Zadanie 0 – Odwzorowanie logistyczne (2/3pkt)

Napisz program, który przeprowadzi symulację rozwoju populacji zwierząt wg modelu logistycznego. Model jest zdefiniowany poprzez ciąg rekurencyjny:

$$x_{n+1} = ax_n(1 - x_n), \tag{1}$$

gdzie $0 < a < 4$ jest parametrem wzrostu, a $0 \leqslant x_k \leqslant 1$ populacją unormowaną do jedynki ($x = 1$ odpowiada zużyciu całych zasobów i maksymalnej dopuszczalnej wielkości populacji).
Program powinien pobierać wartości a i x_0 jako parametry programu oraz sprawdzić ich zakresy. Jako trzeci parametr powinien pobierać liczbę iteracji ($\geqslant 0$). Do liczenia wyrazów ciągu użyj rekurencji. Wynikiem działania programu powinna być informacja o numerze iteracji i wielkości populacji (każda iteracja w nowej linii). W przypadku podania niepoprawnych argumentów program powinien kończyć pracę z wyświetleniem komunikatu o błędzie.

Przykład działania:
./zad1 0.7 0.2
Użyj: ./zad1 a x0 n

./zad1 0.7 0.2 10
ii x
0 0.2
1 0.112
2 0.0696192
3 0.0453407
4 0.0302994
5 0.020567
6 0.0141008
7 0.00973136
8 0.00674566
9 0.00469011
10 0.00326768

Zaobserwuj następujące zjawiska:

1. Dla $0 < a < 1$ populacja wyginie. Oznacza to, że ciąg będzie zbieżny do zera. Fachowo mówi się, że zero jest *atraktorem*.

2. Dla $1 < a < 2$ populacja zbiega szybko do $\frac{a-1}{a}$ niezależnie od początkowej populacji.

3. Dla $2 < a <= 3$ populacja zbiega powoli do $\frac{a-1}{a}$ wcześniej oscylując.

4. Dla $3 < a < 3.44949$ populacja będzie oscylować wokół dwóch wartości zależnych od r. Rozszczepienie atraktora na dwa nazywa się *bifurkacją*.

5. Dla $3.44949 < a < 3.54409$ (*approx*) populacja będzie oscylować wokół czterech wartości zależnych od r. Kolejna bifurkacja. Dla większych wartości będzie 8, 16, 32 punkty oscylacji itd.

6. Dla $a > 3.56995$ większość[1] wartości a i $x0$ prowadzi do zjawiska *chaosu deterministycznego* (atraktor nazywamy chaotycznym). Oznacza to, że chociaż kolejne wyrazy ciągu są ściśle zdefiniowane, to ciąg wydaje się być losowy, nie wystepują w nim oscylacje (można powiedzieć, że są oscylacje o okresie nieskończonym).

Wartość x_0 ma mniejsze znaczenie dla zachowania ciągu niż wartość a.

Uwaga:Nie wolno używać tablic ani kolekcji. Sprawdź ilość i poprawność argumentów w mainie. Funkcja licząca wyrazy ciągu ma być rekurencyjna (żadnych pętli!).
Podpowiedź: Przy sprawdzaniu unikaj podawania brzegowych wartości a. Zacznij od małego n, powiedzmy 30. Jeżeli nie widzisz zbieżności/oscylacji to zwiększ n kilka razy. **To zadanie jest podobne do zadania na rekurencję z poprzednich zajęć.**

Zadanie 1 – całkowanie Monte Carlo (1 pkt)

Chcemy znaleźć przybliżoną wartość liczby π. Rozważmy koło jednostkowe (o promieniu $r = 1$) wpisane w kwadrat o boku 2 (patrz obrazek poniżej). Zastanówmy się, jakie jest prawdopodobieństwo, że jeśli wylosujemy punkt na kwadracie to będzie on leżał wewnątrz koła? To prawdopodobieństwo wynosi tyle co stosunek pól koła do kwadratu, czyli $\pi/4$. Jeżeli oszacujemy empirycznie to prawdopodobieństwo to będziemy mogli wyznaczyć przybliżoną wartość liczby π.

Procedura jest następująca:

