Laboratorium – 20.04.2021

1. Napisz funkcję silnia(), która obliczy silnię liczby $n \in \mathbb{Z}$:
   - Obliczenia wykonaj przy użyciu jednej pętli for,
   - Zmienna, przechowująca wynik działania funkcji deklarowana jest przed pętlą for i inicjowana jest wartością 1. W pętli zmienna ta jest mnożona przez indeks pętli i przypisywana jest do siebie samej (wykorzystaj operator mnożenia z przypisaniem *=),
   - Nie korzystaj z instrukcji warunkowej, w celu obliczenia silni liczby 0,
   - Niech funkcja zwraca wynik w postaci liczby typu long int (dlaczego nie int?),
   - Funkcja powinna przyjmować jeden argument typu int,
   - Stwórz prototyp funkcji w nagłówku programu.

Czy konieczne jest podawanie nazwy zmiennej w argumencie funkcji? Czy potrafisz wskazać zalety wypisania prototypów funkcji przed funkcją main?

2. Wykorzystaj napisaną funkcję obliczającą silnię, aby napisać funkcję obliczającą eksponentę dowolnej liczby rzeczywistej. Niech funkcja zwraca wynik typu double, natomiast przyjmuje dwa argumenty: wykładnik eksponenty double x oraz dokładność numeryczną wyznaczenia eksponenty double eps. Przykładowa deklaracja wygląda następująco: double my_exponent(double x, double eps).
   a. W celu obliczenia eksponenty najwygodniej skorzystać z rozwinięcia funkcji w szereg Tylora:

$$e^x = \exp(x) = \sum_{i=0}^{N} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots, N \in \mathbb{Z}.$$

   b. Szereg Taylora jest zbieżny dla x bliskiego 0, dlatego np. dla $x > 1$ dokładność obliczonej przez nas eksponenty byłaby bardzo niska, a wręcz dla dużych x dostalibyśmy ciąg rozbieżny. Zapewnienie zbieżności naszego algorytmu można zapewnić wykorzystując własność potęgowania. Jeśli $x > 1$, to:

$$\exp(x) = \exp(y)\exp(z),$$

   gdzie y – część całkowita liczby x: $y = x - x \bmod 1$ oraz $z = x - y$.
   Wystarczy wtedy rozwinąć w szereg eksponentę części ułamkowej x, a na koniec pomnożyć wynik rozwinięcia w szereg przez całkowita potęgę liczby $e = 2{,}7128 \ldots$.
   - Część całkowitą można ławo dostać wykorzystując rzutowanie liczby typu double na int: int y= (int) x;
   - double z=x-y.

   c. Obliczenia przeprowadź dla N takiego, że $\frac{z^N}{n!} < eps$. Ten warunek oznacza, że jeśli N-ty element rozwinięcia funkcji $\exp(z)$ w szereg Taylora jest mniejszy niż zadana przez użytkownika dokładność, to algorytm obliczania eksponenty się przerywa i funkcja zwraca wynik.

d. Na koniec porównaj wynik działania swojej funkcji z wbudowaną funkcją exp(double) zdefiniowaną z bibliotece math.h.

Implementacja zadania w języku C może przebiegać następująco:

- Tworzymy nagłówek funkcji: double my_exponent(double x, double eps),
- Tworzymy zmienną int y=(int) x – przechowuje część całkowitą wykładnika eksponenty
- Tworzymy kolejną zmienną double z=x-y – przechowuje część ułamkową wykładnika eksponenty,
- N, dla którego sumujemy elementy rozwinięcia w szereg Taylora znajdujemy, wykorzystując pętlę while, która wykonywać się będzie dopóki fabs(pow(z, N)/silnia(N))<eps. Zmienna N na początku działania pętli ma wartość 0, natomiast zwiększa się o 1, po każdym wykonaniu pętli. Pętlę można przerwać instrukcją break albo definiując odpowiedni warunek w wywołaniu pętli,
- Po znalezieniu N, tworzymy pętlę for, w której sumujemy kolejne składniki szeregu Taylora. Indeks pętli zmienia się od 0 do N włącznie. Wynik sumowania przechowujemy w zmiennej suma. Zmienną tę deklarujemy przed utorzeniem pętli for i inicjujemy wartością 0,
- Ostateczny wynik orzymujemy mnożąc wynik działania funkcji pow(M_E, y) przez zmienną suma (M_E to stała matematyczna zdefiniowana w bibliotece math.h, natomiast pow to funkcja obliczająca potęgę, zdefiniowana w tej samej bibliotece. Przyjmuje dwa argumenty, z których pierwszy jest podstawą potęgi, a drugi jest wykładnikiem potęgi),
- Na koniec zwracamy wartość instrukcją return.

