Α.Τ.Ε.Ι. Θεσσαλονίκης
Τμήμα Πληροφορικής

Εργαστηριακές Ασκήσεις Αριθμητικής Ανάλυσης

στη Γλώσσα Προγραμματισμού C

Γουλιάνας Κώστας
Επίκουρος Καθηγητής Α.Τ.Ε.Ι.Θ

Θεσσαλονίκη 2007

Email: gouliana@it.teithe.gr
Ιστοσελίδα : www.it.teithe.gr/~gouliana

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1 Πινακοποίηση Συνάρτησης

Απλό if
If…else
While
Do …While
For
Functions

- Στο Κεφάλαιο που ακολουθεί παρουσιάζονται οι βασικές εντολές της γλώσσας προγραμματισμού C.

Πρόβλημα

- Δίνεται η πραγματική συνάρτηση μιας πραγματικής μεταβλητής :

  $y = f(x) | x \in [a,b]$, $a, b, x, y \in \Re$

- Να γίνει πρόγραμμα C, το οποίο να δημιουργεί πίνακα τιμών για τη συνάρτηση αυτή όταν :

α) Οι τιμές του `x` είναι τυχαίες στο διάστημα `[a,b]` δοσμένες κατά αύξουσα (φθίνουσα) τάξη.

β) Το διάστημα `[a, b]` είναι ισοδιάστατα διηρημένο ($x : a(h)b$)

1.1 Πινακοποίηση Συνάρτησης Μιας Μεταβλητής - if

α) Αλγόριθμος (1ος Τρόπος)

1) **Διαβάζουμε** τα άκρα του διαστήματος `a, b` και το βήμα `h`.
2) Δίνουμε **αρχική** τιμή στο `x = a`.
3) **Αν** το `x` έχει ξεπεράσει την τιμή του δεξιού άκρου (`b`) του διαστήματος, το πρόγραμμα **τερματίζει**.
4) **Διαφορετικά** :

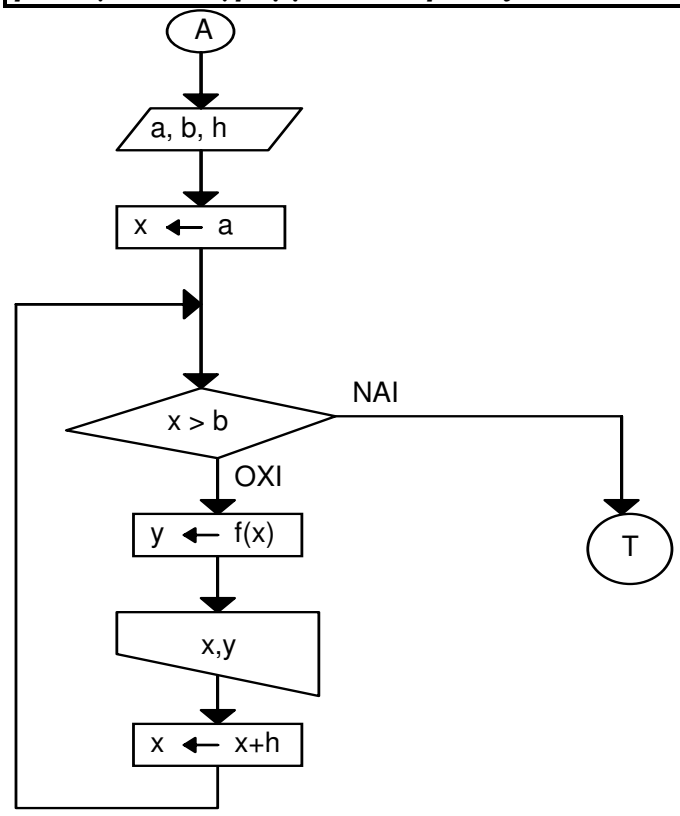
a) **Υπολογίζουμε** το `y = f(x)`.

b) **Εμφανίζουμε** τις τιμές των `x, y`.

c) **Αυξάνουμε** το `x` κατά το βήμα `h`.

d) **Πηγαίνουμε** στο βήμα 3.

β) Λογικό Διάγραμμα (1ος Τρόπος)



γ) Πρόγραμμα (1ος Τρόπος)

```
/*  Άσκηση 1.1
  Πινακοποίηση Συνάρτησης μιας Μεταβλητής
   y = f( x )=1/( 1+x^2 )  */
#include <stdio.h>
#include <stdlib.h>

main()
{
   float a,b,h,x,y;
   printf("Askhsh 1.1\n");
   printf("Pinakopoihsh Synarthshs me if…goto\n");
   printf("\n");
   //
   // Διαβάζουμε τα άκρα του διαστήματος a, b και το βήμα h
   //
   printf("Dose times gia ta a,b,h : ");
   scanf ("%f  %f  %f",&a,&b,&h);
   //
   // Δίνουμε αρχική τιμή στο x = a.
   //
   x = a;
   loop:
   if ( x > b )  goto telos; // το πρόγραμμα τερματίζει
   //
   // Υπολογίζουμε το y = f(x)
   //
   y=1/(1+x*x);
   //
   // Εμφανίζουμε τις τιμές των x, y
   //
   printf("x = %12.8f      y = %5.2f\n", x, y);
   //
   // Αυξάνουμε το x κατά το βήμα h
   //
   x = x + h;
   goto loop;// Πηγαίνουμε στο loop
telos:
   system("Pause");

}
```

δ) Δομή ενός Προγράμματος C

- Ένα πρόγραμμα σε C αποτελείται απ' τα παρακάτω τμήματα :

	Τμήμα Προγράμματος
/* Άσκηση 1.1 Πινακοποίηση Συνάρτησης μιας Μεταβλητής $y = f(x) = 1/(1+x^2)$ */	**Σχόλιο**
#include <stdio.h> #include <stdlib.h>	**Προαιρετικές Εντολές #include**
main()	**Δήλωση Προγράμματος**
{	**Αρχή block Εντολών**
float a,b,h,x,y;	**Δήλωση Τοπικών Μεταβλητών**

printf("Askhsh 1.1\n");
printf("Pinakopoihsh Synarthshs me if…goto\n");
printf("\n");
//
// Σχόλια
//
printf("Dose times gia ta a,b,h : ");
scanf ("%f %f %f",&a,&b,&h);
x = a;
loop:
if (x > b) goto telos;// Σχόλια
y=1/(1+x*x);
printf("x=%12.8f y=%5.2f\n", x, y);
x = x + h;
goto loop;
telos:
system("Pause");
	Εντολές
	
	Χωρισμένες με ;

}	**Τέλος block Εντολών**

ε) Αλγόριθμος – Εντολές

Εντολές	Ενέργεια
/* Άσκηση 1.1 Πινακοποίηση Συνάρτησης μιας Μεταβλητής $y = f(x) = 1/(1+x^2)$ */	**Σχόλια**
#include <stdio.h> #include <stdlib.h>	**Προαιρετικές Εντολές #include**
main()	**Δήλωση Προγράμματος**
float a,b,h,x,y;	**Δηλώσεις Μεταβλητών**

printf("Askhsh 1.1\n");
printf("Pinakopoihsh Synarthshs me if…goto\n");
printf("\n");
	Εμφανίζουμε τον Αριθμό Άσκησης

printf ("Dose times gia ta a,b,h:");	**Εμφανίζουμε Μήνυμα στην Οθόνη**
scanf ("%f %f %f", &a, &b, &h);	**Διαβάζουμε τα άκρα του διαστήματος a, b και το βήμα h**
x = a;	**Δίνουμε αρχική τιμή στο x = a**
loop: if (x > b) goto telos;	**Αν το x έχει ξεπεράσει την τιμή του δεξιού άκρου (b) του διαστήματος, το πρόγραμμα τερματίζει.**
y=1/(1+x*x);	**Υπολογίζουμε το f(x)**
printf("x=%12.8f y=%5.2f\n", x, y);	**Εμφανίζουμε τις τιμές των x, y**
x = x + h;	**Αυξάνουμε το x κατά το βήμα h**
goto loop;	**Πηγαίνουμε στο label loop**
telos: system("Pause");	**Τερματισμός Προγράμματος**
}	**Τέλος εντολών Προγράμματος**

1.1.1 Σχόλια

- Τα σχόλια γράφονται σε οποιοδήποτε σημείο του προγράμματος και μπορεί να καταλαμβάνουν οσεσδήποτε γραμμές, αρκεί να υπάρχουν οι οριοθέτες /* και */. Η σύνταξή τους γενικά είναι : /* <Σχόλια> */. Αν θέλουμε σχόλια στο τέλος μιας εντολής, γράφουμε μετά την εντολή : // <Σχόλια>.

Παράδειγμα 1.1.1

```
/*  Άσκηση 1.1
  Πινακοποίηση Συνάρτησης μιας Μεταβλητής
   y = f(x) = 1/(1+x^2)  */
```

1.1.2 Ενσωμάτωση Αρχείων

❖ 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**.

❖ 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 συνάρτηση `system("Pause")` που χρησιμοποιείται για το «πάγωμα» της οθόνης μετά την εμφάνιση των αποτελεσμάτων απαιτούν την ενσωμάτωση του αρχείου `stdlib.h`, το οποίο γίνεται με την εντολή :

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

1.1.3 Δήλωση Προγράμματος

❖ Η κύρια δομική μονάδα ενός προγράμματος στη C είναι η συνάρτηση (function), η οποία λειτουργεί σαν τις κλασσικές functions άλλων γλωσσών ή τις μεθόδους της Java (επιστρέφοντας 1 τιμή ή καμία) και σαν τα procedures ή subroutines. Η συνάρτηση που αποτελεί και το κύριο πρόγραμμα είναι η συνάρτηση main(). Δε χρειάζεται να επιστρέφει κάποιον τύπο ούτε ορίσματα, οπότε η γενική σύνταξη της εντολής μπορεί να είναι :

```
main()
{
σώμα...
}
```

1.1.4 Δηλώσεις Μεταβλητών

❖ Οι δηλώσεις μεταβλητών γίνονται στην αρχή κάθε συνάρτησης ή στην αρχή ενός block εντολών, μεταξύ { και } και γίνονται με τις εντολές :

```
int <κατάλογος_μεταβλητών>;
long int <κατάλογος_μεταβλητών>;
float <κατάλογος_μεταβλητών>;
double <κατάλογος_μεταβλητών>;
long double <κατάλογος_μεταβλητών>;
char <κατάλογος_μεταβλητών>;
```

❖ Η C υποστηρίζει τους παρακάτω τύπους δεδομένων :

Τύποι Δεδομένων	**Δήλωση**	Είδος Δεδομένων
Χαρακτήρες (char)	**Char** **signed char** **unsigned char**	Η μεταβλητή αποθηκεύει ένα χαρακτήρα μεγέθους 8 bits
Ακέραιοι (integers)	**short int** **signed short int** **unsigned short int** **signed int** **unsigned int** **long int** **signed long int** **unsigned long int**	Η μεταβλητή αποθηκεύει ακεραίους αριθμούς διαφόρων μεγεθών, προσημασμένους ή μη προσημασμένους.
Αριθμοί Κινητής Υποδιαστολής (floating point)	**float** **double** **long double**	Η μεταβλητή αποθηκεύει αριθμούς κινητής υποδιαστολής με mantissa και εκθέτη διαφόρων μεγεθών απλής ή διπλής ακρίβειας.

Παρατηρήσεις

❖ Μια μεταβλητή μπορεί να περιέχει οποιοδήποτε κεφαλαίο ή πεζό χαρακτήρα, ένα αριθμητικό ψηφίο (0 μέχρι 9), και το χαρακτήρα κάτω παύλα (_). Ο πρώτος χαρακτήρας της μεταβλητής δεν μπορεί να είναι αριθμητικό ψηφίο ή κάτω παύλα. Τα ονόματα των μεταβλητών είναι διαφορετικά με κεφαλαία ή μικρά. Π.χ.

```
float a,b,h,x,y;
```

❖ Με τη δήλωση μιας μεταβλητής μπορούμε να της δώσουμε και αρχική τιμή. Π. χ,

```
float pi = 3.14159265358979;
```

❖ Όταν δίνουμε τιμή σε μια μεταβλητή τύπου **long**, στο τέλος της τιμής προσθέτουμε το χαρακτήρα **L** ή **l** . Π.χ.

```
long int count, sum = 0L;
```

❖ Όταν δίνουμε τιμή σε μια μεταβλητή κινητής υποδιαστολής (τύπου **float** ή **double**), μπορεί να χρησιμοποιηθεί η εκθετική μορφή, με το **E** κεφαλαίο ή μικρό. Π.χ.

```
float x, y, z = 123.45e-6;

double light_speed = 2.99792577389E8;
```

❖ Στον επόμενο πίνακα φαίνονται το **Μέγεθος** και η **Κλίμακα Τιμών** των διαφόρων τύπων μεταβλητών.

Τύπος	**Μέγεθος**	**Κλίμακα Τιμών**
unsigned char	8 bits	0 έως 255
char	8 bits	-128 έως 127
unsigned int	16 bits	0 έως 65,535
short int	16 bits	-32,768 έως 32,767
Int	16 bits	-32,768 έως 32,767
unsigned long	32 bits	0 έως 4,294,967,295
long	32 bits	-2,147,483,648 έως 2,147,483,647
float	32 bits	1.17549435 * (10^-38) έως 3.40282347 * (10^+38)
double	64 bits	2.2250738585072014 * (10^-308) έως 1.7976931348623157 * (10^+308)
long double	80 bits	3.4 * (10^-4932) έως 1.1 * (10^4932)

1.1.4.1 Εκτέλεση Αριθμητικών Πράξεων (Αριθμητικοί Τελεστές)

- Οι τελεστές για τις αριθμητικές εκφράσεις της C είναι :

Τελεστής	***Πράξη***
+	Πρόσθεση
-	Αφαίρεση
*****	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο Διαίρεσης (mod)

Παρατηρήσεις

♦ Στις διάφορες πράξεις μεταξύ των μεταβλητών πρέπει να χρησιμοποιούμε τον ίδιο τύπο μεταβλητών, διαφορετικά μπορεί να προκύψουν απρόσμενα αποτελέσματα. Οι κανόνες που ισχύουν είναι :

1. Πράξεις μεταξύ ακεραίων αριθμών παράγουν ακέραιο.

2. Πράξεις μεταξύ πραγματικών αριθμών παράγουν πραγματικό αριθμό.

3. Πράξεις μεταξύ ακεραίων και πραγματικών αριθμών παράγουν πραγματικό αριθμό.

4. Αν αποθηκεύσουμε το αποτέλεσμα μιας πράξης μεταξύ ακεραίων αριθμών σε μια πραγματική μεταβλητή, ο αριθμός γίνεται πραγματικός.

5. Αν αποθηκεύσουμε το αποτέλεσμα μιας πράξης μεταξύ πραγματικών αριθμών σε μια ακέραια μεταβλητή, αποκόπτεται το δεκαδικό μέρος.

6. Μπορούν να αποθηκευθούν ακέραιες τιμές σε μεταβλητές τύπου `char` (οπότε αποθηκεύεται ο ASCII κώδικας του αντίστοιχου χαρακτήρα) ή χαρακτήρες σε ακέραιες μεταβλητές. Π.χ. η δήλωση :

   ```
   char zero = 48;
   ```

   έχει σαν αποτέλεσμα να αποθηκευθεί ο χαρακτήρας `'0'` με ASCII κώδικα = `48` στη μεταβλητή χαρακτήρα zero.

   Η δήλωση :

   ```
   int number = '0';
   ```

   έχει σαν αποτέλεσμα να αποθηκευθεί στην ακέραια μεταβλητή number ο αριθμός `48` που είναι ο ASCII κώδικας του χαρακτήρα `'0'`.

❖ Μπορούμε να μετατρέψουμε τα δεδομένα ενός τύπου σε δεδομένα άλλου τύπου, χωρίς να αλλάξουμε το περιεχόμενο της μεταβλητής στην οποία είναι αποθηκευμένα. Π. χ. με την εντολή

```
f = (float)n;
```

Η τιμή της ακέραιας μεταβλητής `n` μετατρέπεται σε πραγματικό αριθμό και αποθηκεύεται στην πραγματική μεταβλητή `f`, χωρίς να αλλάξει το περιεχόμενο της μεταβλητής `n`.

1.1.4.2 Τελεστής Αντικατάστασης

- Ο τελεστές αντικατάστασης – ανάθεσης τιμής σε μια μεταβλητή είναι το "**=**". Π.χ.

```
x = a;
```

Παρατηρήσεις

❖ Με τη χρήση των Αριθμητικών Τελεστών μπορούμε να χρησιμοποιήσουμε τους **Συντομευμένους τελεστές αντικατάστασης**, όπως φαίνονται στον επόμενο πίνακα:

Τελεστής	*Έκφραση*	*Αποτέλεσμα*
+=	`x+=y`	Πρόσθεση του y στο x και το αποτέλεσμα στο x
−=	`x−=y`	Αφαίρεση του y απ' το x και το αποτέλεσμα στο x
***=**	`x *=y`	Πολλαπλασιασμός του y με το x και το αποτέλεσμα στο x
/=	`x /=y`	Διαίρεση του x με το y και το αποτέλεσμα στο x
%=	`x %=y`	Διαίρεση του x με το y και το Υπόλοιπο της Διαίρεσης στο x

1.1.4.3 Τελεστές Αύξησης και Μείωσης

- Οι τελεστές **++** και `−−` αυξάνουν ή μειώνουν κατά `1` την τιμή της έκφρασης στην οποία ενεργούν. Π. χ. η εντολή

```
x++;
```

έχει σαν αποτέλεσμα να αυξηθεί το `x` κατά `1` και ισοδυναμεί με την εντολή

```
x = x + 1;
```

ενώ η εντολή

```
x--;
```

έχει σαν αποτέλεσμα να μειωθεί το `x` κατά `1` και ισοδυναμεί με την εντολή

```
x = x - 1;
```

Παρατηρήσεις

- Οι τελεστές ++ και `−−` μπορούν να είναι **πριν** ή **μετά** την έκφραση στην οποία ενεργούν.

- Αν οι τελεστές ++ και `−−` είναι **πριν** την έκφραση στην οποία ενεργούν, **πρώτα** αυξάνεται ή μειώνεται η τιμή της έκφρασης και **μετά** χρησιμοποιείται η νέα τιμή. Π. χ. στις εντολές

```
a = 0;
b = ++a;
```

πρώτα αυξάνεται το `a` κατά `1` (οπότε γίνεται `1`) και το `b` παίρνει την ίδια τιμή `1`.

- Αν οι τελεστές ++ και -- είναι **μετά** την έκφραση στην οποία ενεργούν, **πρώτα** χρησιμοποιείται η αρχική τιμή της έκφρασης και **μετά** αυξάνεται ή μειώνεται η τιμή της έκφρασης κατά 1. Π. χ. στις εντολές

```
a = 0;
b = a++;
```

πρώτα το `b` παίρνει την τιμή του `a = 0`, και **μετά** το `a` αυξάνεται κατά 1 (οπότε γίνεται 1).