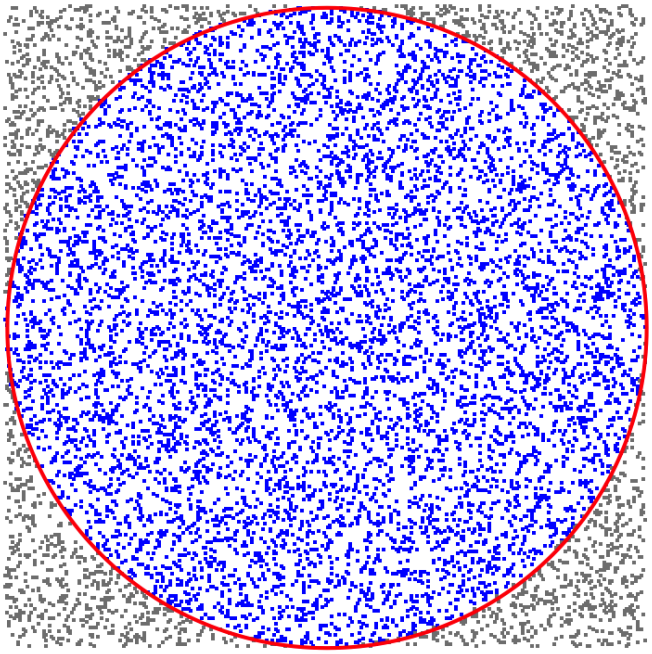
1. Wygeneruj losowo n punktów z rozkładu jednostajnego na kwadracie.
2. Policz ile punktów znajduje się wewnątrz koła, powiedzmy, że jest to k punktów.
3. Stosunek k/n jest estymatą prawdopodobieństwa. Ale wiemy, że prawdopodobieństwo wynosi ściśle $\pi/4$, więc mnożąc estymatę przez 4 dostajemy przybliżoną wartość liczby pi.

Napisz program, który pobiera jeden parametr w momencie uruchomienia będący liczbą punktów do wylosowania. Nazwijmy ten parametr N. Następnie wygeneruj 2N liczb pseudolosowych z rozkładu jednostajnego, w ten sposób dostaniesz N punktów na kwadracie, każdy o 2 współrzędnych. Policz ile punktów jest wewnątrz koła i podziel to przez liczbę wszystkich punktów N, żeby dostać estymatę prawdopodobieństwa. Na koniec wypisz wynik na standardowe wyjście. Proszę przetestować program dla co najmniej 3 różnych wartości N, każda innego rzędu wielkości.

Przykładowy output:
./zadanie0 10
pi = 2.8

./zadanie0 100
pi = 3.28

[1]Dla niektórych wartości parametrów istnieją tzw. okna stabilności, gdzie nie obserwuje się chaosu. Jeżeli trafiłeś na takie wartości to gratulacje! Zainteresowanych odsyłam np do Wikipedii. Jest też wiele fajnych książek o chaosie na różnym poziomie zaawansowania matematycznego. Sam chaos deterministyczny wywodzi się z badań nad prognozowaniem pogody, ale pojawia się w wielu dziedzinach fizyki, a jak pokazuje przykład również w innych naukach.



Rysunek 1: Źródło: Google Image

./zadanie0 1000
pi = 3.16

./zadanie0 10000
pi = 3.1556

./zadanie0 100000
pi = 3.14592

./zadanie0 1000000
pi = 3.14231

./zadanie0 10000000
pi = 3.14091

./zadanie0 100000000
pi = 3.14151

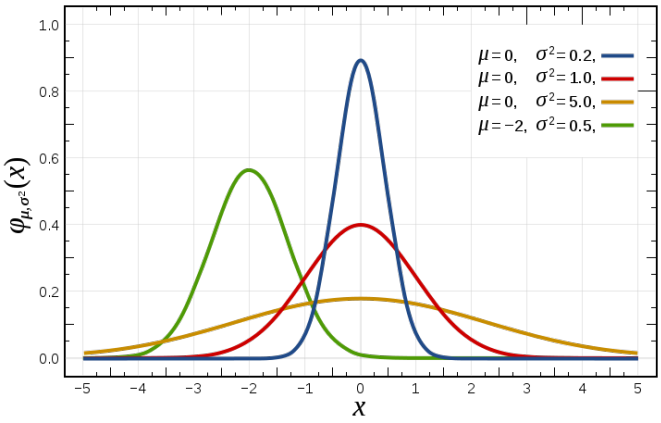
Uwaga: Nie używać tablic ani kolekcji. Nie używać std::cin. Sprawdzić, że argument N jest podany (jeśli nie to komunikat i koniec programu) oraz, że jest dodatni (jeśli nie to komunikat i koniec programu). Uważać z podawaniem dużych N, bo program może się przywiesić na chwilę.
Podpowiedź: Generowanie liczb pseudolosowych jest opisane krótko w teoretycznym minimum oraz w ćwiczeniach do tych zajęć. Zamiast porównywać pole koła jednostkowego i kwadratu o boku 2, można np. porównać ćwiartkę koła jednostkowego na kwadracie o boku 1 (stosunek pól jest taki sam). Jeżeli dostajesz jako wynik ciągle 0, to prawdopodobnie nie zrobiłeś konwersji z int na double

Zadanie 2 – Monte Carlo kontratakuje (1 pkt)

Rozkład Gaussa dany jest wzorem:

$$\rho(\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2}$$

gdzie μ jest średnią, a σ odchyleniem standardowym.



