Δ.Π.Θ

Εισαγωγή στην Επιστήμη των Υπολογιστών

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ & ΔΙΟΙΚΗΣΗΣ

1ο Φυλλάδιο Ασκήσεων

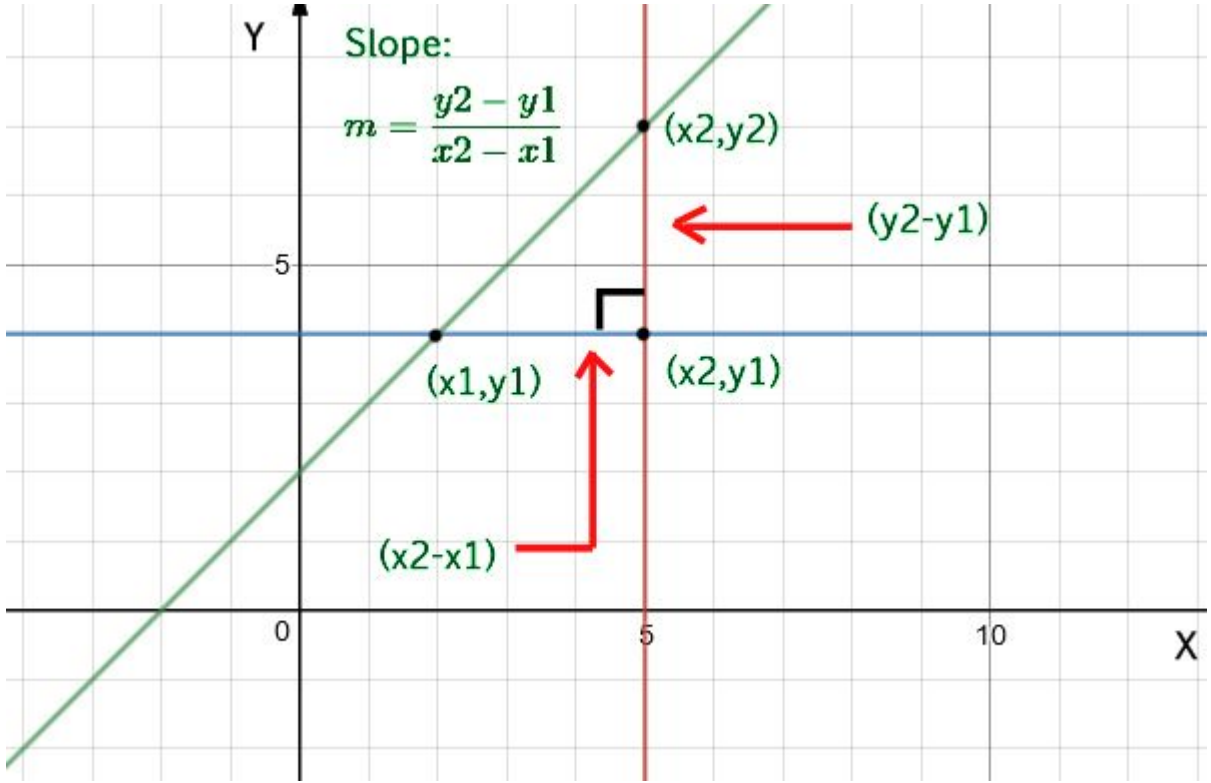
(Ενδεικτικές Λύσεις)

Dr. Αθανάσιος Μπαλαφούτης
Εργαστηριακό Διδακτικό Προσωπικό
Τομέας Συστημάτων Παραγωγής
Εργαστήριο Ρομποτικής και Αυτοματισμών
abalafou@pme.duth.gr
Γραφείο 304, τηλ.: 25410 – 79892

Άσκηση 1

Να γραφεί αλγόριθμος που θα δέχεται ως είσοδο τις συντεταγμένες δύο σημείων (X1,Y1), (X2,Y2), θα υπολογίζει και θα εμφανίζει την κλίση της ευθείας που σχηματίζουν τα σημεία αυτά.
Να ελεγχθούν όλες οι δυνατές περιπτώσεις.

Άσκηση 1



Άσκηση 1 - Λύση


```c
#include <stdio.h>

int main()
{
   float x1, x2, y1, y2, slope;
   printf("Enter X1, Y1 for the first point: ");
   scanf("%f %f", &x1, &y1);
   printf("Enter X2, Y2 for the second point: ");
   scanf("%f %f", &x2, &y2);

   if ((x2 - x1) != 0){
      slope = (y2 - y1)/ (x2 - x1);
      printf("The slope is: %.2f", slope);
   }
   else printf("The slope is undefined");

   return 0;
}
```

Άσκηση 2

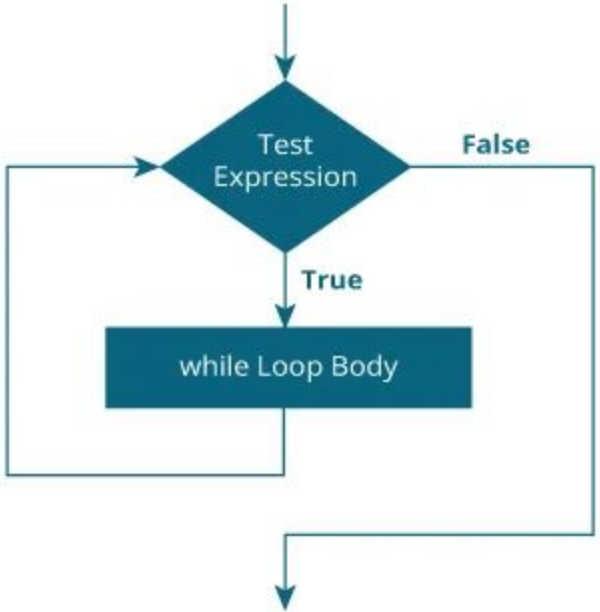
Ένας πωλητής λαμβάνει προμήθεια 4% για πωλήσεις μέχρι τις 200.000 € και η προμήθεια διπλασιάζεται (δηλ. γίνεται 8%) μόνον για τις πωλήσεις που υπερβαίνουν τις 200.000 € .
Να γράψετε έναν αλγόριθμο για να υπολογίσετε το συνολικό ποσό προμήθειας, αν δίνεται ως δεδομένο το ποσό των πωλήσεων .

Άσκηση 2 - Πρώτη Λύση

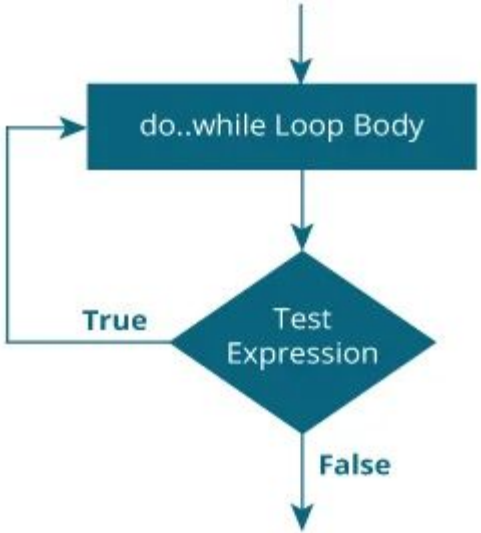

```c
#include <stdio.h>

int main()
{
   float poliseis, promithia;
   printf("Δώσε τις πωλήσεις: ");
   scanf("%f", &poliseis);

   if(poliseis <= 200000)
      promithia = poliseis * 0.04;
   else
      promithia = 200000 * 0.04 + (poliseis - 200000) * 0.08;

   printf("Η προμήθεια θα είναι: %.2f", promithia);
   return 0;
}
```