1.1.4.4 Λογικοί Τελεστές (Boolean)

Οι λογικοί τελεστές ενεργούν πάνω σε αριθμητικά δεδομένα θεωρώντας κάθε **μη μηδενική** τιμή σαν αλήθεια (**true**) και επιστρέφουν την τιμή 1, ενώ την τιμή 0 σαν ψευδή (**false**) και επιστρέφουν την τιμή 0. Οι λογικοί τελεστές είναι :

Τελεστής	***Έκφραση***	***Αποτέλεσμα***
&&	***<έκφραση1> && <έκφραση2>***	Λογικό ΚΑΙ στην ***<έκφραση1>*** και ***<έκφραση2>***. Επιστρέφει 1, αν και οι 2 εκφράσεις είναι αληθείς, διαφορετικά 0.
||	***<έκφραση1> && <έκφραση2>***	Λογικό Ή στην ***<έκφραση1>*** και ***<έκφραση2>***. Επιστρέφει 0, αν και οι 2 εκφράσεις είναι ψευδείς, διαφορετικά 1.

Παρατηρήσεις

Κατά την εκτέλεση των λογικών πράξεων που εκτελούνται από αριστερά προς τα δεξιά, δεν υπολογίζονται οι τιμές όλων των παραστάσεων, αλλά μόνο όσων είναι απαραίτητες. Π.χ. στην έκφραση

```
A && B
```

αν η παράσταση `A` είναι ψευδής, δεν υπολογίζεται η παράσταση `B`, ενώ στην έκφραση

```
A || B
```

αν η παράσταση `A` είναι αληθής, δεν υπολογίζεται η παράσταση `B`.

1.1.4.5 Προτεραιότητα Τελεστών

Όταν υπάρχουν παρενθέσεις σε μια έκφραση, υπολογίζονται πρώτα αυτές. Η προτεραιότητα των τελεστών κατά αύξουσα τάξη φαίνεται στον επόμενο πίνακα :

Τελεστής	Είδος
!	Λογικό ΟΧΙ
++ --	Τελεστές αύξησης - μείωσης
* / %	Τελεστές πολλαπλασιασμού - διαίρεσης
+ -	Τελεστές πρόσθεσης - αφαίρεσης.
< > <= >=	Τελεστές ανισότητας
== !=	Τελεστές ισότητας
&&	Λογικό ΚΑΙ
||	Λογικό Ή
?:	Conditional.
= += -=	Καταχώρηση.

1.1.5 Εισαγωγή Δεδομένων – Εμφάνιση Αποτελεσμάτων

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) .

1.1.5.1 Εμφάνιση Αποτελεσμάτων 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
%d	Εμφανίζει την τιμή της **ακέραιας** μεταβλητής στο δεκαδικό σύστημα
%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.ddddddE+ddd`

Παράδειγμα

Με τις παρακάτω εντολές για την εμφάνιση ενός ακεραίου (μεταβλητή `i`) και του πραγματικού αριθμού `0.1234567` σαν πραγματικό αριθμό απλής και διπλής ακριβείας (μεταβλητές `sp` και `dp`) και σε εκθετική μορφή (μεταβλητή `sp`) :

```
int i = 5;
float sp = 0.1234567;
double dp = 0.1234567;
printf("i = %d, sp = %f, dp = %f, fp = %e \n", i, sp, dp, sp);
```

θα εμφανιστεί :

```
i = 5, sp = 0.123457, dp = 0.123457, fp = 1.234567e-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 `0.1234567` με `7` 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`.

1.1.5.2 Εμφάνιση Αποτελεσμάτων με τη Μορφή που Επιθυμεί ο Προγραμματιστής

- 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 :

```
int i = 5;
float sp = 0.1234567;
double dp = 0.1234567;
printf("i=%5d, sp=%16.12f, dp=%16.12lf, fp=%e\n",i, sp, dp, sp);
```

θα εμφανιστεί :

```
i=^^^^5, sp=^^0.123456701636, dp=^^0.123456700000, fp=1.234567e-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 (`16` στο παράδειγμα), με τόσα δεκαδικά ψηφία μετά την υποδιαστολή όσο και ο δεύτερος αριθμός (`12` 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 (π.χ. `^^0.123456700000` για τη μεταβλητή διπλής ακρίβειας, ενώ η μεταβλητή απλής ακρίβειας εμφανίζει το `^^0.123456701636`, επειδή ο αριθμός `0.1234567` αποθηκεύεται με σφάλμα).

- 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 :

```
int i = 5;
float sp = 0.1234567;
double dp = 0.1234567;
printf("i=%-5d, sp=%-16.12f, dp=%-16.12lf, fp=%e\n",i,sp,dp,sp);
```

θα εμφανιστεί :

```
i=5^^^^, sp=0.123456701636^^, dp=0.123456700000^^, fp=1.234567e-001
```

1.1.5.3 Χαρακτήρες Ελέγχου

- 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.

1.1.5.4 Εισαγωγή Δεδομένων με Προκαθορισμένη Μορφή

- Η γενική σύνταξη της εντολής `scanf` είναι :

```
scanf ("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**, 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.

1.1.6 Η Εντολή `goto`

- Με την εντολή `goto`, ο έλεγχος πηγαίνει σε κάποιο σημείο του προγράμματος, το οποίο καθορίζει ο αριθμός εντολής με τον οποίο συντάσσεται η εντολή `goto`. Η σύνταξη της εντολής `goto` είναι :

```
goto <label>
```

όπου :

`<label>` = όνομα ετικέτας

και συνήθως χρησιμοποιείται για την κατασκευή βρόχων ή για τη διακοπή της εκτέλεσης κάποιων εντολών.

Παράδειγμα

- Με την εντολή του προγράμματος :

```
goto loop;
```

ο έλεγχος πηγαίνει στην εντολή που έχει label `loop` (την εντολή `if`) και χρησιμοποιείται για ανακύκλωση.

1.1.7 Τελεστές Σύγκρισης

- Η C διαθέτει τους παρακάτω Τελεστές Σύγκρισης :

Τελεστής	Σύμβολο Πληκτρολογίου
≤	<=
<	<
=	==
≠	!=
≥	>=
>	>

1.1.8 Η Εντολή `if`

- Με την εντολή `if` ελέγχεται κάποια συνθήκη και αν ισχύει, εκτελείται μια εντολή ή εντολές. Στο παράδειγμα χρησιμοποιείται για τον τερματισμό των επαναλήψεων. Η γενική σύνταξη είναι :

```
if (<συνθήκη>) <εντολή>;
```

```
if (<συνθήκη>)
{
    <εντολές>;
}
```

Παράδειγμα

```
if ( x > b ) goto telos;
```

- Με την προηγούμενη εντολή, το πρόγραμμα τερματίζει, αν `x > b`.

1.1.9 Τερματισμός Προγράμματος (Τέλος εντολών ή return)

- Το πρόγραμμα τερματίζει όταν τελειώσουν οι εντολές με το τελευταίο "}", όταν η κύρια συνάρτηση `main()` δηλώνεται χωρίς τύπο ή με τύπο void ή με την εντολή **`return`**. Η εντολή **`return`** μπορεί να εμφανίζεται σε πολλά σημεία του προγράμματος.

1.2 Πινακοποίηση Συνάρτησης Μιας Μεταβλητής (2ος Τρόπος – if…else)

α) Αλγόριθμος (2ος Τρόπος)

1) **Διαβάζουμε** τα άκρα του διαστήματος `a, b` και το βήμα `h`.

2) Δίνουμε **αρχική** τιμή στο `x = a`.

3) **Αν** το `x` έχει ξεπεράσει την τιμή του δεξιού άκρου (`b`) του διαστήματος, το πρόγραμμα **τερματίζει**.

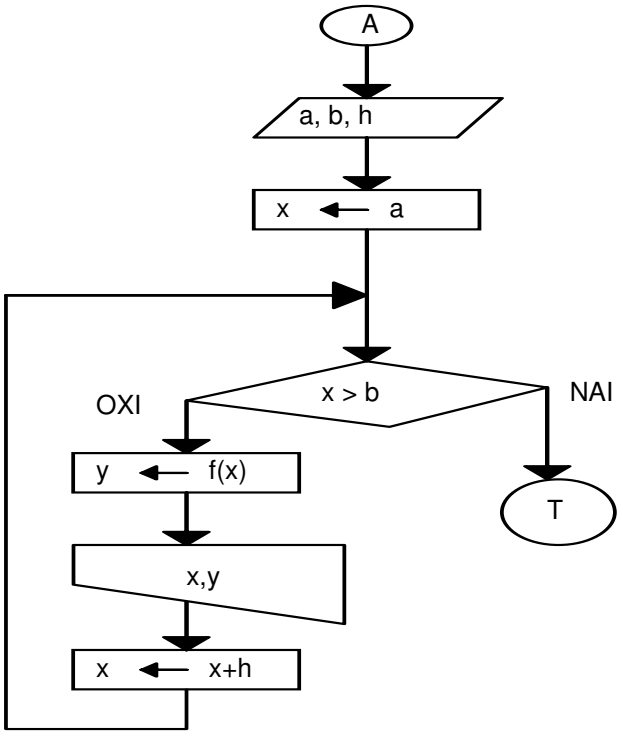
4) **Διαφορετικά**, αν το `x` είναι ≤ `b`, τότε :

a) **Υπολογίζουμε** το `y=f(x)`.

b) **Εμφανίζουμε** τις τιμές των `x, y`.

c) **Αυξάνουμε** το `x` κατά το βήμα `h` και **πηγαίνουμε** στο βήμα 3.

β) Λογικό Διάγραμμα (2ος Τρόπος)



γ) Πρόγραμμα (2ος Τρόπος)

```
/*  Άσκηση 1.2
  Πινακοποίηση Συνάρτησης μιας Μεταβλητής
   y = f(x)=1/(1+x^2)  */
#include <stdio.h>
#include <stdlib.h>
main()
{
   float a,b,h,x,y;
   printf("Askhsh 1.2\n");
   printf("Pinakopoihsh Synarthshs me if…else\n");
   printf("\n");
   //
   // Διάβασμα άκρων του διαστήματος a, b και βήματος h
   //
   printf("Dose times gia ta a,b,h : ");
   scanf ("%f  %f  %f",&a,&b,&h);
   x = a; // Αρχική Τιμή στο x
   loop:
   if ( x > b )  goto telos;
   else
    {
    y=1/(1+x*x); // Υπολογισμός y = f(x)
    printf("x = %12.8f      y = %5.2f\n", x, y); // Εμφάνιση x, y
    x = x + h; // Αύξηση x κατά h
    goto loop; // Πηγαίνουμε στο loop
    }
   telos:
    system("Pause");
}
```

1.2.1 if...else

- Με τη χρήση του `if…else`, μπορούμε να εκτελέσουμε ένα block εντολών, ανάλογα με το αν ισχύει κάποια συνθήκη ή όχι. Η σύνταξή του είναι :

```
if (<συνθήκη>) <εντολή>; ή { <εντολές>; }
else
     <εντολή>; ή { <εντολές>; }
```

Παράδειγμα

```
if ( x > b )  goto telos;
    else
    {
    y=1/(1+x*x); // Υπολογισμός y = f(x)
    printf("x = %12.8f      y = %5.2f\n", x, y); // Εμφάνιση x, y
    x = x + h; // Αύξηση x κατά h
    goto loop; // Πηγαίνουμε στο loop
    }
```

1.3 Πινακοποίηση Συνάρτησης Μιας Μεταβλητής (3ος Τρόπος - while)

α) Αλγόριθμος (3ος Τρόπος)

1) **Διαβάζουμε** τα άκρα του διαστήματος `a, b` και το βήμα `h`.

2) Δίνουμε **αρχική** τιμή στο `x = a.`

3) **Για όσο** το `x` δεν έχει ξεπεράσει την τιμή του δεξιού άκρου (`b`) του διαστήματος :

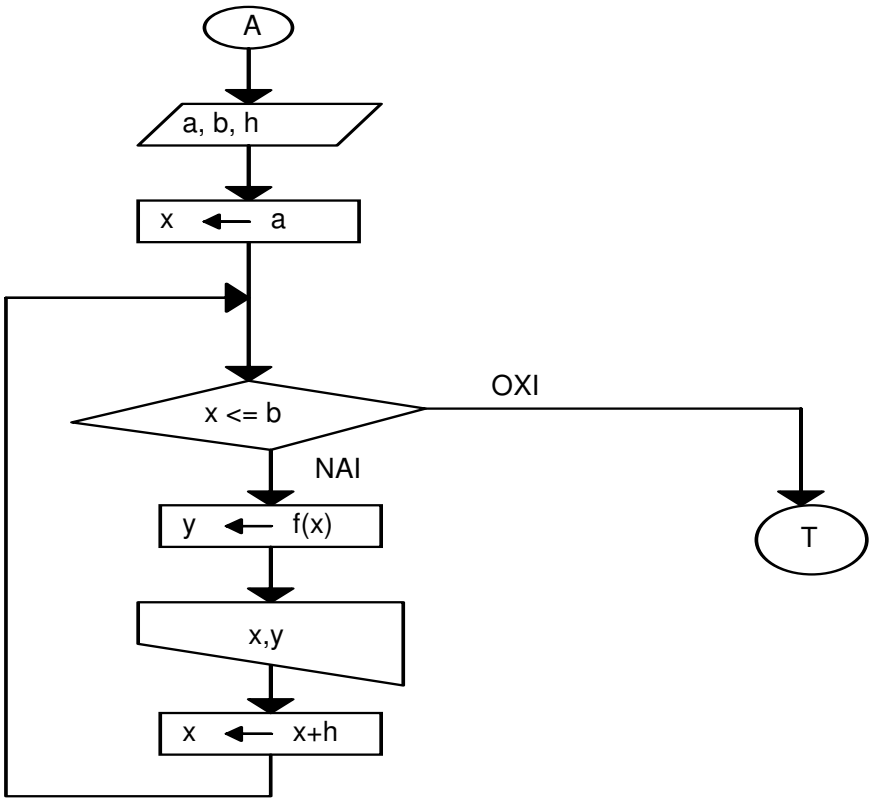
   a) **Υπολογίζουμε** το `y=f(x).`

   b) **Εμφανίζουμε** τις τιμές των `x, y`.

   c) **Αυξάνουμε** το `x` κατά το βήμα `h`.

4) **Όταν** το `x` ξεπεράσει την τιμή του `b` το πρόγραμμα **τερματίζει**.

β) Λογικό Διάγραμμα (3ος Τρόπος)



γ) Πρόγραμμα (3ος Τρόπος)

```
/*  Άσκηση 1.3
  Πινακοποίηση Συνάρτησης μιας Μεταβλητής με while
   y = f(x)=1/(1+x^2)  */
#include <stdio.h>
#include <stdlib.h>
main()
{
   float a,b,h,x,y;
   printf("Askhsh 1.3\n");
   printf("Pinakopoihsh Synarthshs me while\n");
   printf("\n");
   //
   // Διάβασμα άκρων του διαστήματος a, b και βήματος h
   //
   printf("Dose times gia ta a,b,h : ");
   scanf ("%f  %f  %f",&a,&b,&h);
   x = a; // Αρχική Τιμή στο x
   while (x <= b)
     {
     y=1/(1+x*x); // Υπολογισμός y = f(x)
     printf("x = %12.8f     y = %5.2f\n", x, y); // Εμφάνιση x, y
     x = x + h; // Αύξηση x κατά h
     }
   system("Pause");
}
```

1.3.1 Η Εντολή `while`

- Με το `while` εκτελούνται κάποιες εντολές **για όσο** μια συνθήκη είναι αληθής. Η σύνταξη της εντολής είναι :

```
while ( <συνθήκη> )
   <εντολή>;
```

```
while ( <συνθήκη> )
   {
      εντολές…
   }
```

Παρατηρήσεις

- Ο έλεγχος της συνθήκης γίνεται **πριν** την εκτέλεση της εντολής ή των εντολών.
- Αν η συνθήκη είναι εξ αρχής **ψευδής**, οι εντολές δεν εκτελούνται **καμιά** φορά.
- Οι εντολές εκτελούνται **για όσο** η συνθήκη είναι **αληθής** και **παύουν** να εκτελούνται, όταν η συνθήκη γίνει **ψευδής**.
- Η **μεταβλητή** που ελέγχεται στη συνθήκη θα πρέπει να **έχει** τιμή ***πριν το while***, για να εκτελεστούν οι εντολές τουλάχιστον μια φορά και να **αλλάζει** τιμή ***μέσα στο while***, για να τερματίσει ο βρόχος.

- Ο έλεγχος της συνθήκης μπορεί να γίνεται **μετά** την εκτέλεση των εντολών του `while`, οπότε οι εντολές εκτελούνται **τουλάχιστον** μια φορά. Σ' αυτή την περίπτωση, η σύνταξη της εντολής while που αποτελεί μια μορφή της εντολής **repeat…until** θα είναι :

```
do
    <εντολή>
while ( <συνθήκη> );
```

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

και το πρόγραμμα 1.3 θα είναι :

```
/*  Άσκηση 1.3a
  Πινακοποίηση Συνάρτησης μιας Μεταβλητής με  do … while
  y = f(x)=1/(1+x^2)  */

#include <stdio.h>
#include <stdlib.h>
main()
{
   float a,b,h,x,y;
   printf("Askhsh 1.3a\n");
   printf("Pinakopoihsh Synarthshs me do … while\n");
   printf("\n");
   //
   // Διάβασμα άκρων του διαστήματος a, b και βήματος h
   //
   printf("Dose times gia ta a,b,h : ");
   scanf ("%f  %f  %f",&a,&b,&h);
   x = a; // Αρχική Τιμή στο x
   do
     {
     y=1/(1+x*x); // Υπολογισμός y = f(x)
     printf("x = %12.8f      y = %5.2f\n", x, y); //Εμφάνιση x, y
     x = x + h; // Αύξηση x κατά h
     }
   while (x <= b);
   system("Pause");