Rysunek 2: Źródło: `https://en.wikipedia.org/wiki/Gaussian_function#/media/File:Normal_Distribution_PDF.svg`

Jak policzyć pole pod krzywą Gaussa dla pewnego przedziału? Moglibyśmy postąpić podobnie jak w Zadaniu nr 1, to jest wygenerować z rozkładu normalnego punkty w odpowiednio dużym prostokącie, np. 100σ na 1. Następnie dla każdego x z zadanego przedziału policzylibyśmy wartość funkcji Gaussa i porównali z wylosowanym y. Jeżeli wylosowany y byłby mniejszy, oznaczałoby to, że punkt jest pod krzywą i zaakceptowalibyśmy go, w przeciwnym razie odrzucilibyśmy. Dzieląc przez liczbę wszystkich wygenerowanych punktów dostalibyśmy pole pod krzywą, a więc całkę oznaczoną.
Ale jest lepsza i prostsza metoda, która wymaga znacznie mniej operacji i nie narzuca ograniczenia się do jakiegoś przedziału (np. od -50σ do 50σ). Możemy wygenerować liczby pseudolosowe z rozkładu normalnego i policzyć ile z nich wpada do określonego przedziału całkowania. Im większa wartość ρ w jakimś przedziale tym więcej będzie tam liczb. Na koniec wystarczy podzielić przez liczbę wszystkich wygenerowanych liczb.
Napisz program realizujący następujące kroki:

1. Niech program przyjmuje 3 parametry w momencie uruchomienia: średnią μ, odchylenie standardowe σ i liczbę liczb do wylosowania N.
2. Użyj tych parametrów, żeby wygenerować N liczb z rokładu normalnego.
3. Policz, ile spośród tych liczb trafiło do przedziału $(-\sigma, \sigma)$ a ile do $(-2\sigma, 2\sigma)$.
4. Podziel te liczby przez N i wypisz **czytelny** komunikat.
5. Przetestuj porgram. Niezależnie od μ i σ, dla dostatecznie dużej liczby N powinieneś dostać odpowiednio 0.68 i 0.95 (około).

Przykładowy output:
./zadanie1 3.1224 5.223 0
Liczba punktow do wygenerowania musi być większa niż 0!

./zadanie1 3.1224 5.223 100
Całka na przedziale (-2.1006, 8.3454) = 0.63
Całka na przedziale (-7.3236, 13.5684) = 0.98

./zadanie1 -3.1224 10.223 10000
Całka na przedziale (-13.3454, 7.1006) = 0.6843
Całka na przedziale (-23.5684, 17.3236) = 0.9518

Uwaga: Nie używać tablic ani kolekcji. Nie używać std::cin. Sprawdzić liczbę i poprawność argumentów. Uważać z podawaniem dużych N, bo program może się przywiesić na chwilę.
Podpowiedź: Generowanie liczb pseudolosowych jest opisane krótko w teoretycznym minimum oraz w ćwiczeniach do tych zajęć. Jeżeli dostajesz jako wynik ciągle 0, to prawdopodobnie nie zrobiłeś konwersji z int na double

Zadanie 3 – szyfr Cezara (1 pkt)

Szyfr Cezara to bardzo prosta metoda szyfrowania wiadomości. Szyfrowanie polega na zamianie każdej litery na inną, oddaloną od oryginalnej o pewną stałą odległość w alfabecie. Dla wszystkich liter szyfrowanej wiadomości stosuje się identyczne przesunięcie w tę samą stronę, np. jeżeli ograniczymy się do alfabetu angielskiego i przesuwamy do przodu o dwa, to zachodzą nastepujące podmiany:

1. $a \rightarrow c$
2. $b \rightarrow d$
3. $c \rightarrow e$
4. ...
5. $y \rightarrow a$
6. $z \rightarrow b$

Zauważmy, że alfabet jest zapętlony, zatem ostatnie litery przechodzą na pierwsze.
Napisz program, który przyjmuje dwa parametry wywołania: przesunięcie oraz napis do zaszyfrowania. Niech rezultatem działania programu będzie wypisanie zaszyfrowanej wiadomości na standardowe wyjście. Przesunięcie może być zarówno ujemne, zero, lub dodatnie.

Przykładowy output:
./zadanie3 -3 ”abc-def-ghi”
xyz-abc-def

./zadanie3 13 ”ala ma kota, a jacek ma 2 psy: azora i puszka.”

nyn zn xbgn, n wnprx zn 2 cfl: nmben v chfmxn.

Uwaga: Nie tworzyć nowych tablic ani kolekcji. Pamiętaj, żeby sprawdzić liczbę argumentów. Zazwyczaj szyfr Cezara ignoruje wielkość liter, dlatego możesz założyć, że wiadomość nie zawiera wielkich liter, tylko małe od a do z. Wiadomość może zawierać inne znaki, np.: kropkę, przecinek, myślnik, cyfry – te znaki powinny pozostać bez zmian!
Podpowiedź: Ciągi znaków są traktowane jakby były tablicami, dlatego możemy myśleć o argv jak o dwuwymiarowej tablicy: pierwszy wymiar określa argument będący ciągiem znaków, a drugi konkretne znaki. Np. **argv[1][3]** zwróci 3 znak pierwszego argumentu podanego podczas uruchamiania.
Dla chętnych: Zrób, żeby działało też dla wielkich liter, tzn. żeby wielkie litery przechodziły w wielkie, a małe litery w małe.