Εντολές Επανάληψης: while και do…while

Εντολή while



Εντολή do…while



Άσκηση 2 - Καλύτερη Λύση


```c
#include <stdio.h>

int main()
{
   float poliseis, promithia;
   do{
      printf("Δώσε τις πωλήσεις: ");
      scanf("%f", &poliseis);
   }
   while(poliseis <= 0);

   if(poliseis <= 200000)
      promithia = poliseis * 0.04;
   else
      promithia = 20000 * 0.04 + (poliseis - 200000) * 0.08;

   printf("Η προμήθεια θα είναι: %.2f", promithia);
   return 0;
}
```

Έλεγχος εγκυρότητας τιμών

Άσκηση 3

Δίνεται ο πίνακας τιμών για τους καταναλωτές φυσικού αερίου :

Κατανάλωση φυσικού αερίου	Κόστος σε €
Έως 50 m^3	Ελάχιστο κόστος 7
Επόμενα 150 m^3	0.75 / m^3
Επόμενα 200 m^3	0.65 / m^3
Άνω των 400 m^3	0.45 / m^3

Να γραφεί αλγόριθμος που θα υπολογίζει και θα εμφανίζει το κόστος για μια δεδομένη ποσότητα κατανάλωσης.
Δεδομένα εισόδου θα είναι η παρούσα και η προηγούμενη ένδειξη του μετρητή κατανάλωσης.
Κατά την εισαγωγή πρέπει να ελέγχεται η συνθήκη: **παρούσα ένδειξη >= προηγούμενη ένδειξη**.
Η εμφάνιση θα γίνεται αναλυτικά για κάθε κατηγορία τιμολόγησης και στο τέλος θα υπάρχει το συνολικό κόστος.
Η χρέωση είναι κλιμακωτή.

Άσκηση 3

Ανάλυση πίνακα κλιμακωτής χρέωσης

Κατανάλωση φυσικού αερίου	Κόστος σε €
Έως 50 m^3	Ελάχιστο κόστος 7
Επόμενα 150 m^3	0.75 / m^3
Επόμενα 200 m^3	0.65 / m^3
Άνω των 400 m^3	0.45 / m^3

Κατανάλωση φυσικού αερίου	Κόστος σε €
Έως 50 m^3	7€
Από 51 m^3 μέχρι 200 m^3	0.75€ / m^3
Από 201 m^3 μέχρι 400 m^3	0.65€ / m^3
Άνω των 400 m^3	0.45€ / m^3

Άσκηση 3 - Πρώτη Λύση (1/2)


```c
#include <stdio.h>

int main()
{
    int parousa_endiksi, palia_endiksi, katanalosi;
    float kostos;

    do{
        printf("Δώσε την παρούσα και την παλιά ένδειξη του μετρητή: ");
        scanf("%d %d", &parousa_endiksi, &palia_endiksi);
    }
    while(parousa_endiksi <= palia_endiksi);

    katanalosi = parousa_endiksi - palia_endiksi;

    // ... συνεχίζεται ...
```

Άσκηση 3 - Πρώτη Λύση (2/2)


```
    // ...

    if(katanalosi <= 50)
        kostos = 7;
    else if(katanalosi <= 200)
        kostos = 7 + 0.75 * (katanalosi - 50);
    else if(katanalosi <= 400)
        kostos = 7 + 0.75 * 150 + 0.65 * (katanalosi - 200);
    else
        kostos = 7 + 0.75 * 150 + 0.65 * 200 + 0.45 * (katanalosi - 400);

    printf("Το συνολικό κόστος θα είναι: %.2f ", kostos);
    return 0;
}
```

Η λύση αυτή **δεν** εμφανίζει το αναλυτικό κόστος κάθε κατηγορίας τιμολόγησης

Άσκηση 3 - Καλύτερη Λύση (1/3)


```c
#include <stdio.h>

int main()
{
    int parousa_endiksi, palia_endiksi, katanalosi;
    float kostos_katigorias, sinoliko_kostos;

    do{
        printf("Δώσε την παρούσα και την παλιά ένδειξη του μετρητή: ");
        scanf("%d %d", &parousa_endiksi, &palia_endiksi);
    }
    while(parousa_endiksi < palia_endiksi);

    katanalosi = parousa_endiksi - palia_endiksi;
    sinoliko_kostos = 0;

    // ... συνεχίζεται ...
```

Άσκηση 3 - Καλύτερη Λύση (2/3)


```
// ...

if(katanalosi > 400){
    kostos_katigorias = 0.45 * (katanalosi - 400);
    printf("Άνω των 400 : %.2f \n", kostos_katigorias);
    sinoliko_kostos += kostos_katigorias;
    katanalosi -= (katanalosi - 400);
}
if(katanalosi > 200){
    kostos_katigorias = 0.65 * (katanalosi - 200);
    printf("Επόμενα 200 : %.2f \n", kostos_katigorias);
    sinoliko_kostos += kostos_katigorias;
    katanalosi -= (katanalosi - 200);
}

// ... συνεχίζεται ...
```

Άσκηση 3 - Καλύτερη Λύση (3/3)


```
    // ...

    if(katanalosi > 50){
        kostos_katigorias = 0.75 * (katanalosi - 50);
        printf("Επόμενα 150 : %.2f \n", kostos_katigorias);
        sinoliko_kostos += kostos_katigorias;
        katanalosi -= (katanalosi - 50);
    }
    if(katanalosi > 0){
        kostos_katigorias = 7;
        printf("Έως 50 : %.2f \n", kostos_katigorias);
        sinoliko_kostos += kostos_katigorias;
    }

    printf("Το συνολικό κόστος θα είναι: %.2f ", sinoliko_kostos);
    return 0;
}
```

Τύποι Δεδομένων (1/2)

Τύπος Δεδομένων	Μέγεθος (bytes)	Εύρος τιμών	Μορφοποίηση
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld

Τύποι Δεδομένων (2/2)

Τύπος Δεδομένων	Μέγεθος (bytes)	Εύρος τιμών	Μορφοποίηση
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%hd
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

Άσκηση 4

Το τετράγωνο ενός ακεραίου αριθμού Ν μπορεί να υπολογιστεί προσθέτοντας όλους τους ακέραιους από το 1 έως το Ν και επιστρέφοντας πάλι πίσω στο 1,

π.χ.: $4^2 = 1 + 2 + 3 + 4 + 3 + 2 + 1 = 16$

Να γραφεί ο κατάλληλος αλγόριθμος που θα υπολογίζει και θα εμφανίζει το τετράγωνο οποιουδήποτε ακέραιου Ν χρησιμοποιώντας τη μέθοδο αυτή.
Ο αλγόριθμος θα δέχεται ως είσοδο τον αριθμό Ν.
Κατά την εισαγωγή θα ελέγχεται η συνθήκη Ν > 0.

Άσκηση 4 - Λύση


```c
#include <stdio.h>

int main()
{
    int n, tetragono;
    do{
        printf("Δώσε έναν θετικό ακέραιο αριθμό: ");
        scanf("%d", &n);
    }
    while(n <= 0);
    tetragono = 0;
    for(int i = 1; i <= n; i++)
        tetragono += i;
    for(int i = n-1; i >= 1; i--)
        tetragono += i;

    printf("Το τετράγωνο του %d, είναι: %d", n, tetragono);
    return 0;
}
```

Άσκηση 5

Να γραφεί αλγόριθμος που θα βρίσκει σε ποιον όρο το άθροισμα: 1+2+3+… γίνεται μεγαλύτερο του 2000.

