Άσκηση 1.8

Για τον υπολογισμό της τιμής του n! όταν η τιμή του n είναι μεγάλη, χρησιμοποιείται ο προσεγγιστικός τύπος του Stirling:

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

Να γραφούν 2 διαφορετικές συναρτήσεις που να δέχονται ως είσοδο έναν ακέραιο αριθμός και να υπολογίζουν το παραγοντικό του με:

- τον προσεγγιστικό τύπο του Stirling
- χρήση του τύπου: n ! = 1 x 2 x 3 x . . . x n

Και οι δύο συναρτήσεις θα επιστρέφουν τιμή τύπου double.
Να υλοποιήσετε ένα πρόγραμμα που να δέχεται έναν ακέραιο αριθμό και να υπολογίζει το παραγοντικό του με κατάλληλη κλίση των παραπάνω συναρτήσεων. Να βρίσκει επίσης και να εκτυπώνει τη διαφορά (ως ποσοστό %) που παρουσιάζουν οι 2 μέθοδοι υπολογισμού.

Άσκηση 1.8 - Συνάρτηση Stirling


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

// Συνάρτηση υπολογισμού παραγοντικού με τον τύπο του Stirling

double factorial_stirling(int n) {
    if (n == 0) return 1.0; // Το 0! είναι 1
    return sqrt(2 * M_PI * n) * pow(n / M_E, n);
}
```

$$n! \approx e^{-n} n^n \sqrt{2\pi n} \quad \Longrightarrow \quad n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Άσκηση 1.8 - Συνάρτηση factorial


```
// Συνάρτηση υπολογισμού παραγοντικού με ακριβή μέθοδο

double factorial_exact(int n) {
    double result = 1.0;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Άσκηση 1.8 - main


```c
int main() {
    int n;
    printf("Εισάγετε έναν ακέραιο αριθμό: ");
    scanf("%d", &n);

    double exact = factorial_exact(n);
    double stirling = factorial_stirling(n);

    printf("Ακριβές παραγοντικό του %d: %.10e\n", n, exact);
    printf("Προσέγγιση με Stirling για το %d!: %.10e\n", n, stirling);

    double percentage_difference = fabs((exact - stirling) / exact) * 100;
    printf("Διαφορά (%%): %.5f%%\n", percentage_difference);

    return 0;
}
```

Άσκηση 1.9

Χρησιμοποιήστε τη συνάρτηση της προηγούμενης άσκησης που υπολογίζει το παραγοντικό ενός αριθμού, για να υπολογίσετε το παρακάτω άθροισμα:

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

με ακρίβεια 10^{-5} (σταματάμε την επανάληψη όταν ο τρέχων όρος γίνει μικρότερος από 10^{-5})

Η τιμή του x είναι τύπου double και εισάγεται με χρήση της scanf.
Το μέγιστο επιτρεπόμενο πλήθος επαναλήψεων είναι 15.

Άσκηση 1.9 - Συνάρτηση factorial


```
// Συνάρτηση για υπολογισμό του n!

double factorial(int n) {
    double result = 1.0;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Άσκηση 1.9 - Συνάρτηση calculate_exp


```
// Συνάρτηση για υπολογισμό του e^x με ακρίβεια 10^-5
double calculate_exp(double x) {
    double sum = 1.0; // Ο πρώτος όρος είναι πάντα 1
    double term = 1.0; // Για τον υπολογισμό κάθε νέου όρου

    for (int n = 1; n <= 15; n++) {
        term = pow(x, n) / factorial(n);
        sum += term;

        // Έλεγχος ακρίβειας
        if (fabs(term) < 1e-5) {
            break;
        }
    }
    return sum;
}
```

Άσκηση 1.9 - main


```c
int main() {
    double x;
    printf("Εισάγετε την τιμή του x: ");
    scanf("%lf", &x);

    double result = calculate_exp(x);
    printf("Το e^%.2f είναι περίπου: %.5f\n", x, result);

