέσης.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



Αλγόριθμος `main()`

1. **Διάβασε** τιμή για το `n`
2. **Για όσο** `(n<=5)`
3. Γέμισμα θέσεων πίνακα `p[]` με τυχαίες τιμές στο `[1,10]`
4. Εμφάνιση στοιχείων πίνακα `p` – κλήση `showPin(n,p)`
5. **Για** `plithos = n:2:-1`
   Κλήση `returnThesiMax(plithos, p)`, αποθήκευση στο `thesiMax` της θέσης
   Κλήση `swapPiPj(thesiMax, plithos-1, p)`, ανταλλαγή στοιχείων στις θέσεις `thesiMax` και `plithos-1`.
   Κλήση `showPin(n, p)`,

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>


void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}


void swapPiPj(int i, int j, int p[])
{
    int temp;
    temp = p[i];
    p[i] = p[j];
    p[j] = temp;
}


int returnThesiMax(int n, int p[])
{
   int i, max, thesiMax;
   max = p[0];
   thesiMax = 0;
   for (i = 1;i <= n-1;i++)
      if (p[i] >= max )
      {
        max = p[i];
        thesiMax = i;
      }
   return thesiMax;
}


main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με τυχαίους
ακέραιους στο [1,10], εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης
showPin(), και ταξινομεί τα στοιχεία του με την κλήση της συνάρτησης
returnThesiMax(), με την οποία βρίσκει κάθε φορά το μέγιστο όλων των στοιχείων
και την κλήση της συνάρτησης swapPiPj (), με την οποία τοποθετεί στο τέλος του
πίνακα το κάθε φορά μέγιστο στοιχείο */
```

```
{
    int i, n, thesiMax, plithos;

    // Διάβασμα n > 5
    do
    {
       printf("Give an integer n > 5 : ");
       scanf("%d", &n);
    }
    while ( n <= 5 );

    // Δήλωση πίνακα
    int p[n];

    // Γέμισμα πίνακα με τυχαίους ακέραιους στο [1,10]
    for (i = 0;i <= n-1;i++)
      p[i] = (rand() % (10 - 1 + 1)) + 1;

    // Εμφάνιση στοιχείων Πίνακα
    showPin( n,  p);

   for (plithos = n;plithos >= 2;plithos--)
      {
       thesiMax = returnThesiMax(plithos,  p);
       swapPiPj(thesiMax, plithos-1, p);
       printf("bhma = %d p = ", n-plithos+1);
       showPin( n,  p);
      }

   system("Pause");
}
```

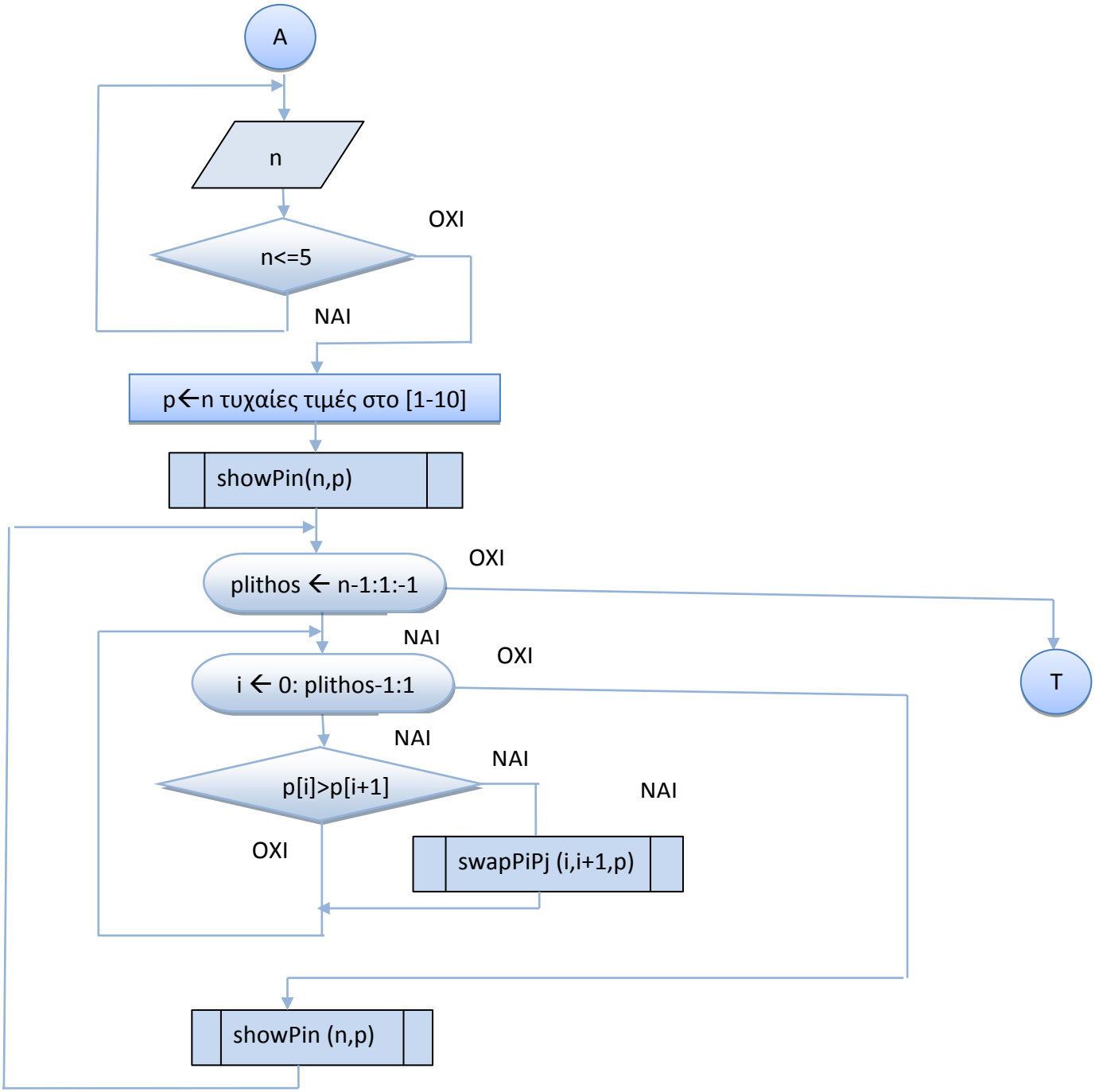
Έξοδος Προγράμματος :

```
Give an integer n > 5 : 10
p = 2 8 5 1 10 5 9 9 3 5
bhma 1, p = 2 8 5 1 5 5 9 9 3 10
bhma 2, p = 2 8 5 1 5 5 9 3 9 10
bhma 3, p = 2 8 5 1 5 5 3 9 9 10
bhma 4, p = 2 3 5 1 5 5 8 9 9 10
bhma 5, p = 2 3 5 1 5 5 8 9 9 10
bhma 6, p = 2 3 5 1 5 5 9 9 9 10
bhma 7, p = 2 3 1 5 5 5 9 9 9 10
bhma 8, p = 2 1 3 5 5 5 9 9 9 10
bhma 9, p = 1 2 3 5 5 5 9 9 9 10
Press any key to continue . . .
```

6.5.2 Ταξινόμηση των Στοιχείων ενός Πίνακα με την Προώθηση του Μεγίστου Κάθε Φορά Στοιχείου προς το Τέλος – Μέθοδος Φυσαλίδας

- Να γραφεί Πρόγραμμα που να διαβάζει έναν ακέραιο αριθμό `n>5`, να δημιουργεί και να γεμίζει έναν πίνακα ακεραίων `n` θέσεων με τυχαίες τιμές στο διάστημα `[1,10]`, να εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης `showPin()` και να καλεί `n-1` φορές τη συνάρτηση `swapPiPj()`, με την οποία θα ανταλλάσσει κάθε φορά δυο διαδοχικά στοιχεία του πίνακα, τα οποία δεν βρίσκονται στη σωστή διάταξη.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



Αλγόριθμος `main()`

1. **Διάβασε** τιμή για το `n`
2. **Για όσο** `(n<=5)`
3. Γέμισμα θέσεων πίνακα `p[]` με τυχαίες τιμές στο `[1,10]`
4. Εμφάνιση στοιχείων πίνακα `p` – κλήση `showPin(n,p)`
5. **Για** `plithos = n-1:1:-1`
   Για `i=0:plithos-1:1`
   Αν `(p[i] > p[i+1])`
   Κλήση `swapPiPj(i, i+1, p)`, ανταλλαγή στοιχείων στις θέσεις `i` και `i+1`.
   Κλήση `showPin(n, p)`,

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}


void swapPiPj(int i, int j, int p[])
{
    int temp;
    temp = p[i];
    p[i] = p[j];
    p[j] = temp;
}



main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με τυχαίους
ακέραιους στο [1,10], εμφανίζει τα στοιχεία του με την κλήση της συνάρτησης
showPin(), και ταξινομεί τα στοιχεία του με την κλήση της συνάρτησης swapPiPj
(), με την οποία τοποθετεί στο τέλος του πίνακα το κάθε φορά μέγιστο στοιχείο
συγκρίνοντας το κάθε στοιχείο με το επόμενό του, αν βρίσκονται στη σωστή θέση
*/
{
    int i, n, plithos;

    // Διάβασμα n > 5
    do
    {
       printf("Give an integer n > 5 : ");
       scanf("%d", &n);
    }
    while ( n <= 5 );

    // Δήλωση πίνακα
    int p[n];

    // Γέμισμα πίνακα με τυχαίους ακέραιους στο [1,10]
    for (i = 0;i <= n-1;i++)
      p[i] = (rand() % (10 - 1 + 1)) + 1;

    // Εμφάνιση στοιχείων Πίνακα
    showPin( n,  p);

   for (plithos = n-1;plithos >= 1;plithos--)
      {
       for (i = 0;i<=plithos-1 >= 1;i++)
         if ( p[i] > p[i+1] )
             swapPiPj(i, i+1, p);
       printf("bhma = %d p = ", n-plithos);
       showPin( n,  p);
      }
   system("Pause");
}
```

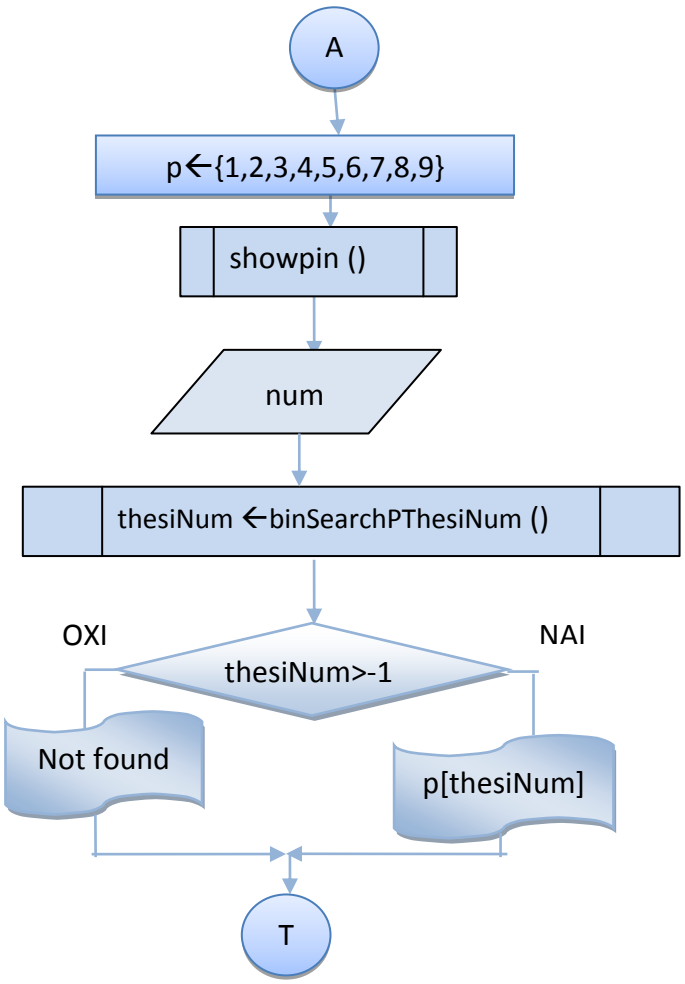
Έξοδος Προγράμματος :

```
Give an integer n > 5 : 10
p = 2 8 5 1 10 5 9 9 3 5
bhma 1, p = 2 5 1 8 5 9 9 3 5 10
bhma 2, p = 2 1 5 5 8 9 3 5 9 10
bhma 3, p = 1 2 5 5 8 3 5 9 9 10
bhma 4, p = 1 2 5 5 3 5 8 9 9 10
bhma 5, p = 1 2 5 3 5 5 8 9 9 10
bhma 6, p = 1 2 3 5 5 5 8 9 9 10
bhma 7, p = 1 2 3 5 5 5 8 9 9 10
bhma 8, p = 1 2 3 5 5 5 8 9 9 10
bhma 9, p = 1 2 3 5 5 5 8 9 9 10
Press any key to continue . . .
```

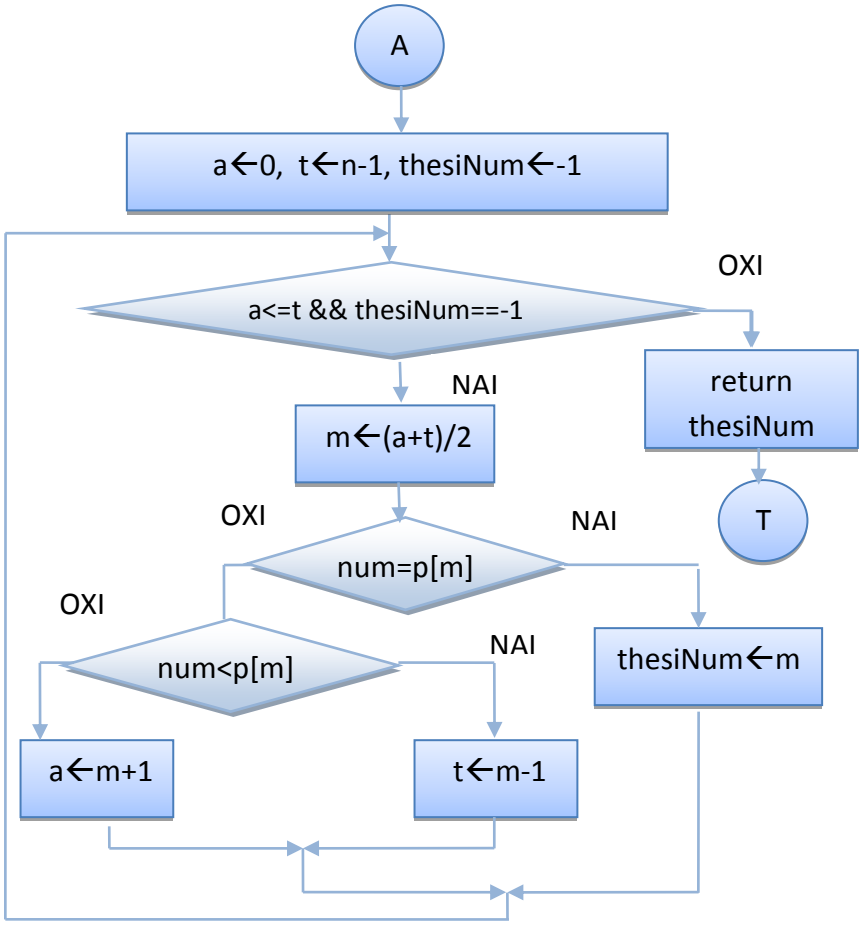
6.5.3 Εύρεση της Πρώτης Εμφάνισης ενός Στοιχείου σε έναν Ταξινομημένο Πίνακα, Επιστροφή της Θέσης – Δυαδική Αναζήτηση

- Να γραφεί Πρόγραμμα που να γεμίζει με δυναμική αρχικοποίηση έναν πίνακα ακεραίων με τις τιμές `{1,2,3,4,5,6,7,8,9}`, να διαβάζει έναν ακέραιο αριθμό `num` και να καλεί τη συνάρτηση `binSearchPThesiNum()`, η οποία σύμφωνα με τη σχέση του αριθμού που ψάχνουμε με το μεσαίο στοιχείο του πίνακα, θα ψάχνει κάθε φορά το αντίστοιχο αριστερό ή δεξί κομμάτι του πίνακα, ώστε να βρει και να επιστρέψει τη θέση του πίνακα, στην οποία βρέθηκε ο αριθμός `num`. Αν δεν βρεθεί ο αριθμός `num`, επιστρέφει το `-1`.