Άσκηση 5 - Λύση


```c
#include <stdio.h>

int main()
{
    int sum, i;

    sum = 0;
    i = 0;

    while(sum <= 2000){
        i++;
        sum += i;
    }

    printf("Το άθροισμα γίνεται > 2000 για i: %d", i);
    return 0;
}
```

Άσκηση 5 - Διαφορετική Λύση


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

int main()
{
    int sum, i;

    sum = 0;
    i = 0;

    while(true){
        i++;
        sum += i;
        if(sum > 2000) break;
    }

    printf("Το άθροισμα γίνεται > 2000 για i: %d", i);
    return 0;
}
```

Άσκηση 5 - Μια ακόμη λύση


```c
#include <stdio.h>

int main()
{
    int sum, i;

    sum = 0;
    i = 0;

    for(;;){
        i++;
        sum += i;
        if(sum > 2000) break;
    }

    printf("Το άθροισμα γίνεται > 2000 για i: %d", i);
    return 0;
}
```

Άσκηση 6

Να γραφεί ένας αλγόριθμος που θα δέχεται επαναληπτικά αριθμούς από το πληκτρολόγιο μέχρις να εισαχθούν 100 αριθμοί ή το άθροισμά τους ξεπεράσει ένα γνωστό και δεδομένο όριο Μ.
Ο αλγόριθμος θα εμφανίζει στο τέλος το πλήθος των αριθμών που έχουν εισαχθεί καθώς και το άθροισμά τους.

Άσκηση 6 - Λύση


```c
#include <stdio.h>
int main() {
    int count;
    float x, m, sum;
    sum = 0;
    count = 0;
    printf("Δώσε τον αριθμό Μ:");
    scanf("%f", &m);
    do{
        printf("Δώσε έναν αριθμό:");
        scanf("%f", &x);
        sum += x;
        count++;
    }
    while(sum <= m && count < 100);
    printf("Πλήθος αριθμών που δόθηκαν: %d\n", count);
    printf("Άθροισμα: %f", sum);
    return 0;
}
```

Μηδενικά μπροστά από αριθμούς

Θέλω ο κωδικός του εργαζομένου να είναι τριψήφιος ακέραιος.
Στην περίπτωση όμως που δώσω τον κωδικό: **001**
η εντολή:

```
printf("Ο κωδικός του εργαζομένου είναι: %d", code);
```

Θα τυπώσει:

```
Ο κωδικός του εργαζομένου είναι: 1
```

Αν θέλω να εκτυπωθούν και τα μηδενικά μπροστά από το 1, θα πρέπει να χρησιμοποιήσω τον προσδιοριστή: **%03d**

```
printf("Ο κωδικός του εργαζομένου είναι: %03d", code);
```

Πόσα ψηφία έχει ένας ακέραιος;

Θέλω να υπολογίσω το πλήθος των ψηφίων ενός ακεραίου αριθμού **n**

```c
#include <stdio.h>

int main(){
    int n = 20901;
    int count = 0;

    while(n>0){
        count++;
        n = n/10;
    }
    printf("Ο αριθμός %d, έχει %d ψηφία", n, count);
    return 0;
}
```

Επειδή το **n** είναι δηλωμένος ως ακέραιος (int), η πράξη:

```c
n = n/10;
```

υπολογίζει μόνο το **πηλίκο** της διαίρεσης (ακέραιο μέρος του **n)**

Άσκηση 7

Δεδομένα που αφορούν Ν εργαζόμενους (Ν = γνωστό) εισάγονται με την εξής σειρά :

- κωδικός (τριψήφιος ακέραιος),
- τμήμα (1 ή 2)
- Μισθός

Να γραφεί αλγόριθμος που θα βρίσκει και θα εμφανίζει :

- το μεγαλύτερο μισθό σε κάθε τμήμα και τον κωδικό εργαζομένου που λαμβάνει τον μεγαλύτερο μισθό (αν υπάρχουν περισσότεροι από ένας εργαζόμενοι θα εμφανίζεται ο πρώτος)
- το μέσο μισθό κάθε τμήματος

Άσκηση 7 - Λύση (1/3)


```c
#include <stdio.h>

int main()
{
    int code, tmima, n, count1, count2, max_code1, max_code2;
    float misthos, max1, max2, sum1, sum2;

    printf("Δώσε τον αριθμό εργαζομένων: ");
    scanf("%d", &n);

    max1 =0; max2 = 0;
    count1 = 0; count2 = 0;
    sum1 = 0; sum2 = 0;

    // ... συνεχίζεται ...
```

Άσκηση 7 - Λύση (2/3)


```
// ...

for(int i = 0; i < n; i++){
        printf("Δώσε κωδικό / τμήμα / μισθό: ");
        scanf("%d %d %f", &code, &tmima, &misthos);

        if (tmima == 1){
            sum1 += misthos;
            count1++;
            if(misthos > max1){
               max1 = misthos;
               max_code1 = code;
            }
        }
```

```
        else if(tmima == 2){
            sum2 += misthos;
            count2++;
            if(misthos > max2){
                max2 = misthos;
                max_code2 = code;
            }
        }
    }

    // ... συνεχίζεται ...
```

Άσκηση 7 - Λύση (3/3)


```
// ...

if (count1 > 0){
        printf("Μεγαλύτερος μισθός και κωδικός εργαζομένου στο τμήμα 1: %.2f, %03d \n", max1, max_code1);
        printf("Μέσος μισθός στο τμήμα 1: %.2f\n", sum1/count1);
    }
    else printf("Δεν βρέθηκαν εργαζόμενοι στο τμήμα 1\n");


    if (count2 > 0){
        printf("Μεγαλύτερος μισθός και κωδικός εργαζομένου στο τμήμα 2: %.2f, %03d \n", max2, max_code2);
        printf("Μέσος μισθός στο τμήμα 2: %.2f\n", sum2/count2);
    }
    else printf("Δεν βρέθηκαν εργαζόμενοι στο τμήμα 2\n");

    return 0;
}
```

Ποσοστιαία Μεταβολή

1. Προσδιορίστε την αρχική και την τελική τιμή.
2. Αφαιρέστε την τελική τιμή από την αρχική τιμή.
3. Διαιρέστε αυτόν τον αριθμό με την αρχική τιμή.
4. Πολλαπλασιάστε με το 100 για να βρείτε την ποσοστιαία μεταβολή

Ποσοστό μεταβολής = [(αρχική τιμή – τελική τιμή) / αρχική τιμή] × 100]

Άσκηση 8

Δεδομένα εισόδου είναι ζεύγη πραγματικών θετικών αριθμών που αντιστοιχούν στο επιθυμητό και το πραγματικό μήκος μεταλλικών ράβδων που παράγει μια μονάδα παραγωγής.
Η μέγιστη αποδεκτή απόκλιση μεταξύ επιθυμητού και πραγματικού μήκους είναι 5%.
Η εισαγωγή δεδομένων θα σταματά όταν το ποσοστό των ζευγών με απόκλιση μεγαλύτερη από τη μέγιστη αποδεκτή απόκλιση ξεπεράσει το 20% του συνόλου όλων των ζευγών που έχουν εισαχθεί μέχρι εκείνη τη στιγμή.
Ο αλγόριθμος θα εμφανίζει στο τέλος το πλήθος των ζευγών που έχουν εισαχθεί.

Άσκηση 8 - Λύση


```c
#include <stdio.h>
int main()
{
    float epithimito, pragmatiko, metaboli, apodekti_apoklisi;
    int total_count, count;
    total_count = 0; count = 0;
    do{
        printf("Δώσε το επιθυμητό και το πραγματικό μέγεθος της ράβδου: ");
        scanf("%f %f", &epithimito, &pragmatiko);

        metaboli = (fabs(epithimito - pragmatiko)/epithimito) * 100;
        total_count++;
        if(metaboli > 5)
           count++;
        apodekti_apoklisi = (float)count / (float)total_count * 100;
        printf("metaboli: %.2f, apoklisi: %.2f\n", metaboli, apodekti_apoklisi);
    }
    while(apodekti_apoklisi <= 20);
    printf("Πλήθος ζευγών: %d", total_count);
    return 0;
}
```

Άσκηση 9

Ο σημερινός αριθμός αυτοκινήτων που κυκλοφορούν σε μια πόλη είναι Α.
Αν ο αριθμός αυτός αυξάνεται με ετήσιο ρυθμό c%, να γραφεί αλγόριθμος που να υπολογίζει σε πόσα έτη ο αριθμός των αυτοκινήτων θα ξεπεράσει μια δεδομένη γνωστή τιμή Β
(να υποθέσετε ότι θα ισχύει Β > Α).
Ο αλγόριθμος θα εμφανίζει στο τέλος τον αριθμό των ετών καθώς και τον τελικό αριθμό των αυτοκινήτων.

Άσκηση 9 - Λύση