Przykład:
Spróbujmy obliczyć $\exp(2{,}5)$ z dokładnością $eps = 0.001$.

Rozwiązanie:

$$\exp(2{,}5) = \exp(2) \cdot \exp(0{,}5) = \exp(2) \cdot \sum_{i=0}^{N} \frac{0{,}5^i}{i!}.$$

Ponieważ $\frac{z^N}{N!} < eps$ dla $N = 5$, to ostateczny wzór na obliczenie eksponenty z zadanej wartości to:

$$\exp(2{,}5) \approx e \cdot e \cdot \left(1 + 0{,}5 + \frac{0{,}5^2}{2} + \frac{0{,}5^3}{6} + \frac{0{,}5^4}{24} + \frac{0{,}5^5}{120}\right).$$

Uwaga! Powyższa implementacja jest dokładnie tą, która została użyta w bibliotece math.h!

Czy potrafisz uogólnioć napisaną funkcję tak, żeby obliczenia przeprowadzać dla $x \epsilon \mathbb{C}$? (To jest pytanie wykraczające, nad którym można się zastanowić w domu).

3. Napisz funkcję, która będzie obliczała moment rzędu n oraz moment centralny rzędu k zestawu danych zapisanych w tabeli.
   a. Niech funkcja będzie postaci `void statystyka(double tab[], int n, int k, double tab_momenty[])`, gdzie
      - `tab[]` – tablica o rozmiarze 20, do której przypisujemy liczby z rozkładu jednorodnego przy użyciu funkcji `rand()` z biblioteki standardowej,
      - `n` – rząd momentu rozkładu
      - `k` – rząd momentu centralnego
      - `tab_momenty[]` – tablica, w której będziemy przechowywać wartość momentu (pierwszy element tablicy) oraz wartość momentu centralnego (drugi element)
   b. Estymatorem momenu rzędu n rozkładu prawdopodobieństwa zmiennej losowej x jest:

$$\hat{E}[x^n] = \frac{1}{N}\sum_{i=1}^{N} x_i^n,$$

      natomiast estymatorem momentu centralnego rzędu k jest:

$$\hat{E}\left[x - \hat{E}[x]\right] = \frac{1}{N-1}\sum_{i=1}^{N}\left(x_i - \hat{E}[x]\right)^k,$$

      gdzie i – numer pomiaru, N – liczba pomiarów w zestawie danych, x_i – jeden z pomiarów zmiennej x. Sumowanie najwygodniej przeprowadzić w pętli `for`, której indeks zmiania się od 0 do $N-1$. Aby obliczyć moment centalny, należy wcześnej obliczyć średnią arytmetyczną, używając innej pętli `for`.
   c. Moment proszę zapisać do pierwszego elementu tablicy `tab_momenty` (pamiętajmy, że pierwszy element tablicy ma indeks 0), natomiast moment cenralny do drugiego elementu tablicy `tab_momenty`.

Prosze zawuażyć, że pomino tego, że funkcja `statystyka` nie zwraca wartości explicite (jest typu `void`), to wynik działania funkcji zostaje zachowany dzięki przekazanej w postaci argumentu funkcji zmiennej tablicowej. Jest to dobry sposób zwrócenia przez funkcję kilku wartości tego samego typu. Jeśli chcialibyśmy, żeby funkcja zwróciła kilka wartości **różnego typu**, możemy to zrobić poprzez **strukturę**, o której będziemy się uczyć w niedalekiej przyszłości!