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ `main()`



ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ binSearchPThesiNum **()**



Αλγόριθμος binSearchPThesiNum**()**

1. a←0
2. t←n-1
3. thesiNum←-1
4. **Για Όσο** (a <= t και thesiNum = -1)
   m=(a+t)/2
   Αν το στοιχείο p[m] είναι ίσο με το num
   Κρατάμε στο thesiNum τη θέση m
   Αλλιώς
   Αν το στοιχείο p[m] είναι μεγαλύτερο του num
   Μετακινούμε το τέλος t στη θέση m-1
   Αλλιώς
   Μετακινούμε την αρχή a στη θέση m+1

Αλγόριθμος main()

1. Γέμισμα πίνακα p[]με τις τιμές {1,2,3,4,5,6,7,8,9}
2. Εμφάνιση στοιχείων p[] – κλήση showPin()
3. Διάβασε τιμή για το num
4. Κλήση binSearchPThesiNum(), αποθήκευση στο thesiNum της θέσης (από 0 μέχρι n-1) του στοιχείου, αν βρέθηκε ίδιο με το num , του -1, αν δεν βρέθηκε στον πίνακα p.
5. **Αν** βρέθηκε (thesiNum > -1) είναι ίσο με το num
   Εμφανίζουμε τη θέση thesiNum και το στοιχείο p[thesiNum]
   Αλλιώς
   Εμφανίζουμε το μήνυμα “not Found”

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

int binSearchPThesiNum(int num, int n, int p[])
{
   int a=0, t=n-1, m;
   int thesiNum = -1;
   while ( a<=t && thesiNum == -1)
    {
     m = (a+t)/2;
     if ( num == p[m] )
       thesiNum = m;
    else
      if ( num < p[m] )
         t = m-1;
      else
         a = m+1;
    printf("a = %d t = %d\n",a,t);
    }
   return thesiNum;
}
```

```
void showPin(int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf("%d ", p[i]);
   printf("\n");
}


main()

/* Πρόγραμμα που δημιουργεί και γεμίζει έναν πίνακα ακεραίων με τους ακέραιους
στο [1,9], διαβάζει έναν ακέραιο αριθμό num και με την κλήση της συνάρτησης
binsearchPThesiNum() ψάχνει όλα τα στοιχεία του πίνακα p να βρει την πρώτη θέση
στον πίνακα, στην οποία υπάρχει ο αριθμός num και την επιστρέφει */
{
    int i, num, n;

    // Δήλωση Γέμισμα πίνακα με τους ακέραιους στο [1,9]
    int p[] = {1,2,3,4,5,6,7,8,9};

    // Εμφάνιση στοιχείων π[] – κλήση showPin()
    showPin(10, p)

    // Διάβασμα num
    printf("Give an integer num : ");
    scanf("%d", &num);

    // Κλήση της συνάρτησης binSearchPThesiNum()
    thesiNum = binsearchPThesiNum( num,  10,  p);

    if ( thesiNum > -1 )
      printf("Found num = %d in position %d, p[%d] = %d\n", num, thesiNum,
      thesiNum, p[thesiNum]);
    else
     printf("num = %d not Found\n", num);
 system("Pause");
}
```

Έξοδος Προγράμματος :

```
p = 1 2 3 4 5 6 7 8 9 10
Give an integer num : 4
Found num = 4 in position 3, p[3] = 4
Press any key to continue . . .

p = 1 2 3 4 5 6 7 8 9 10
Give an integer num : 14
num = 14 not Found
Press any key to continue . . .
```

6.6 Πίνακες 2 Διαστάσεων

Πίνακας 2 διαστάσεων είναι μια διάταξη στοιχείων σε **γραμμές** και **στήλες** που το καθένα τους έχει κάποια θέση στην αντίστοιχη γραμμή και στήλη. Καταλαμβάνει μια περιοχή μνήμης με το λογικό όνομα του πίνακα, ενώ κάθε στοιχείο του ξεχωρίζει με **δύο δείκτες**, γραμμή - στήλη.

Παράδειγμα

Πίνακας 2 διαστάσεων, 3 γραμμών και 2 στηλών με περιεχόμενα τους αριθμούς 1-6.

	Στήλη 0	Στήλη 1
Γραμμή 0	1	2
Γραμμή 1	3	4
Γραμμή 2	5	6

Δήλωση Πίνακα 2 Διαστάσεων

Η Δήλωση ενός Πίνακα 2 διαστάσεων στη C γίνεται με τη δήλωση του **τύπου** των στοιχείων που θα αποθηκεύσει, το **όνομά** του και το **μέγεθός** του (αριθμός γραμμών και στηλών σε ξεχωριστά ζεύγη αγκυλών).

Παράδειγμα

```
int p1[3][2];
```

όπου δηλώνουμε τον πίνακα p1, ο οποίος θα αποθηκεύσει 6 ακέραιους αριθμούς.

Ο αριθμός των γραμμών και στηλών του πίνακα μπορεί να είναι οι τιμές δύο μεταβλητών. Οι επόμενες εντολές έχουν το ίδιο αποτέλεσμα :

```
int m = 3, n = 2;
int p1[m][n];
```

Δημιουργία Πίνακα 2 Διαστάσεων με Δυναμική Αρχικοποίηση

Ένας πίνακας 2 διαστάσεων μπορεί να δημιουργηθεί με **δυναμική αρχικοποίηση**, όπου οι τιμές του περικλείονται σε ζεύγη αγκίστρων για την κάθε γραμμή και χωρίζονται με κόμμα.

Παράδειγμα

```
int pin2[][] = {{1, 2}, {3, 4}, {5, 6}};
```

όπου δηλώνουμε τον πίνακα pin2, ο οποίος θα αποθηκεύσει τους ακέραιους αριθμούς από το 1 μέχρι το 6 σε 3 γραμμές από 2 στοιχεία η καθεμιά.

Πρόσβαση σε Κάποιο Στοιχείο Πίνακα 2 Διαστάσεων

Η **πρόσβαση** σε κάποιο στοιχείο του πίνακα γίνεται με το όνομά του και τη θέση του (δύο δείκτες γραμμής και στήλης) μέσα σε αγκύλες. Π.χ. το pin1[2][1] αναφέρεται στο στοιχείο του pin1 που βρίσκεται στην τρίτη γραμμή και δεύτερη στήλη, δηλαδή το 6.

6.6.1 Παράδειγμα Δημιουργίας – Γεμίσματος 2 πινάκων 2 διαστάσεων με random τιμές και Δυναμική Αρχικοποίηση

- Να γραφεί Αλγόριθμος/Πρόγραμμμα το οποίο γεμίζει έναν πίνακα 2 διαστάσεων (3x2), με **δυναμική αρχικοποίηση** και δίνει τις τιμές από το 1 μέχρι το 6, δηλώνει έναν άλλον πίνακα (mxn), θέσεων, τον οποίο γεμίζει με τυχαίες τιμές και εμφανίζει τα περιεχόμενα των 2 πινάκων.

Αλγόριθμος

1. Διαβάζω τιμές για τα m, n >1.
2. Δήλωση - Αρχικοποίηση πίνακα p1
3. Δήλωση - Γέμισμα πίνακα p2 με τυχαίες τιμές
   Για κάθε γραμμή i = 0 μέχρι (m – 1)
      Για κάθε στήλη j = 0 μέχρι (n – 1)
         Εκχωρούμε μια τυχαία τιμή μεταξύ 1 και 10 στο στοιχείο p2[i][j]
4. Εμφάνιση στοιχείων πίνακα pin1
   Για κάθε γραμμή i = 0 μέχρι (3 – 1)
      Για κάθε στήλη j = 0 μέχρι (2 – 1)
         Εμφανίζουμε το στοιχείο p1[i][j]
5. Εμφάνιση στοιχείων πίνακα p2
   Για κάθε γραμμή i = 0 μέχρι (m – 1)
      Για κάθε στήλη j = 0 μέχρι (n – 1)
         Εμφανίζουμε το στοιχείο p2[i][j]

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*
    Το πρόγραμμα δημιουργεί 2 πίνακες 2 διαστάσεων ( 3x2 ), τον πρώτο με
δυναμική αρχικοποίηση και τον γεμίζει με τις ακέραιες τιμές από 1 μέχρι 6,
ενώ το δεύτερο τον γεμίζει με τυχαίες τιμές από 1 μέχρι 10 και εμφανίζει
τα στοιχεία των 2 πινάκων.
    */
```

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

        // Δήλωση - Αρχικοποίηση πίνακα 1
        int p1[][] = {{1,2},{3,4},{5,6}};

        // Εκχώρηση ακέραιας τιμής > 1 στα μεγέθη γραμμών-στηλών του πίνακα 2
       do
       {
         printf("Give two integers m, n > 1 : ");
         scanf("%d %d", &m, &n);
       }
       while ( m <= 1 || n <= 1 );

        // Δήλωση πίνακα 2
       int p2[m][n];

        // Γέμισμα πίνακα 2 με τυχαίες ακέραιες τιμές
        for (i = 0;i < m;i++)
         for (j = 0;j < n;j++)
           p2[i][j] = (rand() % (10 - 1 + 1)) + 1;


        // Εμφάνιση στοιχείων πίνακα 1
        printf("p1 = \n");
        for ( i=0;i<=3-1;i++)
        {
         for ( j=0;j<=2-1;j++)
           printf("%3d ", p1[i][j]);
         printf("\n");
        }

        // Εμφάνιση στοιχείων πίνακα 2
        printf("p2 = \n");
        for ( i=0;i<=m-1;i++)
        {
         for ( j=0;j<=n-1;j++)
           printf("%3d ", p2[i][j]);
         printf("\n");
        }
 }
```

Έξοδος Προγράμματος :

```
Give two integers m, n > 1 : 3 4
p1 =
 1  2
 3  4
 5  6

p2 =
 2  8  5  1
10  5  9  9
 3  5  6  6
Press any key to continue . . .
```

6.6.2 Παράδειγμα Δημιουργίας – Γεμίσματος 2 πινάκων 2 Διαστάσεων με Τυχαίες Τιμές και Δυναμική Αρχικοποίηση με τη Χρήση Συναρτήσεων

- Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο γεμίζει έναν πίνακα 2 διαστάσεων (3x2), με **δυναμική αρχικοποίηση** και δίνει τις τιμές από το 1 μέχρι το 6, δηλώνει έναν άλλον πίνακα (mxn), θέσεων, τον οποίο γεμίζει με τυχαίες τιμές και εμφανίζει τα περιεχόμενα των 2 πινάκων καλώντας συναρτήσεις για το γέμισμα του πίνακα των τυχαίων τιμών και την εμφάνιση των 2 πινάκων. Μετά καλεί τη συνάρτηση findMax() με την οποία βρίσκει και εμφανίζει το μεγαλύτερο στοιχείο του πίνακα p2 και τη γραμμή και στήλη του πίνακα, στις οποίες βρίσκεται.

Αλγόριθμος

1. Διαβάζω τιμές για τα m, n > 1.
2. Δήλωση - Αρχικοποίηση πίνακα 1
3. Δήλωση - Γέμισμα πίνακα 2 με τυχαίες τιμές καλώντας τη μέθοδο fillPin2D()
4. Εμφάνιση στοιχείων πίνακα 1 καλώντας τη συνάρτηση showPin2D()
5. Εμφάνιση στοιχείων πίνακα 2 καλώντας τη συνάρτηση showPin2D()
6. Εύρεση μεγίστου στοιχείου p2, εμφάνιση στοιχείου και γραμμής-στήλης καλώντας τη συνάρτηση findMax()

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define nmax 100   // απαιτείται για το πέρασμα των παραμέτρων

/* Το πρόγραμμα δημιουργεί 2 πίνακες 2 διαστάσεων ( 3x2 ), τον πρώτο με
δυναμική αρχικοποίηση και τον γεμίζει με τις ακέραιες τιμές από 1 μέχρι 6,
ενώ το δεύτερο τον γεμίζει με τυχαίες τιμές από 1 μέχρι 10 καλώντας τη
συνάρτηση fillPin2D() και εμφανίζει τα στοιχεία των 2 πινάκων καλώντας τις
συναρτήσεις showPin2D(). Μετά καλεί τη συνάρτηση findMax()για την εύρεση του
μεγίστου στοιχείου του p2 και την εμφάνιση του μεγίστου στοιχείου και της
γραμμής-στήλης, στις οποίες βρέθηκε   */


void fillPin2D(int m, int n, int p[][nmax])
// Συνάρτηση δημιουργίας στοιχείων πίνακα 2D
{
   int i,j;
   for ( i=0;i<=m-1;i++)
    for ( j=0;j<=n-1;j++)
      p[i][j] = (rand() % (10 - 1 + 1)) + 1;
}

void showPin2D(int m, int n, int p[][nmax])
// Συνάρτηση Εμφάνισης στοιχείων πίνακα 2
{
   int i, j;
   for ( i=0;i<=m-1;i++)
    {
     for ( j=0;j<=n-1;j++)
       printf("%3d ", p[i][j]);
     printf("\n");
    }
}
```

```
void findMax(int m, int n, int p[][nmax])
// Εύρεση, Εμφάνιση Μεγίστου – Γραμμής – Στήλης
{
   int i, j, max, lineMax, colMax, lineColMax;
   max = p[0][0];
   lineMax = 0;
   colMax = 0;
   for (i = 0;i <= m-1;i++)
    for ( j=0;j<=n-1;j++)
      if (p[i][j] >= max )
      {
        max = p[i][j];
        lineMax = i;
        colMax = j;
      }
   printf("max = %d in line %d and col %d\n", p[lineMax][colMax], lineMax,
colMax);
}

main()
     int i, j;
        // Δήλωση - Αρχικοποίηση πίνακα 1
        int p1[][nmax] = {{1,2},{3,4},{5,6}};
       // Εκχώρηση ακέραιας τιμής>1 στα μεγέθη γραμμών-στηλών του πίνακα 2
       do
       {
         printf("Give two integers m, n > 1 : ");
         scanf("%d %d", &m, &n);
       }
       while ( m <= 1 || n <= 1 );
        // Δήλωση πίνακα 2
        int p2[m][nmax];

        // Γέμισμα πίνακα 2 με τυχαίες ακέραιες τιμές
        fillPin2D( m, n, p2);

        // Εμφάνιση στοιχείων πίνακα 1
        printf("p1 = \n");
        showPin2D( 3, 2,  p1);

        // Εμφάνιση στοιχείων πίνακα 2
        printf("p2 = \n");
        showPin2D( m, n,  p2);

       // Εύρεση, Εμφάνιση Μεγίστου – Γραμμής – Στήλης
       findMax( m, n,  p2);
       system("Pause");
     }
}
```

Έξοδος Προγράμματος :

```
Give two integers m, n > 1 : 3 4
p1 =
 1  2
 3  4
 5  6

p2 =
 2  8  5  1
10  5  9  9
 3  5  6  6
max = 10 in line 1 and column 0
Press any key to continue . . .
```

7 ΣΥΜΒΟΛΟΣΕΙΡΕΣ – STRINGS - Πίνακες Χαρακτήρων

Οι **Συμβολοσειρές – Strings** στη C υλοποιούνται σαν **Πίνακες** Χαρακτήρων (δεν υπάρχει ο τύπος String, υπάρχει όμως η βιβλιοθήκη συναρτήσεων **<string.h>**, την οποία μπορούμε να συμπεριλάβουμε με την εντολή **#include**). Η Δήλωση μιας Συμβολοσειράς γίνεται με τη δήλωση του **τύπου** char των στοιχείων που θα αποθηκεύσει, το **όνομά** της και τη σταθερά τύπου String μέσα σε " ".

Παράδειγμα

```
char s1[] = "Kostas Goulianas";
char s2[17] = "Kostas Goulianas";
char s3[20] = "KOSTAS GOULIANAS";
char s4[7] = {'K','o','s','t','a','s','\0'};
char s5[20];
printf("Give a string : ");
scanf("%s", s5);
```

όπου δηλώνουμε την Συμβολοσειρά s1, χωρίς τον αριθμό χαρακτήρων, η οποία θα αποθηκεύσει το όνομα "Kostas Goulianas", την Συμβολοσειρά s2, σαν πίνακα με 17 χαρακτήρες, η οποία θα αποθηκεύσει το όνομα "Kostas Goulianas" (16 χαρακτήρες + τον χαρακτήρα τερματισμού '\0'), την Συμβολοσειρά s3, σαν πίνακα με 20 χαρακτήρες, η οποία θα αποθηκεύσει το όνομα "KOSTAS GOULIANAS" και την Συμβολοσειρά s4, με 7 χαρακτήρες, τους 6 χαρακτήρες της λέξης "Kostas" αναλυτικά και σαν τελευταίο χαρακτήρα, τον χαρακτήρα τερματισμού '\0', η οποία θα αποθηκεύσει το όνομα "Kostas". Όλες οι παραπάνω Συμβολοσειρές έχουν δημιουργηθεί με **δυναμική αρχικοποίηση**. Η Συμβολοσειρά s5[20] δηλώνεται απλώς (χρειάζεται το μέγεθος) και παίρνει τιμή με την εντολή scanf(). Επειδή το s5 είναι πίνακας, δεν χρειάζεται το '&'.

Εμφάνιση Συμβολοσειρών

Η Εμφάνιση των Συμβολοσειρών σταθερών ή μεταβλητών γίνεται με την εντολή printf().

Παράδειγμα

```
printf("S1 = %s\n", s1);
printf("S2 = %s\n", s2);
printf("S3 = %s\n", s3);
printf("S4 = %s\n", s4);
printf("S5 = %s\n", s5);
```

Έξοδος Προγράμματος

```
Give a string : kostas
S1 = Kostas Goulianas
S2 = Kostas Goulianas
S3 = KOSTAS GOULIANAS
S4 = Kostas
S5 = kostas
Press any key to continue . . .
```

Μήκος Συμβολοσειράς

Για να βρούμε το **Μήκος** μιας Συμβολοσειράς χρησιμοποιούμε τη συνάρτηση `strlen()` , η οποία βρίσκεται στο **<string.h>** και επιστρέφει τον αριθμό χαρακτήρων που είναι αποθηκευμένοι στον πίνακα. Π.χ. ο πίνακας `s3` έχει `20` διαθέσιμους χαρακτήρες, αλλά αποθηκεύονται `16`.

Παράδειγμα

```
printf("length of S1 = %d\n", strlen(s1));
printf("length of S2 = %d\n", strlen(s2));
printf("length of S3 = %d\n", strlen(s3));
printf("length of S4 = %d\n", strlen(s4));
printf("length of S5 = %d\n", strlen(s5));
```

Έξοδος Προγράμματος

```
length of S1 = 16
length of S2 = 16
length of S3 = 16
length of S4 = 6
length of S5 = 6
Press any key to continue . . .
```

8.1 Συναρτήσεις Χειρισμού Συμβολοσειρών

Η βιβλιοθήκη **<string.h>** περιέχει τις παρακάτω συναρτήσεις για σύγκριση, αναζήτηση κάποιου χαρακτήρα, εξαγωγή Υπο-Συμβολοσειράς (`substring`) κ.λ.π..