```c
#include <stdio.h>

int main()
{
    int a, b, years;
    float c;

    printf("Δώσε τα A, B, c: ");
    scanf("%d %d %f", &a, &b, &c);

    years = 0;
    while(b > a){
        a = a + a * c/100;
        years++;
        printf("%d\n", a);
    }
    printf("Έτη: %d, Αυτοκίνητα: %d", years, a);
    return 0;
}
```

Άσκηση 10

Να γραφεί αλγόριθμος που θα υπολογίζει και θα εμφανίζει τον πρώτο ακέραιο και θετικό αριθμό το τετράγωνο του οποίου διαφέρει από το τετράγωνο του επομένου του τουλάχιστον κατά 50.

Άσκηση 10 - Λύση


```c
#include <stdio.h>

int main()
{
    int x;

    x = 1;

    while((x+1)*(x+1) - x*x < 50){
        x++;
    }
    printf("Ο ζητούμενος ακέραιος είναι ο: %d", x);
    return 0;
}
```

Έλεγχος ισότητας πραγματικών αριθμών

Τι θα εκτυπώσει το παρακάτω πρόγραμμα;

```c
#include<stdio.h>
int main()
{
    float x = 0.1;
    if (x == 0.1)
        printf("Το x είναι 0.1");
    else
        printf("Το x ΔΕΝ είναι 0.1");
    return 0;
}
```

Έλεγχος ισότητας πραγματικών αριθμών

Ας ελέγξουμε πόσο bit καταλαμβάνουν στην μνήμη η μεταβλητή **x** και ο αριθμός **0.1**:

```c
#include<stdio.h>
int main()
{
   float x = 0.1;
   printf("%d %d %d", sizeof(x), sizeof(0.1), sizeof(0.1f));
   return 0;
}
```

x: **4 bit**

0.1: **8 bit**

0.1f: **4 bit**

Το πρόβλημα είναι ότι ο αριθμός **0.1** αποθηκεύεται ως double και όχι ως float

Για να γίνει ο αριθμός **0.1 float**, θα πρέπει να γραφτεί ως: **0.1f**

Άσκηση 11

Να γραφεί αλγόριθμος που θα εισάγει ένα άγνωστο πλήθος μετρήσεων (οι μετρήσεις αντιστοιχούν σε πραγματικούς αριθμούς), θα υπολογίζει και θα εκτυπώνει :

- Τη μέγιστη τιμή και τη θέση της στο πλήθος
- Την ελάχιστη τιμή και τη θέση της στο πλήθος
- Τη μέση τιμή

Ο τελευταίος αριθμός θα είναι ο αριθμός –999.9 και δεν αποτελεί μέτρηση (ο τελευταίος αριθμός καθορίζει και το τέλος εισαγωγής των δεδομένων).

Άσκηση 11 - Λύση


```c
#include <stdio.h>

int main() {
    float x, max_x, min_x, sum;
    int i, max_i, min_i;

    printf("Δώσε το x:");
    scanf("%f", &x);
    i = 1;
    sum = 0;

    max_x = x; min_x = x;
    max_i = 1; min_i= 1;
    while(x != -999.9f){
        if(x > max_x){
            max_x = x; max_i = i;
        }
        if(x < min_x){
            min_x = x; min_i = i;
        }
        sum += x;
        printf("Δώσε το x:");
        scanf("%f", &x);
        i++;
    }
    printf("Μέγιστο %.2f στη θέση %d\n", max_x, max_i);
    printf("Ελάχιστο %.2f στη θέση %d\n", min_x, min_i);
    printf("Μέση τιμή: %.2f ", sum/i);
    return 0;
}
```

Άσκηση 12

Ένας θετικός ακέραιος λέγεται ατελής (deficient), τέλειος (perfect) ή πλούσιος (abundant) αν το άθροισμα των διαιρετών του εκτός από τον ίδιο τον αριθμό είναι αντίστοιχα μικρότερο, ίσο ή μεγαλύτερο από τον αριθμό αυτό.
π.χ. το 6 είναι τέλειος αριθμός διότι 6=1+2+3 και οι αριθμοί 1,2,3 είναι οι διαιρέτες του.
Να γραφεί αλγόριθμος για την εμφάνιση της αντίστοιχης λέξης για μια περιοχή 100 συνεχόμενων ακεραίων αριθμών (π.χ. 200 έως 300, 490 έως 590, 8120 έως 8130).
Τα όρια της περιοχής θα εισάγονται από το πληκτρολόγιο.

Άσκηση 12 - Λύση


```c
#include <stdio.h>
int main() {
      int first, last, sum;

      printf("Δώσε τον πρώτο και τον τελευταίο ακέραιο: ");
      scanf("%d %d", &first, &last);

      for(int i = first; i <= last; i++){
         sum = 0;
         for(int j = 1; j < i; j++){
             if(i % j == 0)
                sum += j;
         }
         if (sum == i)  printf("Ο αριθμός %d είναι τέλειος \n", i, sum);
         else if (sum < i)  printf("Ο αριθμός %d είναι ατελής \n", i, sum);
         else  printf("Ο αριθμός %d είναι πλούσιος \n", i, sum);
     }
      return 0;
}
```

Άσκηση 13

Για την εύρεση μιας ρίζας ενός πολυωνύμου χρησιμοποιείται ο παρακάτω αλγόριθμος (μέθοδος Newton):

1. εισάγονται ως δεδομένα η τάξη του πολυωνύμου και οι συντελεστές κάθε όρου
2. εισάγονται ως δεδομένα δύο τυχαίες τιμές (με ονόματα posX, negX) εντός των οποίων ενδέχεται να υπάρχει μια ρίζα x
3. ο αλγόριθμος ελέγχει αν υπάρχει ρίζα μεταξύ αυτών των δύο τιμών (οι τιμές του πολυωνύμου για κάθε μια από τις δύο αυτές τιμές posX, negX πρέπει να έχουν αντίθετο πρόσημο). Αν αυτό δεν ισχύει ο αλγόριθμος θα σταματά και θα εμφανίζει το κατάλληλο μήνυμα.
4. Ο αλγόριθμος θα χρησιμοποιεί ως προσεγγιστική τιμή της ρίζας x τη μέση τιμή των δύο αυτών τιμών posX, negX επαναληπτικά.
5. Αν για τη νέα τιμή x η τιμή του πολυωνύμου είναι θετική θα αντικαθιστά το posX με το x, ενώ αν είναι αρνητική θα αντικαθιστά το negX με το x.
6. Ο αλγόριθμος θα επαναλαμβάνει τα βήματα 4 και 5 μέχρις ότου δύο διαδοχικές τιμές του x θα διαφέρουν τουλάχιστον κατά μια σταθερά $\varepsilon=10^{-4}$. Ο αλγόριθμος σε κάθε βήμα πρέπει να εμφανίζει όλες τις υπολογιζόμενες τιμές.

Άσκηση 13 - Λύση (1/3)


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

// Υπολογισμός της τιμής του πολυωνύμου στο σημείο x

double polynomial(double coefficients[], int n, double x) {
    double result = 0;
    for (int i = 0; i <= n; i++) {
        result += coefficients[i] * pow(x, n - i);
    }
    return result;
}
```

Λόγω χρήσης της βιβλιοθήκης math.h, το προγραμμα στο τέλος θα πρέπει να γίνει compile με την παράμετρο **–lm**

```
gcc -o f1_13 f1_13.c -lm
./f1_13
```