    return 0;
}
```

Άσκηση 1.10

Να γράψετε συνάρτηση που να δέχεται έναν ακέραιο αριθμό `k ≥ 2`, και να ελέγχει αν αυτός ο αριθμός είναι **πρώτος (prime)** ή όχι, επιστρέφοντας κατάλληλη τιμή.

Στη συνέχεια να γραφεί ένα πρόγραμμα που να καλεί τη συνάρτηση και να εμφανίζει στην οθόνη όλους τους πρώτους αριθμούς που είναι μικρότεροι μιας γνωστής μέγιστης τιμής `max` που ορίζεται μέσω μιας δήλωσης `#define`.

Άσκηση 1.10 - is_prime


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

// Συνάρτηση για τον έλεγχο αν ο αριθμός k είναι πρώτος
bool is_prime(int k) {
    if (k <= 1) {
        return false;  // Οι αριθμοί <= 1 δεν είναι πρώτοι
    }
    for (int i = 2; i <= k; i++) {
        if (k % i == 0) {
            return false;  // Αν βρούμε διαιρετό αριθμό, ο αριθμός δεν είναι πρώτος
        }
    }
    return true;  // Αν δεν βρούμε διαιρετό, ο αριθμός είναι πρώτος
}
```

Άσκηση 1.10 - main


```c
#define MAX 100  // Ορίζουμε τη μέγιστη τιμή για τον έλεγχο

int main() {
    printf("Πρώτοι αριθμοί μικρότεροι από το %d:\n", MAX);
    for (int i = 2; i < MAX; i++) {
        if (is_prime(i)) {
            printf("%d ", i);
        }
    }
    printf("\n");

    return 0;
}
```

Άσκηση 1.11

Οι H.A.Brothers και J.A. Knox ανακάλυψαν ότι καθώς η τιμή του x γίνεται μεγαλύτερη, η τιμή της παρακάτω έκφρασης:

$$\left(\frac{2x+1}{2x-1}\right)^x$$

προσεγγίζει τη μαθηματική σταθερά e.
Να γράψετε συνάρτηση που να υλοποιεί την παραπάνω μαθηματικό τύπο.
Στη συνέχεια να γραφεί ένα πρόγραμμα που να καλεί επαναληπτικά τη συνάρτηση αυτή, μέχρι η απόλυτη τιμή της διαφοράς του x από τη συνάρτηση `exp` (της βιβλιοθήκης `math.h`) να γίνει μικρότερη του 10^{-6}.
Το πρόγραμμα να εμφανίζει την τελική τιμή του x.

Άσκηση 1.11 - compute_expression


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

double compute_expression(double x) {
   return pow((2 * x + 1) / (2 * x - 1), x);
}
```

$$\left(\frac{2x+1}{2x-1}\right)^{x}$$

Άσκηση 1.11 - main


```c
int main() {
    double x = 1.0; // Αρχική τιμή του x
    double result;
    double epsilon = 1e-6; // Όριο ακρίβειας
    while (1) {
        result = compute_expression(x);
        // Έλεγχος αν η διαφορά με τον αριθμό e είναι μικρότερη από το όριο ακρίβειας
        if (fabs(result - exp(1)) < epsilon)
            break;
        x += 1.0; // Αύξηση του x για την προσέγγιση
    }
    printf("Η τελική τιμή του x είναι: %.0lf\n", x);
    return 0;
}
```

Άσκηση 1.12

Ένας ακέραιος αριθμός, ονομάζεται αριθμός **Armstrong** αν ισχύει:
(άθροισμα των ψηφίων του, καθένα υψωμένο στη δύναμη του πλήθους των ψηφίων) = ο ίδιος ο αριθμός.

Παραδείγματα: $153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27$
$9474 = 9^4 + 4^4 + 7^4 + 4^4$

Να γραφεί συνάρτηση που δέχεται έναν θετικό ακέραιο αριθμό, να ελέγχει αν είναι Armstrong, επιστρέφοντας κατάλληλη τιμή.

Στη συνέχεια να γραφεί ένα πρόγραμμα που να εμφανίζει όλους τους αριθμούς Armstrong που υπάρχουν στο διάστημα [10, 9999].

Άσκηση 1.12 - is_armstrong


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

int is_armstrong(int num) {
    int sum = 0;
    int digits = 0;
    int temp;
    // Υπολογισμός του πλήθους των ψηφίων
    temp = num;
    while (temp != 0) {
        digits++;
        temp /= 10;
    }
    // Υπολογισμός του αθροίσματος των ψηφίων υψωμένων στη δύναμη του πλήθους των ψηφίων

    temp = num;
    while (temp != 0) {
        int digit = temp % 10;
        sum += pow(digit, digits);
        temp /= 10;
    }

    return sum == num;
}
```

Άσκηση 1.12 - main


```c
int main() {
    printf("Αριθμοί Armstrong στο διάστημα [10, 9999]:\n");
    for (int i = 10; i <= 9999; i++) {
        if (is_armstrong(i)) {
            printf("%d\n", i);
        }
    }

    return 0;
}
```