Η Συνάρτηση `strcmp()`

Με τη Συνάρτηση `strcmp()` συγκρίνουμε `2` Συμβολοσειρές. Αν η πρώτη είναι μεγαλύτερη από τη δεύτερη, επιστρέφει μια τιμή μεγαλύτερη του μηδενός, αν η πρώτη είναι μικρότερη από τη δεύτερη, επιστρέφει μια τιμή μικρότερη του μηδενός και αν η πρώτη είναι ίση με τη δεύτερη, επιστρέφει την τιμή μηδέν. Η σύνταξή της είναι :

```
strcmp(<Συμβολοσειρά-1>,<Συμβολοσειρά-2>)
```

Παράδειγμα

```
char s1[] = "Kostas Goulianas";
char s2[17] = "Kostas Goulianas";
char s3[20] = "KOSTAS GOULIANAS";
char s4[7] = {'K','o','s','t','a','s','\0'};

if (strcmp(s1,s2) > 0 )
      printf("s1 = %s > s2 = %s\n", s1,s2);
else
      if (strcmp(s1,s2) < 0 )
        printf("s1 = %s < s2 = %s\n", s1,s2);
      else
        printf("s1 = %s = s2 = %s\n", s1,s2);

if (strcmp(s1,s3) > 0 )
      printf("s1 = %s > s3 = %s\n", s1,s3);
else
      if (strcmp(s1,s3) < 0 )
        printf("s1 = %s < s3= %s\n", s1,s3);
      else
        printf("s1 = %s = s3 = %s\n", s1,s3);
```

Έξοδος Προγράμματος

```
s1 = Kostas Goulianas = s2 = Kostas Goulianas
s1 = Kostas Goulianas > s3 = KOSTAS GOULIANAS
Press any key to continue . . .
```

Η Συνάρτηση `strcspn()`

Με τη Συνάρτηση `strcspn()` βρίσκουμε τη Θέση ενός **χαρακτήρα** της **πρώτης** του εμφάνισης σε μια Συμβολοσειρά. Αν ο **χαρακτήρας** δεν υπάρχει στη Συμβολοσειρά, η Συνάρτηση επιστρέφει την τελευταία θέση της Συμβολοσειράς. Η σύνταξή της είναι :

```
strcspn(<Συμβολοσειρά>, "χαρακτήρες προς αναζήτηση")
```

Παράδειγμα

```
String s1 = "Kostas Goulianas";
i = strcspn (s1,"a");
printf("look for a in string %s, i= %d\n", s1, i);
i = strcspn (s1,"f");
printf("look for f in string %s, i= %d\n", s1, i);
i = strcspn (s1,"as");
printf("look for as in string %s, i= %d\n", s1, i);
```

Έξοδος Προγράμματος

```
look for a in string Kostas Goulianas, i= 4 //  a θέση 4
look for f in string Kostas Goulianas, i= 16 //  f θέση 16, δεν υπάρχει
look for as in string Kostas Goulianas, i= 2 //  s θέση 2
Press any key to continue . . .
```

Προσομοίωση της Συνάρτησης `substring()`

Με τη Συνάρτηση `substring()` εξάγουμε ένα τμήμα μιας Συμβολοσειράς σε μια άλλη μεταβλητή τύπου πίνακα χαρακτήρων. Η αρχική Συμβολοσειρά παραμένει ως έχει. Ο κώδικας της συνάρτησης θα μπορούσε να είναι ο παρακάτω :

Παράδειγμα

```
void substring( int from, int to, char ch[], char sub[20])
{

    int i=0;
    while ( i < from )
     i++;

    while ( i < to )
    {
        sub[i-from] = ch[i];
        i++;
    }

}

main()
{
 String s1 = "Kostas Goulianas";
 int from = strcspn (s1," ");  //Βρίσκω τη θέση του κενού
 // Παίρνω τους χαρακτήρες απ' το κενό μέχρι το τέλος
 substring( from, strlen(s1), s1,  sub);
 printf("sub = %s\n", sub);
 system("Pause");
}
```

Έξοδος Προγράμματος

```
sub = Goulianas
Press any key to continue . . .
```

Η Συνάρτηση `strcpy()`

Με τη Συνάρτηση `strcpy()` αντιγράφουμε το περιεχόμενο μιας Συμβολοσειράς σε μιαν άλλη, της δεύτερης στην πρώτη (δεν επιτρέπεται η απ' ευθείας ανάθεση τιμής, π.χ. η εντολή `s1 = s2` θα βγάλει μήνυμα λάθους). Η σύνταξή της είναι :

`strcpy(<Συμβολοσειρά-1>, <Συμβολοσειρά-2>)`

Παράδειγμα

```
char s1[] = "Kostas Goulianas";
char s2[17] = "Kostas Goulianas";
char s3[20] = "KOSTAS GOULIANAS";
char s4[7] = {'K','o','s','t','a','s','\0'};
strcpy(s2, s3);
printf("strcpy(s2, s3) : s2 = %s s3 = %s\n", s2, s3);
strcpy(s3, s1);
printf("strcpy(s3, s1) : s1 = %s s3 = %s\n", s1, s3);
strcpy(s3, s4); ;
printf("strcpy(s3, s4) : s4 = %s s3 = %s\n", s4, s3);
strcpy(s4, s1); ;
printf("strcpy(s4, s1) : s4 = %s s1 = %s\n", s4, s1);
system("Pause");
```

Έξοδος Προγράμματος

```
strcpy(s2, s3) : s2 = KOSTAS GOULIANAS s3 = KOSTAS GOULIANAS
strcpy(s3, s1) : s3 = Kostas Goulianas s1 = Kostas Goulianas
strcpy(s4, s3) : s3 = Kostas s4 = Kostas
strcpy(s4, s3) : s4 = Kostas Goulianas s3 = Kostas Goulianas
Press any key to continue . . .
```

Συνένωση Συμβολοσειρών - `strcat()`

Η συνένωση Συμβολοσειρών γίνεται με την συνάρτηση `strcat()`. Η σύνταξή της είναι :

`strcat(<Συμβολοσειρά-1>, <Συμβολοσειρά-2>)`

Παράδειγμα

```
char s6[] = "Kostas";
printf("S6 = %s\n", s6);
strcat(s6," T. Goulianas");
printf("s6 = %s\n", s6);
system("Pause");
```

Έξοδος Προγράμματος

```
S6 = Kostas
S6 = Kostas T. Goulianas
Press any key to continue . . .
```

8 ΔΟΜΕΣ - structures

- Να γραφεί Αλγόριθμος/Πρόγραμμα το οποίο δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για 2 φοιτητές και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή.

Αλγόριθμος

1. Εισαγωγή Στοιχείων 1ου φοιτητή
2. Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή
3. Εισαγωγή Στοιχείων 2ου φοιτητή
4. Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμα το οποίο δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό
Μαθήματος,τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για 2 φοιτητές και
Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή */

main()
{
    char name[50];
    int aM, kM;
    double bTh, bErg, tB;

    printf("Give name of foititis 1 : ");
    scanf("%s", name);
    printf("Give aM of foititis 1 : ");
    scanf("%d", &aM);
    printf("Give kM of foititis 1 : ");
    scanf("%d", &kM);
    printf("Give bTh of foititis 1 : ");
    scanf("%lf", &bTh);
    printf("Give bErg of foititis 1 : ");
    scanf("%lf", &bErg);
    tB = bTh * 0.7 + bErg * 0.3;
    printf("tB = %lf\n", tB);

    printf("Give name of foititis 2 : ");
    scanf("%s", name);
    printf("Give aM of foititis 2 : ");
    scanf("%d", &aM);
    printf("Give kM of foititis 2 : ");
    scanf("%d", &kM);
    printf("Give bTh of foititis 2 : ");
    scanf("%lf", &bTh);
    printf("Give bErg of foititis 2 : ");
    scanf("%lf", &bErg);
    tB = bTh * 0.7 + bErg * 0.3;
    printf("tB = %lf\n", tB);

    system("Pause");
}
```

Έξοδος Προγράμματος :

```
Give name of foititis 1 : Kostas
Give aM of foititis 1 : 181112
Give kM of foititis 1 : 6101
Give bTh of foititis 1 : 7.0
Give bErg of foititis 1 : 6.5
tB = 6.850000
Give name of foititis 2 : Nikos
Give aM of foititis 2 : 182345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 8.5
Give bErg of foititis 2 : 5
tB = 7.450000
Press any key to continue . . .
```

8.1 Δημιουργία Δομής με Πεδία και Τύπους Δεδομένων

Στο προηγούμενο πρόγραμμα, όλα τα στοιχεία Όνομα, Αριθμός Μητρώου, Κωδικός Μαθήματος, Βαθμός Θεωρίας, Βαθμός Εργαστηρίου και Τελικός Βαθμός αφορούν τους φοιτητές. Θα μπορούσαμε να ορίσουμε ένα **ΝΕΟ** σύνθετο τύπο δεδομένων, μια δομή με το όνομα `foititis`, η οποία να περιέχει τα παραπάνω στοιχεία και μετά να δημιουργούμε μεταβλητές του τύπου της δομής, οι οποίες και έχουν **φυσική υπόσταση**, καταλαμβάνουν μνήμη.

Ορισμός Δομής

```
struct <όνομα_δομής>
{
Δηλώσεις μεταβλητών - πεδίων
...
};
```

Παράδειγμα

```
struct foititis
{
   char name[50]; // Όνομα Φοιτητή
   int aM;        // Αριθμός Μητρώου Φοιτητή
   int kM;        // Κωδικός Μαθήματος
   double bTh;    // Βαθμός Θεωρίας
   double bErg;   // Βαθμός Εργαστηρίου
};
```

Δήλωση Μεταβλητής Τύπου `struct`

Η δήλωση μιας μεταβλητής τύπου `struct` μπορεί να γίνει οπουδήποτε στη `main()` ή σε οποιαδήποτε συνάρτηση, , όπως φαίνεται στο επόμενο παράδειγμα :

```
struct <όνομα_ δομής>  <όνομα_μεταβλητής>;
```

Παράδειγμα 1

```
struct foititis f1, f2, f[50];
```

ή στον ορισμό της δομής, όπως φαίνεται στο επόμενο παράδειγμα :

Παράδειγμα 2

```
struct foititis
{
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
} f1, f2, f[50];
```

Οι μεταβλητές `f1, f2, f[i],i=0:49` θα περιέχουν **τα δικά τους αντίγραφα** των μεταβλητών-πεδίων της δομής `foititis`.

Πρόσβαση στα Πεδία της Μεταβλητής Τύπου `struct`

Η πρόσβαση στα πεδία της μεταβλητής τύπου `struct` γίνεται με τον τελεστή **'.'**. Η γενική της μορφή είναι `<όνομα_μεταβλητής>.<όνομα_πεδίου>`.

Παράδειγμα

Με την εντολή

```
f1.aM = 14013;
```

δίνουμε την τιμή `14013` στο πεδίο `aM` (Αριθμός Μητρώου) της μεταβλητής `f1` τύπου `struct foititis`.

8.1.1 Παράδειγμα Δημιουργίας Δομής με Πεδία

- Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο δηλώνει `2` μεταβλητές τύπου `struct foititis`, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους `2` φοιτητές και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή.

Αλγόριθμος `main()`

1. Εισαγωγή Στοιχείων 1ου φοιτητή
2. Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή
3. Εισαγωγή Στοιχείων 2ου φοιτητή
4. Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο δημιουργεί 2 μεταβλητές τύπου struct foititis, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2 φοιτητές και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή */

    struct foititis
    {
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
    };


   main()
   {
    double tB;
    struct foititis f1, f2;
    // Εισαγωγή Στοιχείων Φοιτητή 1
    printf("Give name of foititis 1 : ");
    scanf("%s", f1.name);
    printf("Give aM of foititis 1 : ");
    scanf("%d", &f1.aM);
    printf("Give kM of foititis 1 : ");
    scanf("%d", &f1.kM);
    printf("Give bTh of foititis 1 : ");
    scanf("%lf", &f1.bTh);
    printf("Give bErg of foititis 1 : ");
    scanf("%lf", &f1.bErg);

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 1
    tB = f1. bTh * 0.7 + f1. bErg * 0.3;
    printf("tB = %lf\n", tB);

    // Εισαγωγή Στοιχείων Φοιτητή 2
    printf("Give name of foititis 2 : ");
    scanf("%s", f2.name);
    printf("Give aM of foititis 2 : ");
    scanf("%d", &f2.aM);
    printf("Give kM of foititis 2 : ");
    scanf("%d", &f2.kM);
    printf("Give bTh of foititis 2 : ");
    scanf("%lf", &f2.bTh);
    printf("Give bErg of foititis 2 : ");
    scanf("%lf", &f2.bErg);

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 2
    tB = f2. bTh * 0.7 + f2. bErg * 0.3;
    printf("tB = %lf\n", tB);
}
```

Έξοδος Προγράμματος :

```
Give name of foititis 1 : ioannou
Give aM of foititis 1 : 191234
Give kM of foititis 1 : 6101
Give bTh of foititis 1 : 7
Give bErg of foititis 1 : 6
tB = 6.700000
Give name of foititis 2 : georgiou
Give aM of foititis 2 : 192345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 6
Give bErg of foititis 2 : 8
tB = 6.600000
Press any key to continue . . .
```

8.2 Συναρτήσεις Επεξεργασίας Δεδομένων της Δομής

Τα δεδομένα μιας δομής, μπορούμε να τα επεξεργαστούμε με συναρτήσεις, οι οποίες βρίσκονται στο ίδιο αρχείο που βρίσκεται η `main()`.

8.2.1 Παράδειγμα Δημιουργίας Δομής η οποία περιέχει Δεδομένα και Επεξεργασία τους με Συναρτήσεις

- Να γραφεί Αλγόριθμος/Πρόγραμμμα, το οποίο δηλώνει `2` μεταβλητές τύπου `struct foititis`, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, το Βαθμό Θεωρίας και το Βαθμό Εργαστηρίου για τους `2` φοιτητές και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή **με την κλήση των** συναρτήσεων **`findTB()` και `returnTB()`** . Η συνάρτηση **`findTB()`** θα **υπολογίζει** και θα **εμφανίζει** τον Τελικό Βαθμό του 1ου φοιτητή, ενώ η συνάρτηση **`returnTB()`** θα **υπολογίζει** και θα **επιστρέφει** στη `main()` τον Τελικό Βαθμό του 2ου φοιτητή, τον οποίο και θα εμφανίζει.

Αλγόριθμος `main()`

1. Εισαγωγή Στοιχείων 1ου φοιτητή
2. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή με κλήση συνάρτησης `findTB()`**
3. Εισαγωγή Στοιχείων 2ου φοιτητή
4. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή με κλήση συνάρτησης `returnTB()`**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο δημιουργεί 2 μεταβλητές τύπου foititis, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2 φοιτητές και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή με την κλήση των συναρτήσεων findTB() και returnTB() */

    struct foititis
    {
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
    };


   // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
   double returnTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    return tB;
   }

   // Συνάρτηση Υπολογισμού - Εμφάνισης Τελικού Βαθμού
   void findTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    printf("call findTB() - tB = %lf\n", tB);
   }

   main()
   {
    double tB;
    struct foititis f1, f2;
    // Εισαγωγή Στοιχείων Φοιτητή 1
    printf("Give name of foititis 1 : ");
    scanf("%s", f1.name);
    printf("Give aM of foititis 1 : ");
    scanf("%d", &f1.aM);
    printf("Give kM of foititis 1 : ");
    scanf("%d", &f1.kM);
    printf("Give bTh of foititis 1 : ");
    scanf("%lf", &f1.bTh);
    printf("Give bErg of foititis 1 : ");
    scanf("%lf", &f1.bErg);

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 1
    findTB(f1);

    // Εισαγωγή Στοιχείων Φοιτητή 2
    printf("Give name of foititis 2 : ");
    scanf("%s", f2.name);
    printf("Give aM of foititis 2 : ");
    scanf("%d", &f2.aM);
    printf("Give kM of foititis 2 : ");
    scanf("%d", &f2.kM);
    printf("Give bTh of foititis 2 : ");
    scanf("%lf", &f2.bTh);
    printf("Give bErg of foititis 2 : ");
    scanf("%lf", &f2.bErg);
```

```
    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 2
    tB = returnTB(f1);
    printf("call returnTB() - tB = %lf\n", tB);
}
```

Έξοδος Προγράμματος :

```
Give name of foititis 1 : ioannou
Give aM of foititis 1 : 191234
Give kM of foititis 1 : 6101
Give bTh of foititis 1 : 7
Give bErg of foititis 1 : 6
call findTB() - tB = 6.700000
Give name of foititis 2 : georgiou
Give aM of foititis 2 : 192345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 6
Give bErg of foititis 2 : 8
call returnTB() - tB = 6.600000
Press any key to continue . . .
```

8.3 Συνάρτηση για Το Γέμισμα των Πεδίων μιας Δομής

Μπορούμε να χρησιμοποιήσουμε μια συνάρτηση, με την οποία δίνουμε τιμές (σταθερές ή παραμετρικά) στα πεδία της μεταβλητής τύπου δομής.

Παράδειγμα

Συνάρτηση που γεμίζει παραμετρικά όλα τα πεδία μιας μεταβλητής δομής τύπου `foititis`.