Άσκηση 13 - Λύση (2/3)


```c
int main() {
    int n; // Τάξη του πολυωνύμου
    printf("Δώσε την τάξη του πολυωνύμου: ");
    scanf("%d", &n);

    double coefficients[n + 1]; // Συντελεστές του πολυωνύμου
    printf("Δώσε τους συντελεστές του πολυωνύμου:\n");
    for (int i = 0; i <= n; i++) {
        printf("Συντελεστής για x^%d: ", n - i);
        scanf("%lf", &coefficients[i]);
    }

    double posX, negX, x, prevX, epsilon = 1e-4;
    int iterations = 0;

    printf("Δώσε την αρχική τιμή posX: ");
    scanf("%lf", &posX);
    printf("Δώσε την αρχική τιμή negX: ");
    scanf("%lf", &negX);
```

Άσκηση 13 - Λύση (3/3)


```
while (1) {
        x = (posX + negX) / 2;
        double polyValue = polynomial(coefficients, n, x);
        printf("Βήμα %d: x = %.6lf, f(x) = %.6lf\n", iterations, x, polyValue);

        if (fabs(x - prevX) < epsilon)
            break;
        if (polyValue > 0)
            posX = x;
        else if (polyValue < 0)
            negX = x;
        else
            break; // Η ρίζα είναι ακριβώς 0.
        prevX = x;
        iterations++;
    }
    printf("Η ρίζα του πολυωνύμου είναι: %.6lf\n", x);
    return 0;
}
```

Άσκηση 14

Για ορισμένους θετικούς ακέραιους τριψήφιους αριθμούς ισχύει η εξής ιδιότητα:

1. υπολογίζουμε το άθροισμα των κύβων των ψηφίων του αριθμού
2. υπολογίζουμε το άθροισμα των ψηφίων του αριθμού
3. υπολογίζουμε τη διαφορά των τιμών που προέκυψαν από τα βήματα (1) και (2) και στη συνέχεια την τετραγωνική ρίζα του αριθμού που προκύπτει από την αφαίρεση.
4. υπολογίζουμε τον διψήφιο αριθμό που προκύπτει από τα 2 πρώτα ψηφία του αρχικού αριθμού και από αυτόν αφαιρούμε το τελευταίο ψηφίο του αρχικού αριθμού

Οι αριθμοί από τα βήματα (3) και (4) είναι ίσοι!

Π.χ. ο αριθμός 153 : $\sqrt{\left(1^3+5^3+3^3\right)-\left(1+5+3\right)}=15-3$

Να γραφεί πρόγραμμα σε γλώσσα C που θα βρίσκει και θα εμφανίζει, έναν σε κάθε γραμμή, όλους τους θετικούς τριψήφιους ακέραιους αριθμούς που πληρούν την παραπάνω ιδιότητα.

Άσκηση 14 - Λύση (1/2)


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

int main() {
    for (int num = 100; num <= 999; num++) {
        int sum_of_cubes = 0;
        int sum_of_digits = 0;

        // Υπολογισμός αθροίσματος κύβων των ψηφίων
        int temp = num;
        while (temp > 0) {
            int digit = temp % 10;
            sum_of_cubes += digit * digit * digit;
            temp /= 10;
        }
```

```c
        // Υπολογισμός αθροίσματος ψηφίων
        temp = num;
        while (temp > 0) {
            int digit = temp % 10;
            sum_of_digits += digit;
            temp /= 10;
        }
```

Άσκηση 14 - Λύση (2/2)


```
        // Υπολογισμός της διαφοράς και της τετραγωνικής ρίζας
        int diff = sum_of_cubes - sum_of_digits;
        int sqrt_diff = sqrt(abs(diff));

        // Υπολογισμός των δύο πρώτων ψηφίων και του τελευταίου ψηφίου
        int first_two_digits = num / 10;
        int last_digit = num % 10;

        // Υπολογισμός του διψήφιου αριθμού και έλεγχος ισότητας
        int two_digit_number = first_two_digits - last_digit;
        if (two_digit_number == sqrt_diff) {
            printf("%d\n", num);
        }
    }

    return 0;
}
```

Άσκηση 15

Ο αριθμός 24 έχει την παρακάτω ιδιότητα. Εάν επιλέξουμε έναν οποιονδήποτε πρώτο (prime) αριθμό, μεγαλύτερο του 3, τον υψώσουμε στο τετράγωνο, και από το αποτέλεσμα που θα προκύψει αφαιρέσουμε την μονάδα τότε το 24 είναι πάντοτε ένας διαιρέτης αυτού του αποτελέσματος.
Π.χ.
Επιλέγω τον 17 (είναι πρώτος αριθμός).
Το τετράγωνό του είναι 289.
Αφαιρώντας 1 προκύπτει 288.
Η διαίρεση 288/12 δίνει αποτέλεσμα 24.

Να γραφεί αλγόριθμος ή πρόγραμμα σε γλώσσα C που θα επιβεβαιώνει την παραπάνω ιδιότητα για όλους τους πρώτους αριθμούς που είναι μεγαλύτεροι του 3 και μικρότεροι του 100.

Άσκηση 15

ΥΠΟΔΕΙΞΗ :

Το παρακάτω τμήμα κώδικα σε γλώσσα C βρίσκει αν ένας ακέραιος και θετικός αριθμός k ≥ 2 είναι πρώτος (prime) αριθμός .

```
i=2;
flag=0;
while ((i<=k/2) && (flag==0)){
   if (k%i==0) flag=1;
   i++;
}
if (flag==0) printf("number %4d is prime \n ",k);
```

Άσκηση 15 - Λύση


```c
#include <stdio.h>
int main(){
    int i,k,flag,square_k;
     for (k=4;k<100;k++){
          i=2; flag=0;
          while ((i<=k/2) && (flag==0)){
            if (k%i==0) flag=1;
            i++;
          }
          if (flag==0){
               printf("Ο αριθμός %4d είναι πρώτος ",k);
               square_k=k*k;
               square_k=square_k-1;
               if (square_k % 24 == 0)
                   printf("Ο αριθμός %4d διαιρείται με το 24\n",square_k);
          }
     }
     return 0;
}
```

Άσκηση 16 i.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$\frac{1}{3}-\frac{2}{5}+\frac{3}{7}-\frac{4}{9}+\dots$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 i. - Λύση (1/2)


```c
#include <stdio.h>

int main()
{
    float sum, term, accuracy, old_term;
    int sign, nom, denom;

    sign = 1;
    accuracy = 1e-4;
    nom = 1;
    denom = 3;
    term = (float)nom/(float)denom;
    sum = term;
    old_term = 0;

    // ... συνεχίζεται ...
```

Άσκηση 16 i. - Λύση (2/2)


```
    // ...

    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.2f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        nom++;
        denom +=2;
        sign = -sign;
        term = sign * (float)nom/(float)denom;
        sum += term;
    }
    printf("Όροι ακολουθίας: %d", nom);
    return 0;
}
```

Άσκηση 16 ii.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$\frac{1}{1+\sqrt{2}} - \frac{1}{\sqrt{2}+\sqrt{3}} + \frac{1}{\sqrt{3}+\sqrt{4}} - \frac{1}{\sqrt{4}+\sqrt{5}} + \dots$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 ii. - Λύση (1/2)


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

int main()
{
    float sum, term, accuracy, old_term;
    int sign, denom1, denom2;

    sign = 1;
    accuracy = 1e-4;
    denom1 = 1;
    denom2= 2;
    term = 1.0/((float)denom1 + sqrt((float)denom2));
    sum = term;
    old_term = 0;

    // ... συνεχίζεται ...
```

Άσκηση 16 ii. - Λύση (2/2)


```
    // ...

    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.2f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        denom1++;
        denom2++;
        sign = -sign;
        term = sign * 1.0/((float)denom1 + sqrt((float)denom2));
        sum += term;
    }
    printf("Όροι ακολουθίας: %d", denom1);
    return 0;
}
```

Άσκηση 16 iii.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$ln\,x = (x-1) - \frac{1}{2}(x-1)^2 + \frac{1}{3}(x-1)^3 - \frac{1}{4}(x-1)^4 + \ldots$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 iii. - Λύση (1/2)


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

int main(){
    float x, sum, term, accuracy, old_term;
    int sign, denom;

    sign = 1;
    accuracy = 1e-4;
    denom = 1;
    printf("Δώσε τον αριθμό x:");
    scanf("%f", &x);
    term = pow((x - 1), denom) / (float)denom;
    printf("%f", term);
    sum = term;
    old_term = 0;

    // ... συνεχίζεται ...
```

Άσκηση 16 iii. - Λύση (2/2)


```
    // ...