}
```

1.4 Πινακοποίηση Συνάρτησης Μιας Μεταβλητής - for)

α) Αλγόριθμος (4ος Τρόπος)

1) **Διαβάζουμε** τα άκρα του διαστήματος `a, b` και το βήμα `h`.

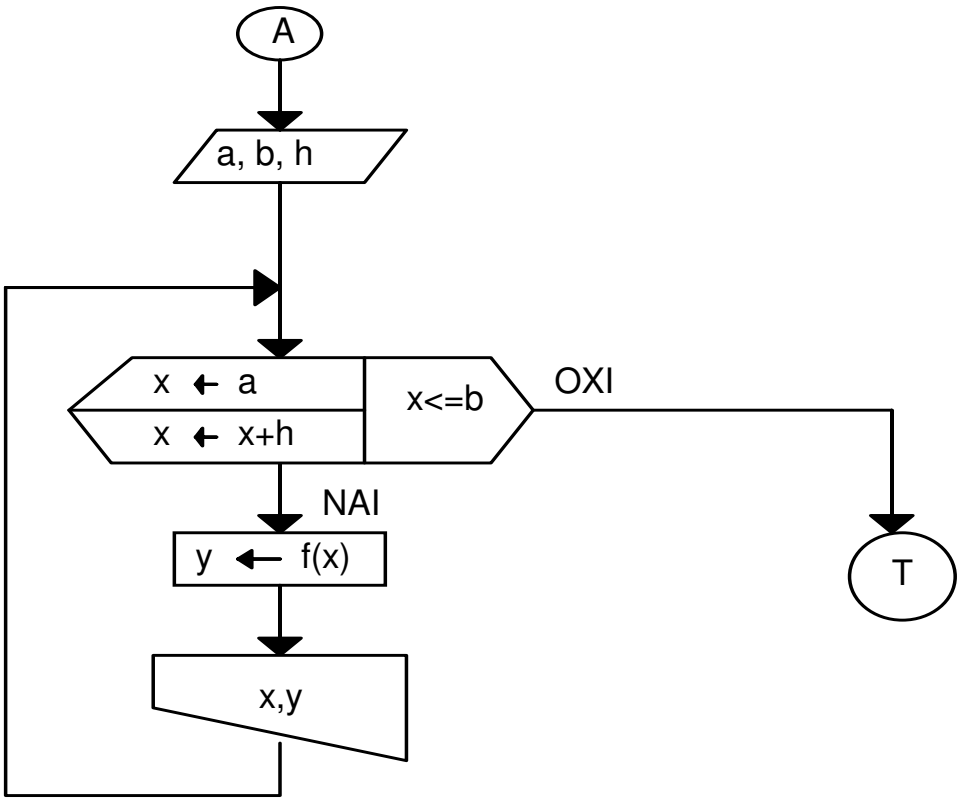
2) **Για** τιμές του `x` απ' το `a` μέχρι το `b` με βήμα `h` :

   a) **Υπολογίζουμε** το `y=f(x)` **.**

   b) **Εμφανίζουμε** τις τιμές των `x, y`.

3) **Όταν** το `x` ξεπεράσει την τιμή του δεξιού άκρου (`b`) του διαστήματος το πρόγραμμα **τερματίζει**.

β) Λογικό Διάγραμμα (4ος Τρόπος)



γ) Πρόγραμμα (4ος Τρόπος)

```
/*  Άσκηση 1.4
  Πινακοποίηση Συνάρτησης μιας Μεταβλητής Με for
   y = f(x)=1/(1+x^2)  */
#include <stdio.h>
#include <stdlib.h>
main()
{
    float a,b,h,x,y;
    printf("Askhsh 1.4 - Pinakopoihsh Synarthshs me for\n");
    printf("\n");
    //
    // Διάβασμα άκρων του διαστήματος a, b και βήματος h
    //
    printf("Dose times gia ta a,b,h : ");
    scanf ("%f  %f  %f",&a,&b,&h);
    for (x = a; x <= b; x = x + h) {
     y=1/(1+x*x); // Υπολογισμός y = f(x)
     printf("x = %12.8f      y = %5.2f\n", x, y); // Εμφάνιση x, y
     }
    system("Pause");
}
```

1.4.1 Ανακύκλωση με την εντολή for

- Όταν ξέρουμε τον αριθμό των επαναλήψεων, χρησιμοποιούμε την εντολή **for**, η οποία μπορεί να εμφανιστεί, στη γενική της μορφή, με τον παρακάτω τύπο :

```
for ( < έκφραση1>;< έκφραση2>; < έκφραση3> )
     εντολή;

for ( < έκφραση1>;< έκφραση2>; < έκφραση3> )    {
     εντολές;
     }
```

Παρατηρήσεις

- Η **< έκφραση1>** υπολογίζεται πριν την πρώτη επανάληψη.
- Μετά από κάθε επανάληψη υπολογίζεται η **< έκφραση3>**.
- Οι εντολές εκτελούνται για όσο η **<έκφραση2>** είναι αληθής (έχει τιμή 1) και μέχρι να γίνει ψευδής (τιμή 0).
- Ο έλεγχος για το αν ισχύει η **<έκφραση2>** γίνεται πριν την εκτέλεση των εντολών.
- Μπορούν να χρησιμοποιηθούν οι συντομευμένοι τελεστές. Π.χ.

```
for (x = a; x <= b; x +=h)
```

♦ Μπορούν να χρησιμοποιηθούν οι τελεστές αύξησης και μείωσης. Π.χ. η εντολή

```
for (x = 1; x <= 10; x ++)
```

ισοδυναμεί με την εντολή

```
for (x = 1; x <= 10; x = x + 1)
```

♦ Η **<έκφραση1>**, η **<έκφραση2>** και η **<έκφραση3>** μπορούν να παραλειφθούν, αρκεί να υπάρχουν τα ";". Π.χ. η εντολή

```
for (;;)
```

αποτελεί έναν ατέρμονο βρόχο.

1.4.2 Εσωτερικές Συναρτήσεις της C

- Η C δε διαθέτει τον τελεστή ύψωσης σε δύναμη, διαθέτει όμως το δικό της ρεπερτόριο εσωτερικών συναρτήσεων (intrinsic functions), για τον υπολογισμό των στοιχειωδών Μαθηματικών και Τριγωνομετρικών Συναρτήσεων, μία εκ των οποίων υπολογίζει την ύψωση σε δύναμη. Οι συναρτήσεις αυτές απαιτούν τη χρήση της Μαθηματικής Βιβλιοθήκης που γίνεται με την εντολή :

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

▪ Το πρόγραμμα 1.4 με τη χρήση της συνάρτησης `pow(x)` θα είναι :

```
/* Άσκηση 1.4a
   Πινακοποίηση Συνάρτησης μιας Μεταβλητής Με τη Χρήση της
   Συνάρτησης pow     y = f(x) = 1/(1+x^2)  */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main()
{
   float a,b,h,x,y;
   printf("Askhsh 1.4a\n");
   printf("Pinakopoihsh Synarthshs me function pow(x,e)\n");
   printf("\n");
   //
   // Διάβασμα άκρων του διαστήματος a, b και βήματος h
   //
   printf("Dose times gia ta a,b,h : ");
   scanf ("%f  %f  %f",&a,&b,&h);
   for (x = a; x <= b; x = x + h)
     {
     y=1/(1+pow(x,2)); // Υπολογισμός y = f(x)
     printf("x = %12.8f      y = %5.2f\n", x, y); // Εμφάνιση x, y
     }
   system("Pause");
}
```

- Ο Επόμενος πίνακας παρουσιάζει τις πιο συνηθισμένες εσωτερικές συναρτήσεις με το όνομά τους, τη χρήση τους και το αποτέλεσμα που επιστρέφουν :

Συνάρτηση	**Δήλωση**	**Αποτέλεσμα**
Acos	**double acos(double** *x***);**	Επιστρέφει το το Τόξο Συνημιτόνου του x μεταξύ *0* και π ($-1 \le x \le 1$)
Asin	**Double asin(double** *x***);**	Επιστρέφει το το Τόξο Ημιτόνου του x μεταξύ $-\frac{\pi}{2}$ και $\frac{\pi}{2}$ ($-1 \le x \le 1$)
Atan	**double atan(double** *x***);**	Επιστρέφει το το Τόξο Εφαπτομένης του x μεταξύ $-\frac{\pi}{2}$ και $\frac{\pi}{2}$
Cos	**double cos(double** *x***);**	Επιστρέφει το Συνημίτονο του x
Cosh	**double cosh(double** *x***);**	Επιστρέφει το Υπερβολικό Συνημίτονο του x
Sin	**double sin(double** *x***);**	Επιστρέφει το Ημίτονο του x
Sinh	**double sinh(double** *x***);**	Επιστρέφει το Υπερβολικό Ημίτονο του x
Tan	**double tan(double** *x***);**	Επιστρέφει το Τόξο Εφαπτομένης του x
Tanh	**double tanh(double** *x***);**	Επιστρέφει την Υπερβολική Εφαπτομένη του x
Exp	**double exp(double** *x***);**	Επιστρέφει την Εκθετική Συνάρτηση (e^x)
Frexp	**double frexp(double** *x***, int ****exponent***);**	Αναλύει το x σε mantissa και εκθέτη, επιστρέφοντας στη συνάρτηση τη mantissa και στην ακέραια μεταβλητή *exponent* τον εκθέτη ($x = mantissa * 2^{exponent}$)
Ldexp	**double ldexp(double** *x***, int** *exponent***);**	Επιστρέφει το γινόμενο του x * το 2 στην δύναμη *exponent* ($x * 2^{exponent}$)
Log	**double log(double** *x***);**	Επιστρέφει το Φυσικό Λογάριθμο του x ($\log_e$ x).
log10	**double log10(double** *x***);**	Επιστρέφει το Δεκαδικό Λογάριθμο του x ($\log_{10}$ x).
Modf	**double modf(double** *x***, double ****integer***);**	Επιστρέφει το δεκαδικό μέρος ενός δεκαδικού αριθμού, ενώ το ακέραιο μέρος επιστρέφει στη μεταβλητή *integer*
Pow	**double pow(double** *x***, double** *y***);**	Επιστρέφει την ύψωση του x στην δύναμη y (x^y), με $x \ge 0$, αν y = κλάσμα και $x \ne 0$, αν $y \le 0$
Sqrt	**double sqrt(double** *x***);**	Επιστρέφει την Τετραγωνική Ρίζα του x ($\sqrt{x}$), με $x \ne 0$
Ceil	**double ceil(double** *x***);**	Επιστρέφει το μικρότερο ακέραιο ≥ του x
Fabs	**double fabs(double** *x***);**	Επιστρέφει την, Απόλυτη Τιμή του x ($\lvert x \rvert$)
floor	**double floor(double** *x***);**	Επιστρέφει το μεγαλύτερο ακέραιο ≤ του x
fmod	**double fmod (double** *x***, double** *y***);**	Επιστρέφει το υπόλοιπο της διαίρεσης του x με το y. Αν y = 0, επιστρέφει 0 ή μήνυμα λάθους

1.4.3 Ορισμός macro-Εντολών

- Η συνάρτηση $y = f(x) = 1 + x^2$ που χρησιμοποιείται στο πρόγραμμα της Πινακοποίησης μπορεί να δηλωθεί στην αρχή του προγράμματος με τη δήλωση **#define**. Η δήλωσή της θα έχει τη μορφή :

```
#define f(x) (1+x*x) // Προσοχή…Η έκφραση μέσα σε παρενθέσεις
```

Ο προ-επεξεργαστής της C, όπου βρει μια έκφραση που περιέχει το `f(x)`, θα αντικαταστήσει την τιμή του `x` στην έκφραση. Οι δηλώσεις macro-Εντολών έχουν την παρακάτω γενική μορφή :

<Όνομα_function>(Τυπικές_Παράμετροι) (<Αριθμητική_Έκφραση>)

όπου :

<Όνομα_function> = Μεταβλητή (Τύπου integer ή real).

(Τυπικές_Παράμετροι) = Μεταβλητές.

<Αριθμητική_Έκφραση> = Αριθμητική Έκφραση ως προς τις παραμέτρους.

Πρόγραμμα (Με τη χρήση macro- Εντολής)

- Το πρόγραμμα 1.4 με τη χρήση της macro-Εντολής θα είναι :

```
/* Άσκηση 1.4b
   Πινακοποίηση Συνάρτησης μιας Μεταβλητής
   Με τη Χρήση macro-εντολής   y = f(x) = 1/(1+x^2)
*/
#include <stdio.h>
#include <stdlib.h>
#define f(x) (1/(1+x*x)) // Δήλωση Συνάρτησης macro-εντολής
main()
{
   float a,b,h,x,y;
   printf("Askhsh 1.4b - Pinakopoihsh Synarthshs me macro-entolh\n");
   printf("\n");
   //
   // Διάβασμα άκρων του διαστήματος a, b και βήματος h
   //
   printf("Dose times gia ta a,b,h : ");
   scanf ("%f  %f  %f",&a,&b,&h);
   for (x = a; x ≤ b; x = x + h)  {
     y = f(x) ; // Υπολογισμός y = f(x)
     printf("x = %12.8f      y = %5.2f\n", x, y); // Εμφάνιση x, y
     }
   system("Pause");
}
```

Παρατηρήσεις

- Η Συνάρτηση - Εντολή πρέπει να τοποθετείται **πριν** από κάθε εκτελέσιμη εντολή του προγράμματος αλλά **μετά** τις δηλώσεις Μεταβλητών - Πινάκων.
- Μπορεί να υπάρχει οποιοσδήποτε αριθμός Τυπικών Παραμέτρων.
- Οι παρενθέσεις στον ορισμό της Συνάρτησης - Εντολής είναι υποχρεωτικές.
- Το όνομα της συνάρτησης πρέπει να ακολουθεί τους κανόνες των μεταβλητών.
- Οι Πραγματικές Παράμετροι μπορεί να είναι σταθερές ή μεταβλητές.
- Η συνάρτηση σε κάθε κλήση της δίνει **μια** τιμή.
- Ο αριθμός, ο τύπος και η σειρά των Πραγματικών Παραμέτρων στη χρήση της συνάρτησης στο κυρίως πρόγραμμα πρέπει να συμφωνεί με τον αριθμό, τον τύπο και τη σειρά των Τυπικών Παραμέτρων στον Ορισμό της συνάρτησης.
- Τα ονόματα των Πραγματικών Παραμέτρων μπορεί να είναι ή να μην είναι ίδια με τα ονόματα των Τυπικών Παραμέτρων.

Εφαρμογή 1.1

• Με τη χρήση των εντολών `if`, `while`, `do while`, `for` να γίνει πινακοποίηση της συνάρτησης

$$f(x)=\begin{cases}\dfrac{\sin(x)}{\sqrt{x}} & , \quad \alpha\nu\ x>0\\ 0 & , \quad \alpha\nu\ x=0\\ \dfrac{\sin(x)}{x} & , \quad \alpha\nu\ x<0\end{cases}$$

Εφαρμογή 1.2

• Με τη χρήση των εντολών `if`, `while`, `do while`, `for` να γίνει πινακοποίηση της συνάρτησης

$$f(x,y)=\begin{cases}\dfrac{x\cdot\sin(y)}{\sqrt{x^2-y^2}} & , \quad \alpha\nu\ |x|>|y|\\ 0 & , \quad \alpha\nu\ \ x=0\\ \dfrac{x\cdot\sin(y)}{\sqrt{y^2-x^2}} & , \quad \alpha\nu\ |x|<|y|\end{cases}$$

Εργαστηριακή Άσκηση 1a

- Να γίνει πρόγραμμα C, το οποίο να κατασκευάζει πίνακα τιμών για τη συνάρτηση $y = f(x)$ στο διάστημα `[a,b]`. (Να γίνουν 2 προγράμματα, με **`if`** και με **`if…else`**)

Αλγόριθμος

1) **Διαβάζουμε** τα άκρα του διαστήματος `a, b` και το βήμα `h`.

2) **Δίνουμε** αρχική τιμή στο `x = a`.

3) **Αν** το `x` έχει ξεπεράσει την τιμή του δεξιού άκρου (`b`) του διαστήματος, το πρόγραμμα τερματίζει.

4) **Διαφορετικά** :

   a) **Υπολογίζουμε** το $y = f(x)$.

   b) **Εμφανίζουμε** τις τιμές των `x, y`.

   c) **Αυξάνουμε** το `x` κατά το βήμα `h`.

   d) **Πηγαίνουμε** στο βήμα 3.

Απαιτούμενες Γνώσεις C

- Δομή Προγράμματος C
- Σχόλια
- Επικεφαλλίδα Προγράμματος C
- Δηλώσεις Ακεραίων – Πραγματικών Μεταβλητών
- Αριθμητικές Πράξεις – Τελεστές
- Είσοδος Δεδομένων – Εμφάνιση Αποτελεσμάτων
- Εσωτερικές Συναρτήσεις C
- Η **Εντολή *`goto`***
- Τελεστές Συσχέτισης
- Η **Εντολή *`if`***
- Η **Εντολή *`if…else`***
- Τερματισμός Προγράμματος C

Εργαστηριακή Άσκηση 1b

- Να γίνει πρόγραμμα C, το οποίο να κατασκευάζει πίνακα τιμών για τη συνάρτηση $y = f(x)$ στο διάστημα [a, b]. (Να γίνουν 3 προγράμματα, με ***while***, ***do while*** και ***for***)

Αλγόριθμος με την εντολή *while*

1) **Διαβάζουμε** τα άκρα του διαστήματος a, b και το βήμα h.

2) Δίνουμε **αρχική** τιμή στο x = a.

3) **Για όσο** το x δεν έχει ξεπεράσει την τιμή του δεξιού άκρου (b) του διαστήματος :

   a) **Υπολογίζουμε** το $y = f(x)$.

   b) **Εμφανίζουμε** τις τιμές των x, y.

   c) **Αυξάνουμε** το x κατά το βήμα h.

4) **Όταν** το x ξεπεράσει την τιμή του b το πρόγραμμα **τερματίζει**.

Αλγόριθμος με την εντολή *do while*

1) **Διαβάζουμε** τα άκρα του διαστήματος a, b και το βήμα h.

2) Δίνουμε **αρχική** τιμή στο x = a.

3) a)**Υπολογίζουμε** το $y = f(x)$.

   b) **Εμφανίζουμε** τις τιμές των x, y.

   c) **Αυξάνουμε** το x κατά το βήμα h.

Για όσο το x δεν έχει ξεπεράσει την τιμή του δεξιού άκρου (b) του διαστήματος

4) **Όταν** το x ξεπεράσει την τιμή του b το πρόγραμμα **τερματίζει**.

Αλγόριθμος με την εντολή *for*

1) **Διαβάζουμε** τα άκρα του διαστήματος a, b και το βήμα h.

2) **Για** τιμές του x απ' το a μέχρι το b με βήμα h :

   a) **Υπολογίζουμε** το $y = f(x)$.

   b) **Εμφανίζουμε** τις τιμές των x, y.

3) **Όταν** το x ξεπεράσει την τιμή του b το πρόγραμμα **τερματίζει**.

Απαιτούμενες Γνώσεις C

- Οι **Εντολές *while, do while, for***

2 Συναρτήσεις (Functions)

Χρήση Συναρτήσεων (functions)
Υπολογισμός του $\sum_{i=0}^{\infty} \frac{1}{x^i}$

- Στο κεφάλαιο που ακολουθεί παρουσιάζονται αλγόριθμοι και προγράμματα σε προβλήματα που έχουν να κάνουν με τον υπολογισμό Σειρών, όπου εισάγεται η έννοια της Συνάρτησης (function) της C, αντίστοιχη συναρτήσεων και υποπρογραμμάτων άλλων γλωσσών προγραμματισμού.

Πρόβλημα

- Να γίνει πρόγραμμα C, το οποίο να υπολογίζει την τιμή της σειράς :

$$\sum_{i=0}^{\infty} \frac{1}{x^i} = 1 + \frac{1}{x} + \frac{1}{x^2} + \frac{1}{x^3} + \cdots + \frac{1}{x^n} + \cdots$$

Η διαδικασία τελειώνει, όταν το σφάλμα αποκοπής (δηλαδή ο όρος που προστίθεται κάθε φορά) γίνει μικρότερο του 10^{-6}.

2.1 Υπολογισμός του $\sum_{i=0}^{\infty} \frac{1}{x^i}$ με Αναγωγικό Τύπο

Αλγόριθμος (1ος Τρόπος)

1. **Δίνουμε** την **αρχική** τιμή 1 στις μεταβλητές **mysum** και **oros** (τον κάθε όρο του αθροίσματος).
2. **Διαβάζουμε** την τιμή του x
3. **Για όσο** ο όρος είναι μεγαλύτερος του 0.0000001 (δεν πλησιάζει το μηδέν) :

   a) **Διαιρούμε** τον όρο δια του x βρίσκοντας το νέο όρο (αναγωγικός τύπος).

   b) **Προσθέτουμε** τον νέο όρο στο mysum.

   c) **Εμφανίζουμε** την τιμή του x και την τιμή του mysum.
4. **Εμφανίζουμε** την τιμή του x και την τελική τιμή του mysum.

Πρόγραμμα (1ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
main() // Άσκηση 2.1 – Υπολογισμός του Σ 1/x^i
{
float x;
printf("Askhsh 2.1 - Ypologismos tou 1/x^0+1/x^1+…\n");
printf("\n");
// Βήμα 1 – Αρχική τιμή 1 στις μεταβλητές mysum και oros
float sum = 1, oros = 1;
// Βήμα 2 – Διαβάζουμε την τιμή του x
printf("Dose timh gia to x > 1 : ");
scanf ("%f",&x);
// Βήμα 3 – Εύρεση Αθροίσματος
while (oros > 0.0000001)
   {
   oros = oros / x;  // Βήμα 3a – Εύρεση Νέου Όρου
   sum = sum + oros; // Βήμα 3b – Πρόσθεση στο Άθροισμα
   // Βήμα 3c – Εμφάνιση x, Αθροίσματος
    printf("x = %12.8f  ----   y = %12.8f\n", x, sum);
   }
// Βήμα 4 – Εμφάνιση x, Αθροίσματος
printf("x = %12.8f  ----   y = %12.8f\n", x, sum);
system("Pause");
}
```