```
foititis gemismaPedion( char n[], int am, int km, double bth, double berg)
{
    struct foititis f;
    strcpy(f.name, n);
    f.aM = am;
    f.kM = km;
    f.bTh = bth;
    f.bErg = berg;
    return f;
}
```

8.3.1 Παράδειγμα Δημιουργίας Δομής, Γεμίσματος των Πεδίων με τη Χρήση Συνάρτησης και Επεξεργασία των Δεδομένων τους με Συναρτήσεις

- Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο δηλώνει 2 μεταβλητές τύπου `foititis`, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2 φοιτητές με την κλήση της συνάρτησης `gemismaPedion()`, με την οποία περνάει σταθερές τιμές για τα πεδία του 1ου φοιτητή, τις οποίες και εμφανίζει, ενώ για τα πεδία του 2ου φοιτητή διαβάζει τιμές απ' το πληκτρολόγιο σε τοπικές μεταβλητές, τις οποίες μετά περνάει παραμετρικά με την κλήση της συνάρτησης `gemismaPedion()`, τις οποίες και εμφανίζει και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή **με την κλήση των** συναρτήσεων **`findTB()` και `returnTB()`**.

Αλγόριθμος `main()`

1. Εισαγωγή Στοιχείων 1ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
2. Εμφάνιση στοιχείων 1ου Φοιτητή
3. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή με κλήση συνάρτησης `findTB()`**
4. Εισαγωγή Στοιχείων 2ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
5. Εμφάνιση στοιχείων 2ου Φοιτητή
6. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή με κλήση συνάρτησης `returnTB()`**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο δημιουργεί 2 μεαβλητές τύπου foititis, με την κλήση
της συνάρτησης gemismaPedion()  δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου,
τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2
φοιτητές σταθερές και παραμετρικά, τις οποίες και εμφανίζει και Υπολογίζει
και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή με την κλήση των συναρτήσεων
findTB() και returnTB() */

    struct foititis
    {
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
    };

  // Συνάρτηση Κατασκευής - Δομητής
  foititis gemismaPedion( char n[], int am, int km, double bth, double berg)
  {
    struct foititis f;
    strcpy(f.name, n);
    f.aM = am;
    f.kM = km;
    f.bTh = bth;
    f.bErg = berg;
    return f;
  }

   // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
   double returnTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    return tB;
   }

   // Συνάρτηση Υπολογισμού - Εμφάνισης Τελικού Βαθμού
   void findTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    printf("call findTB() - tB = %lf\n", tB);
   }
```

```
  main()
   {
    char name[50];
    int aM, kM;
    double bTh, bErg, tB;
    struct foititis f1, f2;
    // Εισαγωγή Στοιχείων Φοιτητή 1
    f1 = gemismaPedion("ioannou",191234,6101,7.5,6.0);

    // Εμφάνιση Στοιχείων Φοιτητή 1
    printf("Foititis f1 : \n");
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n", f1.name,
f1.aM, f1.kM, f1.bTh, f1.bErg);

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 1
    findTB(f1);

    // Εισαγωγή Στοιχείων Φοιτητή 2
    printf("Give name of foititis 2 : ");
    scanf("%s", name);
    printf("Give aM of foititis 2 : ");
    scanf("%d", &aM);
    printf("Give kM of foititis 2 : ");
    scanf("%d", &kM);
    printf("Give bTh of foititis 2 : ");
    scanf("%lf", &bTh);
    printf("Give bErg of foititis 2 : ");
    scanf("%lf", &bErg);
    f2 = gemismaPedion(name,aM,kM,bTh,bErg);

    // Εμφάνιση Στοιχείων Φοιτητή 2
    printf("Foititis f2 : \n");
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n", f1.name,
f1.aM, f1.kM, f1.bTh, f1.bErg);

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 2
    tB = returnTB(f1);
    printf("call returnTB() - tB = %lf\n", tB);
}
```

Έξοδος Προγράμματος :

```
foititis f1 :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
call findTB() - tB = 7.0500000
Give name of foititis 2 : georgiou
Give aM of foititis 2 : 192345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 7
Give bErg of foititis 2 : 8
foititis f2 :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
call returnTB() - tB = 7.300000
Press any key to continue . . .
```

8.4 Εμφάνιση των Δεδομένων Δομής με τη Συνάρτηση `emfanishPedion()`

Η προηγούμενη εντολή `printf()` με την οποία εμφανίσαμε τις τιμές των πεδίων των μεταβλητών `f1` και `f2` θα μπορούσε να ανήκει σε μια συνάρτηση `emfanishPedion()`, στην κλήση της οποίας θα περνάει σαν παράμετρος η αντίστοιχη μεταβλητή τύπου `foititis`.

Παράδειγμα

```
void emfanishPedion(foititis f)
{
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n",
f.name, f.aM, f.kM, f.bTh, f.bErg);
}
```

8.4.1 Παράδειγμα Δημιουργίας Δομής, Γεμίσματος των Πεδίων με τη Χρήση Συνάρτησης και Επεξεργασία – Εμφάνιση των Δεδομένων τους με Συναρτήσεις

- Να γραφεί Αλγόριθμος/Πρόγραμμμα, το οποίο δηλώνει 2 μεταβλητές τύπου `foititis`, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2 φοιτητές με την κλήση της συνάρτησης `gemismaPedion()`, με την οποία περνάει σταθερές τιμές για τα πεδία του 1ου φοιτητή, τις οποίες και εμφανίζει με την κλήση της συνάρτησης `gemismaPedion()`,, ενώ για τα πεδία του 2ου φοιτητή διαβάζει τιμές απ' το πληκτρολόγιο σε τοπικές μεταβλητές, τις οποίες μετά περνάει παραμετρικά με την κλήση της συνάρτησης `emfanihPedion()`, τις οποίες και εμφανίζει με την κλήση της συνάρτησης `emfanihPedion()`, και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή **με την κλήση των** συναρτήσεων **`findTB()` και `returnTB()`**.

Αλγόριθμος `main()`

1. Εισαγωγή Στοιχείων 1ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
2. Εμφάνιση στοιχείων 1ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
3. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή με κλήση συνάρτησης `findTB()`**
4. Εισαγωγή Στοιχείων 2ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
5. Εμφάνιση στοιχείων 2ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
6. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή με κλήση συνάρτησης `returnTB()`**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο δημιουργεί 2 μεαβλητές τύπου foititis, με την κλήση
της συνάρτησης gemismaPedion()  δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου,
τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2
φοιτητές σταθερές και παραμετρικά, τις οποίες και εμφανίζει με την κλήση της
συνάρτησης emfanishPedion()και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του
κάθε φοιτητή με την κλήση των συναρτήσεων findTB() και returnTB() */


    struct foititis
    {
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
    };


  // Συνάρτηση Κατασκευής - Δομητής
  foititis gemismaPedion( char n[], int am, int km, double bth, double berg)
  {
    struct foititis f;
    strcpy(f.name, n);
    f.aM = am;
    f.kM = km;
    f.bTh = bth;
    f.bErg = berg;
    return f;
  }


 // Συνάρτηση Εμφάνισης τιμών πεδίων
 void emfanishPedion(foititis f)
 {
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n", f.name,
f.aM, f.kM, f.bTh, f.bErg);
 }


   // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
   double returnTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    return tB;
   }


   // Συνάρτηση Υπολογισμού - Εμφάνισης Τελικού Βαθμού
   void findTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    printf("call findTB() - tB = %lf\n", tB);
   }
```

```
main()
 {
  char name[50];
  int aM, kM;
  double bTh, bErg, tB;
  struct foititis f1, f2;
  // Εισαγωγή Στοιχείων Φοιτητή 1
  f1 = gemismaPedion("ioannou",191234,6101,7.5,6.0);

  // Εμφάνιση Στοιχείων Φοιτητή 1
  printf("Foititis f1 : \n");
  emfanishPedion( f1 );

  // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 1
  findTB(f1);

  // Εισαγωγή Στοιχείων Φοιτητή 2
  printf("Give name of foititis 2 : ");
  scanf("%s", name);
  printf("Give aM of foititis 2 : ");
  scanf("%d", &aM);
  printf("Give kM of foititis 2 : ");
  scanf("%d", &kM);
  printf("Give bTh of foititis 2 : ");
  scanf("%lf", &bTh);
  printf("Give bErg of foititis 2 : ");
  scanf("%lf", &bErg);
  f2 = gemismaPedion(name,aM,kM,bTh,bErg);

  // Εμφάνιση Στοιχείων Φοιτητή 2
  printf("Foititis f2 : \n");
  emfanishPedion( f2 );

  // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 2
  tB = returnTB(f1);
  printf("call returnTB() - tB = %lf\n", tB);
}
```

Έξοδος Προγράμματος :

```
foititis f1 :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
call findTB() - tB = 7.0500000
Give name of foititis 2 : georgiou
Give aM of foititis 2 : 192345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 7
Give bErg of foititis 2 : 8
foititis f2 :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
call returnTB() - tB = 7.300000
Press any key to continue . . .
```

8.5 Συναρτήσεις Επεξεργασίας Δεδομένων Πολλών Μεταβλητών τύπου Δομής

Οι προηγούμενες συναρτήσεις `findTB()`, `returnTB()`, `gemismaPedion()` και `emfanishPedion()` επεξεργάζονταν δεδομένα μιας μεταβλητής τύπου δομής. Μπορούμε όμως να έχουμε και συναρτήσεις που να επεξεργάζονται δεδομένα πολλών διαφορετικών μεταβλητών τύπου δομής. Τέτοιες συναρτήσεις θα μπορούσαν να περιέχουν ονόματα μεταβλητών τύπου δομής ή ονόματα πεδίων μεταβλητών τύπου δομής, όπως στο επόμενο παράδειγμα.

8.5.1 Παράδειγμα Δημιουργίας 2 μεταβλητών τύπου Δομής και Επεξεργασία των Δεδομένων τους με Συναρτήσεις

- Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο δηλώνει `2` μεταβλητές `f1` και `f2` τύπου `foititis`, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους `2` φοιτητές με την κλήση της συνάρτησης `gemismaPedion()`, με την οποία περνάει σταθερές τιμές για τα πεδία του 1ου φοιτητή, τις οποίες και εμφανίζει με την κλήση της συνάρτησης `emfanishPedion()`, ενώ για τα πεδία του 2ου φοιτητή διαβάζει τιμές απ' το πληκτρολόγιο σε τοπικές μεταβλητές, τις οποίες μετά περνάει παραμετρικά με την κλήση της συνάρτησης `gemismaPedion()`, τις οποίες και εμφανίζει με την κλήση της συνάρτησης `emfanishPedion()` και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή με την κλήση των συναρτήσεων `findTB()` και `returnTB()`. Στη συνέχεια καλεί τη συνάρτηση `findMaxTB()` με την οποία βρίσκει το μεγαλύτερο Τελικό βαθμό των φοιτητών `f1` και `f2` και ανταλλάσσει τα περιεχόμενα των μεταβλητών `f1` και `f2` στη `main()` και μετά ανταλλάσσει ξανά τα περιεχόμενα των μεταβλητών `f1` και `f2` με την κλήση της συνάρτησης `swapF1F2()`.

Αλγόριθμος `main()`

1. Εισαγωγή Στοιχείων 1ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
2. Εμφάνιση στοιχείων 1ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
3. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή με κλήση συνάρτησης `findTB()`**
4. Εισαγωγή Στοιχείων 2ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
5. Εμφάνιση στοιχείων 2ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
6. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή με κλήση συνάρτησης `returnTB()`**
7. **Υπολογισμός - Εμφάνιση Μέγιστου Τελικού Βαθμού 1ου - 2ου φοιτητή με κλήση συνάρτησης `findMaxTB()`**
8. **Ανταλλαγή περιεχομένων 1ου - 2ου φοιτητή**
9. Εμφάνιση στοιχείων 1ου – 2ου Φοιτητή – κλήση συνάρτησης `emfanishPedion()`
10. **Ανταλλαγή περιεχομένων 1ου - 2ου φοιτητή** – κλήση συνάρτησης `swapF1F2()`
11. Εμφάνιση στοιχείων 1ου – 2ου Φοιτητή – κλήση συνάρτησης `emfanishPedion()`

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο δημιουργεί 2 μεαβλητές f1, f2 τύπου foititis, με την
κλήση της συνάρτησης gemismaPedion()  δίνει τιμές στο Όνομα, τον Αριθμό
Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου
για τους 2 φοιτητές σταθερές και παραμετρικά, τις οποίες και εμφανίζει με την
κλήση της συνάρτησης emfanishPedion()και Υπολογίζει και Εμφανίζει τον Τελικό
Βαθμό του κάθε φοιτητή με την κλήση των συναρτήσεων findTB() και returnTB().
Στη συνέχεια καλεί τη συνάρτηση findMaxTB() με την οποία βρίσκει το
μεγαλύτερο Τελικό βαθμό των φοιτητών f1 και f2 και ανταλλάσσει τα περιεχόμενα
των μεταβλητών f1 και f2 στη main() και μετά ανταλλάσσει ξανά τα περιεχόμενα
των μεταβλητών f1 και f2  με την κλήση της συνάρτησης swapF1F2(). */


    struct foititis
    {
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
    };


  // Συνάρτηση Κατασκευής - Δομητής
  foititis gemismaPedion( char n[], int am, int km, double bth, double berg)
  {
    struct foititis f;
    strcpy(f.name, n);
    f.aM = am;
    f.kM = km;
    f.bTh = bth;
    f.bErg = berg;
    return f;
  }


 // Συνάρτηση Εμφάνισης τιμών πεδίων
 void emfanishPedion(foititis f)
 {
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n", f.name,
f.aM, f.kM, f.bTh, f.bErg);
 }


   // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
   double returnTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    return tB;
   }


   // Συνάρτηση Υπολογισμού - Εμφάνισης Τελικού Βαθμού
   void findTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    printf("call findTB() - tB = %lf\n", tB);
   }
```

```
void findMaxTB(foititis f1, foititis f2)
{
    if ( returnTB(f1) >  returnTB(f2))
      printf("TB f1 = %lf > TB f2 = %lf\n", returnTB(f1), returnTB(f2));
    else
      printf("TB f2 = %lf > TB f1 = %lf\n", returnTB(f2), returnTB(f1));
}


void swapF1F2(foititis f1, foititis f2)
{
    struct foititis temp = f1;
    f1 = f2;
    f2 = temp;
}



  main()
   {
    char name[50];
    int aM, kM;
    double bTh, bErg, tB;
    struct foititis f1, f2;
    // Εισαγωγή Στοιχείων Φοιτητή 1
    f1 = gemismaPedion("ioannou",191234,6101,7.5,6.0);

    // Εμφάνιση Στοιχείων Φοιτητή 1
    printf("Foititis f1 : \n");
    emfanishPedion( f1 );

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 1
    findTB(f1);

    // Εισαγωγή Στοιχείων Φοιτητή 2
    printf("Give name of foititis 2 : ");
    scanf("%s", name);
    printf("Give aM of foititis 2 : ");
    scanf("%d", &aM);
    printf("Give kM of foititis 2 : ");
    scanf("%d", &kM);
    printf("Give bTh of foititis 2 : ");
    scanf("%lf", &bTh);
    printf("Give bErg of foititis 2 : ");
    scanf("%lf", &bErg);
    f2 = gemismaPedion(name,aM,kM,bTh,bErg);

    // Εμφάνιση Στοιχείων Φοιτητή 2
    printf("Foititis f2 : \n");
    emfanishPedion( f2 );

    // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 2
    tB = returnTB(f1);
    printf("call returnTB() - tB = %lf\n", tB);

    // Υπολογισμός - Εμφάνιση Μέγιστου Τελικού Βαθμού Φοιτητών f1, f2
    findMaxTB(f1, f2);

    // Ανταλλαγή περιεχομένων  f1, f2
    struct foititis temp = f1;
    f1 = f2;
    f2 = temp;

    // Εμφάνιση Στοιχείων Φοιτητή 1
    printf("Foititis f1 after swap f1 f2:\n");
    emfanishPedion( f1 );
```

```
      // Εμφάνιση Στοιχείων Φοιτητή 1
      printf("Foititis f2 after swap f1 f2 :\n");
      emfanishPedion( f2 );

      // Ανταλλαγή περιεχομένων  f1, f2 – κλήση swapF1F2
      swapF1F2(f1,f2);

     // Εμφάνιση Στοιχείων Φοιτητή 1
     printf("Foititis f1 after call swapF1F2() : \n");
     emfanishPedion( f1 );

     // Εμφάνιση Στοιχείων Φοιτητή 1
     printf("Foititis f2 after call swapF1F2() : \n");
     emfanishPedion( f2 );

  }
```

Έξοδος Προγράμματος :

```
foititis f1 :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
call findTB() - tB = 7.0500000
Give name of foititis 2 : georgiou
Give aM of foititis 2 : 192345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 7
Give bErg of foititis 2 : 8
foititis f2 :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
call returnTB() - tB = 7.300000
TB f2 = 7.300000 > TB f1 = 7.050000
Foititis f1 after swap f1 f2 :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
Foititis f2 after swap f1 f2 :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
Foititis f1 after call swapF1F2() :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
Foititis f2 after call swapF1F2() :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
Press any key to continue . . .
```

8.6 Πίνακες Μεταβλητών Τύπου Δομής

Στις περισσότερες εφαρμογές απαιτείται η δημιουργία ενός πίνακα μεταβλητών τύπου δομής, όταν οι μεταβλητές που θα χρειαστούμε είναι πολλές. Για τον σκοπό αυτό θα χρειαστεί να δηλώσουμε **έναν** πίνακα και **μετά** να γεμίσουμε τις θέσεις του πίνακα.

Παράδειγμα

```
struct foititis
{
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
        double tB;
};

struct foititis f[50]; // Δημιουργία Πίνακα 50 φοιτητών
```

8.6.1 Παράδειγμα Δημιουργίας Πίνακα Μεταβλητών Τύπου Δομής – Επεξεργασία Δεδομένων του Πίνακα με χρήση Συναρτήσεων

- Να γραφεί Αλγόριθμος/Πρόγραμμα διαβάζει την τιμή μιας μεταβλητής `n` για το μέγεθος του πίνακα `n` μεταβλητών τύπου `foititis`. **Η δομή `foititis` περιέχει και το πεδίο `tB` για τον Τελικό Βαθμό**. Όλα τα πεδία των φοιτητών του πίνακα, εκτός του Τελικού Βαθμού γεμίζουν με την κλήση της συνάρτησης `gemismaPedion()`. Με την κλήση της συνάρτησης `setTBAll()`, η οποία καλεί τη συνάρτηση `returnTB()`, γεμίζουν όλα τα πεδία των `n` φοιτητών του πίνακα, τα οποία και εμφανίζει με την κλήση της συνάρτησης `emfanishPedion()`. Μετά δημιουργεί έναν τυχαίο ακέραιο αριθμό `index` μεταξύ `0` και `n-2` και καλεί τη συνάρτηση `swapFiFi1()`, με την οποία ανταλλάσσει τα περιεχόμενα των `f[index]`, `f[index+1]` και εμφανίζει ξανά τα στοιχεία όλων των φοιτητών. Στη συνέχεια καλεί τη συνάρτηση `returnThesiMaxTB()` με την οποία βρίσκει το φοιτητή που έχει το μεγαλύτερο Τελικό βαθμό και εμφανίζει τα στοιχεία του.