    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.2f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        denom++;
        sign = -sign;
        term = sign * pow((x - 1), denom) / (float)denom;
        sum += term;
    }
    printf("Όροι ακολουθίας: %d", denom);
    return 0;
}
```

Άσκηση 16 iv.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$1+\frac{x^2}{1\cdot 3}+\frac{x^4}{3\cdot 5}+\frac{x^6}{5\cdot 7}+\ldots+\frac{x^k}{m\cdot n}$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 iv. - Λύση (1/2)


```c
#include <stdio.h>
#include <math.h>
int main(){
    float x, sum, term, accuracy, old_term;
    int denom1, denom2, count;

    accuracy = 1e-4;
    denom1 = 1;
    denom2 = 3;

    printf("Δώσε τον αριθμό x:");
    scanf("%f", &x);

    term = pow(x, denom1+1)/(float)(denom1*denom2);
    sum = 1 + term;
    old_term = 1;
    count = 2;
    // ... συνεχίζεται ...
```

Άσκηση 16 iv. - Λύση (2/2)


```
// ...

    while(fabs(term - old_term) > accuracy){
        printf("Τρέχον άθροισμα: %.2f, Διαφορά: %.6f \n", sum, fabs(term - old_term));
        old_term = term;
        denom1 += 2;
        denom2 += 2;
        term = pow(x, denom1+1)/(float)(denom1*denom2);
        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d", count);
    return 0;
}
```

Άσκηση 16 ν.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 ν. - Λύση (1/2)


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

// Συνάρτηση που υπολογίζει το παραγοντικό ενός ακεραίου
unsigned int factorial(unsigned int n)
{
    int result, i;

    result = 1;

    for (i = 2; i <= n; i++) {
        result *= i;
    }

    return result;
}
```

```c
int main(){

    float x, sum, term, accuracy, old_term;
    int denom, count;

    accuracy = 1e-4;
    denom = 1;

    printf("Δώσε τον αριθμό x:");
    scanf("%f", &x);

    term = pow(x, denom)/(float)factorial(denom);
    sum = 1 + term;
    old_term = 1;
    count = 2;

    // ... συνεχίζεται ...
```

Άσκηση 16 ν. - Λύση (2/2)


```
// ...

    while(fabs(term - old_term) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(term - old_term));
        old_term = term;
        denom++;
        term = pow(x, denom)/(float)factorial(denom);
        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d", count);

    return 0;
}
```

Άσκηση 16 vi.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 vi. - Λύση (1/2)


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

// Συνάρτηση που υπολογίζει το παραγοντικό ενός ακεραίου
unsigned int factorial(unsigned int n)
{
    int result, i;

    result = 1;

    for (i = 2; i <= n; i++) {
        result *= i;
    }

    return result;
}
```

```c
int main(){

    float x, sum, term, accuracy, old_term;
    int sign, denom, count;

    accuracy = 1e-4;
    denom = 1;
    sign = -1;

    printf("Δώσε τον αριθμό x:");
    scanf("%f", &x);

    term = pow(x, denom)/(float)factorial(denom);
    sum = 1 - term;
    old_term = 1;
    count = 2;

    // ... συνεχίζεται ...
```

Άσκηση 16 vi. - Λύση (2/2)


```
// ...

    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        denom++;
        sign = -sign;
        term = sign * pow(x, denom)/(float)factorial(denom);
        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", sum);

    return 0;
}
```

Άσκηση 16 vii.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$e^{-x^2} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - \dots$$

Η τιμή του x θα δίνεται από το πληκτρολόγιο. Ο αλγόριθμος, σε κάθε βήμα του, πρέπει να εμφανίζει όλες τις διαδοχικές τιμές του αθροίσματος καθώς και τη διαφορά τους μέχρι να επιτευχθεί η επιθυμητή ακρίβεια. Μετά τον υπολογισμό και την εμφάνιση του αποτελέσματος να εμφανίσετε το πλήθος των όρων που χρησιμοποιήθηκαν στον υπολογισμό.

Άσκηση 16 vii. - Λύση (1/2)


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

// Συνάρτηση που υπολογίζει το παραγοντικό ενός ακεραίου
unsigned int factorial(unsigned int n)
{
    int result, i;

    result = 1;

    for (i = 2; i <= n; i++) {
        result *= i;
    }

    return result;
}
```

```c
int main(){

    float x, sum, term, accuracy, old_term;
    int sign, denom, count, power;

    accuracy = 1e-4;
    denom = 1;
    power = 2;
    sign = -1;

    printf("Δώσε τον αριθμό x:");
    scanf("%f", &x);

    term = pow(x, power)/(float)factorial(denom);
    sum = 1 - term;
    old_term = 1;
    count = 2;
    // ... συνεχίζεται ...
```

Άσκηση 16 vii. - Λύση (2/2)


```
// ...

    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        denom++;
        power += 2;
        sign = -sign;
        term = sign * pow(x, power)/(float)factorial(denom);
        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", sum);

    return 0;
}
```

Άσκηση 16 viii.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$1+\frac{1}{4}+\frac{1\times3}{4\times8}+\frac{1\times3\times5}{4\times8\times12}+\ldots$$

Άσκηση 16 viii. - Λύση (1/2)


```c
#include <stdio.h>

int main(){
    float sum_numerator, sum_denom, total_sum, term, accuracy, old_term;
    int denom, numerator, count;

    accuracy = 1e-4;
    numerator = 1;
    denom = 4;
    sum_numerator = numerator;
    sum_denom = denom;

    term = sum_numerator / sum_denom;
    total_sum = 1 + term;
    old_term = 1;
    count = 2;
    // ... συνεχίζεται ...
```

Άσκηση 16 viii. - Λύση (2/2)


```c
// ...

    while(fabs(term - old_term) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", total_sum, fabs(term - old_term));
        old_term = term;
        numerator += 2;
        denom += 4;
        sum_numerator *= numerator;
        sum_denom *= denom;
        term = sum_numerator / sum_denom;
        total_sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", total_sum);

    return 0;
}
```

Άσκηση 16 ix.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$1 + \frac{1}{3} + \frac{1 \cdot 3}{3 \cdot 6} + \frac{1 \cdot 3 \cdot 5}{3 \cdot 6 \cdot 9} + \ldots$$

Άσκηση 16 ix. - Λύση (1/2)


```c
#include <stdio.h>

int main(){
    float sum_numerator, sum_denom, total_sum, term, accuracy, old_term;
    int denom, numerator, count;

    accuracy = 1e-4;
    numerator = 1;
    denom = 3;
    sum_numerator = numerator;
    sum_denom = denom;

    term = sum_numerator / sum_denom;
    total_sum = 1 + term;
    old_term = 1;
    count = 2;
    // ... συνεχίζεται ...
```

Άσκηση 16 ix. - Λύση (2/2)


```
// ...

    while(fabs(term - old_term) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", total_sum, fabs(term - old_term));
        old_term = term;
        numerator += 2;
        denom += 3;
        sum_numerator *= numerator;
        sum_denom *= denom;
        term = sum_numerator / sum_denom;
        total_sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", total_sum);

    return 0;
}
```