2.2 Υπολογισμός του $\sum_{i=0}^{\infty} \frac{1}{x^i}$ με function

Αλγόριθμος function main() (2ος Τρόπος)

1. **Διαβάζουμε** την τιμή του x
2. **Εμφανίζουμε** την τιμή του x και την τιμή του myfun.

Αλγόριθμος function myfun() (2ος Τρόπος)

1. **Δίνουμε** την **αρχική** τιμή 1 στις μεταβλητές **sum** και **oros** (τον κάθε όρο).
2. **Για όσο** ο όρος είναι μεγαλύτερος του 0.000001 (δεν πλησιάζει το μηδέν) :

   a) **Διαιρούμε** τον όρο δια του x βρίσκοντας το νέο όρο (αναγωγικός τύπος).

   b) **Προσθέτουμε** τον νέο όρο στο sum.

   c) **Εμφανίζουμε** την τιμή του x και την τιμή του sum.
3. **Επιστρέφουμε** στη συνάρτηση main() την τιμή του sum.

Πρόγραμμα (2ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
float myfun(float x); // Δήλωση Συνάρτησης
main() // Άσκηση 2.2 – Υπολογισμός του Σ 1/x^i με function
{
float x;
printf("Askhsh 2.2”);
printf(“Ypologismos tou 1/x^0+1/x^1+… me function\n");
printf("\n");
//
// Βήμα 1 – Διαβάζουμε την τιμή του x
//
printf("Dose timh gia to x > 1 : ");
scanf ("%f",&x);
//
// Βήμα 2 – Εμφάνιση x, Αθροίσματος
//
printf("x = %12.8f  ----   y = %12.8f\n", x, myfun(x));
```

```
system("Pause");
}

float myfun(float x)
{
//
// Βήμα 1 - Αρχική τιμή 1 στις μεταβλητές mysum  και oros
//
    float oros = 1.0, sum = 1;
//
// Βήμα 2 - Εύρεση Αθροίσματος
//
    while (oros > 0.0000001)
     {
     oros = oros / x;  // Βήμα 2a - Εύρεση Νέου Όρου
     sum = sum + oros; // Βήμα 2b - Πρόσθεση στο Άθροισμα
     //
     // Βήμα 2c - Εμφάνιση x, Αθροίσματος
     //
     printf("myfun = %12.8f\n", sum);
     }
//
// Βήμα 3 - Επιστροφή Αθροίσματος στη συνάρτηση main
//
return (sum);
}
```

2.2.1 Συναρτήσεις (Functions)

- Εκτός απ' τη δυνατότητα να γράφουμε συναρτήσεις μιας γραμμής με τη macro-εντολή **#define**, η C μας παρέχει και τη δυνατότητα να γράφουμε συναρτήσεις που αποτελούνται από πολλές εντολές. Η γενική τους μορφή είναι :

```
<Τύπος_Function > <Όνομα_Function> ( Τυπικές_Παράμετροι )
Δηλώσεις του τύπου των Παραμέτρων
{
    εντολές
}
```

Παρατηρήσεις

- Η Συνάρτηση πρέπει να τοποθετείται **στην αρχή ή στο τέλος** του προγράμματος. Αν τοποθετηθεί μετά το πρόγραμμα `main()`, τότε ο τύπος της συνάρτησης πρέπει να δηλωθεί και στο κυρίως πρόγραμμα **πριν** την εντολή `main()`.

- Μπορεί να μην υπάρχει καμία Τυπική Παράμετρος.

- Οι παρενθέσεις στον ορισμό της Συνάρτησης είναι υποχρεωτικές.

- Το όνομα της συνάρτησης πρέπει να ακολουθεί τους κανόνες των μεταβλητών.

♦ Η συνάρτηση πρέπει να επιστρέφει μία τιμή. Η τιμή επιστρέφει με την εντολή **return** <μεταβλητή> , όπου <μεταβλητή> = τοπική μεταβλητή της συνάρτησης. Π.χ.

```
return sum;
```

2.2.2 Χρήση Συναρτήσεων

- Μια Συνάρτηση μπορεί να χρησιμοποιηθεί σαν παράμετρος της εντολής `printf`, σε κάποια συνθήκη ή στο δεύτερο μέλος οποιασδήποτε εντολής ανάθεσης, όπως και κάθε μεταβλητή. Η γενική της μορφή είναι :

  var = <Όνομα_function> (Πραγματικές_Παράμετροι)

  όπου :

  var, Πραγματικές_Παράμετροι = Μεταβλητές

Παράδειγμα 2.2.3

```
printf("x = %12.8f  ----   y = %12.8f\n", x, myfun(x));
```

Παρατηρήσεις

♦ Οι Πραγματικές Παράμετροι μπορεί να είναι Σταθερές, Μεταβλητές, Ονόματα Πινάκων, Στοιχεία Πινάκων, Αριθμητικές Εκφράσεις. Άλλες Συναρτήσεις.

♦ Η συνάρτηση σε κάθε κλήση της δίνει **μια** τιμή.

♦ Ο αριθμός, ο τύπος και η σειρά των Πραγματικών Παραμέτρων στη χρήση της συνάρτησης στο κυρίως πρόγραμμα πρέπει να συμφωνεί με τον αριθμό, τον τύπο και τη σειρά των Τυπικών Παραμέτρων στον Ορισμό της συνάρτησης.

♦ Τα ονόματα των Πραγματικών Παραμέτρων μπορεί να είναι ή να μην είναι ίδια με τα ονόματα των Τυπικών Παραμέτρων.

♦ Η εντολή **return** μπορεί να εμφανίζεται πολλές φορές στη Συνάρτηση.

♦ Μια συνάρτηση μπορεί να καλείται από το κύριο πρόγραμμα, όπως τα procedures ή subroutines άλλων γλωσσών. Έτσι, αντί να χρησιμοποιούμε τη συνάρτηση στο σημείο του κυρίως προγράμματος που θέλουμε να μας επιστρέψει κάποια τιμή, **καλούμε** πρώτα τη συνάρτηση, η οποία **επιστρέφει** μία ή περισσότερες **τιμές** στις **παραμέτρους**, όπως φαίνεται στην επόμενη εφαρμογή :

2.3 Υπολογισμός του $\sum_{i=0}^{\infty} \frac{1}{x^i}$ με function – procedure

Αλγόριθμος function `main()` (3ος Τρόπος)

1. **Διαβάζουμε** την τιμή του `x`
2. **Καλούμε** τη συνάρτηση `myfun`.
3. **Εμφανίζουμε** την τιμή του `x` και την τιμή του `y[0]`.

Αλγόριθμος procedure-function `myfun()` (3ος Τρόπος)

1. **Δίνουμε** την **αρχική** τιμή 1 στις μεταβλητές **`sum`** και **`oros`** (τον κάθε όρο).
2. **Για όσο** ο όρος είναι μεγαλύτερος του `0.000001` (δεν πλησιάζει το μηδέν) :
   a) **Διαιρούμε** τον όρο δια του `x` βρίσκοντας το νέο όρο (αναγωγικός τύπος).
   b) **Προσθέτουμε** τον νέο όρο στο `sum`.
   c) **Εμφανίζουμε** την τιμή του `x` και την τιμή του `sum`.
3. **Επιστρέφουμε** στη μεταβλητή `y[0]` της συνάρτησης `main()` την τιμή του `sum`.

Πρόγραμμα (3ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
float myfun(float x, float y[]);
main() // Υπολογισμός του Σ 1/x^i με function – procedure
{
float x, y[1];
printf("Askhsh 2.3”);
printf(“Ypologismos tou 1/x^0+1/x^1+… me function-procedure\n");
printf("\n");
//
// Βήμα 1 – Διαβάζουμε την τιμή του x
//
printf("Dose timh gia to x : ");
scanf ("%f",&x);
//
// Βήμα 2 – Κλήση Συνάρτησης Υπολογισμού Αθροίσματος
//
myfun(x, y);
//
// Βήμα 3 – Εμφάνιση x, Αθροίσματος
//
```

```
printf("x = %12.8f  ----   y = %12.8f\n", x, y[0]);
system("Pause");
}

float myfun(float x, float  y[])
{
//
// Βήμα 1 - Αρχική τιμή 1 στις μεταβλητές mysum  και oros
//
float oros = 1.0, sum = 1;
//
// Βήμα 2 - Εύρεση Αθροίσματος
//
 while (oros > 0.0000001)
      {
      oros = oros / x; // Βήμα 2a - Εύρεση Νέου Όρου
      sum = sum + oros; // Βήμα 2b - Πρόσθεση στο Άθροισμα
      //
      // Βήμα 2c - Εμφάνιση x, Αθροίσματος
      //
      printf("x =   %12.8f   myfun = %12.8f\n", x, sum);
      }
//
// Βήμα 3 - Επιστροφή Αθροίσματος στην παράμετρο y,
//          Μεταβλητή της συνάρτησης main
//
y[0] = sum;
}
```

2.3.1 Κλήση Συναρτήσεων - procedures

- Η κλήση της Συνάρτησης–procedure γίνεται με το όνομά της και τις πραγματικές παραμέτρους. Π.χ.

```
myfun(x, y);
```

Παρατηρήσεις

- Η συνάρτηση–procedure μπορεί να επιστρέφει περισσότερες από μία τιμές. Οι τιμές επιστρέφουν στις παραμέτρους.

- 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. Για τις απλές μεταβλητές θα πρέπει να χρησιμοποιήσουμε δείκτες – pointers, οι πίνακες όμως δεν έχουν τέτοιο πρόβλημα, γιατί, όταν περνάει πίνακας σαν παράμετρος, περνάει η **διεύθυνση** του πίνακα.

2.3.2 Δήλωση Μονοδιάστατων Πινάκων

- Η δήλωση γενικά ενός πίνακα `pin 10` θέσεων για πραγματικούς αριθμούς γίνεται με την εντολή :

```
float pin[10];
```

με στοιχεία `pin[0], p[1], …, p[9].`

Στο προηγούμενο παράδειγμα, δηλώσαμε τον πίνακα

```
float y[1];
```

με μοναδικό στοιχείο το `y[0]`.

2.3.3 Πέρασμα Πινάκων σαν Παραμέτρων σε functions

Για το πέρασμα Πινάκων σαν παραμέτρων ισχύουν τα παρακάτω :

- Οι Πίνακες - Παράμετροι θα πρέπει να δηλωθούν στο κυρίως Πρόγραμμα.
- Ο Πίνακας περνάει σαν παράμετρος με το όνομά του και αγκύλες.

2.3.4 Είδη Μεταβλητών

Οι μεταβλητές στη C – ανεξάρτητα απ' τον τύπο των δεδομένων που αποθηκεύουν - μπορούν να ταξινομηθούν στις παρακάτω κατηγορίες :

Αυτόματες Μεταβλητές (*auto variables*)

Είναι ***τοπικές μεταβλητές*** που ενεργοποιούνται μόνο μέσα στη ***συνάρτηση*** ή στο ***block εντολών*** της C - μεταξύ { } – που τις δηλώνουμε, ενώ δεν είναι δυνατή η χρήση τους από άλλες συναρτήσεις του προγράμματος. Η δήλωσή τους γίνεται με τον προσδιοριστή ***auto***. Αν δεν υπάρχει κανένας προσδιοριστής, οι μεταβλητές θεωρούνται αυτόματες.

Μεταβλητές Καταχωρητών (*register variables*)

Είναι **αυτόματες μεταβλητές** τύπου **int** ή **char** που αντί για την κλασσική μνήμη χρησιμοποιούν για **μεγαλύτερη ταχύτητα** κάποιον **καταχωρητή** (register) της ΚΜΕ. Στην ίδια συνάρτηση μπορούν να συνυπάρχουν μόνο 3-4 μεταβλητές καταχωρητών.

Στατικές Μεταβλητές (static *variables*)

Είναι όπως και οι αυτόματες μεταβλητές. Ενεργοποιούνται μόνο μέσα στη ***συνάρτηση*** ή στο ***block εντολών*** της C - μεταξύ { } – που τις δηλώνουμε, ενώ δεν είναι δυνατή η χρήση τους από άλλες συναρτήσεις του προγράμματος. Σε αντίθεση με τις αυτόματες, **διατηρούν την τιμή τους ανάμεσα σε διαδοχικές κλήσεις της συνάρτησης**. Η δήλωσή τους γίνεται με τον προσδιοριστή **static**.

Παράδειγμα

```
main()
{     incr( );
      incr( );
      incr( );
}
      incr( )
      { static int x = 0;
         x = x + 1;
         printf("%d\n", x);
       }
```

Με την πρώτη κλήση της συνάρτησης incr( ) το x παίρνει την τιμή 1
Με τη δεύτερη κλήση της συνάρτησης incr( ) το x παίρνει την τιμή 2
Με την τρίτη κλήση της συνάρτησης incr( ) το x παίρνει την τιμή 3

Εξωτερικές Μεταβλητές (*global* (*variables*)

Είναι μεταβλητές που διατηρούν την τιμή τους και μπορούν να χρησιμοποιηθούν σε οποιοδήποτε σημείο του προγράμματος. Η δήλωσή τους γίνεται **πριν** από τον ορισμό του κυρίως προγράμματος **main ( ).** Για να τις χρησιμοποιήσουμε όμως σε μια **συνάρτηση** του προγράμματος, θα πρέπει να τις **δηλώσουμε ξανά** με τον προσδιοριστή **extern** (αν δε βρίσκεται η συνάρτηση στο ίδιο αρχείο με τη συνάρτηση main()).

Παρατηρήσεις

- Αντί να χρησιμοποιήσουμε παράμετρο στην οποία θα επιστρέψει η τιμή της συνάρτησης–procedure, θα μπορούσαμε να επιστρέψουμε την τιμή σε μια εξωτερική μεταβλητή (global variable) του προγράμματος `main()`. Τότε το πρόγραμμα 2.3 θα είχε την παρακάτω μορφή :