Αλγόριθμος `main()`

1. Διάβασμα τιμής `<= 50` στη μεταβλητή `n`.
2. Εισαγωγή Στοιχείων `n` φοιτητών – κλήση συνάρτησης `gemismaPedion()`
3. **Γέμισμα πεδίου** `tB` **των** `n` φοιτητών - **κλήση συνάρτησης** `setTBAll()`
4. Εμφάνιση στοιχείων `n` φοιτητών – κλήση συνάρτησης `emfanihPedion()`
5. Δημιουργία τυχαίου ακεραίου `index` στο [`0, n-2`].
6. **Ανταλλαγή περιεχομένων** `f[index], f[index+1]` – κλήση συνάρτησης `swapFiFi1()`
7. Εμφάνιση στοιχείων `n` φοιτητών – κλήση συνάρτησης `emfanihPedion()`
8. **Εύρεση φοιτητή με Μέγιστο Τελικό Βαθμό - κλήση συνάρτησης** `returnThesiMaxTB()`
9. **Εμφάνιση** στοιχείων **φοιτητή με Μέγιστο Τελικό Βαθμό.**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμα το οποίο διαβάζει την τιμή μιας μεταβλητής n για το μέγεθος
του πίνακα n μεταβλητών τύπου foititis. Η δομή foititis περιέχει και το πεδίο
tB για τον Τελικό Βαθμό. Όλα τα πεδία των φοιτητών του πίνακα, εκτός του
Τελικού Βαθμού γεμίζουν με την κλήση της συνάρτησης gemismaPedion(). Με την
κλήση της συνάρτησης setTBAll(), η οποία καλεί τη συνάρτηση returnTB(),
γεμίζουν όλα τα πεδία των n  φοιτητών του πίνακα,  τα οποία και εμφανίζει με
την κλήση της συνάρτησης emfanishPedion(). Μετά δημιουργεί έναν τυχαίο
ακέραιο αριθμό index μεταξύ 0 και n-2 και καλεί τη συνάρτηση swapFiFi1(), με
την οποία ανταλλάσσει τα περιεχόμενα των f[index], f[index+1] και εμφανίζει
ξανά τα στοιχεία όλων των φοιτητών.  Στη συνέχεια καλεί τη συνάρτηση
returnThesiMaxTB() με την οποία βρίσκει το φοιτητή που έχει το μεγαλύτερο
Τελικό βαθμό και εμφανίζει τα στοιχεία του*/


    struct foititis
    {
        char name[50];
        int aM;
        int kM;
        double bTh;
        double bErg;
       double tB;
    };


  // Συνάρτηση Κατασκευής - Δομητής
  foititis gemismaPedion( char n[], int am, int km, double bth, double berg)
  {
    struct foititis f;
    strcpy(f.name, n);
    f.aM = am;
    f.kM = km;
    f.bTh = bth;
    f.bErg = berg;
    return f;
  }


 // Συνάρτηση Εμφάνισης τιμών πεδίων
 void emfanishPedion(foititis f)
 {
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n", f.name,
f.aM, f.kM, f.bTh, f.bErg);
 }

   // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
   double returnTB(foititis f)
   {
    double tB = f.bTh * 0.7 + f.bErg * 0.3;
    return tB;
   }

   // Συνάρτηση Υπολογισμού - Τελικού Βαθμού Όλων
   void setTBAll( foititis f[], int n)
   {
    int i;
    for (  i=0;i<=n-1;i++)
      f[i].tB =  returnTB(f[i]);
   }
```

```
   // Ανταλλαγή περιεχομένων  f[i], f[j] – κλήση swapFiFi1
   void swapFiFj(foititis f[], int i, int j)
   {
    struct foititis temp = f[i];
    f[i] = f[j];
    f[j] = temp;
   }

   // Υπολογισμός Θέσης Μέγιστου Τελικού Βαθμού Φοιτητών f[i], i=0:n-1

  int returnThesiMaxTB(foititis f[], int n)
  {
   double maxTB = f[0].tB;
   int i, thesiTB = 0;
   for ( i=1;i<=n-1;i++)
     if ( f[i].tB >  maxTB)
      {
       maxTB = f[i].tB;
       thesiTB = i;
      }
   return thesiTB;
}


 main()
  {
   char name[50];
   int aM, kM;
   double bTh, bErg, tB;
   struct foititis f[50];

   // Διάβασμα n
   do
   {
      printf("Give n <= 50 : ");
      scanf("%d", &n);
   }
   while ( n > 50 );


   // Εισαγωγή Στοιχείων n Φοιτητών
  for ( i=0;i<=n-1;i++)
  {
   printf("Give name of foititis f[%d] : ", i);
   scanf("%s", name);
   printf("Give aM of foititis f[%d] : ", i);
   scanf("%d", &aM);
   printf("Give kM of foititis f[%d] : ", i);
   scanf("%d", &kM);
   printf("Give bTh of foititis f[%d] : ", i);
   scanf("%lf", &bTh);
   printf("Give bErg of foititis f[%d] : ", i);
   scanf("%lf", &bErg);
   f[i] = gemismaPedion(name,aM,kM,bTh,bErg);
  }

   // Υπολογισμός Τελικού Βαθμού n Φοιτητών
   setTBAll(f, n);

   // Εμφάνιση Στοιχείων n Φοιτητών
   for ( i=0;i<=n-1;i++)
    {
       printf("Foititis f[%d] : \n", i);
       emfanishPedion( f[i] );
   }
```

```
      // Δημιουργία τυχαίας τιμής στο index
       int index = rand() % (n-2);

      // Ανταλλαγή περιεχομένων  f1, f2 – κλήση swapFiFi1()
      swapFiFj( f,  index,  index+1);

      printf("pinakas after Swap f[%d] <--> f[%d]\n", index,  index+1);

      // Εμφάνιση Στοιχείων n Φοιτητών
      for ( i=0;i<=n-1;i++)
       {
          printf("Foititis f[%d] : \n", i);
          emfanishPedion( f[i] );
      }

      // Υπολογισμός Θέσης  Μέγιστου Τελικού Βαθμού Φοιτητών
      thesiTB = returnThesiMaxTB(f, n);

     // Εμφάνιση Στοιχείων Φοιτητή με Μέγιστο Τελικό Βαθμό
     printf("Pedia foitith me maxTB :\n");
     emfanishPedion( f[thesiTB] );

     system("Pause");
  }
```

Έξοδος Προγράμματος :

```
Give name of foititis f[0] : ioannou
Give aM of foititis f[0] : 191234
Give kM of foititis f[0] : 6101
Give bTh of foititis f[0] : 7.5
Give bErg of foititis f[0] : 6.0
Give name of foititis f[1] : georgiou
Give aM of foititis f[1]  : 192345
Give kM of foititis f[1]  : 6101
Give bTh of foititis f[1]  : 7
Give bErg of foititis f[1]  : 8
Give name of foititis f[2] : aleksiou
Give aM of foititis f[2]  : 193456
Give kM of foititis f[2]  : 6101
Give bTh of foititis f[2]  : 9
Give bErg of foititis f[2]  : 8
Foititis f[0] :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
tB : 7.050000
Foititis f[1] :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
tB : 7.300000
Foititis f[2] :
name : aleksiou
aM : 193456
kM : 6101
bTh : 9
bErg : 8
tB : 8.700000
```

```
pinakas after Swap f0] <--> f[1]
Foititis f[0] :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
tB : 7.300000
Foititis f[1] :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
tB : 7.050000
Foititis f[2] :
name : aleksiou
aM : 193456
kM : 6101
bTh : 9
bErg : 8
tB : 8.700000
Pedia foitith me maxTB :
name : aleksiou
aM : 193456
kM : 6101
bTh : 9
bErg : 8
tB : 8.700000
Press any key to continue . . .
```

9 ΔΕΙΚΤΕΣ - Pointers

Ένας δείκτης-pointer είναι μια μεταβλητή που **αποθηκεύει** τη **διεύθυνση** στη μνήμη μιας άλλης μεταβλητής. Έτσι, αν γράψουμε

```
px = &x;
```

ο δείκτης `px` αποθηκεύει τη **διεύθυνση** στη μνήμη της μεταβλητής `x`. Αν θέλουμε να επεξεργαστούμε τα περιεχόμενα της μεταβλητής `x`, μπορούμε να χρησιμοποιήσουμε το δείκτη `px`, αντί της μεταβλητής `x`. Ο τελεστής `'*'` πριν από το όνομα κάποιου δείκτη, επιστρέφει τα περιεχόμενα της μεταβλητής, η διεύθυνση της οποίας είναι αποθηκευμένη στο δείκτη. Π.χ. η εντολή :

```
z = *px;
```

ή

```
z = *(&x);
```

αποθηκεύει στη μεταβλητή `z`, τα περιεχόμενα της μεταβλητής `x`, η διεύθυνση της οποίας είναι αποθηκευμένη στο δείκτη `px`.

9.1 Δηλώσεις Μεταβλητών Δεικτών

Οι δηλώσεις των μεταβλητών – δεικτών γίνονται όπως οι άλλες δηλώσεις μεταβλητών, αλλά πριν το όνομα της μεταβλητής βάζουμε τον τελεστή '*'. Π.χ. με την εντολή

```
int *px;
```

ή

```
int* px;
```

δηλώνουμε τη μεταβλητή `px` που θα αποθηκεύσει διευθύνσεις ακεραίων μεταβλητών.

Το λειτουργικό σύστημα με τη δήλωση κάποιων μεταβλητών σε κάποιο πρόγραμμα μας δίνει μια περιοχή μνήμης από τη μνήμη σωρού (`stack memory`), η οποία είναι περιορισμένου μεγέθους και χρησιμοποιείται για τις μεταβλητές της `main()` και των συναρτήσεων που καλεί. Μπορούμε να δούμε τα περιεχόμενα αυτών των θέσεων εμφανίζοντας τα περιεχόμενα των δεικτών. Δεν μπορούμε να κάνουμε πράξεις με δείκτες, εκτός από το να εκχωρήσουμε την τιμή ενός δείκτη σε μια μεταβλητή ή να προσθέσουμε ή να αφαιρέσουμε κάποιον αριθμό σε κάποιον δείκτη. Η πρόσθεση ή η αφαίρεση του 1 π.χ. σε κάποιον δείκτη έχει σαν αποτέλεσμα να προστεθεί ή να αφαιρεθεί όχι ο αριθμός 1, αλλά τόσα `bytes`, όσα `bytes` καταλαμβάνει ο τύπος της μεταβλητής, στην οποία δείχνει ο δείκτης.

Με τις παρακάτω εντολές :

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    char ch = '$';
    short s = 1;
    int i = 5;
    float f = 1.5;
    double d = 2.5;
```

δηλώνουμε μεταβλητές των πιο γνωστών τύπων, στις οποίες δίνουμε και κάποια τιμή.

Με τις παρακάτω εντολές :

```
    char *pch;
    short *ps;
    int *pi;
    float *pf;
    double *pd;
```

δηλώνουμε και τους αντίστοιχους δείκτες στις παραπάνω μεταβλητές.

Με τις παρακάτω εντολές :

```
    pch = &ch;
    ps = &s;
    pi = &i;
    pf = &f;
    pd = &d;
```

αποθηκεύουμε τις διευθύνσεις των μεταβλητών στους αντίστοιχους δείκτες.

Με τις παρακάτω εντολές :

```
printf("*pch = %c *ps = %hd  *pi = %d *pf = %f *pd = %lf\n", *pch, *ps, *pi,  *pf, *pd);
printf("pch = %d ps = %d pi = %d pf = %d pd = %d\n", pch, ps, pi, pf, pd);
```

εμφανίζουμε τα περιεχόμενα των μεταβλητών, στις οποίες δείχνουν οι δείκτες και τις διευθύνσεις αυτών των μεταβλητών. Έτσι, το πρόγραμμά μας θα εμφανίσει τα παρακάτω :

```
*pch = $ *ps = 1  *pi = 5 *pf = 1.500000 *pd = 2.500000
pch = 2293575 ps = 2293572 pi = 2293568 pf = 2293564 pd = 2293552
```

Μπορούμε να δούμε ότι οι διευθύνσεις είναι στην ίδια περιοχή μνήμης, η `pd` τύπου `double` καταλαμβάνει `8 bytes` από το `2293552` μέχρι το `2293559`, η `pf` τύπου `float` καταλαμβάνει `4 bytes` από το `2293564` μέχρι το `2293567`, η `pi` τύπου `int` καταλαμβάνει `4 bytes` από το `2293568` μέχρι το `2293571`, η `ps` τύπου `short` καταλαμβάνει `2 bytes` από το `2293572` μέχρι το `2293573`, η `pch` τύπου `char` καταλαμβάνει `1 byte` το `2293575`.

Με τις παρακάτω εντολές :

```
pch+=1;
ps+=1;
pi+=1;
pf+=1;
pd+=1;
```

Προσθέτουμε τόσα bytes όσα και ο τύπος της κάθε μεταβλητής. Με τις εντολές

```
printf("pch = %d ps = %d pi = %d pf = %d pd = %d\n", pch, ps, pi, pf, pd);
printf("*pch = %c *ps = %hd  *pi = %d *pf = %f *pd = %lf\n", *pch, *ps, *pi,  *pf, *pd)
```

θα δούμε τις αλλαγές στις διευθύνσεις και τα περιεχόμενα των δεικτών, οι οποίοι δεν δείχνουν πλέον στις αρχικές μεταβλητές.

```
pch = 2293576 ps = 2293574 pi = 2293572 pf = 2293568 pd = 2293560
```

Μπορούμε να δούμε ότι οι νέες διευθύνσεις είναι στην ίδια περιοχή μνήμης, η pd τύπου double καταλαμβάνει 8 bytes από το 2293560 μέχρι το 2293567, (προστέθηκαν 8 bytes), η pf τύπου float καταλαμβάνει 4 bytes από το 2293568 μέχρι το 2293571, (προστέθηκαν 4 bytes), η pi τύπου int καταλαμβάνει 4 bytes από το 2293572 μέχρι το 2293575, (προστέθηκαν 4 bytes), η ps τύπου short καταλαμβάνει 2 bytes από το 2293574 μέχρι το 2293575, (προστέθηκαν 2 bytes), η pch τύπου char καταλαμβάνει 1 byte το 2293576.

Τα περιεχόμενα των νέων δεικτών είναι :

```
*pch = x *ps = 9257  *pi = 606666753 *pf = 0.000000 *pd = 0.125000
```

Αν θέλουμε οι δείκτες να δείχνουν ξανά στις αρχικές μεταβλητές, θα πρέπει να αποθηκεύσουμε ξανά τις αρχικές διευθύνσεις των μεταβλητών με τις παρακάτω εντολές :

```
pch = &ch;
ps = &s;
pi = &i;
pf = &f;
pd = &d;
```

9.2 Πέρασμα Παραμέτρων σε Συναρτήσεις με Αναφορά (Διεύθυνση)

Στην κλήση των συναρτήσεων οι παράμετροι περνάνε με τιμή, δηλαδή δημιουργείται ένα αντίγραφο της πραγματικής παραμέτρου, η οποία χρησιμοποιείται στην κλήση της συνάρτησης, οπότε οποιαδήποτε αλλαγή κι αν γίνει μέσα στην συνάρτηση στην τυπική παράμετρο, δεν αποθηκεύεται στην πραγματική παράμετρο-μεταβλητή, αφού δεν περνάει η διεύθυνσή της.

Αν γα παράδειγμα σε ένα πρόγραμμα χρησιμοποιήσουμε μια συνάρτηση `zero_x(int x)` με την οποία μηδενίζουμε την τιμή μιας ακέραιας μεταβλητής `x`, θα μηδενιστεί η τιμή της μέσα στη συνάρτηση, αλλά η αλλαγή δεν θα περάσει στην πραγματική παράμετρο :

```
#include <stdio.h>
#include <stdlib.h>

void zero_x(int x)
{
    x = 0;
    printf("In zero_x x = %d\n", x);
}
main()
{
     int x = 15;
     printf("In main before  zero_x(x) x = %d\n", x);
     zero_x(x);
     printf("In main after  zero_x(x) x = %d\n", x);
     system("Pause");
}
```

Έξοδος Προγράμματος :

```
In main before  zero_x(x) x = 15
In zero_x x = 0
In main after  zero_x(x) x = 15
Press any key to continue . . .
```

- Επειδή οι παράμετροι περνάνε στη συνάρτηση **με τιμή** (`by value`), αν θέλουμε να περάσουν **με αναφορά** (`by reference`), δηλαδή αν οι μεταβλητές αλλάξουν τιμή στη συνάρτηση, οι νέες τιμές να περάσουν στο κυρίως πρόγραμμα, θα πρέπει αντί της μεταβλητής, να **περάσουμε** στη συνάρτηση τη **διεύθυνση** της μεταβλητής που γίνεται με το σύμβολο '**&**'. (Το σύμβολο '**&**' πριν από κάποια μεταβλητή επιστρέφει τη διεύθυνση που βρίσκεται αποθηκευμένη η μεταβλητή. Π.χ. το `&x` είναι η διεύθυνση της μεταβλητής `x`). Αν λοιπόν θέλουμε να αλλάξουμε τα περιεχόμενα της μεταβλητής `x`, θα πρέπει να χρησιμοποιήσουμε τον τελεστή '*' με ένα δείκτη στις τυπικές παραμέτρους, μέσω του οποίου θα περάσει η διεύθυνση της πραγματικής παραμέτρου. Έτσι το προηγούμενο πρόγραμμα θα γίνει :

```
#include <stdio.h>
#include <stdlib.h>

void zero_px(int *px)
{
    *px = 0;
    printf("In zero_px *px = %d\n", *px);
}
```

```
main()
{
      int x = 15;
      printf("In main before  zero_px(&x) x = %d\n", x);
      zero_px(&x);
      printf("In main after  zero_px(&x) x = %d\n", x);
      system("Pause");
}
```

Έξοδος Προγράμματος :

```
In main before  zero_px(&x) x = 15
In zero_px x = 0
In main after  zero_px(&x) x = 0
Press any key to continue . . .
```

Με τον ίδιο τρόπο μπορούμε να ανταλλάξουμε και τις τιμές 2 ακέραιων μεταβλητών προσαρμόζοντας ανάλογα και την συνάρτηση `swap(a,b)` :

```
#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int *b)
{
     int temp;
     temp = *a;
     *a = *b;
     *b = temp;
     printf("In swap *a = %d *b = %d\n", *a, *b);
}

main()
{
     int a = 5, b = 6;
     printf("In main before swap(&a, &b) a = %d b = %d\n", a, b);
     swap(&a, &b);
     printf("In main after swap(&a, &b) a = %d b = %d\n", a, b);
     system("Pause");
}
```

Έξοδος Προγράμματος :

```
In main before  swap(&a, &b) a = 5 b = 6
In swap *a = 6 *b = 5
In main after  swap(&a, &b) a = 6 b = 5
Press any key to continue . . .
```

9.3 Δηλώσεις Μεταβλητών Δεικτών Πινάκων

Με τη δήλωση ενός πίνακα, το όνομά του είναι δείκτης στο πρώτο στοιχείο του πίνακα.