Άσκηση 16 χ.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$\frac{1}{1\cdot 2}\cdot\frac{1}{2}+\frac{1}{2\cdot 3}\cdot\frac{1}{2^2}+\frac{1}{3\cdot 4}\cdot\frac{1}{2^3}+\dots$$

Άσκηση 16 x. - Λύση (1/2)


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

int main(){

    float sum, term, accuracy, old_term;
    int denom, count;

    accuracy = 1e-4;
    denom = 1;

    term = 1.0/ (denom * (denom + 1) * 2);
    sum = term;

    old_term = 0;
    count = 1;
    // ... συνεχίζεται ...
```

Άσκηση 16 x. - Λύση (2/2)


```
// ...

    while(fabs(term - old_term) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(term - old_term));
        old_term = term;
        denom++;

        term = 1 / (denom * (denom + 1) * pow(2, denom));
        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", total_sum);

    return 0;
}
```

Άσκηση 16 xi.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$\frac{1}{2\cdot 3\cdot 4}+\frac{2}{3\cdot 4\cdot 5}+\frac{3}{4\cdot 5\cdot 6}+\dots$$

Άσκηση 16 xi. - Λύση (1/2)


```c
#include <stdio.h>

int main(){

    float sum, term, accuracy, old_term, numerator, denom;
    int count;

    accuracy = 1e-4;
    numerator = 1;
    denom = 2;

    term = numerator / (denom * (denom + 1) * (denom + 2));
    sum = term;

    old_term = 0;
    count = 1;

// ... συνεχίζεται ...
```

Άσκηση 16 xi. - Λύση (2/2)


```
// ...

    while(fabs(term - old_term) > accuracy){
       printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(term - old_term));
       old_term = term;
       numerator++;
       denom++;

       term = numerator / (denom * (denom + 1) * (denom + 2));
       sum += term;
       count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", total_sum);

 return 0;
}
```

Άσκηση 16 xii.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$1 - \frac{1}{3 \cdot 3} + \frac{3}{3^2 \cdot 5} - \frac{5}{3^3 \cdot 7} + \ldots$$

Άσκηση 16 xii. - Λύση (1/2)


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

int main(){
    float sum, term, accuracy, old_term, numerator, denom;
    int power, count, sign;

    accuracy = 1e-4;
    numerator = 1;
    denom = 3;
    power = 1;
    sign = -1;

    term = numerator / (3 * denom );
    sum = 1 - term;
    old_term = 0;
    count = 2;
// ... συνεχίζεται ...
```

Άσκηση 16 xii. - Λύση (2/2)


```
// ...
    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        numerator += 2;
        denom += 2;
        power++;
        sign = -sign;

        term = sign * numerator / (pow(3, power) * denom);
        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", sum);

    return 0;
}
```

Άσκηση 16 xiii.

Να γραφεί αλγόριθμος ή/και πρόγραμμα σε γλώσσα C που θα υπολογίζει τα παρακάτω αθροίσματα με ακρίβεια 10^{-4}.
Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1}$$

Άσκηση 16 xiii. - Λύση (1/2)


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

int main(){

    float sum, term, accuracy, old_term;
    int count, k;

    accuracy = 1e-4;
    k = 0;

    term = pow(-1, k) / (2 * k +1);
    sum = term;
    old_term = 0;
    count = 1;

// ... συνεχίζεται ...
```

Άσκηση 16 xiii. - Λύση (2/2)


```
// ...

    while(fabs(fabs(term) - fabs(old_term)) > accuracy){
        printf("Τρέχον άθροισμα: %.6f, Διαφορά: %.6f \n", sum, fabs(fabs(term) - fabs(old_term)));
        old_term = term;
        k++;

        term = pow(-1, k) / (2 * k +1);

        sum += term;
        count++;
    }
    printf("Όροι ακολουθίας: %d \n", count);
    printf("Αποτέλεσμα: %.6f \n", sum);

    return 0;
}
```

Άσκηση 17

Δίνονται Ν σειρές ακέραιων και θετικών αριθμών (Ν= γνωστό).
Σε κάθε σειρά ο πρώτος αριθμός δείχνει το πλήθος αυτών που υπάρχουν στη συνέχεια.
Να γραφεί αλγόριθμος που θα βρίσκει και θα εμφανίζει το πλήθος των άρτιων και των περιττών αριθμών που υπάρχουν σε κάθε σειρά. Κατά την εισαγωγή θα ελέγχεται ότι όλοι οι αριθμοί είναι > 0.

Παράδειγμα για Ν=3 :

5, 2,45,77,4,33	άρτιοι = 2	περιττοί=3
4,17,27,44,55	άρτιοι = 1	περιττοί=3
2, 3,9	άρτιοι = 0	περιττοί=2

Άσκηση 17 - Λύση (1/2)


```c
#include <stdio.h>

int main() {

    int n, num;
    printf("Εισάγετε τον αριθμό των σειρών: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        int length;
        int evenCount = 0;
        int oddCount = 0;

        printf("Εισάγετε το πλήθος αριθμών στη σειρά: ");
        scanf("%d", &length);

// ... συνεχίζεται ...
```

Άσκηση 17 - Λύση (2/2)


```
// ...

    for (int j = 0; j < length; j++) {
            do{
                printf("Εισάγετε έναν αριθμό: ");
                scanf("%d", &num);
            }
            while(num <= 0);

            if (num % 2 == 0) evenCount++;
            else  oddCount++;
        }
        printf("Σειρά %d: Άρτιοι αριθμοί: %d, Περιττοί αριθμοί: %d\n", i + 1, evenCount, oddCount);
    }

    return 0;
}
```

Άσκηση 18

Να γραφεί αλγόριθμος για το εξής πρόβλημα:
Δεδομένα είναι μια λίστα 5 ακεραίων αριθμών και μια τιμή, έστω sum.
Ο αλγόριθμος θα βρίσκει και θα εμφανίζει ένα υποσύνολο των αριθμών της λίστας το άθροισμα των οποίων είναι ίσο με την τιμή sum, αν υπάρχει ένα τέτοιο υποσύνολο ή ένα κατάλληλο μήνυμα, αν δεν υπάρχει τέτοιο υποσύνολο.

Παράδειγμα : αν η λίστα των αριθμών είναι 5, 13, 23, 9, 3 και sum=27 τότε ο αλγόριθμος θα βρίσκει το υποσύνολο 5, 13, 9.

Θα υλοποιήσουμε την απλή περίπτωση του προβλήματος, όπου οι αριθμοί εξετάζονται με τη σειρά.

(Η γενική λύση απαιτή τη χρήση αναδρομικών συναρτήσεων που θα μελετηθούν αργότερα.)

Άσκηση 18 - Λύση (1/2)


```c
#include <stdio.h>

int main()
{
    int count, sum, current_sum, arr[5], results[5];
    for(int i = 0; i < 5; i++){
        printf("Δώσε το %do στοιχείο του πίνακα: ", i+1);
        scanf("%d", &arr[i]);
    }
    printf("Δώσε το επιθυμητό άθροισμα:");
    scanf("%d", &sum);


// ... συνεχίζεται ...
```

Άσκηση 18 - Λύση (2/2)