Πρόγραμμα (4ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
float y; // Δήλωση global varible y
float myfun(float x);
main() // Άσκηση 2.4 – Υπολογισμός του Σ_{i=0}^{∞} 1/x^i με global varible
{
float x;
printf("Askhsh 2.4”);
printf(“Ypologismos tou 1/x^0+1/x^1+… me global variable\n");
printf("\n");
//
// Βήμα 1 – Διαβάζουμε την τιμή του x
//
printf("Dose timh gia to x : ");
scanf ("%f",&x);
myfun(x); // Βήμα 2 – Κλήση Συνάρτησης
//
// Βήμα 3 – Εμφάνιση x, Αθροίσματος
//
printf("x = %12.8f  ----   y = %12.8f\n", x, y);
system("Pause");
}

float myfun(float x)
{
//
// Βήμα 1 – Αρχική τιμή 1 στις μεταβλητές mysum  και oros
//
float oros = 1.0, sum = 1;
//
// Βήμα 2 – Εύρεση Αθροίσματος
//
 while (oros > 0.0000001)
      {
      oros = oros / x; // Βήμα 2a – Εύρεση Νέου Όρου
      sum = sum + oros; // Βήμα 2b – Πρόσθεση στο Άθροισμα
      // Βήμα 2c – Εμφάνιση x, Αθροίσματος
      printf("x =   %12.8f   myfun = %12.8f\n", x, sum);
      }
y = sum; //Βήμα 3 – Επιστροφή Αθροίσματος στη Μεταβλητή y
}
```

Εργαστηριακή Άσκηση 2a – 2b

- Να γίνει πρόγραμμα C, το οποίο, δοθέντος κάποιου x, να υπολογίζει το $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$ (Να γίνουν 2 προγράμματα, το ένα με **function** και το άλλο με **procedure-function**).

Εργαστηριακή Άσκηση 2a

Αλγόριθμος function `main()`

1. **Διαβάζουμε** την τιμή του `x`
2. **Εμφανίζουμε** την τιμή του `x`, του `exp(x)` και την τιμή του `myexp(x)`.

Αλγόριθμος function `myexp()`

1. **Δίνουμε** την αρχική τιμή `1` στις μεταβλητές **`i`, `sum`** και **`oros`** (τον κάθε όρο).
2. **Για όσο** ο όρος είναι μεγαλύτερος του `0.000001` (δεν πλησιάζει το μηδέν) :
   a) **Πολλαπλασιάζουμε** τον όρο επί `x` και διαιρούμε δια `i` βρίσκοντας το νέο όρο.
   b) **Προσθέτουμε** τον νέο όρο στο `sum`.
   c) **Αυξάνουμε** το `i` κατά `1`.
3. **Επιστρέφουμε** στη συνάρτηση `myexp()` την τιμή του `sum`.

Εργαστηριακή Άσκηση 2b

Αλγόριθμος function `main()`

1. **Διαβάζουμε** την τιμή του `x`.
2. **Καλούμε** τη συνάρτηση `myexp(x,y)`.
3. **Εμφανίζουμε** την τιμή του `x`, του `exp(x)` και την τιμή του `y[0]`.

Αλγόριθμος procedure-function `myexp ()` (2ος Τρόπος)

1. **Δίνουμε** την αρχική τιμή 1 στις μεταβλητές **`i`, `sum`** και **`oros`** (τον κάθε όρο).
2. **Για όσο** ο όρος είναι μεγαλύτερος του `0.0000001` (δεν πλησιάζει το μηδέν) :
   a) **Πολλαπλασιάζουμε** τον όρο επί `x` και διαιρούμε δια `i` βρίσκοντας το νέο όρο.
   b) **Προσθέτουμε** τον νέο όρο στο `sum`.
   c) **Αυξάνουμε** το `i` κατά 1.
3. **Επιστρέφουμε** στη μεταβλητή `y[0]` της συνάρτησης `main()` την τιμή του `sum`.

Επίλυση Εξισώσεων

3

Σχήμα Horner
Μέθοδος Διαδοχικών Προσεγγίσεων

- Στο κεφάλαιο που ακολουθεί παρουσιάζονται αλγόριθμοι και προγράμματα σχετικά με μεθόδους για την εύρεση των ριζών μιας συνάρτησης ή ενός πολυωνύμου, καθώς και το σχήμα Horner για την εύρεση της τιμής του σε κάποιο σημείο. Εισάγεται η έννοια του πίνακα στη C και οι εντολές που αφορούν το χειρισμό πινάκων, όπως δήλωση πίνακα, διάβασμα στοιχείων πίνακα και εμφάνιση στοιχείων πίνακα. Το σχήμα του Horner υλοποιείται επίσης με τη χρήση functions και procedures και χρησιμοποιείται για την εφαρμογή της μεθόδου των Διαδοχικών Προσεγγίσεων για την εύρεση των ριζών ενός πολυωνύμου.

3.1 Υπολογισμός Τιμής Πολυωνύμου - Σχήμα του Horner

- Δίδεται το Πολυώνυμο :

$$P(x) = p_0 x^n + p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x^1 + p_n$$

- Να γίνει πρόγραμμα C το οποίο να υπολογίζει τους συντελεστές του πηλίκου και το Υπόλοιπο της Διαίρεσης του Πολυωνύμου P(x) δια του Μονώνυμου :

$$x - \xi$$

με τη βοήθεια του σχήματος του Horner.

α) Αλγόριθμος

1. **Διαβάζουμε** τους συντελεστές του Πολυωνύμου $p_0, p_1, ..., p_n$ στον πίνακα p.

2. **Εμφανίζουμε** τους συντελεστές του Πολυωνύμου $p_0, p_1, ..., p_n$.

3. **Διαβάζουμε** το ξ.

4. **Υπολογίζουμε** τις τιμές του πηλίκου Q(x) στον πίνακα q, σύμφωνα με το σχήμα του Horner :

   a) $q_0 = p_0$
   b) **Για** τους συντελεστές $q_i, i = 1,2,...,n$:
   $$q_i = p_i + q_{i-1} * \xi$$

5) **Εμφανίζουμε** τις τιμές των συντελεστών $q_0, q_1, ..., q_{n-1}$

6) **Εμφανίζουμε** την τιμή του υπολοίπου της διαίρεσης (συντελεστής q_n).

β) Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
main()// Άσκηση 3.1 – Τιμή Πολυωνύμου με το Σχήμα Horner
{
int i;
float ksi,p[n+1],q[n+1];
printf("Askhsh 3.1”);
printf(“Ypologismos Timhs Polywnymou me to Sxhma Horner\n");
printf("\n");
//
// Βήμα 1 – Γέμισμα Πίνακα p
//
for ( i = 0; i<=n; i++ )
{
printf("Dose timh gia to p[%d ] : ",i);
scanf ("%f",&p[i]);
}
//
// Βήμα 2 – Εμφάνιση Πίνακα p
//
for ( i = 0; i<=n; i++ )
{
printf("p[%d] = %f\n",i, p[i]);
}
//
// Βήμα 3 – Διάβασμα ξ
//
printf("Dose timh gia to ksi : ");
scanf ("%f",&ksi);
//
// Βήμα 4 – Εφαρμογή Σχήματος Horner
// – Υπολογισμός Πηλίκου, Υπολοίπου
//
q[0] = p[0];
for ( i = 1; i<=n; i=i+1 )
q[i] = p[i]+q[i-1]*ksi;
//
// Βήμα 5 – Εμφάνιση Πηλίκου
//
for ( i = 0; i<=n-1; i+=1 )
 printf("q[ %d ] = %12.8f\n", i, q[i]);
//
// Βήμα 6 – Εμφάνιση Υπολοίπου = p(ξ)
//
printf("Ypoloipo = q[ %d ] = %12.8f\n", n, q[n]);
system("Pause");
}
```

3.1.1 Αριθμητικές Σταθερές (parameter)

- Η δήλωση του πλήθους των στοιχείων των πινάκων `p` και `q` στο προηγούμενο πρόγραμμα γίνεται με την εντολή:

```
#define n 3
```

Παρατηρήσεις

♦ Οι Σταθερές μπορούν να δηλωθούν στη αρχή του προγράμματος πριν τη συνάρτηση `main()` με την εντολή **`#define`** και προηγούνται των δηλώσεων των μεταβλητών ή με την εντολή **const** στο τμήμα δηλώσεων μεταβλητών. Π.χ.

```
const n = 3;
```

♦ Οι Αριθμητικές Σταθερές στη C μπορεί να είναι Ακέραιες ή Πραγματικές και δεν αλλάζουν τιμή κατά τη διάρκεια του προγράμματος.

3.1.2 Δήλωση Μονοδιάστατων Πινάκων

- Η δήλωση γενικά ενός πίνακα pin 10 θέσεων για πραγματικούς αριθμούς γίνεται με την εντολή :

```
float pin[10];
```

με στοιχεία `pin[0], p[1], …, p[9].`

Παρατηρήσεις

- Η δήλωση του πλήθους των στοιχείων του πίνακα μπορεί να γίνει με τη δήλωση σταθεράς. Έτσι, θα είχαμε το ίδιο αποτέλεσμα με τις εντολές :

```
#define n 10
float pin[n];
```

- Στο πρόγραμμα χρησιμοποιήθηκε για τη δήλωση των πινάκων `p` και `q` η εντολή :

```
float p[n+1], q[n+1];
```

όπου :

`p, q` = τα ονόματα των πινάκων.
`n+1` = το μέγιστο πλήθος των στοιχείων των πινάκων.

και τα στοιχεία των πινάκων `p` και `q` θα είναι και `p[0], p[1], …, p[n]` και `q[0], q[1], …, q[n]`, γιατί χρειαζόμαστε `n+1` συντελεστές.

3.1.3 Διάβασμα Στοιχείων ενός Μονοδιάστατου Πίνακα

- Με την ομάδα εντολών :

```
for ( i = 0; i<=n; i++ )
{
printf("Dose timh gia to p[%d ] : ",i);
scanf ("%f",&p[i]);
}
```

διαβάζουμε n + 1 = 4 πραγματικούς αριθμούς απ' το πληκτρολόγιο χωρίς format (**έναν αριθμό σε κάθε γραμμή**) και τους αποθηκεύουμε στον πίνακα των συντελεστών p.

Παρατήρηση

- Αν θέλαμε να χρησιμοποιήσουμε το πρόγραμμα για συγκεκριμένο πολυώνυμο, π.χ. το $P(x) = x^3 - 4x$, θα μπορούσαμε να δώσουμε τις τιμές των στοιχείων του πίνακα p στην εντολή δήλωσης του πίνακα, οπότε το *Βήμα 1* είναι περιττό, ενώ το τμήμα δηλώσεων θα γινόταν :

```
float ksi,q[n+1];
float p[n+1] = {1,0,-4,0};
```

3.1.4 Εμφάνιση n Στοιχείων Μονοδιάστατου Πίνακα

- Με την ομάδα εντολών :

```
for ( i = 0; i<=n; i++ )
{
printf("p[%d] = %f\n",i, p[i]);
}
```

εμφανίζουμε στην οθόνη n + 1 = 4 πραγματικούς αριθμούς (τα στοιχεία του πίνακα των συντελεστών p και τον αντίστοιχο δείκτη) χωρίς format.

3.1.5 Πέρασμα Πινάκων σαν Παραμέτρων σε functions

➢ Αν θέλαμε να χρησιμοποιήσουμε μια function που θα καλείται από το πρόγραμμα main() να υλοποιήσει το Βήμα 4 θα έπρεπε να περάσουμε τους **πίνακες** p και q σαν **παραμέτρους**. Για το πέρασμα Πινάκων σαν παραμέτρων ισχύουν τα παρακάτω :

♦ Οι Πίνακες - Παράμετροι θα πρέπει να δηλωθούν στο κυρίως Πρόγραμμα.

♦ Ο Πίνακας περνάει σαν παράμετρος με το όνομά του και αγκύλες, όπως φαίνεται στις εφαρμογές που ακολουθούν :

3.2 Ανταλλαγή Στοιχείων Πινάκων με function

- Να γίνει πρόγραμμα C που να **ανταλλάζει** τα **περιεχόμενα** των **πινάκων** `a` και `b` με τη χρήση ενός function που θα επιστρέφει την τιμή του $1^{ου}$ στοιχείου του πίνακα `a, a[0].`

α) Αλγόριθμος Προγράμματος

1) **Εμφανίζουμε** τα περιεχόμενα των Πινάκων `a, b`.

2) **Εμφανίζουμε** το περιεχόμενο του στοιχείου `a[0]` με τη χρήση function.

3) **Εμφανίζουμε** τα νέα περιεχόμενα των Πινάκων `a, b`.

β) Αλγόριθμος function

1) **Για** τα στοιχεία `i = 0,…,n-1`

```
temp = a[i]
a[i] = b[i]
b[i] = temp
```

2) **Επιστροφή** `a[0]`

γ) Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
float swap_a_b( float  a[],  float  b[]);
main() // Ανταλλαγή Στοιχείων Πινάκων a, b με function
{
int i;
float a[n] = {1,2,3};
float b[n] = {-1,-2,-3};
printf("Askhsh 3.2”);
printf(“Antallagh Stoixeiwn Pinakwn a, b me function\n");
printf("\n");
//
// Βήμα 1 – Εμφάνιση Στοιχείων Πινάκων a, b
//
printf("Arxikos Pinakas a\n");
for ( i = 0; i<=n-1; i++ )
 printf("a[%d] = %f\n",i,a[i]);
printf("Arxikos Pinakas b\n");
for ( i = 0; i<=n-1; i++ )
 printf("b[%d] = %f\n",i,b[i]);
//
// Βήμα 2 – Εμφάνιση Στοιχείου a[0] με function
//
printf("\n");
printf("a[0] = %12f\n", swap_a_b(a,b));
//
// Βήμα 3 – Εμφάνιση Νέων Στοιχείων Πινάκων a, b
//
printf("Telikos Pinakas a\n");
for ( i = 0; i<=n-1; i++ )
 printf("a[%d] = %f\n",i,a[i]);
printf("Telikos Pinakas b\n");
for ( i = 0; i<=n-1; i++ )
 printf("b[%d] = %f\n",i,b[i]);
system("Pause");
}

float swap_a_b( float  a[],  float  b[])
{
// Ανταλλαγή Στοιχείων Πινάκων a, b
float temp;
int i;
for ( i = 0; i<n; i++ ) {
 temp = a[i];
 a[i] = b[i];
 b[i] = temp;
 }
return a[0];
}
```

3.3 Ανταλλαγή Στοιχείων Πινάκων με procedure function

- Να γίνει πρόγραμμα C που να ανταλλάζει τα περιεχόμενα των πινάκων `a` και `b` με την κλήση ενός procedure function.

α) Αλγόριθμος Προγράμματος

1) **Εμφανίζουμε** τα περιεχόμενα των Πινάκων `a, b.`

2) **Καλούμε** τη συνάρτηση ανταλλαγής των στοιχείων των πινάκων `a, b.`

3) **Εμφανίζουμε** τα νέα περιεχόμενα των Πινάκων `a, b.`

β) Αλγόριθμος function

Για τα στοιχεία `i = 0,…,n-1`

```
temp = a[i]
a[i] = b[i]
b[i] = temp
```

γ) Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
float swap_a_b( float  a[],  float  b[]);
main()//Ανταλλαγή Στοιχείων Πινάκων a, b με function-procedure
{
int i;
float a[n] = {1,2,3};
float b[n] = {-1,-2,-3};
printf("Askhsh 3.3”);
printf(“Antallagh Stoixeiwn Pinakwn a, b me function-procedure\n");
printf("\n");
//
// Βήμα 1 - Εμφάνιση Στοιχείων Πινάκων a, b
//
printf("\n");
printf("Arxikos Pinakas a\n");
for ( i = 0; i<=n-1; i++ )
 printf("a[%d] = %f\n",i,a[i]);
printf("Arxikos Pinakas b\n");
for ( i = 0; i<=n-1; i++ )
 printf("b[%d] = %f\n",i,b[i]);
//
// Βήμα 2 - Κλήση function Ανταλλαγής Στοιχείων a, b
//
swap_a_b(a,b);
//
// Βήμα 3 - Εμφάνιση Νέων Στοιχείων Πινάκων a, b
//
printf("Telikos Pinakas a\n");
for ( i = 0; i<=n-1; i++ )
 printf("a[%d] = %f\n",i,a[i]);
printf("Telikos Pinakas b\n");
for ( i = 0; i<=n-1; i++ )
 printf("b[%d] = %f\n",i,b[i]);
system("Pause");
}

float swap_a_b( float  a[],  float  b[])
{
// Ανταλλαγή Στοιχείων Πινάκων a, b
//
float temp;
int i;
for ( i = 0; i<n; i++ ) {
 temp = a[i];
 a[i] = b[i];
 b[i] = temp;
 }
}
```

Εργαστηριακή Άσκηση 3.2

- Δίδεται το Πολυώνυμο $P(x) = p_0 x^n + p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x^1 + p_n$. Να γίνει πρόγραμμα C που να χρησιμοποιεί ένα **function** που θα επιστρέφει το Υπόλοιπο της Διαίρεσης του Πολυωνύμου `P(x)` δια του Μονώνυμου $(x - \xi)$ και την τιμή της Παραγώγου του Πολυωνύμου $P'(\xi)$ με το σχήμα του Horner.

α) Αλγόριθμος Προγράμματος

1) ***Διαβάζει*** τους συντελεστές του Πολυωνύμου $p_0, p_1, \cdots p_n$ (στον πίνακα `p`).
2) ***Εμφανίζει*** στην οθόνη τους συντελεστές του Πολυωνύμου $p_0, p_1, \cdots p_n$.
3) ***Διαβάζει*** το συντελεστή του μονώνυμου ξ.
4) ***Χρησιμοποιεί*** τη συνάρτηση `horner_fun(<παράμετροι>)`, για να ***υπολογίσει*** το Υπόλοιπο της διαίρεσης $P(x)/(x-\xi)$**=** η τιμή του Πολυωνύμου $P(\xi)$ στο $\xi = q(n)$ και να το ***εμφανίσει***.
5) ***Εμφανίζει*** τους Συντελεστές του Πηλίκου $q_i, i = 0,1,...,n-1$**.**
6) ***Χρησιμοποιεί*** τη συνάρτηση `horner_fun(<παράμετροι>)`, για να ***υπολογίσει*** το Υπόλοιπο της διαίρεσης $Q(x)/(x-\xi)$**=** η τιμή της Παραγώγου του Πολυωνύμου $P'(\xi) = r(n-1)$ και να το ***εμφανίσει.***
7) ***Εμφανίζει*** τους Συντελεστές του Πηλίκου $r_i, i = 0,1,...,n-2$**..**

Αλγόριθμος function

$q_0 = p_0$

Για τους συντελεστές $q_i, i = 1,2,...,n$:

$$q_i = p_i + q_{i-1} * \xi$$

Επιστροφή q_n

Εργαστηριακή Άσκηση 3.3

- Δίδεται το Πολυώνυμο $P(x) = p_0 x^n + p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x^1 + p_n$. Να γίνει πρόγραμμα C που να **καλεί** ένα **function** που θα υπολογίζει τους συντελεστές του πηλίκου και το Υπόλοιπο της Διαίρεσης του Πολυωνύμου `P(x)` δια του Μονώνυμου $(x - \xi)$ και την τιμή της Παραγώγου του Πολυωνύμου $P'(\xi)$ με το σχήμα του Horner.

α) Αλγόριθμος Προγράμματος

1) ***Διαβάζει*** τους συντελεστές του Πολυωνύμου $p_0, p_1, \cdots p_n$ (στον πίνακα `p`).
2) ***Εμφανίζει*** στην οθόνη τους συντελεστές του Πολυωνύμου $p_0, p_1, \cdots p_n$.
3) ***Διαβάζει*** το συντελεστή του μονώνυμου ξ.
4) ***Καλεί*** το function `horner_sub(<παράμετροι>)` με το οποίο ***υπολογίζει*** το Πηλίκο (Συντελεστές $q_i, i = 0,1,...,n-1$) και το Υπόλοιπο της διαίρεσης $P(x)/(x-\xi)$ = η τιμή του Πολυωνύμου $P(\xi) = q(n)$
5) ***Εμφανίζει*** τους Συντελεστές του **Πηλίκου** $q_i, i = 0,1,...,n-1$.
6) ***Εμφανίζει*** το **Υπόλοιπο** $q(n)$.
7) ***Καλεί*** το function `horner_sub(<παράμετροι>)` με το οποίο ***υπολογίζει*** το Πηλίκο (Συντελεστές $r_i, i = 0,1,...,n-2$) και το Υπόλοιπο της διαίρεσης $Q(x)/(x-\xi)$= η τιμή της Παραγώγου του Πολυωνύμου $P'(\xi) = r(n-1)$.
8) ***Εμφανίζει*** τους Συντελεστές του **Πηλίκου** $r_i, i = 0,1,...,n-2$.
9) ***Εμφανίζει*** το **Υπόλοιπο** $r(n-1)$.