Παράδειγμα

Με την παρακάτω δήλωση του πίνακα `p` :

```
int p[] = {2,4,6,8,10};
for (i=0;i<=sizeof(p)/sizeof(p[0])-1;i++)
     printf("p[%d] = %d\n", i, p[i]);
```

θα εμφανιστούν τα παρακάτω :

```
p[0] = 2
p[1] = 4
p[2] = 6
p[3] = 8
p[4] = 10
```

Η συνάρτηση `sizeof(p)` επιστρέφει το πλήθος των `bytes` όλων των στοιχείων του πίνακα `p`, ενώ η συνάρτηση `sizeof(p[0])` επιστρέφει το πλήθος των `bytes` ενός στοιχείου του πίνακα `p`, οπότε το πηλίκο επιστρέφει το πλήθος των στοιχείων του πίνακα.

Με την εντολή :

```
printf("p[0] = %d\n", *p);
```

θα εμφανιστούν τα παρακάτω :

```
p[0] = 2
```

Σε ένα δείκτη μπορούμε να προσθέσουμε έναν ακέραιο αριθμό. Το αποτέλεσμα είναι μια νέα διεύθυνση που προκύπτει απ' την προηγούμενη, αν προστεθεί ο αριθμός επί το πλήθος των `bytes` του τύπου του δείκτη. Π.χ. το `*(p+1)` είναι το `p[1]`.

Παράδειγμα

Με την εντολή :

```
for (i=0;i<=sizeof(p)/sizeof(p[0])-1;i++)
     printf("*(p + %d) = %d\n", i, *(p+i));
```

θα εμφανιστούν τα παρακάτω :

```
*(p+0) = 2
*(p+1) = 4
*(p+2) = 6
*(p+3) = 8
*(p+4) = 10
Press any key to continue . . .
```

Αν δηλώσουμε ένα δείκτη και αποθηκεύσουμε το όνομα ενός πίνακα, μπορούμε να εμφανίζουμε τα στοιχεία του πίνακα, όπως και στα προηγούμενα παραδείγματα. Με τις παρακάτω εντολές :

```
int *pp; // Δήλωση δείκτη στον πίνακα p
pp = p;

printf("pp = ");
for (i=0;i<=n-1;i++)
 printf(" %d", pp[i]);
printf("\n");
printf("*(pp + i ) = ");
for (i=0;i<=n-1;i++)
 printf(" %d", *(pp+i));
printf("\n");
system("Pause");
```

το πρόγραμμα θα εμφανίσει :

```
pp = 2 4 6 8 10
*(pp + i ) = 2 4 6 8 10
Press any key to continue . . .
```

9.4 Πέρασμα Παραμέτρων Πινάκων σε Συναρτήσεις με Αναφορά (Διεύθυνση)

Αφού με τη δήλωση ενός πίνακα, το όνομά του είναι δείκτης στο πρώτο στοιχείο του πίνακα, όταν περνάμε σαν παράμετρο σε μια συνάρτηση το όνομα ενός πίνακα, είναι σαν να περνάμε τον δείκτη στον πίνακα. Οι παρακάτω δηλώσεις `void fillP( int n, int p[])` και `void fillPP( int n, int *p)` είναι ισοδύναμες, όπως φαίνεται στο πρόγραμμα που ακολουθεί :

```
#include <stdio.h>
#include <stdlib.h>

void showP( int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf(" %d", p[i]);
   printf("\n");
}

void fillP( int n, int p[])
{
   int i;
   for ( i=0;i<=n-1;i++)
    {
     printf("Give p[%d] : ", i);
     scanf("%d", &p[i]);
    }
}
```

```
void fillPP( int n, int *p)
{
   int i;
   for ( i=0;i<=n-1;i++)
    {
     printf("Give p[%d] : ", i);
     scanf("%d", &p[i]);
    }
}

void fillPP1( int n, int *p)
{
   int i;
   for ( i=0;i<=n-1;i++)
    {
     printf("Give p[%d] : ", i);
     scanf("%d", *(p+i));
    }
}

main()
{
    int i, n = 5;
    int p[n];
    fillP( n, p);
    printf("call fillP(int n, int p[]), p = ");
    showP( n, p);
    fillPP( n, p);
    printf("call fillPP(int n, int *p), p = ");
    showP( n, p);
    fillPP1( n, p);
    printf("call fillPP1(int n, int *p), p = ");
    showP( n, p);
    system("Pause");
}
```

Με το τρέξιμο του παραπάνω προγράμματος θα έχουμε :

```
Give p[0] : 2
Give p[1] : 4
Give p[2] : 6
Give p[3] : 8
Give p[4] : 10
call fillP(int n, int p[]), p = 2 4 6 8 10
Give p[0] : 1
Give p[1] : 3
Give p[2] : 5
Give p[3] : 7
Give p[4] : 9
call fillPP(int n, int *p), p = 1 3 5 7 9
Give p[0] : 1
Give p[1] : 2
Give p[2] : 3
Give p[3] : 4
Give p[4] : 5
call fillPP1(int n, int *p), p = 1 2 3 4 5
```

Η συνάρτηση μπορεί επίσης να **επιστρέφει δείκτη σε πίνακα**, πράγμα που πρέπει να δηλωθεί στην δήλωση της συνάρτησης. Η παρακάτω συνάρτηση

```
int* intfillP(int n, int* p)
{
   int i;
   for ( i=0;i<=n-1;i++)
    {
     printf("Give p[%d] : ", i);
     scanf("%d", &p[i]);
    }
    return p;
}
```

επιστρέφει δείκτη σε πίνακα.

9.5 Δείκτες στη Μνήμη Σωρού – Δυναμική Χορήγηση Μνήμης

Η μνήμη στοίβας είναι περιορισμένης χωρητικότητας. Αν δηλώσουμε έναν πίνακα με 1000000 στοιχεία, το λειτουργικό σύστημα δεν θα μπορέσει να μας διαθέσει τη μνήμη που χρειαζόμαστε και το πρόγραμμά μας θα κολλήσει. Μπορούμε να χρησιμοποιήσουμε όμως τη μνήμη σωρού, η οποία είναι μεγαλύτερης χωρητικότητας με τη χρήση δεικτών και της εντολής malloc (memory allocation). Σ' αυτή την περίπτωση δεν χρησιμοποιούμε αγκύλες.

Με τις παρακάτω εντολές δηλώνουμε 3 μεταβλητές (num, ch, pp[10]) στη μνήμη στοίβας και 3 μεταβλητές – δείκτες (*pnum, *ptw, *pp) στην μνήμη σωρού, έναν σε ακέραιο αριθμό, έναν σε ένα πίνακα n ακεραίων και έναν σε έναν πίνακα 40 χαρακτήρων και δεσμεύουμε και τη μνήμη που χρειαζόμαστε.

```
int num;
char ch;
int p[] = {2,4,6,8,10, 12, 14, 16, 18, 20};

int *pnum;
char *ptw;
int *pp;

pnum = (int*)malloc(sizeof(int));
ptw = (char*)malloc(40);
pp = (int*)malloc(sizeof(int) * 10);
```

Με τις παρακάτω εντολές εμφανίζουμε τις διευθύνσεις που καταλαμβάνουν στη μνήμη στοίβας οι 3 μεταβλητές (num, ch, pp[10]) και στην μνήμη σωρού οι 3 μεταβλητές – δείκτες (*pnum, *ptw, *pp):

```
printf("Addresses of num, ch = %d %d\n", &num,&ch);
printf("Addresses of p+i ( i = 0:n-1) = ");
```

```
    for ( i=0;i<=n-1;i++)
          printf(" %d", p+i);

    printf("Addresses of pnum, ptw = %d %d\n", pnum, ptw);
    printf("Addresses of pp+i ( i = 0:n-1) = ");
    for ( i=0;i<=n-1;i++)
         printf(" %d", pp+i);
```

και θα εμφανιστούν τα παρακάτω :

```
Addresses of num, ch = 2293564 2293563
Addresses of p+i ( i = 0:n-1) = 2293504 2293508 2293512 2293516
2293520 2293524 2293528 2293532 2293536 2293540
Addresses of pnum, ptw = 8589136 8589240
Addresses of pp+i ( i = 0:n-1) = 8589288 8589292 8589296 8589300
8589304 8589308 8589312 8589316 8589320 8589324
Press any key to continue . . .
```

Θα μπορούσαμε να χρησιμοποιήσουμε δείκτες σε όλες τις συναρτήσεις που χρησιμοποιήσαμε στο κεφάλαιο 6. Στο επόμενο πρόγραμμα δηλώνουμε έναν δείκτη σε έναν πίνακα ακεραίων 10 θέσεων, δεσμεύουμε την αντίστοιχη θέση μνήμης με την εντολή malloc, καλούμε την συνάρτηση `fillPP( int n, int *p)` για να γεμίσουμε τις θέσεις του πίνακα με τυχαίες ακέραιες τιμές στο [0,9], καλούμε την συνάρτηση `showPP( int n, int *p)` για να εμφανίσουμε τα στοιχεία του πίνακα, καλούμε την συνάρτηση `returnThesiMax( int n, int *p)` για να βρούμε το μέγιστο στοιχείο του πίνακα, το οποίο και εμφανίζουμε, διαβάζουμε έναν ακέραιο αριθμό στη μεταβλητή num και καλούμε την συνάρτηση `findThesiNum( int n, int num, int *p)` για να βρούμε όλες τις θέσεις του πίνακα στις οποίες εμφανίζεται ο αριθμός, καλούμε την συνάρτηση `returnThesiNum( int n, int num, int *p)` για να βρούμε την πρώτη θέση του πίνακα στην οποία υπάρχει ο αριθμός και τον εμφανίζουμε, καλούμε την συνάρτηση `sortMaxPP( int n, int *p)` για να ταξινομήσουμε τα στοιχεία του πίνακα, καλούμε την συνάρτηση `showPP( int n, int *p)` για να εμφανίσουμε τα στοιχεία του πίνακα ταξινομημένα, καλούμε την συνάρτηση `findThesiNumBinSearch( int a, int t, int num, int *p)` για να βρούμε την πρώτη θέση του πίνακα στην οποία υπάρχει ο αριθμός και τον εμφανίζουμε, διαβάζουμε δύο ακέραιους αριθμούς στο [0,9], καλούμε την συνάρτηση `swapPiPj( int i, int j, int *p)` για να ανταλλάξουμε τα στοιχεία του πίνακα σ' αυτές τις θέσεις και στο τέλος καλούμε την συνάρτηση `showPP( int n, int *p)` για να εμφανίσουμε τα στοιχεία του πίνακα μετά την ανταλλαγή :

```
#include <stdio.h>
#include <stdlib.h>

void swapPiPj( int i, int j, int *p)
{
   int temp;
   temp = *(p+i);
   *(p+i) = *(p+j);
   *(p+j) = temp;
}
```

```
void showPP( int n, int *p)
{
   int i;
   for ( i=0;i<=n-1;i++)
      printf(" %d", *(p+i));
   printf("\n");
}


void fillPP( int n, int *p)
{
   int i;
   srand(1);
   for ( i=0;i<=n-1;i++)
      *(p+i) = rand()%10;
}


int returnThesiMax( int n, int *p)
{
   int i, max, thesiMax;
   max = *p;
   thesiMax = 0;
   for ( i=1;i<=n-1;i++)
      if ( *(p+i) > max )
         {
          max =  *(p+i);
          thesiMax = i;
          }
    return thesiMax;
}


int returnThesiNum( int n, int num, int *p)
{
   int i, thesiNum = -1;
   i=0;
   while ( i<=n-1 && thesiNum == -1)
      if ( *(p+i) == num )
          thesiNum = i;
      else
          i++;
    return thesiNum;
}

void findThesiNum( int n, int num, int *p)
{
   int i;

   for ( i=0;i<=n-1;i++ )
      if ( *(p+i) == num )
          printf("*(p+i) = %d in position i = %d\n", *(p+i),
i);
}
```

```
int findThesiNumBinSearch( int a, int t, int num, int *p)
{
   int i, m, thesiNum = -1;

   while ( a<=t && thesiNum == -1)
   {
        m = (a+t)/2;
        if ( *(p+m) == num )
          thesiNum = m;
        else
          if ( *(p+m) > num )
           t = m-1;
          else
           a = m + 1;

}
    return thesiNum;
}


void sortMaxPP( int n, int *p)
{
   int i, thesiMax;
   for ( i=n;i>=2;i--)

     {
      thesiMax = returnThesiMax(  i, p);
      swapPiPj(i-1,thesiMax,p);
      }

}

main()
{

    int i, j, num, n = 10;
    int *pp;

    pp = (int*)malloc(sizeof(int) * n);
    fillPP( n, pp);
    printf("\npp = ");
    showPP( n, pp);
    int thesiMax = returnThesiMax(n, pp);
    printf("\nmax  =  *(pp+thesiMax)  =  %d  in  position   thesiMax  =
%d\n", *(pp+thesiMax) , thesiMax );
    printf("Give num : ");
    scanf("%d", &num);
    findThesiNum(n, num, pp);
    int thesiNum = returnThesiNum(n, num, pp);
    if (thesiNum != -1)
      printf("*(pp+thesiNum)    =    %d    in    thesiNum    =    %d\n",
*(pp+thesiNum), thesiNum );
    else
      printf("not found\n");

    sortMaxPP(  n,  pp);
    printf("\npp sorted = ");
```

```
    showPP( n, pp);
    thesiNum = findThesiNumBinSearch(  0,  n-1,  num,  pp);
    if (thesiNum != -1)
      printf("*(pp+thesiNum)    =    %d    in    thesiNum    =    %d\n",
 *(pp+thesiNum), thesiNum );
    else
      printf("not found\n");
    printf("Give i, j in [0,n-1] : ");
    scanf("%d %d", &i, &j);
    swapPiPj(i,j,pp);
    printf("\npp = ");
    showPP( n, pp);
    system("Pause");
 }
```

Με το τρέξιμο του παραπάνω προγράμματος θα έχουμε :

```
pp = 1 7 4 0 9 4 8 8 2 4
max = *(pp+thesiMax) = 9 in position thesiMax = 4
Give num : 8
*(p+i) = 8 in position i = 6
*(p+i) = 8 in position i = 7
*(pp+thesiNum) = 8 in thesiNum = 6

pp sorted = 0 1 2 4 4 4 7 8 8 9
*(pp+thesiNum) = 8 in thesiNum = 7
Give i, j in [0,n-1] : 0 1

pp = 1 0 2 4 4 4 7 8 8 9
Press any key to continue . . .
```

9.6 Δείκτες σε Πίνακες 2 Διαστάσεων

Όταν δηλώνεται ένας πίνακας 2 διαστάσεων δημιουργούνται τόσοι μονοδιάστατοι πίνακες, όσες και οι γραμμές του διδιάστατου πίνακα. Με τις παρακάτω εντολές

```
int p2d[][3] = {{1,2,3},{4,5,6},{7,8,9}};
```

δηλώνουμε τον πίνακα 2 διαστάσεων `p2d` με 3 γραμμές και 3 στήλες, τον οποίο γεμίζουμε με δυναμική αρχικοποίηση. Οι διευθύνσεις των 3 μονοδιάστατων πινάκων βρίσκονται στα `p2d[0]`, `p2d[1]` και `p2d[2]`. Με τις παρακάτω εντολές :

```
printf("\nAddresses of p2d = \n");

for ( i=0;i<=2;i++)
    {
    for ( j=0;j<=2;j++)
     printf(" %d", p2d[i]+j);
    printf("\n");
    }
```

```
printf("\ndata of p2d = \n");
for ( i=0;i<=2;i++)
{
for ( j=0;j<=2;j++)
 printf(" %d", *(p2d[i]+j));
printf("\n");
}
```

βλέπουμε τις διευθύνσεις όλων των στοιχείων του πίνακα, τις οποίες κατόπιν χρησιμοποιούμε να εμφανίσουμε τα περιεχόμενά τους και η έξοδος θα είναι :

```
Addresses of p2d =
2293456 2293460 2293464
2293468 2293472 2293476
2293480 2293484 2293488


data of p2d =
 1 2 3
 4 5 6
 7 8 9
```

Μπορούμε να περνάμε σαν παράμετρο έναν δείκτη σε έναν πίνακα 2 διαστάσεων, οπότε σ' αυτή την περίπτωση δεν είναι αναγκαίο να περνάμε παραμετρικά στις αγκύλες και το πλήθος των στηλών του πίνακα. Έτσι το παράδειγμα 6.4 μπορεί να γίνει με δείκτες και μπορεί να είναι το παρακάτω :

- Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο γεμίζει έναν πίνακα `2` διαστάσεων (`3x2`), με **δυναμική αρχικοποίηση** και δίνει τις τιμές από το `1` μέχρι το `6`, δηλώνει έναν άλλον πίνακα (`mxn`), θέσεων, τον οποίο γεμίζει με τυχαίες τιμές και εμφανίζει τα περιεχόμενα των `2` πινάκων καλώντας συναρτήσεις για το γέμισμα του πίνακα των τυχαίων τιμών και την εμφάνιση των `2` πινάκων. Μετά καλεί τη συνάρτηση `findMax()` με την οποία βρίσκει και εμφανίζει το μεγαλύτερο στοιχείο του πίνακα `p2` και τη γραμμή και στήλη του πίνακα, στις οποίες βρίσκεται.