```
// ...
    current_sum = 0;
    count = 0;
    for(int i = 0; i < 5; i++){
        if(current_sum + arr[i] <= sum){
            current_sum += arr[i];
            results[count] = arr[i];
            count++;
        }
    }
    if (current_sum == sum){
        for (int i = 0; i < count; i++)
           printf("%d ", results[i]);
    }
    else
       printf("Δεν βρέθηκε το ζητούμενο άθροισμα");
    return 0;
}
```

Άσκηση 19

Η τετραγωνική ρίζα ενός αριθμού N μπορεί να υπολογιστεί κατά προσέγγιση με επαναληπτική εφαρμογή του τύπου:

$$NG = 0.5 (LG + N / LG)$$

όπου:
NG (Next Guess) είναι η επόμενη εκτίμηση για την τιμή της τετραγωνικής ρίζας και
LG (Last Guess) είναι η τελευταία υπολογισμένη εκτίμηση για την τιμή της τετραγωνικής ρίζας.

Να γραφεί ένας αλγόριθμος για την υλοποίηση του παραπάνω υπολογισμού.
Δεδομένα εισόδου είναι ο αριθμός N και μια αρχική εκτίμηση για την τιμή της τετραγωνικής ρίζας.

Η επαναληπτική διαδικασία να τερματίζει όταν **LG - NG < 10^{-6}.**

Άσκηση 19 - Λύση


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

int main(){
    double N, NG, LG;
    int accuracy = 1e-6;

    printf("Δώσε τον αριθμό και την αρχική εκτίμηση της τετραγωνικής του ρίζας:");
    scanf("%lf %lf", &N, &LG);
    NG = 0.5 * (LG + N / LG);

    while(fabs(LG - NG ) > accuracy){
        LG = NG;
        NG = 0.5 * (LG + N / LG);
    }
    printf("%lf", NG);
    return 0;
}
```

Άσκηση 20

Δύο θετικοί ακέραιοι αριθμοί είναι «φίλιοι» αν ο καθένας ισούται με το άθροισμα όσων διαιρούν τον άλλον (λαμβάνονται υπόψη μόνον οι γνήσιοι διαιρέτες).
Οι πιο διάσημοι «φίλιοι» αριθμοί είναι οι αριθμοί 220 και 284 (αποδίδονται στον Πυθαγόρα).

Διαιρέτες του 220 : 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 (άθροισμα=284)
Διαιρέτες του 284 : 1, 2, 4, 71, 142 (άθροισμα=220)

Να γραφεί αλγόριθμος που θα βρίσκει και θα εμφανίζει όλα τα ζεύγη των φίλιων αριθμών με τον περιορισμό και οι δύο να είναι μικρότεροι του 1000.
(προσπαθήστε να χρησιμοποιήσετε κατάλληλες αλγοριθμικές δομές που θα ελαχιστοποιούν το πλήθος των επαναλήψεων).

Άσκηση 20 - Λύση (1/2)


```c
#include <stdio.h>

int getDivisorsSum(int num){

    int sum = 0;

    for(int i = 1; i < num; i++){
        if(num % i == 0)
            sum = sum + i;
    }
    return sum;
}

// ... συνεχίζεται ...
```

Άσκηση 20 - Λύση (2/2)


```
// ...

int main()
{
    int sum1, sum2;

    for(int i = 1; i < 1000; i++){
        sum1 = getDivisorsSum(i);
        if(sum1 != i){
            sum2 = getDivisorsSum(sum1);
            if(sum2 == i)
               printf("%d - %d \n", i, sum1);
        }
    }
    return 0;
}
```

Άσκηση 21

Ένας τριγωνικός αριθμός (triangular number), έστω x_n, προσδιορίζεται από τη σχέση:

$x_n = n(n + 1) / 2$, όπου n 1, 2,3,... .

Μερικοί τριγωνικοί αριθμοί είναι : 1, 3, 6, 10, 15, 21, 28, 36, 45,…
Ένας αριθμός λέγεται τέλειος (perfect number) αν είναι ίσος με το άθροισμα των γνησίων διαιρετών του.

Π.χ. οι αριθμοί 6 και 28 είναι τέλειοι διότι : 6=1+2+3 και 28=1+2+4+7+14.

Να γράψετε έναν αλγόριθμο ή ένα πρόγραμμα σε γλώσσα C που:

1. Θα εισάγει κατά σειρά M (M = γνωστό, ορίζεται ως σταθερά στην αρχή) τυχαίες θετικές ακέραιες τιμές, μέσω της συνάρτησης rand( ) στην περιοχή [1,200].
2. Θα δημιουργεί για κάθε μία από τις M τυχαίες θετικές ακέραιες τιμές τον αντίστοιχο τριγωνικό αριθμό x_n.
3. Θα ελέγχει αν ο τριγωνικός αριθμός είναι τέλειος και εφόσον είναι θα τον εμφανίζει στην οθόνη.

Άσκηση 21 - Λύση (1/2)


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

#define M 10

int main()
{
    int xn, sum;
    srand(time(0));

    for(int i = 0; i < M; i++){
        int random = rand() % 200 + 1;
        xn = random * (random + 1) / 2;

// ... συνεχίζεται ...
```

Άσκηση 21 - Λύση (2/2)


```
// ...

        sum = 0;
        for(int i = 1; i < xn; i++){
            if(xn % i == 0)
               sum += i;
        }
        printf("Ο Τριγωνικός αριθμός: %d: ", xn);
        if (sum == xn)
           printf("είναι τέλειος \n");
        else
           printf("δεν είναι τέλειος \n");
    }
    return 0;
}
```

Άσκηση 22

Να γραφεί πρόγραμμα σε γλώσσα C που θα υπολογίζει το παρακάτω άθροισμα με ακρίβεια epsilon και κατά μέγιστο m επαναλήψεις. Να καθορίσετε ως σταθερές τις τιμές των epsilon και m . Η τιμή του x εισάγεται από το πληκτρολόγιο.

Η ακρίβεια υπολογισμού είναι η απόλυτη τιμή της διαφοράς δύο διαδοχικών τιμών του αθροίσματος.

$$\sum_{n=0}^{\infty}(-1)^n \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1}$$

Για τον υπολογισμό της τιμής του παραγοντικού για n > 10 θα πρέπει να χρησιμοποιηθεί ο τύπος του Stirling:

$$n! \approx e^{-n} n^n \sqrt{2\pi n}$$

Το πρόγραμμα πρέπει να εμφανίζει, σε κάθε επανάληψη, τις διαδοχικές τιμές του αθροίσματος και της ακρίβειας.

Άσκηση 22 - Λύση (1/3)


```c
#include <stdio.h>
#include <math.h>
#define epsilon 1e-4
#define m 30

double factorial10(int n){
    double f;
    f = exp(-n) * pow(n, n) * sqrt(2 * M_PI * n);
    return f;
}
double factorial(int n){
    double result;
    result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}
// ... συνεχίζεται ...
```

Άσκηση 22 - Λύση (2/3)


```c
// ...

int main()
{
    double x, term, old_term, sum;
    int n;

    printf("Δώσε τον αριθμό x: ");
    scanf("%lf", &x);

    n = 0;
    term = pow(-1, n) * factorial(2*n) / (pow(4, n) * pow(factorial(n), 2) * (2*n +1)) * pow(x, 2*n+1);
    sum = term;
    old_term = 0;

    // ... συνεχίζεται ...
```

Άσκηση 22 - Λύση (3/3)


```
// ...
while(fabs(fabs(term) - fabs(old_term)) > epsilon && n < m){
 printf("Τρέχον άθροισμα: %.6lf, Ακρίβεια: %.6lf \n", sum, fabs(fabs(term) - fabs(old_term)));
 old_term = term;
 n++;
 if(n <= 5)
  term = pow(-1, n) * factorial(2*n) / (pow(4, n) * pow(factorial(n), 2) * (2*n +1)) * pow(x, 2*n+1);
 else if(n <= 10)
  term = pow(-1, n) * factorial10(2*n) / (pow(4, n) * pow(factorial(n), 2) * (2*n +1)) * pow(x, 2*n+1);
 else
  term = pow(-1, n) * factorial10(2*n) / (pow(4, n) * pow(factorial10(n), 2) * (2*n +1)) * pow(x, 2*n+1);

 sum += term;
}
printf("Όροι ακολουθίας: %d \n", n);
printf("Αποτέλεσμα: %.6lf \n", sum);
return 0;
}
```