Αλγόριθμος function - procedure

$q_0 = p_0$

Για τους συντελεστές $q_i, i = 1,2,...,n$:

$$q_i = p_i + q_{i-1} * \xi$$

3.4 Αριθμητική Επίλυση Εξισώσεων - Μέθοδος Διαδοχικών Προσεγγίσεων

- Δίνεται η εξίσωση :

$$f(x) = 0 | x \in [a,b] \subseteq \Re$$

Να γίνει πρόγραμμα C, το οποίο να υπολογίζει **μια** ρίζα της εξίσωσης που περιέχεται στο διάστημα `[a, b]` με τη βοήθεια της μεθόδου των **Διαδοχικών Προσεγγίσεων.** Η αρχική εξίσωση παίρνει τη μορφή

$$x = g(x)$$

ενώ η ρίζα βρίσκεται απ' τον αναγωγικό τύπο :

$$x_{n+1} = g(x_n), \left| g'(x_n) \right| < 1$$

Αλγόριθμος

1. **Δηλώνουμε** με τη χρήση του **`#define`** τις συναρτήσεις **`f(x), g(x)`** και **`gp(x)`,** όπου `gp(x)` η παράγωγος της `g(x)`**.**

2**. Διαβάζουμε** τα όρια του διαστήματος της ρίζας **`a, b`.**

3. **Αν** υπάρχει ρίζα στο διάστημα $[a,b]$ ($f(a) * f(b) < 0$) **τότε**

   a) **Διαβάζουμε** μια αρχική τιμή για το `x`

4. **Εμφανίζουμε** τις τιμές του `x` και $f(x)$.

5. **Για** όσο $\left| f(x) \right| > 0.000001$ και $\left| g'(x) \right| < 1$:

   α) **Υπολογίζουμε** το νέο `x = g(x)`**.**

   β) **Εμφανίζουμε** τις τιμές του `x` και $f(x)$.

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
//
// Βήμα 1 - Δήλωση Συναρτήσεων f(x), g(x), g'(x)
//
#define f(x) (pow(x,3)-4*x)
#define g(x) (x/2+2/x)
#define gp(x) (1/2.0-2/pow(x,2))
main() // Άσκηση 3.4 - Μέθοδος Διαδοχικών Προσεγγίσεων
float a, b, x;
printf("Askhsh 3.4\n");
printf("Methodos Diadoxikwn Proseggisewn -f(x)=x^3-4*x,g(x)= x/2 + 2/x\n");
printf("\n");
//
// Βήμα 2 - Διάβασμα Άκρων Διαστήματος [a,b]
//
printf("Dose timh gia ta a b : ");
scanf ("%f %f",&a, &b);
//
// Βήμα 3 - Έλεγχος Ύπαρξης Ρίζας στο [a,b]
//
if (f(a) * f(b) < 0 )
{
//
// Βήμα 3α - Διάβασμα Αρχικής Τιμής για το x
//
printf("Dose timh gia to x0 sto [a b] : ");
scanf ("%f",&x);
}
//
// Βήμα 4 - Εμφάνιση x, f(x)
//
printf("x = %f  f(x) = %f\n",x,f(x));
//
// Βήμα 5 - Επανάληψη για όσο δε βρέθηκε η Ρίζα
//
while (( fabs(f(x)) > 0.00000001) &&  (fabs(gp(x)) <1 ))
 {
//
// Βήμα 5α - Εύρεση Νέας Προσεγγιστικής Τιμής του x
//
x = g(x);
//
// Βήμα 5β - Εμφάνιση x, f(x)
//
printf("x = %f  f(x) = %f\n",x,f(x));
}
system("Pause");
}
```

Εργαστηριακή Άσκηση 3.4

- Δίνεται η εξίσωση :

$$f(x) = 0 | x \in [a,b] \subseteq \Re$$

- Να γίνει πρόγραμμα C, το οποίο να υπολογίζει **μια** ρίζα της εξίσωσης που περιέχεται στο διάστημα `[a, b]` με τη βοήθεια της μεθόδου των **Newton-Raphson.** Η ρίζα βρίσκεται απ' τον αναγωγικό τύπο :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, f'(x_n) \neq 0$$

Αλγόριθμος

1. **Δηλώνουμε** με τη χρήση του **`#define`** τις συναρτήσεις **`f(x)`** και **`pf(x)`,** όπου `pf(x)` η παράγωγος της `f(x)`.

2. **Διαβάζουμε** τα όρια του διαστήματος της ρίζας **`a, b`**.

3. **Αν** υπάρχει ρίζα στο διάστημα $[a,b]$ ($f(a) * f(b) < 0$) **τότε**

   a) **Διαβάζουμε** μια αρχική τιμή για το `x`

   b) **Εμφανίζουμε** τις τιμές του `x` και $f(x)$.

4. **Για** όσο $|f(x)| > 0.000001$ και $|f'(x)| \neq 0$:

   α) **Υπολογίζουμε** το νέο $x = x - \frac{f(x)}{f'(x)}$

   β) **Εμφανίζουμε** τις τιμές του `x` και $f(x)$.

3.5 Μέθοδος Διαδοχικών Προσεγγίσεων – Εύρεση Όλων των Ριζών στο [a,b]

- Να τροποποιηθεί το πρόγραμμα 3.4, ώστε να υπολογίζει όλες τις ρίζες της εξίσωσης που περιέχονται στο διάστημα `[a, b]` με τη βοήθεια της μεθόδου των **Διαδοχικών Προσεγγίσεων.** Το διάστημα `[a, b]` θα χωρίζεται με το βήμα `h` στα υπο-διαστήματα $[a, a+h]$, $[a+h, a+2h]$,…., $[b-h, b]$, στο καθένα απ' τα οποία θα ελέγχεται αν υπάρχει ρίζα. Σ' αυτή την περίπτωση, θα χρησιμοποιείται ο αλγόριθμος 3.4 για την εύρεσή της.

Αλγόριθμος

1) **Δηλώνουμε** με τη χρήση του **`#define`** τις συναρτήσεις **`f(x)`, `g(x)`** και **`gp(x)`,** όπου `gp(x)` η παράγωγος της `g(x)`.

2) **Διαβάζουμε** τα όρια του διαστήματος των ριζών `a, b` και το βήμα `h`.

3) **Για** $x0 = a, b, h$

a) **Αν** υπάρχει ρίζα στο υποδιάστημα $[x_0, x_0+h]$ ($f(x_0) * f(x_0+h) < 0$) **τότε**

i. **Δίνουμε** αρχική τιμή στο `x` το `x0`

ii. **Εμφανίζουμε** τις τιμές του `x` και $f(x)$

iii. **Για** όσο $|f(x)| > 0.000001$ και $|g'(x)| < 1$

A) **Υπολογίζουμε** το νέο `x = g(x)`.

B) **Εμφανίζουμε** τις τιμές του `x` και $f(x)$.

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
//
// Βήμα 1 - Δήλωση Συναρτήσεων f(x), g(x), g'(x)
//
#define  f(x) (pow(x,3)-4*x)
#define g(x) (x/2+2/x)
#define gp(x) (1/2.0-2/pow(x,2))
```

```
main() // Άσκηση 3.5 - Διαδοχικές Προσεγγίσεις - Όλες οι Ρίζες
{
float a, b, h, x0, x1, x;
printf("Askhsh 3.5 - Diadoxikes Proseggiseis - Eyresh Olwn twn
Rizwn Polywnymou - f(x)=x^3-4*x, g(x)= x/2 + 2/x\n");
printf("\n");
//
// Βήμα 2 - Διάβασμα Άκρων Διαστήματος [a,b]και βήματος h
//
printf("Dose timh gia ta a b h : ");
scanf ("%f %f %f",&a, &b, &h);
//
// Βήμα 3 - Δημιουργία Υπο-διαστημάτων
//
for (x0 = a; x0 <= b; x0 = x0 + h)
//
// Βήμα 3a - Έλεγχος ύπαρξης Ρίζας στο υπο-διάστημα
//
{
  x1 = x0 + h;
  if ((f(x0) * f(x1)) < 0 )
  {
   //
   // Βήμα 3ai - Αρχική Τιμή στο x το x0
   //
   x = x0;
   //
   // Βήμα 3aii - Εμφάνιση τιμών x, f(x),
   // αν η μέθοδος συγκλίνει
   //
   if ((fabs(gp(x0)) <1 ) && (fabs(gp(x1)) < 1 ))
    printf("x = %f  f(x) = %f\n",x,f(x));
   //
   // Βήμα 3aiii - Εύρεση Ρίζας
   //
   while (( fabs(f(x))> 0.00000001) && (fabs(gp(x)) <1 ))
   {
   //
   // Βήμα 3aiiiA - Εύρεση Νέας Τιμής x
   //
   x = g(x);
   //
   // Βήμα 3aiiiB - Εμφάνιση Νέων τιμών x, f(x)
   //
   printf("x = %f  f(x) = %f\n",x,f(x));
   } //while
   system("Pause");
  } //if
  } //for
system("Pause");
}
```

Εργαστηριακή Άσκηση 3.5 (a – b)

- Δίνεται το Πολυώνυμο $P(x) = p_0 x^n + p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x^1 + p_n$. Να γίνει πρόγραμμα C, το οποίο να υπολογίζει **όλες** τις ρίζες του Πολυωνύμου που περιέχονται στο διάστημα `[a, b]` με τη βοήθεια της μεθόδου των **Newton-Raphson. .** Το διάστημα `[a, b]` θα χωρίζεται με το βήμα `h` στα υπο-διαστήματα $[a, a+h]$, $[a+h, a+2h]$,…., $[b-h, b]$, στο καθένα απ' τα οποία θα ελέγχεται αν υπάρχει ρίζα. Για τον υπολογισμό της Τιμής του Πολυωνύμου και της τιμής της Παραγώγου του Πολυωνύμου θα χρησιμοποιηθεί η συνάρτηση Horner της Εργαστηριακής Άσκησης 3.1 και 3.2.

Αλγόριθμος

1. ***Διαβάζουμε*** τους συντελεστές του Πολυωνύμου $p_0, p_1, \cdots p_n$ (στον πίνακα `p`).
2. ***Εμφανίζουμε*** στην οθόνη τους συντελεστές του Πολυωνύμου $p_0, p_1, \cdots p_n$.
3. **Διαβάζουμε** τα όρια του διαστήματος των ριζών `a, b` και το βήμα `h`.
4. **Για** $x0 = a, b, h$

   a) **Αν** υπάρχει ρίζα στο υποδιάστημα $[x_0, x_0 + h]$ ($f(x_0) * f(x_0 + h) < 0$) **τότε**

   i) Δίνουμε αρχική τιμή στο `x` το `x0`

   ii) Εμφανίζουμε τις τιμές του `x` και $p(x)$

   iii) Για όσο $|p(x)| > 0.000001$ και $|p'(x)| \neq 0$

   A) Υπολογίζουμε το νέο $x = x - \frac{p(x)}{p'(x)}$

   B) Εμφανίζουμε τις τιμές του `x` και $p(x)$.

Παρατηρήσεις

- Οπουδήποτε στον αλγόριθμο εμφανίζεται η Τιμή του Πολυωνύμου ή της Παραγώγου του Πολυωνύμου, π.χ. $p(x), p(a), p(b), p'(x)$ θα χρησιμοποιείται η συνάρτηση `horner(…)` των Εργαστηριακών Ασκήσεων 3.1 (function) και 3.2 (function - procedure).
- Στην Εργαστηριακή Άσκηση 3.1 η function χρησιμοποιείται αντί των $p(x)$ κ.λ.π., ενώ στην Εργαστηριακή Άσκηση 3.2 η function – procedure καλείται και χρησιμοποιείται αντί των $p(x)$ κ.λ.π., η τιμή που επιστρέφει στο $q(n)$ ή στο $r(n-1)$ αντίστοιχα.

Επίλυση Γραμμικών Συστημάτων

4

Αριθμητική Επίλυση Διαγωνίου Συστήματος
Αριθμητική Επίλυση Άνω Τριγωνικού Συστήματος
Αριθμητική Επίλυση Κάτω Τριγωνικού Συστήματος
Μέθοδος Απαλοιφής Gauss
Επαναληπτικές Μέθοδοι Επίλυσης
Γραμ. Συστημάτων (Μέθοδος Gauss-Seidel - Jacobi)

- Στο κεφάλαιο που ακολουθεί παρουσιάζονται αλγόριθμοι και προγράμματα σε C για την επίλυση συστημάτων. Με τον αλγόριθμο για την αριθμητική επίλυση ενός Διαγωνίου Συστήματος εισάγονται οι έννοιες του **δισδιάστατου** πίνακα της C και των εντολών που απαιτούνται για το διάβασμα, την εμφάνιση και τη χρήση δισδιάστατου πίνακα. Τα στοιχεία των πινάκων διαβάζονται από το πληκτρολόγιο και παρουσιάζονται οι εντολές της C για το διάβασμα τα στοιχείων των πινάκων και την εμφάνισή τους στην οθόνη. Ενδεικτικά παρουσιάζονται η μέθοδος Gauss για την επίλυση ενός γραμμικού συστήματος απ' τις άμεσες μεθόδους και η μέθοδος Gauss-Seidel για την επίλυση ενός γραμμικού συστήματος απ' τις επαναληπτικές μεθόδους.

4.1 Αριθμητική Επίλυση Διαγωνίου Συστήματος

- Να γίνει πρόγραμμα το οποίο να επιλύει το Διαγώνιο Σύστημα:

$$A \cdot x = b$$

ή το σύστημα :

$$\begin{bmatrix} a_{11} & 0 & ... & 0 \\ 0 & a_{22} & 0 & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{bmatrix}$$

ή σε μορφή εξισώσεων το σύστημα :

$$\begin{array}{cccccccccccc} a_{11}x_1 & & & & & & & & & = & b_1 \\ & & a_{22}x_2 & & & & & & & = & b_2 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ & & & & ... & a_{ii}x_i & & & & = & b_i \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ & & & & ... & ... & & a_{nn}x_n & = & b_n \end{array}$$

Αλγόριθμος

1. Δίνουμε **τιμές** στα στοιχεία των πινάκων `A` και `b`.
2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3. **Για** τις τιμές του $x_i, i = 1,2,...,n$:

   Υπολογίζουμε το $x_i = \frac{b_i}{a_{i,i}}$.

4. **Εμφανίζουμε** στην οθόνη τα στοιχεία του `x`.

Πρόγραμμα (1ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
main()//Άσκηση 4.1 – Επίλυση Διαγωνίου Συστήματος
{
int i, j;
float a[n+1][n+1], b[n+1], x[n+1];
printf("Askhsh 4.1 – Epilysh Diagwniou Systhmatos\n");
printf("\n");
// Διάβασμα Στοιχείων Πίνακα A
for ( i = 1; i<=n; i++ )
 for ( j = 1; j<=n; j++ )
  if ( i == j )
    {
    printf("Dose timh gia to a[%d,%d] : ",i,j);
    scanf ("%f",&a[i][j]);
    }
  else
    a[i][j] = 0;
// Διάβασμα Στοιχείων Πίνακα b
for ( i = 1; i<=n; i++ )
 {
  printf("Dose timh gia to b[%d] : ",i);
  scanf ("%f",&b[i]);
  }
// Εμφάνιση Στοιχείων Πίνακα A
printf("Pinakas A :\n”);
for ( i = 1; i<=n; i++ )
 {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
 printf("\n");
 }
// Εμφάνιση Στοιχείων Πίνακα b
printf("Pinakas b :  ”);
for ( i = 1; i<=n; i++ )
   printf("%6.1f",b[i]);
printf("\n");
//
// Επίλυση Συστήματος
//
for ( i = 1; i<=n; i++ )
 x[i] = b[i]/a[i][i];
// Εμφάνιση Στοιχείων Πίνακα x
printf("Pinakas x :  ”);
for ( i = 1; i<=n; i++ )
   printf("%6.1f",x[i]);
printf("\n");
system("Pause");
}
```

4.1.1 Δήλωση Πινάκων Δύο Διαστάσεων

- Με την εντολή :

```
float a[n+1][n+1];
```

δηλώσαμε τον δισδιάστατο πίνακα `A` με `n+1` γραμμές και `n+1` στήλες.

- Η δήλωση γενικά ενός δισδιάστατου πίνακα `pin 10×10` θέσεων για πραγματικούς αριθμούς γίνεται με την εντολή :

```
float pin[10][10];
```

με στοιχεία `pin[0][0], pin[0][1],…, pin[0][9], pin[1][0], p[1][1],…, p[1][9],..., pin[9][0], pin[9][1],…, pin[9][9].`

Παρατηρήσεις

- Η δήλωση του πλήθους των στοιχείων του πίνακα μπορεί να γίνει με τη δήλωση σταθεράς. Έτσι, θα είχαμε το ίδιο αποτέλεσμα με τις εντολές :

```
#define n 10
float pin[n][n];
```

- Στο πρόγραμμα χρησιμοποιήθηκε για τη δήλωση του πίνακα `A` η εντολή :

```
float a[n+1][n+1];
```

με στοιχεία `a[0][0], a[0][1],…, a[0][n], a[1][0], a[1][1],…, a[1][n],..., a[n][0], a[n][1],…, a[n][n]` απ' τα οποία δε χρησιμοποιούμε τα στοιχεία της $1^{ης}$ γραμμής και της $1^{ης}$ στήλης.

- Όταν αναφερόμαστε στα στοιχεία ενός δισδιάστατου πίνακα πρέπει να χρησιμοποιούμε δύο μεταβλητές για δείκτες. π.χ. `a[i][j]` είναι το στοιχείο του πίνακα που βρίσκεται στην `i` γραμμή και στη `j` στήλη.

4.1.2 Διάβασμα Στοιχείων ενός Δισδιάστατου Πίνακα

- Με την ομάδα εντολών :

```
for ( i = 1; i<=n; i++ )
 for ( j = 1; j<=n; j++ )
   if ( i == j )
     {
     printf("Dose timh gia to a[%d,%d] : ",i,j);
     scanf ("%f",&a[i][j]);
     }
   else
     a[i][j] = 0;
```

διαβάζουμε `n = 3` πραγματικούς αριθμούς απ' το πληκτρολόγιο χωρίς format **(έναν αριθμό σε κάθε γραμμή)** και τους αποθηκεύουμε στον πίνακα `A`.

Παρατηρήσεις

- Θα μπορούσαμε να δώσουμε συγκεκριμένες τιμές στα στοιχεία του πίνακα `A` στην εντολή δήλωσης του πίνακα, οπότε το *Βήμα 1* είναι περιττό, ενώ το τμήμα δηλώσεων θα γινόταν :

```
#define n 3
float a[n+1][n+1]={ 0,0,0,0,0,1,0,0,0,0,2,0,0,0,0,3};
```

και θα δημιουργούνταν ο πίνακας $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$ από τον οποίο χρησιμοποιούμε το διαγώνιο υπο-πίνακα $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$.

- Αν θέλαμε να διαβάσουμε τα στοιχεία ενός Δισδιάστατου πίνακα κατά γραμμές (`n` στοιχεία στην κάθε γραμμή) θα χρησιμοποιούσαμε την ομάδα εντολών :