Αλγόριθμος

1. Διαβάζω τιμές για τα `m, n > 1`.
2. Δήλωση - Γέμισμα πίνακα `p` με τυχαίες τιμές καλώντας τη μέθοδο `fillPin2D()`
3. Εμφάνιση στοιχείων πίνακα `p` καλώντας τη συνάρτηση `showPin2D()`
4. Εύρεση μεγίστου στοιχείου `p`, εμφάνιση στοιχείου και γραμμής-στήλης καλώντας τη συνάρτηση `findMax()`

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/* Το πρόγραμμα δημιουργεί 1 πίνακα 2 διαστάσεων ( 3x2 ) και τον γεμίζει με
τυχαίες τιμές από 1 μέχρι 10 καλώντας τη  συνάρτηση fillPin2D() και εμφανίζει
τα στοιχεία του  πίνακα καλώντας τη  συνάρτηση showPin2D(). Μετά καλεί τη
συνάρτηση findMax()για την εύρεση του μεγίστου στοιχείου του p και την
εμφάνιση του μεγίστου στοιχείου και της γραμμής-στήλης, στις οποίες βρέθηκε
*/


void fillPin2D(int m, int n, int *p)
// Συνάρτηση δημιουργίας στοιχείων πίνακα 2D
{
   int i,j;
   for ( i=0;i<=m-1;i++)
    for ( j=0;j<=n-1;j++)
      *(p+i*n+j) = (rand() % (10 - 1 + 1)) + 1;
}

void showPin2D(int m, int n, int *p)
// Συνάρτηση Εμφάνισης στοιχείων πίνακα 2
{
   int i, j;
   for ( i=0;i<=m-1;i++)
    {
     for ( j=0;j<=n-1;j++)
       printf("%3d ", *(p+i*n+j));
     printf("\n");
    }
}
void findMax(int m, int n, int *p)
// Εύρεση, Εμφάνιση Μεγίστου – Γραμμής – Στήλης
{
   int i, j, max, lineMax, colMax, lineColMax;
   max = *p;
   lineMax = 0;
   colMax = 0;
   for (i = 0;i <= m-1;i++)
    for ( j=0;j<=n-1;j++)
      if (*(p+i*n+j) >= max )
      {
        max = *(p+i*n+j);
        lineMax = i;
        colMax = j;
      }
   printf("max = %d in line %d and col %d\n", *(p+ lineMax *n+ colMax),
lineMax, colMax);
}

main()
     int i, j;
       // Εκχώρηση ακέραιας τιμής>1 στα μεγέθη γραμμών-στηλών του πίνακα 2
       do
       {
         printf("Give two integers m, n > 1 : ");
         scanf("%d %d", &m, &n);
       }
       while ( m <= 1 || n <= 1 );

        // Δήλωση πίνακα p
        int p[m][n];
```

```
        // Εκχώρηση διεύθυνσης πίνακα p στο δείκτη pp
        int *pp = &p[m][n];// Εναλλακτικά, int *pp = p[0];

        // Γέμισμα πίνακα με τυχαίες ακέραιες τιμές
        fillPin2D( m, n, pp);

        // Εμφάνιση στοιχείων πίνακα p
        printf("p = \n");
        showPin2D( m, n,  pp);

       // Εύρεση, Εμφάνιση Μεγίστου – Γραμμής – Στήλης
       findMax( m, n,  pp);
       system("Pause");
     }
}
```

Έξοδος Προγράμματος :

```
Give two integers m, n > 1 : 3 4

p =
 2  8  5  1
10  5  9  9
 3  5  6  6
max = 10 in line 1 and column 0
Press any key to continue . . .
```

9.7 Δείκτες σε Δομές - structures

Όπως είδαμε στο παράδειγμα 8.5.1, δεν μπορούμε να αλλάξουμε τα περιεχόμενα των δεικτών σε δομές ή τα περιεχόμενα των κελιών τους με την κλήση κάποιας συνάρτησης, γιατί οι δομές περνάνε σαν παράμετροι με τιμή και μάλιστα περνάει όλη η δομή, με αποτέλεσμα να έχουμε σπατάλη μνήμης. Για να περάσουν οι δομές με αναφορά, θα πρέπει να είναι δείκτες.

9.7.1 Παράδειγμα Δημιουργίας 2 μεταβλητών Δεικτών Τύπου Δομής και Επεξεργασία των Δεδομένων τους με Συναρτήσεις

- Να γραφεί Αλγόριθμος/Πρόγραμμα, το οποίο δηλώνει 2 μεταβλητές f1 και f2 τύπου foititis, δίνει τιμές στο Όνομα, τον Αριθμό Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου για τους 2 φοιτητές με την κλήση της συνάρτησης gemismaPedion(), με την οποία περνάει σταθερές τιμές για τα πεδία του 1ου φοιτητή, τις οποίες και εμφανίζει με την κλήση της συνάρτησης emfanishPedion(), ενώ για τα πεδία του 2ου φοιτητή διαβάζει τιμές απ' το πληκτρολόγιο σε τοπικές μεταβλητές, τις οποίες μετά περνάει παραμετρικά με την κλήση της συνάρτησης gemismaPedion(), τις οποίες και εμφανίζει με την κλήση της συνάρτησης emfanishPedion() και Υπολογίζει και Εμφανίζει τον Τελικό Βαθμό του κάθε φοιτητή με την κλήση των συναρτήσεων findTB() και returnTB(). Στη συνέχεια καλεί τη συνάρτηση findMaxTB() με την οποία βρίσκει το μεγαλύτερο Τελικό βαθμό των φοιτητών f1 και f2 και ανταλλάσσει τα περιεχόμενα των μεταβλητών f1 και f2 στη main() και μετά ανταλλάσσει ξανά τα περιεχόμενα των μεταβλητών f1 και f2 με την κλήση της συνάρτησης swapF1F2(). Μετά θα καλεί τη

συνάρτηση `swapAM()` με την οποία θα ανταλλάσσει τπους Αριθμούς Μητρώου των 2 φοιτητών και τη συνάρτηση `setKM()` με την οποία θα αλλάζει την τιμή του Κωδικού μαθήματος για το φοιτητή 2 και με τη συνάρτηση `emfanishPedion()` θα εμφανίζει τα νέα περιεχόμενα των φοιτητών `f1` και `f2`.

Αλγόριθμος `main()`

1. Εισαγωγή Στοιχείων 1ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
2. Εμφάνιση στοιχείων 1ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
3. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 1ου φοιτητή με κλήση συνάρτησης `findTB()`**
4. Εισαγωγή Στοιχείων 2ου φοιτητή – κλήση συνάρτησης `gemismaPedion()`
5. Εμφάνιση στοιχείων 2ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
6. **Υπολογισμός - Εμφάνιση Τελικού Βαθμού 2ου φοιτητή με κλήση συνάρτησης `returnTB()`**
7. **Υπολογισμός - Εμφάνιση Μέγιστου Τελικού Βαθμού 1ου - 2ου φοιτητή με κλήση συνάρτησης `findMaxTB()`**
8. **Ανταλλαγή περιεχομένων 1ου - 2ου φοιτητή**
9. Εμφάνιση στοιχείων 1ου – 2ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
10. **Ανταλλαγή περιεχομένων 1ου - 2ου φοιτητή** – κλήση συνάρτησης `swapF1F2()`
11. Εμφάνιση στοιχείων 1ου – 2ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`
12. **Ανταλλαγή περιεχομένων ΑΜ 1ου - 2ου φοιτητή** – κλήση συνάρτησης `swapAM()`
13. **Αλλαγή πεδίου ΚΜ φοιτητή 2** – κλήση συνάρτησης `setKM()`
14. Εμφάνιση στοιχείων 1ου – 2ου Φοιτητή – κλήση συνάρτησης `emfanihPedion()`

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο δημιουργεί 2 μεαβλητές f1, f2 τύπου foititis, με την
κλήση της συνάρτησης gemismaPedion()  δίνει τιμές στο Όνομα, τον Αριθμό
Μητρώου, τον Κωδικό Μαθήματος, τον Βαθμό Θεωρίας και τον Βαθμό Εργαστηρίου
για τους 2 φοιτητές σταθερές και παραμετρικά, τις οποίες και εμφανίζει με την
κλήση της συνάρτησης emfanishPedion()και Υπολογίζει και Εμφανίζει τον Τελικό
Βαθμό του κάθε φοιτητή με την κλήση των συναρτήσεων findTB() και returnTB().
Στη συνέχεια καλεί τη συνάρτηση findMaxTB() με την οποία βρίσκει το
μεγαλύτερο Τελικό βαθμό των φοιτητών f1 και f2 και ανταλλάσσει τα περιεχόμενα
των μεταβλητών f1 και f2 στη main() και μετά ανταλλάσσει ξανά τα περιεχόμενα
των μεταβλητών f1 και f2  με την κλήση της συνάρτησης swapF1F2().Μετά θα
καλεί τη συνάρτηση swapAM() με την οποία θα ανταλλάσσει τπους Αριθμούς
Μητρώου των 2 φοιτητών και τη συνάρτηση setKM() με την οποία θα αλλάζει την
τιμή του Κωδικού μαθήματος για το φοιτητή 2 και με τη συνάρτηση
emfanihPedion() θα εμφανίζει τα νέα περιεχόμενα των φοιτητών f1 και f2.*/


    typedef struct foititis
    {
        char *name;
        int aM;
        int kM;
        double bTh;
        double bErg;
   }Foititis;
```

```
    // Συνάρτηση Κατασκευής - Δομητής
    void gemismaPedion( char *n, int am, int km, double bth, double berg,
Foititis *f)
{
    f->name = n;
    f->aM = am;
    f->kM = km;
    f->bTh = bth;
    f->bErg = berg;
}

   // Συνάρτηση Εμφάνισης τιμών πεδίων
   void emfanishPedion(Foititis *f)
  {
      printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\n", f->name,
  f->aM, f->kM, f->bTh, f->bErg);
  }

  // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
  double returnTB(Foititis *f)
  {
      double tB = f->bTh * 0.7 + f->bErg * 0.3;
      return tB;
  }

  // Συνάρτηση Υπολογισμού - Εμφάνισης Τελικού Βαθμού
  void findTB(Foititis *f)
  {
      double tB = f->bTh * 0.7 + f->bErg * 0.3;
      printf("call findTB() - tB = %lf\n", tB);
  }

  // Συνάρτηση Υπολογισμού - Εμφάνισης Μεγίστου Τελικού Βαθμού
  void findMaxTB(Foititis *f1, Foititis *f2)
  {
      if ( returnTB(f1) >  returnTB(f2))
        printf("TB f1 = %lf > TB f2 = %lf\n", returnTB(f1), returnTB(f2));
      else
        printf("TB f2 = %lf > TB f1 = %lf\n", returnTB(f2), returnTB(f1));
  }

  // Συνάρτηση Ανταλλαγής  aM f1, f2
  void swapAM(Foititis *f1, Foititis *f2)
  {
      int temp = f1->aM;
      f1->aM = f2->aM;
      f2->aM = temp;
  }

  // Συνάρτηση Ανταλλαγής  f1, f2
  void swapF1F2(Foititis *f1, Foititis *f2)
  {
      Foititis *temp = f1;
      f1 = f2;
      f2 = temp;
  }

  // Συνάρτηση Αλλαγής  kM
  void setKM( int km, Foititis *f)
  {
      f->kM = km;
  }
```

```
main()
 {
  char *n = (char*)malloc(sizeof(40));
  int aM, kM;
  double bTh, bErg, tB;
  Foititis *f1 = (Foititis*)malloc(sizeof(Foititis));
  Foititis *f2 = (Foititis*)malloc(sizeof(Foititis));

  // Εισαγωγή Στοιχείων Φοιτητή 1
  gemismaPedion("ioannou",191234,6101,7.5,6.0,f1);

  // Εμφάνιση Στοιχείων Φοιτητή 1
  printf("Foititis f1 : \n");
  emfanishPedion( f1 );

  // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 1
  findTB(f1);

  // Εισαγωγή Στοιχείων Φοιτητή 2
  printf("Give name of foititis 2 : ");
  scanf("%s", n);
  printf("Give aM of foititis 2 : ");
  scanf("%d", &aM);
  printf("Give kM of foititis 2 : ");
  scanf("%d", &kM);
  printf("Give bTh of foititis 2 : ");
  scanf("%lf", &bTh);
  printf("Give bErg of foititis 2 : ");
  scanf("%lf", &bErg);
  gemismaPedion(n,aM,kM,bTh,bErg,f2);

  // Εμφάνιση Στοιχείων Φοιτητή 2
  printf("Foititis f2 : \n");
  emfanishPedion( f2 );

  // Υπολογισμός - Εμφάνιση Τελικού Βαθμού Φοιτητή 2
  tB = returnTB(f2);
  printf("call returnTB() - tB = %lf\n", tB);

  // Υπολογισμός - Εμφάνιση Μέγιστου Τελικού Βαθμού Φοιτητών f1, f2
  findMaxTB(f1, f2);

  // Ανταλλαγή περιεχομένων  f1, f2
 Foititis *temp;
  temp = f1;
  f1 = f2;
  f2 = temp;

  // Εμφάνιση Στοιχείων Φοιτητή 1
  printf("Foititis f1 after swap f1 f2:\n");
  emfanishPedion( f1 );

  // Εμφάνιση Στοιχείων Φοιτητή 2
  printf("Foititis f2 after swap f1 f2 :\n");
  emfanishPedion( f2 );

  // Ανταλλαγή περιεχομένων  f1, f2 – κλήση swapF1F2
  swapF1F2(f1,f2);

 // Εμφάνιση Στοιχείων Φοιτητή 1
 printf("Foititis f1 after call swapF1F2() : \n");
 emfanishPedion( f1 );

 // Εμφάνιση Στοιχείων Φοιτητή 2
 printf("Foititis f2 after call swapF1F2() : \n");
 emfanishPedion( f2 );
```

```
    // Ανταλλαγή περιεχομένων  aM f1, f2
    swapAM(f1,f2);

    // Αλλαγή περιεχομένων  kM f2
    setKM( 6102, f2);

     // Εμφάνιση Στοιχείων Φοιτητή 1
     printf("Foititis f1 after call swapAM() : \n");
     emfanishPedion( f1 );

     // Εμφάνιση Στοιχείων Φοιτητή 2
     printf("Foititis f2 after call swapAM and setKM( ) : \n");
     emfanishPedion( f2 );
  }
```

Έξοδος Προγράμματος :

```
foititis f1 :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
call findTB() - tB = 7.0500000
Give name of foititis 2 : georgiou
Give aM of foititis 2 : 12345
Give kM of foititis 2 : 6101
Give bTh of foititis 2 : 8
Give bErg of foititis 2 : 7
foititis f2 :
name : georgiou
aM : 12345
kM : 6101
bTh : 8
bErg : 7
call returnTB() - tB = 7.700000
TB f2 = 7.700000 > TB f1 = 7.050000

Foititis f1 after swap f1 f2 :
name : georgiou
aM : 12345
kM : 6101
bTh : 8
bErg : 7
Foititis f2 after swap f1 f2 :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0

Foititis f1 after call swapF1F2() :
name : georgiou
aM : 12345
kM : 6101
bTh : 8
bErg : 7
Foititis f2 after call swapF1F2() :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
Foititis f1 after call swapAM() :
```

```
name : georgiou
aM : 191234
kM : 6101
bTh : 8
bErg : 7
Foititis f2 after call swapAM and setKM() :
name : ioannou
aM : 12345
kM : 6102
bTh : 7.5
bErg : 6.0
Press any key to continue . . .
```

Παρατηρήσεις

- Με τις εντολές

```
typedef struct foititis
{
    char *name;
    int aM;
    int kM;
    double bTh;
    double bErg;
}Foititis;
```

ορίζουμε τον τύπο δομής `Foititis`, τον οποίο μπορούμε να χρησιμοποιούμε αντί του `struct foititis`.

- Το πεδίο **`name`** έγινε **δείκτης σε πίνακα χαρακτήρων**, για να μπορούμε πιο εύκολα να διαβάζουμε πίνακες χαρακτήρων ή αλφαριθμητικές σταθερές.

- Η πρόσβαση σε κάθε **πεδίο** της δομής με δείκτη δεν πλέον γίνεται με την **τελεία**, αλλά με το **->**.

- Οι δομές περνάνε με **αναφορά** σαν **παράμετροι** στις συναρτήσεις, εκτός από τη συνάρτηση `swapF1F2()`, στην οποία περνάνε με τιμή.

9.7.2 Παράδειγμα Δημιουργίας Πίνακα Μεταβλητών Δεικτών Τύπου δομής – Επεξεργασία Δεδομένων του Πίνακα με Χρήση Συναρτήσεων

- Να γραφεί Αλγόριθμος/Πρόγραμμμα διαβάζει την τιμή μιας μεταβλητής `n` για το μέγεθος του πίνακα `n` μεταβλητών τύπου `foititis`. **Η δομή `foititis` περιέχει και το πεδίο `tB` για τον Τελικό Βαθμό**. Όλα τα πεδία των φοιτητών του πίνακα, εκτός του Τελικού Βαθμού γεμίζουν με την κλήση της συνάρτησης `gemismaPedion()`. Με την κλήση της συνάρτησης `setTBAll()`, η οποία καλεί τη συνάρτηση `returnTB()`, γεμίζουν όλα τα πεδία των `n` φοιτητών του πίνακα, τα οποία και εμφανίζει με την κλήση της συνάρτησης `emfanishPedion()`. Μετά δημιουργεί έναν τυχαίο ακέραιο αριθμό `index` μεταξύ `0` και `n-2` και καλεί τη συνάρτηση `swapFiFi1()`, με την οποία

ανταλλάσσει τα περιεχόμενα των f[index], f[index+1] και εμφανίζει ξανά τα στοιχεία όλων των φοιτητών. Στη συνέχεια καλεί τη συνάρτηση returnThesiMaxTB() με την οποία βρίσκει το φοιτητή που έχει το μεγαλύτερο Τελικό βαθμό και εμφανίζει τα στοιχεία του.

Αλγόριθμος main()

1. Διάβασμα τιμής <= 50 στη μεταβλητή n.
2. Εισαγωγή Στοιχείων n φοιτητών – κλήση συνάρτησης gemismaPedion()
3. **Γέμισμα πεδίου** tB **των** n φοιτητών - **κλήση συνάρτησης** setTBAll **()**
4. Εμφάνιση στοιχείων n φοιτητών – κλήση συνάρτησης emfanihPedion()
5. Δημιουργία τυχαίου ακεραίου index στο [0, n-2].
6. **Ανταλλαγή περιεχομένων** f[index], f[index+1] – κλήση συνάρτησης swapFiFi1()
7. Εμφάνιση στοιχείων n φοιτητών – κλήση συνάρτησης emfanihPedion()
8. **Εύρεση φοιτητή με Μέγιστο Τελικό Βαθμό - κλήση συνάρτησης** returnThesiMaxTB**()**
9. **Εμφάνιση** στοιχείων **φοιτητή με Μέγιστο Τελικό Βαθμό.**

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο διαβάζει την τιμή μιας μεταβλητής n για το μέγεθος
του πίνακα n μεταβλητών τύπου foititis. Η δομή foititis περιέχει και το πεδίο
tB για τον Τελικό Βαθμό. Όλα τα πεδία των φοιτητών του πίνακα, εκτός του
Τελικού Βαθμού γεμίζουν με την κλήση της συνάρτησης gemismaPedion(). Με την
κλήση της συνάρτησης setTBAll(), η οποία καλεί τη συνάρτηση returnTB(),
γεμίζουν όλα τα πεδία των n  φοιτητών του πίνακα,  τα οποία και εμφανίζει με
την κλήση της συνάρτησης emfanishPedion(). Μετά δημιουργεί έναν τυχαίο
ακέραιο αριθμό index μεταξύ 0 και n-2 και καλεί τη συνάρτηση swapFiFi1(), με
την οποία ανταλλάσσει τα περιεχόμενα των f[index], f[index+1] και εμφανίζει
ξανά τα στοιχεία όλων των φοιτητών.  Στη συνέχεια καλεί τη συνάρτηση
returnThesiMaxTB() με την οποία βρίσκει το φοιτητή που έχει το μεγαλύτερο
Τελικό βαθμό και εμφανίζει τα στοιχεία του*/


    typedef struct foititis
    {
        char *name;
        int aM;
        int kM;
        double bTh;
        double bErg;
        double tB;
   }Foititis;

  // Συνάρτηση Κατασκευής - Δομητής
  Foititis* gemismaPedion( char *nm, int am, int km, double bth, double berg)
  {
  Foititis *f = (Foititis*)malloc(sizeof(Foititis));
  f->name = nm;
  f->aM = am;
  f->kM = km;
```

```
  f->bTh = bth;
  f->bErg = berg;
  return f;
 }

 // Συνάρτηση Εμφάνισης τιμών πεδίων
void emfanishPedion(Foititis *f)
{
    printf("name : %s\naM = %d\nkM = %d\nbTh = %lf\nbErg = %lf\ntB = %lf\n",
f->name, f->aM, f->kM, f->bTh, f->bErg, f->tB);
}

   // Συνάρτηση Υπολογισμού - Επιστροφής Τελικού Βαθμού
double returnTB(Foititis *f)
{
    double tB = f->bTh * 0.7 + f->bErg * 0.3;
    return tB;
}

   // Συνάρτηση Υπολογισμού - Τελικού Βαθμού Όλων
void setTBAll( Foititis *f, int n)
{
    int i;
    for (  i=0;i<=n-1;i++)
      (f+i)->tB =  returnTB(f+i);
}

    // Ανταλλαγή περιεχομένων  f[i], f[j] – swapFiFj
void swapFiFj(Foititis *f, int i, int j)
{
    Foititis temp;
    temp = *(f+i);
    *(f+i) = *(f+j);
    *(f+j) = temp;
}

    // Υπολογισμός Θέσης Μέγιστου Τελικού Βαθμού Φοιτητών f[i], i=0:n-1
int returnThesiMaxTB(Foititis *f, int n)
{
    double maxTB = (f+0)->tB;
    int i, thesiTB = 0;
    for ( i=1;i<=n-1;i++)
      if ( (f+i)->tB >  maxTB)
       {
        maxTB = (f+i)->tB;
        thesiTB = i;
       }
    return thesiTB;
}



  main()
   {
    int i,  n, index, thesiTB, aM, kM;
    double bTh, bErg, tB;
    Foititis *temp;

    // Διάβασμα n
    do
    {
       printf("Give n <= 50 : ");
       scanf("%d", &n);
    }
    while ( n > 50 );
```

```
     Foititis *f = (Foititis*)malloc(sizeof(Foititis)*n);

    // Εισαγωγή Στοιχείων n Φοιτητών
    for ( i=0;i<=n-1;i++)
    {
     char *nm = (char*)malloc(sizeof(40));
     printf("Give name of foititis f[%d] : ", i);
     scanf("%s", nm);
     printf("Give aM of foititis f[%d] : ", i);
     scanf("%d", &aM);
     printf("Give kM of foititis f[%d] : ", i);
     scanf("%d", &kM);
     printf("Give bTh of foititis f[%d] : ", i);
     scanf("%lf", &bTh);
     printf("Give bErg of foititis f[%d]3 : ", i);
     scanf("%lf", &bErg);
     printf("name = %s\n", nm);
     temp = gemismaPedion(nm,aM,kM,bTh,bErg);
     *(f+i) = *temp;
    }

     // Υπολογισμός Τελικού Βαθμού n Φοιτητών
     setTBAll(f, n);

     // Εμφάνιση Στοιχείων n Φοιτητών
     for ( i=0;i<=n-1;i++)
      {
         printf("Foititis f[%d] : \n", i);
         emfanishPedion( f+i );    }

     // Δημιουργία τυχαίας τιμής στο index
      int index = rand() % (n-2);

     // Ανταλλαγή περιεχομένων  f1, f2 – κλήση swapFiFj()
     swapFiFj( f,  index,  index+1);

     printf("pinakas after Swap f[%d] <--> f[%d]\n", index,  index+1);

     // Εμφάνιση Στοιχείων n Φοιτητών
     for ( i=0;i<=n-1;i++)
      {
         printf("Foititis f[%d] : \n", i);
         emfanishPedion( f+i );
     }

     // Υπολογισμός Θέσης  Μέγιστου Τελικού Βαθμού Φοιτητών
     thesiTB = returnThesiMaxTB(f, n);

    // Εμφάνιση Στοιχείων Φοιτητή με Μέγιστο Τελικό Βαθμό
    printf("Pedia foitith me maxTB :\n");
    emfanishPedion(f+thesiTB);

    system("Pause");
  }
```

Έξοδος Προγράμματος :

```
Give name of foititis f[0] : ioannou
Give aM of foititis f[0] : 191234
Give kM of foititis f[0] : 6101
Give bTh of foititis f[0] : 7.5
Give bErg of foititis f[0] : 6.0
```

```
Give name of foititis f[1] : georgiou
Give aM of foititis f[1]  : 192345
Give kM of foititis f[1]  : 6101
Give bTh of foititis f[1]  : 7
Give bErg of foititis f[1]  : 8
Give name of foititis f[2] : aleksiou
Give aM of foititis f[2]  : 193456
Give kM of foititis f[2]  : 6101
Give bTh of foititis f[2]  : 9
Give bErg of foititis f[2]  : 8
Foititis f[0] :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
tB : 7.050000
Foititis f[1] :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
tB : 7.300000
Foititis f[2] :
name : aleksiou
aM : 193456
kM : 6101
bTh : 9
bErg : 8
tB : 8.700000

pinakas after Swap f0] <--> f[1]
Foititis f[0] :
name : georgiou
aM : 192345
kM : 6101
bTh : 7
bErg : 8
tB : 7.300000
Foititis f[1] :
name : ioannou
aM : 191234
kM : 6101
bTh : 7.5
bErg : 6.0
tB : 7.050000
Foititis f[2] :
name : aleksiou
aM : 193456
kM : 6101
bTh : 9
bErg : 8
tB : 8.700000
Pedia foitith me maxTB :
name : aleksiou
aM : 193456
kM : 6101
bTh : 9
bErg : 8
tB : 8.700000
Press any key to continue . . .
```

10 ΑΝΑΔΡΟΜΗ

Υπλάρχουν συναρτήσεις, οι οποίες μέσα στον κώδικά τους περιέχουν κλήσεις στον εαυτό τους με διαφορετικές παραμέτρους κάθε φορά. Αυτές οι συαναρτήσεις ονομάζονται αναδρομικές. Όταν κληθεί μια αναδρομική συνάρτηση απ' τη `main()` ή από οποιαδήποτε καλούσα συνάρτηση, παγώνει η εκτέλεση των εντολών της καλούσας συνάρτησης και αρχίζει να εκτελείται το πρώτο στιγμιότυπο της αναδρομικής συνάρτησης, με τη δημιουργία των απαραίτητων τοπικών μεταβλητών. Αυτό το στιγμιότυπο κάπου καλεί τον εαυτό του, οπότε παγώνει η εκτέλεση των εντολών του πρώτου στιγμιότηπου, αρχίζει να εκτελείται το δεύτερο στιγμιότυπο της αναδρομικής συνάρτησης, και αυτό συνεχίζεται μέχρι να ικανοποιηθεί κάποια συνθήκη και να τερματίσει το τελευταίο στιγμιότυπο, οπότε μετά θα συνεχίσει και θα τερματίσει το προ-τελευταίο στιγμιότυπο και θα συνεχιστεί αυτό μέχρι να τερματίσει το πρώτο στιγμιότυπο. Κάθε φορά πού τερματίζει ένα στιγμιότυπο απελευθερώνει και τη μνήμη που χρειάστηκε, οπότε, αν δεν προβλέψουμε τον τερματισμό της αναδρομής, με τις συνεχείς κλήσεις της συνάρτησης θα έχουμε πρόβλημα μνήμης. Γι' αυτό συνήθως υπάρχει ένας έλεγχος με `if` και όχι επανάληψη, γιατί η αναδρομική κλήση είναι επανάληψη. Με αναδρομή λύνονται προβλήματα, στα οποία κάτι εποαναλαμβάνεται με διαφορετικές παραμέτρους κάθε φορά.

10.1 Παράδειγμα Κλήσης Αναδρομικής Συνάρτησης – Υπολογισμός 1+2+…+n

- Να γραφεί Αλγόριθμος/Πρόγραμμμα που να διαβάζει την τιμή μιας μεταβλητής `n >= 5` και να καλεί μια αναδρομική συνάρτηση `sumn()`, η οποία θα βρίσκει και θα επιστρέφει το άθροισμα των αριθμών `1+2+…+n`..

Αλγόριθμος `main()`

1. **Κάνε**
   Διάβασμα τιμής στη μεταβλητή `n`.
2. **Για Όσο** `n < 5`
3. **Κλήση** συνάρτησης `sumn()`
4. Εμφάνιση αθροίσματος

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμμα το οποίο διαβάζει την τιμή μιας μεταβλητής n >= 5 και καλεί μια
αναδρομική συνάρτηση recaddn(), η οποία βρίσκει και επιστρέφει το άθροισμα των
αριθμών 1+2+…+n.*/
```

```
int recaddn(int n)
{

    if ( n == 1 )
     return 1;
    else
     return (n+recaddn(n-1));
}

  main()
   {
    int sum,  n;

    // Διάβασμα n
    do
    {
       printf("Give n >= 5 : ");
       scanf("%d", &n);
    }
    while ( n < 5 );


    // Υπολογισμός – Εμφάνιση Αθροίσματος – Κλήση Συνάρτησης recaddn(int n)
    sum = recaddn(n);
    printf("sum = %d\n", sum);´

    system("Pause");
}
```

Έξοδος Προγράμματος :

```
Give n >= 5 : 6
sum = 21
Press any key to continue . . .
```

10.2 Παράδειγμα Κλήσης Αναδρομικής Συνάρτησης – Υπολογισμός 1*2*…*n(n!)

- Να γραφεί Αλγόριθμος/Πρόγραμμμα που να διαβάζει την τιμή μιας μεταβλητής `n >= 5` και να καλεί μια αναδρομική συνάρτηση `recnpar()`, η οποία θα βρίσκει και θα επιστρέφει το γινόμενο των αριθμών `1*2*…*n=n!`.

Αλγόριθμος `main()`

1. **Κάνε**
   Διάβασμα τιμής στη μεταβλητή `n`.
2. **Για Όσο** `n < 5`
3. **Κλήση** συνάρτησης `recnpar()`.
4. Εμφάνιση παραγοντικού.

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμα το οποίο διαβάζει την τιμή μιας μεταβλητής n >= 5 και καλεί μια
αναδρομική συνάρτηση recnpar(), η οποία βρίσκει και επιστρέφει το γινόμενο των
αριθμών 1*2*…*n=n!.*/


 int recnpar(int n)
 {
     if ( n == 0 || n == 1 )
      return 1;
     else
      return (n*recnpar(n-1));
 }


   main()
    {
     int npar,  n;

     // Διάβασμα n
     do
     {
        printf("Give n >= 5 : ");
        scanf("%d", &n);
     }
     while ( n < 5 );


     // Υπολογισμός – Εμφάνιση παραγοντικού – Κλήση Συνάρτησης recnpar(int n)
     npar = recnpar(n);
     printf("sum = %d\n", npar);´

     system("Pause");
 }
```

Έξοδος Προγράμματος :

```
Give n >= 5 : 6
npar = 120
Press any key to continue . . .
```

10.3 Παράδειγμα Κλήσης Αναδρομικής Συνάρτησης – Μέτρημα Ανάποδα ανά 2

- Να γραφεί Αλγόριθμος/Πρόγραμμα που να διαβάζει την τιμή μιας μεταβλητής `n >= 5` και να καλεί μια αναδρομική συνάρτηση `recnminus2()`, η οποία θα αφαιρεί `2` από το `n` μέχρι να μηδενιστεί. Η συνάρτηση θα είναι τύπου `void` και θα εμφανίζει τις τιμές που θα παίρνει κάθε φορά το `n` πριν και μετά από κάθε κλήση της συνάρτησης.

Αλγόριθμος `main()`

1. **Κάνε**
   Διάβασμα τιμής στη μεταβλητή `n`.
2. **Για Όσο** `n < 5`
3. **Κλήση** συνάρτησης `recnpar()`.
4. Εμφάνιση παραγοντικού.

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>

/*  Πρόγραμμα το οποίο διαβάζει την τιμή μιας μεταβλητής n >= 5 και να καλεί
μια αναδρομική συνάρτηση recnminus2(), η οποία θα αφαιρεί 2 από το n μέχρι να
μηδενιστεί. Η συνάρτηση θα είναι τύπου void και θα εμφανίζει τις τιμές που θα
παίρνει κάθε φορά το n πριν και μετά από κάθε κλήση της συνάρτησης*/


 void recnminus2(int n)
 {
     if ( n != 0 )
      {
      printf("n before call recnminus2 = %d\n", n);
      recnminus2(n-2);
      printf("n after call recnminus2 = %d\n", n);
      }
 }

   main()
    {
     int  n;

     // Διάβασμα n
     do
     {
        printf("Give n >= 5 : ");
        scanf("%d", &n);
     }
     while ( n < 5 );


     // Κλήση Συνάρτησης recnminus2(int n)
     recnminus2(n);

     system("Pause");
 }
```

Έξοδος Προγράμματος :

```
Give n >= 5 : 6
n before call recnminus2 = 6
n before call recnminus2 = 4
n before call recnminus2 = 2
n after call recnminus2 = 2
n after call recnminus2 = 4
n after call recnminus2 = 6
Press any key to continue . . .
```

Παρατήρηση

Με κάθε κλήση της συνάρτησης το `n` παίρνει μια τιμή μικρότερη κατά `2` πριν την επόμενη αναδρομική κλήση, οπότε εμφανίζεται το `6,4,2` (με το `n = 0` τερματίζει η τελευταία κλήση), ενώ με την εντολή εμφάνισης μετά την κλήση εμφανίζονται οι τιμές του `n` που παίρνει στην τελευταία κλήση, μετά στην προ-τελευταία κ.λ.π., `2,4,6`. Χρειάζεται ιδιαίτερη προσοχή στον έλεγχο τερματισμού, όπως φαίνεται και στο επόμενο τρέξιμο του προγράμματος, η οποία έχει σαν αποτέλεσμα να εξαντληθεί όλη η διαθέσιμη μνήμη :

Έξοδος Προγράμματος :

```
Give n >= 5 : 5
n before call recnminus2 = 5
n before call recnminus2 = 3
n before call recnminus2 = 1
n before call recnminus2 = -1
n before call recnminus2 = -3
n before call recnminus2 = -5
...
```