```
for ( i = 1; i<=n; i++ )
{
for ( j = 1; j<=n; j++ )
    scanf ("%f",&a[i][j]);
```

και στην εκτέλεση του προγράμματος θα δίναμε *n* στοιχεία σε κάθε γραμμή χωρισμένα με κενά.

4.1.3 Εμφάνιση n Στοιχείων Δισδιάστατου Πίνακα

- Με την ομάδα εντολών :

```
for ( i = 1; i<=n; i++ )
 {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
 printf("\n");
 }
```

εμφανίζουμε στην οθόνη `n × n` πραγματικούς αριθμούς (τα στοιχεία του πίνακα `A`) χωρίς format.

Εργαστηριακή Άσκηση 4.1

- Να γίνει πρόγραμμα το οποίο να επιλύει το Διαγώνιο Σύστημα:

$$A \cdot x = b$$

ή το σύστημα :

$$\begin{bmatrix} 0 & 0 & \dots & 0 & a_{1n} \\ 0 & 0 & \dots & a_{2,n-1} & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & a_{n-1,2} & \dots & 0 & 0 \\ a_{n1} & 0 & \dots & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_{n-1} \\ b_n \end{bmatrix} \end{bmatrix}$$

ή σε μορφή εξισώσεων το σύστημα :

$$\begin{array}{cccccccccccc} & & & & & & & & a_{1n}x_n & = & b_1 \\ & & & & & & a_{2,n-1}x_{n-1} & & & = & b_2 \\ \dots & & \dots & & \dots & \dots & \dots & & \dots & \dots & \dots & \\ \dots & & \dots & & \dots & \dots & \dots & & \dots & = & & \\ & a_{n-1,2}x_2 & & & & & & & & = & b_{n-1} & \\ a_{n1}x_1 & & & & & & & & & = & b_n & \end{array}$$

Αλγόριθμος

1. Δίνουμε **τιμές** στα στοιχεία των πινάκων `A` και `b`.
2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3. **Για** τις τιμές του x_i, $i = n, n-1, \dots, 1$:

   Υπολογίζουμε το $x_i = \dfrac{b_{n-i+1}}{a_{n-i+1,i}}$.

4. **Εμφανίζουμε** στην οθόνη τα στοιχεία του `x`.

4.2 Αριθμητική Επίλυση Κάτω Τριγ. Συστήματος (1ος Τρόπος)

- Να γίνει C πρόγραμμα, το οποίο να επιλύει το Γραμμικό **Κάτω Τριγωνικό** Σύστημα:

$$A \cdot x = b$$

ή το σύστημα : $\begin{bmatrix} a_{11} & 0 & ... & 0 \\ a_{21} & a_{22} & 0 & 0 \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{bmatrix}$

ή σε μορφή εξισώσεων το σύστημα :

$$\begin{bmatrix} a_{11}x_1 & & & & & & & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & & & & & = & b_2 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ a_{n1}x_1 & + & a_{n2}x_2 & + & ... & + & a_{nn}x_n & = & b_n \end{bmatrix}$$

Αλγόριθμος (1ος Τρόπος)

1) **Διαβάζουμε** τα στοιχεία του πίνακα `A`, ενώ το κάθε στοιχείο του πίνακα `b` είναι το άθροισμα των στοιχείων της αντίστοιχης γραμμές του πίνακα `A`, έτσι ώστε η λύση να είναι μονάδες και να μη χρειαστεί επαλήθευση.
2) **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3) **Λύνουμε** το σύστημα ως προς το $x_1 = \frac{b_1}{a_{1,1}}$.
4) **Για** τις τιμές του $x_i, \quad i = 2,3,...,n$:
   a) **Θέτουμε** `sum = 0`
   b) **Υπολογίζουμε** το άθροισμα $sum = \sum_{k=1}^{i-1} a_{i,k} x_k$.
   c) **Υπολογίζουμε** το $x_i = \frac{b_i - sum}{a_{i,i}}$.
   d) Ε**μφανίζουμε** στην οθόνη τα στοιχεία του `x`.

Πρόγραμμα (1ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
main()//Άσκηση 4.2 – Επίλυση Κάτω Τριγωνικού Συστήματος
{
int i, j;
float a[n+1][n+1], b[n+1], x[n+1], sum;
printf("Askhsh 4.2 – Epilysh Katw Trigwnikou Systhmatos\n");
printf("\n");
// Διάβασμα Στοιχείων Πίνακα A, Δημιουργία Πίνακα b
for ( i = 1; i<=n; i++ )
 {
 b[i] = 0;
 for ( j = 1; j<=n; j++ )
  if ( j <= i ) {
    printf("Dose timh gia to a[%d,%d] : ",i,j);
    scanf ("%f",&a[i][j]);
    b[i] = b[i] + a[i][j];
    }
  else
    a[i][j] = 0;
  }
// Εμφάνιση Στοιχείων Πίνακα A
printf("Pinakas A :\n”);
for ( i = 1; i<=n; i++ ) {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
 printf("\n");
 }
// Εμφάνιση Στοιχείων Πίνακα b
printf("Pinakas b :  ”);
for ( i = 1; i<=n; i++ )
   printf("%6.1f",b[i]);
printf("\n");
// Επίλυση Συστήματος
x[1] = b[1]/a[1][1]; // Υπολογισμός x[1]
for ( i = 2; i <= n; i++ ){
 sum = 0;
 for ( j = 1; j<=i-1; j++ )
  sum = sum + a[i][j] * x[j];
 x[i] = (b[i]-sum)/a[i][i];
}
// Εμφάνιση Στοιχείων Πίνακα x
printf("Pinakas x :  ”);
for ( i = 1; i<=n; i++ )
   printf("%6.1f",x[i]);
printf("\n");
system("Pause");
}
```

4.3 Αριθμητική Επίλυση Κάτω Τριγ. Συστήματος (2ος Τρόπος με function)

- Να γίνει C πρόγραμμα, το οποίο να επιλύει το Γραμμικό **Κάτω Τριγωνικό** Σύστημα:

$$A \cdot x = b$$

ή το σύστημα : $\begin{bmatrix} a_{11} & 0 & ... & 0 \\ a_{21} & a_{22} & 0 & 0 \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{bmatrix}$

με την **κλήση function** – **procedure** για τον **υπολογισμό** των στοιχείων του πίνακα `x`.

Αλγόριθμος προγράμματος (2ος Τρόπος)

1. **Διαβάζουμε** τα στοιχεία του πίνακα `A`, ενώ το κάθε στοιχείο του πίνακα `b` είναι το άθροισμα των στοιχείων της αντίστοιχης γραμμές του πίνακα `A`, έτσι ώστε η λύση να είναι μονάδες και να μη χρειαστεί επαλήθευση.
2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3. **Καλούμε** το **function** – **procedure** για τον **υπολογισμό** των στοιχείων του πίνακα `x`
4. **Εμφανίζουμε** στην οθόνη τα στοιχεία του `x`.

Αλγόριθμος function – procedure (2ος Τρόπος)

1) **Λύνουμε** το σύστημα ως προς το $x_1 = \frac{b_1}{a_{1,1}}$.
2) **Για** τις τιμές του x_i, $i = 2,3,...,n$:
   a) **Θέτουμε** `sum = 0`
   b) **Υπολογίζουμε** το άθροισμα $sum = \sum_{k=1}^{i-1} a_{i,k} x_k$.
   c) **Υπολογίζουμε** το $x_i = \frac{b_i - sum}{a_{i,i}}$.

Πρόγραμμα (2ος Τρόπος)

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
kato_trig(float a[][n+1], float b[], float x[]);
main()//Άσκηση 4.3 – Επίλυση Κάτω Τριγωνικού Συστήματος
//      με κλήση function – procedure
{
int i, j;
float a[n+1][n+1], b[n+1], x[n+1], sum;
printf("Askhsh 4.3 – Epilysh Katw Trigwnikou Systhmatos me
function – procedure\n");
printf("\n");
// Διάβασμα Στοιχείων Πίνακα Α, Δημιουργία Πίνακα b
for ( i = 1; i<=n; i++ )
 {
 b[i] = 0;
 for ( j = 1; j<=n; j++ )
  if ( j <= i )
    {
    printf("Dose timh gia to a[%d,%d] : ",i,j);
    scanf ("%f",&a[i][j]);
    b[i] = b[i] + a[i][j];
    }
  else
    a[i][j] = 0;
  }
// Εμφάνιση Στοιχείων Πίνακα Α
printf("Pinakas A :\n");
for ( i = 1; i<=n; i++ )
 {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
 printf("\n");
 }
// Εμφάνιση Στοιχείων Πίνακα b
printf("Pinakas b :  ");
for ( i = 1; i<=n; i++ )
   printf("%6.1f",b[i]);
printf("\n");
kato_trig(a,b,x);// Επίλυση Συστήματος με κλήση function
// Εμφάνιση Στοιχείων Πίνακα x
printf("Pinakas x :  ");
for ( i = 1; i<=n; i++ )
   printf("%6.1f",x[i]);
printf("\n");
system("Pause");
}
```

```
kato_trig(float a[][n+1], float b[], float x[]);
{
int i, j;
x[1] = b[1]/a[1][1]; // Υπολογισμός x[1]
for ( i = 2; i <= n; i++ )
{
 sum = 0;
 for ( j = 1; j<=i-1; j++ )
  sum = sum + a[i][j] * x[j];
 x[i] = (b[i]-sum)/a[i][i];
}
}
```

4.3.1 Πέρασμα Δισδιάστατων Πινάκων σαν Παραμέτρων σε functions

➢ Για το πέρασμα Δισδιάστατων Πινάκων σαν παραμέτρων σε functions ισχύουν τα παρακάτω :

♦ Οι Πίνακες - Παράμετροι θα πρέπει να δηλωθούν στο κυρίως Πρόγραμμα.

♦ Ο Πίνακας περνάει σαν παράμετρος στη δήλωση της function με το όνομά του και αγκύλες, όπου όμως στο δεύτερο ζεύγος αγκυλών περνάει ο αριθμός των στηλών. Στο προηγούμενο παράδειγμα, η δήλωση της function έγινε με την εντολή :

```
kato_trig(float a[][n+1], float b[], float x[]);
```

♦ Η ίδια δήλωση πρέπει να γίνει και πριν το `main()`

♦ Στη χρήση ή κλήση της function χρησιμοποιείται μόνο το όνομα του πίνακα. Στο προηγούμενο παράδειγμα, η κλήση της function έγινε με την εντολή :

```
kato_trig(a,b,x);//Επίλυση Συστήματος με κλήση function
```

Εργαστηριακή Άσκηση 4.3

- Να γίνει C πρόγραμμα, το οποίο να επιλύει το Γραμμικό **Άνω Τριγωνικό** Σύστημα:

$A \cdot x = b$ ή το σύστημα : $\begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ 0 & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{bmatrix}$

ή σε μορφή εξισώσεων το σύστημα

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$
$$a_{22}x_2 + a_{23}x_3 ..... + a_{2n}x_n = b_2$$
$$.....$$
$$a_{nn}x_n = b_n$$

με την **κλήση function** – **procedure** για τον **υπολογισμό** των στοιχείων του πίνακα `x`.

Αλγόριθμος προγράμματος

1) **Διαβάζουμε** τα στοιχεία του πίνακα `A`, ενώ το κάθε στοιχείο του πίνακα `b` είναι το άθροισμα των στοιχείων της αντίστοιχης γραμμές του πίνακα `A`, έτσι ώστε η λύση να είναι μονάδες και να μη χρειαστεί επαλήθευση.
2) **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3) **Καλούμε** το **function** – **procedure** για τον **υπολογισμό** των στοιχείων του πίνακα `x`
4) **Εμφανίζουμε** στην οθόνη τα στοιχεία του `x`.

Αλγόριθμος function – procedure

1) **Λύνουμε** το σύστημα ως προς το $x_n = \frac{b_n}{a_{nn}}$.
2) **Για** τις τιμές του x_i, $i = n-1, n-2, ..., 1$:

   a) **Θέτουμε** `sum = 0`

   b) **Υπολογίζουμε** το άθροισμα $sum = \sum_{k=i+1}^{n} a_{i,k} x_k$.

   c) **Υπολογίζουμε** το $x_i = \frac{b_i - sum}{a_{i,i}}$.

4.4 Άμεσες Μέθοδοι Επίλυσης Γραμμ. Συστημάτων Μέθοδος Απαλοιφής του Gauss

- Δίνεται το γραμμικό σύστημα :

$$Ax = b,\ A \in \Re^{nxn},\ b \in \Re^n,\ x \in \Re^n,\ \det(A) \neq 0$$

και σε μορφή πινάκων : $\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \\ b_n \end{bmatrix}$

Στη μέθοδο **απαλοιφής** του **Gauss**, πριν από το μηδενισμό των στοιχείων της κάθε στήλης κάτω από τη διαγώνιο, για τη μετατροπή του αρχικού συστήματος σε Άνω Τριγωνικό, θα πρέπει να ελέγχουμε αν το διαγώνιο στοιχείο είναι μηδέν, οπότε σ' αυτή την περίπτωση θα πρέπει να ελέγξουμε όλα τα στοιχεία της αντίστοιχης στήλης κάτω από το διαγώνιο και ανταλλάσσουμε τα στοιχεία της γραμμής στην οποία το βρήκαμε με τα στοιχεία της γραμμής του διαγωνίου στοιχείου. Ας δούμε την παραπάνω διαδικασία σαν μια ανεξάρτητη εφαρμογή, πριν την εντάξουμε στον κύριο αλγόριθμο.

4.4.1 Διασφάλιση Μη Μηδενικών Διαγωνίων Στοιχείων Σε Έναν Δισδιάστατο Πίνακα

- Δίνεται ο Δισδιάστατος Πίνακας :

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Να γίνει πρόγραμμα C, το οποίο θα ελέγχει αν το διαγώνιο στοιχείο είναι μηδέν, οπότε σ' αυτή την περίπτωση θα ελέγχει όλα τα στοιχεία της αντίστοιχης στήλης κάτω από το διαγώνιο και θα ανταλλάσσει τα στοιχεία της γραμμής στην οποία βρέθηκε με τα στοιχεία της γραμμής του διαγωνίου στοιχείου.

Αλγόριθμος

1) **Διαβάζουμε** τα στοιχεία του πίνακα A.

2) **Εμφανίζουμε** στην οθόνη τα στοιχεία του πίνακα A.

3) **Για** κάθε γραμμή $j: j = 1,2,...,n-1$:

   a) **Αν** ($a_{jj} = 0$) τότε

      Για τις υπόλοιπες γραμμές $i: i = j+1, j+2,...,n$:

         Αν ($a_{ji} \neq 0$) τότε **Ανταλλάσσουμε** τα στοιχεία των γραμμών i και j.

4) Εμφανίζουμε στην οθόνη τα στοιχεία του πίνακα A.

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
main()//Άσκηση 4.4.1 – Διασφάλιση Μη Μηδενικών Διαγωνίων Στοιχείων
{
int i, j, jj, k;
float a[n+1][n+1], temp;
printf("Askhsh 4.4.1 - Diasfalish Mh Mhdenikwn Diagwniwn
Stoixeiwn se Disdiastato Pinaka\n");
printf("\n");
// Διάβασμα Στοιχείων Πίνακα A
for ( i = 1; i<=n; i++ )
for ( j = 1; j<=n; j++ )
  {
    printf("Dose timh gia to a[%d,%d] : ",i,j);
    scanf ("%f",&a[i][j]);
  }

// Εμφάνιση Στοιχείων Πίνακα A
printf("Arxikos Pinakas A:\n");
for ( i = 1; i<=n; i++ )
 {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
 printf("\n");
 }
printf("\n");

//Ελεγχος Μηδενικών Διαγωνίων Στοιχείων στις στήλες 1..n-1
for ( j = 1; j<=n-1; j++ )
if ( a[j][j] == 0 ) // Μηδενικό Διαγώνιο Στοιχείο
```

```
 for ( i = j+1; i<=n; i++ ) // Ελεγχος σε γραμμές j+1..n
   if ( a[i][j] != 0 ) //Μη Μηδενικό Στοιχείο γραμμή i
    {
     //Ανταλλαγή Στοιχείων γραμμών i και j
     for ( k = 1; k<=n; k++ )
     {
       temp = a[i][k];
       a[i][k] = a[j][k];
       a[j][k] = temp;
     }

  // Εμφάνιση Στοιχείων Πίνακα A μετά την Ανταλλαγή
  printf("Pinakas A - Antallagh stoixeion grammmon %d %d\n", i,j);
    for ( i = 1; i<=n; i++ )
    {
    for ( jj = 1; jj<=n; jj++ )
     printf("%6.1f",a[i][jj]);
    printf("\n");
    }
   break;
   }

// Εμφάνιση Στοιχείων Τελικού Πίνακα A
printf("Telikos Pinakas A \n");
for ( i = 1; i<=n; i++ )
 {
 for ( jj = 1; jj<=n; jj++ )
   printf("%6.1f",a[i][jj]);
 printf("\n");
 }
system("Pause");
}
```

4.4.1.1 Η Εντολή break

Η Εντολή **break** μπορεί να χρησιμοποιηθεί στο σώμα ενός βρόχου, το οποίο τερματίζεται στο σημείο που εμφανίζεται η εντολή. Η σύνταξή της είναι :

```
break;
```

Εργαστηριακή Άσκηση 4.4.1

- Δίνεται ο Δισδιάστατος Πίνακας :

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Να γίνει πρόγραμμα C, το οποίο θα βρίσκει το μεγαλύτερο κατ' απόλυτη τιμή στοιχείο σε κάθε στήλη, οπότε θα ελέγχει όλα τα στοιχεία της αντίστοιχης στήλης από το διαγώνιο και κάτω και θα ανταλλάσσει τα στοιχεία της γραμμής στην οποία βρέθηκε με τα στοιχεία της γραμμής του διαγωνίου στοιχείου.

Αλγόριθμος

1) **Διαβάζουμε** τα στοιχεία του πίνακα `A`.
2) **Εμφανίζουμε** στην οθόνη τα στοιχεία του πίνακα `A`.
3) **Για** κάθε στήλη $j: j = 1,2,...,n-1$:
   i. $max = |a_{jj}|$
   ii. $grammh = j$
   iii. **Για** τις γραμμές $i: i = j, j+1,...,n$:
      Αν ($|a_{ij}| > max$) τότε
         $max = |a_{ij}|$
         $grammh = i$
   iv. **Αν** ($j \neq grammh$) τότε
         Ανταλλάσσουμε τα στοιχεία των γραμμών `grammh` και `j`.
4) Εμφανίζουμε στην οθόνη τα στοιχεία του πίνακα `A`.

4.4.2 Μέθοδος Απαλοιφής του Gauss (Χωρίς Οδήγηση)

- Δίνεται το γραμμικό σύστημα :

$$Ax = b,\ A \in \Re^{nxn},\ b \in \Re^{n},\ x \in \Re^{n},\ \det(A) \neq 0$$

και σε μορφή πινάκων : $\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \\ b_n \end{bmatrix}$

Να γίνει πρόγραμμα σε C που να μετατρέπει το αρχικό σύστημα σε Άνω Τριγωνικό, αφού πρώτα θα ελέγχει αν το διαγώνιο στοιχείο είναι μηδέν, οπότε σ' αυτή την περίπτωση θα χρησιμοποιεί τον Αλγόριθμο 4.4.1.

Αλγόριθμος

1. **Διαβάζουμε** τα στοιχεία των πινάκων `A` και `b`.
2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3. **Για** κάθε γραμμή $j: j = 1,2,...,n-1$:

   a) **Αν** ($a_{jj} = 0$) τότε

      Για τις υπόλοιπες γραμμές $i : i = j+1, j+2,...,n$:

         Αν ($a_{ji} \neq 0$) τότε

            Ανταλλάσσουμε τα στοιχεία των γραμμών `i` και `j`.

   b) Για τις γραμμές $i : i = j+1,...,n$:

      i) Υπολογίζουμε τα $\lambda_i = \dfrac{a_{ij}}{a_{jj}}$

      ii) Υπολογίζουμε τα νέα $a_{ik} = a_{ik} - \lambda_i * a_{jk},\ k = 1,...,n$

      iii) Υπολογίζουμε το νέο $b_i = b_i - \lambda_i * b_j$

4. Εμφανίζουμε στην οθόνη τα νέα στοιχεία των πινάκων `A` και `b`.

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
main()//Άσκηση 4.4.2 – Απαλοιφή Gauss (Μη Μηδενικά Διαγώνια Στοιχεία)
{
int i, j, ii, jj, k;
float a[n+1][n+1], b[n+1],l[n+1],temp;
printf("Askhsh 4.4.2 - Apaloifh Gauss me Diasfalish Mh
Mhdenikwn Diagwniwn Stoixeiwn se Disdiastato Pinaka\n");
printf("\n");
// Διάβασμα Στοιχείων Πίνακα A
for ( i = 1; i<=n; i++ )
 for ( j = 1; j<=n; j++ )  {
    printf("Dose timh gia to a[%d,%d] : ",i,j);
    scanf ("%f",&a[i][j]);
  }
// Διάβασμα Στοιχείων Πίνακα b
for ( i = 1; i<=n; i++ )  {
    printf("Dose timh gia to b[%d] : ",i);
    scanf ("%f",&b[i]);
  }
// Εμφάνιση Στοιχείων Πίνακα A
printf("Arxikos Pinakas A:\n");
for ( i = 1; i<=n; i++ ) {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
 printf("\n");
 }
printf("\n");
// Εμφάνιση Στοιχείων Πίνακα b
printf("Arxikos Pinakas b:\n");
for ( i = 1; i<=n; i++ )
   printf("%6.1f",b[i]);
 printf("\n");

//Ελεγχος Μηδενικών Διαγωνίων Στοιχείων στις στήλες 1..n-1 - Απαλοιφή
for ( j = 1; j<=n-1; j++ )
if ( a[j][j] == 0 ) // Μηδενικό Διαγώνιο Στοιχείο
 for ( i = j+1; i<=n; i++ ) // Ελεγχος σε γραμμές j+1..n
   if ( a[i][j] != 0 ) //Μη Μηδενικό Στοιχείο γραμμή i
    {
     //Ανταλλαγή Στοιχείων γραμμών i και j
     for ( k = 1; k<=n; k++ )    {
       temp = a[i][k];
       a[i][k] = a[j][k];
       a[j][k] = temp;
     }
      temp = b[i];
      b[i] = b[j];
```

```
      b[j] = temp;
   // Εμφάνιση Στοιχείων Πίνακα A μετά την Ανταλλαγή
   printf("Pinakas A - Antallagh stoixeion grammmon %d %d\n", i,j);
     for ( ii = 1; ii<=n; ii++ )
     {
     for ( jj = 1; jj<=n; jj++ )
      printf("%6.1f",a[ii][jj]);
     printf("\n");
     }
   // Εμφάνιση Στοιχείων Πίνακα b μετά την Ανταλλαγή
   printf("Pinakas b - Antallagh stoixeion grammmon %d %d\n", i,j);
     for ( ii = 1; ii<=n; ii++ )
      printf("%6.1f",b[ii]);
     printf("\n");
   break;
    }
for ( i = j+1; i<=n; i++ )//Απαλοιφή Αγνώστου σε γραμμές j+1..n
    {
     l[i] = a[i][j]/a[j][j];
     for ( k = 1; k<=n; k++ )
       a[i][k]=  a[i][k] - l[i] * a[j][k];
     b[i]=  b[i] - l[i] * b[j];
    }
   // Εμφάνιση Στοιχείων Πίνακα A μετά την Απαλοιφή
   printf("Pinakas A - meta thn Apaloifh\n");
     for ( ii = 1; ii<=n; ii++ )
     {
     for ( jj = 1; jj<=n; jj++ )
      printf("%6.1f",a[ii][jj]);
     printf("\n");
     }
   // Εμφάνιση Στοιχείων Πίνακα b μετά την Απαλοιφή
   printf("Pinakas b - meta thn Apaloifh \n");
     for ( ii = 1; ii<=n; ii++ )
      printf("%6.1f",b[ii]);
     printf("\n");
}
// Εμφάνιση Στοιχείων Τελικού Πίνακα A
printf("Telikos Pinakas A \n");
for ( i = 1; i<=n; i++ )
 {
 for ( jj = 1; jj<=n; jj++ )
   printf("%6.1f",a[i][jj]);
 printf("\n");
 }
// Εμφάνιση Στοιχείων Τελικού Πίνακα b
printf("Telikos Pinakas b \n");
for ( i = 1; i<=n; i++ )
   printf("%6.1f",b[i]);
 printf("\n");
system("Pause");
}
```

Εργαστηριακή Άσκηση 4.4.2

- Δίνεται το γραμμικό σύστημα :

$$Ax = b,\ A \in \Re^{nxn},\ b \in \Re^{n},\ x \in \Re^{n},\ \det(A) \neq 0$$

και σε μορφή πινάκων : $\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \\ b_n \end{bmatrix}$

Να γίνει πρόγραμμα σε C που να μετατρέπει το αρχικό σύστημα σε Άνω Τριγωνικό, χρησιμοποιώντας τον Αλγόριθμο 4.4.2 με τον οποίο θα βρίσκει το μεγαλύτερο κατ' απόλυτη τιμή στοιχείο σε κάθε στήλη και θα το χρησιμοποιεί σαν διαγώνιο στοιχείο. Μετά, θα καλεί τη `function ano_trig()` με την οποία θα λύνει το Άνω Τριγωνικό Σύστημα και θα εμφανίζει τη λύση.

Αλγόριθμος

1. **Διαβάζουμε** τα στοιχεία των πινάκων `A` και `b`.
2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3. **Για** κάθε στήλη $j: j = 1,2,...,n-1$:
    i. Βρίσκουμε το μεγαλύτερο κατ' απόλυτη τιμή στοιχείο στη στήλη
    ii. Κάνουμε απαλοιφή του `j` αγνώστου στη στήλη
4. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
5. **Καλούμε** τη function με την οποία λύνουμε το Άνω Τριγωνικό Σύστημα
6. **Εμφανίζουμε** τη λύση του Άνω Τριγωνικού Συστήματος

4.5 Επαναληπτικές Μέθοδοι Επίλυσης Γραμμ. Συστημάτων Μέθοδος Gauss - Seidel

- Δίνεται το γραμμικό σύστημα :

$$Ax = b,\ A \in \Re^{nxn},\ b \in \Re^{n},\ x \in \Re^{n},\ \det(A) \neq 0$$

Να γίνει πρόγραμμα C, το οποίο να ελέγχει αν για τη μήτρα `A` υπάρχει Υπεροχή Διαγωνίου Στοιχείου Κατά Γραμμές, δηλαδή ισχύει $|a_{i,i}| > \sum_{\substack{i=1 \\ i \neq j}}^{n} |a_{i,j}| \quad \forall i : i = 1,2,...,n$ και αν ναι, να εφαρμόζει τη διαδικασία των διαδοχικών προσεγγίσεων **Gauss-Seidel** και να επιλύει το σύστημα με μία επιθυμητή προσέγγιση `0.000001`.

Αλγόριθμος

1. **Διαβάζουμε** τα στοιχεία του πίνακα `A` και δημιουργούμε τα στοιχεία του πίνακα `b` να είναι το άθροισμα των στοιχείων κάθε γραμμής του πίνακα `A`.
2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.
3. **Ελέγχουμε** αν για τη μήτρα `A` υπάρχει **Υπεροχή** Διαγωνίου Στοιχείου Κατά Γραμμές **:**

   i) **Θέτουμε** `flag = 1`
   ii) **Για** κάθε γραμμή $i : i = 1,2,...,n$:

   a) **Θέτουμε** `sum = 0`

   b) **Υπολογίζουμε** το άθροισμα $sum = \sum_{j=1}^{n} | a_{i,j} |$.

   c) **Αφαιρούμε** απ' το `sum` το $| a_{i,i} |$.

   d) Αν $| a_{i,i} | < sum$, **θέτουμε** στο `flag` την τιμή `0` και **σταματάμε** το βρόχο.

4. Αν **υπάρχει** Υπεροχή του Διαγωνίου Στοιχείου του πίνακα `A ( flag = 1 )`, Επιλύουμε το Σύστημα με τη Μέθοδο gauss-seidel :

   i) **Θέτουμε** αρχικές τιμές στα στοιχεία των πινάκων `x` και `oldx`.

   ii) **Υπολογίζουμε** το σφάλμα $\sum_{i=1}^{n} | x_i - oldx_i |$.

   iii) **Για όσο** το σφάλμα είναι μεγαλύτερο του `0.000001` **:**

   a) **Αποθηκεύουμε** τις προηγούμενες τιμές του πίνακα `x` στον πίνακα `oldx`.

   b) **Για** τις τιμές του $x_i, i = 1,2,...,n$:

i) **Θέτουμε** `sum = 0`

ii) **Υπολογίζουμε** το άθροισμα $sum = \sum_{j=1}^{n} a_{ij} x_j$.

iii) **Αφαιρούμε** απ' το `sum` το $a_{ii} * x_i$.

iv) **Υπολογίζουμε** το νέο $x_i = \frac{b_i - sum}{a_{ii}}$.

c) **Υπολογίζουμε** το νέο σφάλμα $\sum_{i=1}^{n} | x_i - oldx_i |$.

d) **Εμφανίζουμε** στην οθόνη τα στοιχεία του πίνακα `x` και το Σφάλμα.

5. **Εμφανίζουμε** στην οθόνη τα στοιχεία του πίνακα `x` και το Τελικό Σφάλμα.

6. Αν **δεν υπάρχει** Υπεροχή του Διαγωνίου Στοιχείου του πίνακα `A` **εμφανίζουμε** το μήνυμα "Δεν υπάρχει υπεροχή - το σύστημα δε λύνεται".

Πρόγραμμα

```
#include <stdio.h>
#include <stdlib.h>
#define n 3
main() // Ασκηση 4.5 - Μέθοδος Gauss-Seidel με Έλεγχο Υπεροχής Κατά Γραμμές
{
int i, j, flag;
float a[n+1][n+1], b[n+1], x[n+1], oldx[n+1], sum, sfalma;
printf("Askhsh 4.5\n");
printf("Gauss - Seidel me Elegxo Yperoxhs Kata Grammes\n");
printf("\n");
// Γέμισμα Πινάκων A, b
for ( i = 1; i<=n; i++ )
 {
 b[i] = 0;
 for ( j = 1; j<=n; j++ )
  {
    printf("Dose timh gia to a[%d,%d] : ",i,j);
    scanf ("%f",&a[i][j]);
    b[i] = b[i] + a[i][j];
  }
}
// Εμφάνιση Πινάκων A, b
printf("Arxikos Pinakas A\n");
printf("\n");
for ( i = 1; i<=n; i++ )
 {
 for ( j = 1; j<=n; j++ )
   printf("%6.1f",a[i][j]);
   printf("\n");
 }
```

```
printf("\n");
printf("Arxikos Pinakas b\n");
printf("\n");
for ( i = 1; i<=n; i++ )
   printf("%6.1f",b[i]);
printf("\n");
printf("\n");
// Έλεγχος Υπεροχής Κατά Γραμμές
flag = 1;
for ( i = 1; i<=n; i++ )//Για Κάθε Γραμμή i
 {
  sum = 0;
  for ( j = 1; j<=n; j++ )
    sum = sum + fabs(a[i][j]);
  sum = sum - fabs(a[i][i]);
  if ( fabs(a[i][i]) <= sum )
   {
    flag = 0;
    break;
   }
 }
if (flag != 0) // Υπάρχει Υπεροχή Κατά Γραμμές
 {
 // Αρχικές Τιμές στα  x, oldx
for ( i = 1; i<=n; i++ )
  {
   x[i] = 0;
   oldx[i] = 1;
   }
// Υπολογισμός Σφάλματος
sfalma = 0;
for ( i = 1; i<=n; i++ )
  sfalma = sfalma + fabs(x[i] - oldx[i]);
// Εφαρμογή Μεθόδου Gauss-Seidel
while (sfalma >= 0.000001)
 {
  for ( i = 1; i<=n; i++ ) // Αποθήκευση Παλιών Τιμών x
   oldx[i] = x[i];
  for ( i = 1; i<=n; i++ ) // Εύρεση Νέων Τιμών x
  {
   sum = 0;
   for ( j = 1; j<=n; j++ )
    sum = sum + a[i][j] * x[j];
   sum = sum - a[i][i] * x[i];
   x[i] = (b[i] - sum)/a[i][i];
  }
// Υπολογισμός Νέου Σφάλματος

sfalma = 0;
for ( i = 1; i<=n; i++ )
  sfalma = sfalma + fabs(x[i] - oldx[i]);
```

```
// Εμφάνιση x, Σφάλματος
 printf("Pinakas x :  ");
 for ( i = 1; i<=n; i++ )
   printf("%12.6f",x[i]);
 printf("Sfalma = %16.12f\n",sfalma);
 printf("\n");
 }//while
// Εμφάνιση Τελικού x, Σφάλματος
 printf("Pinakas x :  ");
 for ( i = 1; i<=n; i++ )
   printf("%12.6f",x[i]);
printf("Sfalma = %16.12f\n",sfalma);
 printf("\n");
}//if
else // Δεν Υπάρχει Υπεροχή Κατά Γραμμές
 {
 printf("Den mporei na efarmostei h methodos\n");
}
system("Pause");
}
```

Εργαστηριακή Άσκηση 4.5

- Δίνεται το γραμμικό σύστημα :

$$Ax = b,\ A \in \Re^{nxn},\ b \in \Re^n,\ x \in \Re^n,\ \det(A) \neq 0$$

Να γίνει πρόγραμμα C, το οποίο να ελέγχει αν για τη μήτρα `A` υπάρχει Υπεροχή Διαγωνίου Στοιχείου Κατά Γραμμές ή Κατά Στήλες, δηλαδή ισχύει $|a_{i,i}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{i,j}| \quad \forall i:\ i = 1,2,...,n$ ή $|a_{j,j}| > \sum_{\substack{i=1 \\ j \neq i}}^{n} |a_{i,j}| \quad \forall j:\ j = 1,2,...,n$ και αν ναι, να εφαρμόζει τη διαδικασία των διαδοχικών προσεγγίσεων **Jacobi** και να επιλύει το σύστημα με μία επιθυμητή προσέγγιση `0.000001`.

Αλγόριθμος

1. **Διαβάζουμε** τα στοιχεία του πίνακα `A` και δημιουργούμε τα στοιχεία του πίνακα `b` να είναι το άθροισμα των στοιχείων κάθε γραμμής του πίνακα `A`.

2. **Εμφανίζουμε** στην οθόνη τα στοιχεία των πινάκων `A` και `b`.

3. **Ελέγχουμε** αν για τη μήτρα `A` υπάρχει **Υπεροχή** Διαγωνίου Στοιχείου Κατά Γραμμές **:**

i) **Θέτουμε** `flag = 1`

ii) **Για** κάθε γραμμή $i: i = 1,2,...,n$:

a) **Θέτουμε** `sum = 0`

b) **Υπολογίζουμε** το άθροισμα $sum = \sum_{j=1}^{n} |a_{i,j}|$.

c) **Αφαιρούμε** απ' το `sum` το $|a_{i,i}|$.

d) Αν $|a_{i,i}| < sum$, **θέτουμε** στο `flag` την τιμή `0` και **σταματάμε** το βρόχο.

4. **Αν `flag = 0`, Ελέγχουμε** αν για τη μήτρα `A` υπάρχει **Υπεροχή** Διαγωνίου Στοιχείου Κατά Στήλες **:**

i) **Θέτουμε** `flag = 1`

ii) Για κάθε στήλη $j: j = 1,2,...,n$:

a) **Θέτουμε** `sum = 0`

b) **Υπολογίζουμε** το άθροισμα $sum = \sum_{i=1}^{n} | a_{i,j} |$.

c) **Αφαιρούμε** απ' το `sum` το $| a_{j,j} |$.

d) Αν $| a_{jj} | < sum$, **θέτουμε** στο `flag` την τιμή `0` και **σταματάμε** το βρόχο.

5. Αν **υπάρχει** Υπεροχή του Διαγωνίου Στοιχείου του πίνακα `A ( flag = 1 )`, Επιλύουμε το Σύστημα με τη Μέθοδο Jacobi :

   i. **Θέτουμε** αρχικές τιμές στα στοιχεία των πινάκων `x` και `oldx`.
   ii. **Για όσο** το σφάλμα είναι μεγαλύτερο του `0.000001` :

   a) **Αποθηκεύουμε** τις προηγούμενες τιμές του πίνακα `x` στον πίνακα `oldx`.
   b) **Για** τις τιμές του $x_i, i = 1,2,...,n$:

      i) **Θέτουμε** `sum = 0`.

      ii) **Υπολογίζουμε** το άθροισμα $sum = \sum_{j=1}^{n} a_{ij} oldx_j$.

      iii) **Αφαιρούμε** απ' το `sum` το $a_{ii} * old x_i$.

      iv) **Υπολογίζουμε** το νέο $x_i = \frac{b_i - sum}{a_{ii}}$.

   c) **Εμφανίζουμε** στην οθόνη τα στοιχεία του πίνακα `x` και το Σφάλμα.

6. **Εμφανίζουμε** στην οθόνη τα στοιχεία του πίνακα `x` και το Τελικό Σφάλμα.

7. Αν **δεν υπάρχει** Υπεροχή του Διαγωνίου Στοιχείου του πίνακα `A` **εμφανίζουμε** το μήνυμα "Δεν υπάρχει υπεροχή - το σύστημα δε λύνεται".

Παρατήρηση

Υπολογίζουμε το σφάλμα $\sum_{i=1}^{n} | x_i - oldx_i |$ με `function`