3 Statystyka

3.1 Średnia i wariancja próby losowej

Szablon programu należy uzupełnić o definicję funkcji `aver_varian(const double tab[], size_t n, double *arith_average, double *variance)`, która oblicza średnią arytmetyczną oraz wariancję zbioru `n` liczb zapisanych w tablicy `tab`. Obliczone wartości zapisuje w pamięci pod adresami przekazanymi w parametrach `arith_average` i `variance`.

Test 3.1

- **Wejście**
  Numer testu, liczba prób n, n wartości zmiennej losowej.
- **Wyjście**
  Wartości średniej arytmetycznej i wariancji.
- **Przykład:**
  Wejście: 1 4
  -1. 0.5 0. 1.5
  Wyjście: 0.250 0.812

3.2 Tablica wyników prób Bernoulliego

Dla przypomnienia:
Próba Bernoulliego to eksperyment losowy z dwoma możliwymi wynikami, np. rzut monetą z wynikami 0 (reszka, porażka), 1 (orzeł, sukces) .
Przyjmujemy, że moneta nie jest symetryczna, tzn. zadajemy, jakie jest prawdopodobieństwo p rezultatu "orzeł" – wyniku 1 (dla monety symetrycznej byłoby równe 0.5).
Symulację takiego eksperymentu należy zrealizować stosując biblioteczny generator liczb pseudolosowych.
Dla powtarzalności wyników programu należy przyjąć, że wynik próby jest równy 1 gdy wylosowana z przedziału $[0, RAND\_MAX]$ liczba jest mniejsza od $p \cdot (RAND\_MAX + 1)$.

Szablon programu należy uzupełnić o definicję funkcji `bernoulli_gen(...)`, która generuje losowo tablicę n prób Bernoulliego. Elementami tej tablicy mają być wyniki prób.

Test 3.2
Test symuluje wykonanie rzutów monetą. Wczytuje liczbę rzutów i założone prawdopodobieństwo wypadnięcia orła (wyniku = 1) oraz wyprowadza wyniki kolejnych prób.

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób n, prawdopodobieństwo p wypadnięcia orła (jedynki).

- **Wyjście**
  Wyniki symulacji n prób.

- **Przykład:**
  Wejście: 2 2 20 0.7
  Wyjście: 0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1

3.3 Dyskretny rozkład prawdopodobieństwa

Dyskretny rozkład prawdopodobieństwa jest to rozkład prawdopodobieństwa zmiennej losowej, której zbiór możliwych wartości jest przeliczalny. Zdarzeniem losowym w tym zadaniu jest rzut dwoma sześciennymi kostkami do gry w kości. Wartością zmiennej losowej jest suma oczek tych dwóch kostek, czyli liczba z przedziału $[2, 12]$.
Rezultatem tego zadania ma być przybliżony rozkład prawdopodobieństwa tej zmiennej losowej. Przybliżony, bo

1. zamiast rzutu kostkami stosujemy generator liczb pseudolosowych,
2. wykonujemy tylko skończoną liczbę prób.

Szablon programu należy uzupełnić o definicję funkcji
`two_dice_probab_distrib(double distrib[], int throws_num)`, która symuluje wykonanie `throws_num` rzutów dwoma kostkami i zapisuje wartości otrzymanego przybliżonego rozkładu prawdopodobieństwa w 11-elementowej tablicy `distrib`.
Dla uzyskania powtarzalności wyników, losując liczbę oczek jednej kostki należy tylko raz wywołać funkcję `rand()`.

Test 3.3
Test wczytuje dane, wywołuje funkcję `two_dice_probab_distrib(distrib, throws_num)` i wyprowadza wartości obliczonego rozkładu.

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób `throws_num.`

- **Wyjście**
  Wartości rozkładu prawdopodobieństwa w postaci liczbowej.

- **Przykład:**
  Wejście: 3 20 1000
  Wyjście: 0.024 0.063 0.091 0.095 0.146 0.159 0.146 0.107 0.089 0.056 0.024

3.4 Dysrybuanta (ang. Cumulative Distribution Function)

Szablon programu należy uzupełnić o definicję funkcji
`cum_discret_distrib(double distrib[], size_t n)`,
która na podstawie danego dyskretnego rozkładu prawdopodobieństwa (zapisanego w `n` elementowej tablicy `distrib`) oblicza wartości dystrybuanty dla każdej wartości zmiennej losowej [1]). Obliczone wartości dystrybuanty zapisuje w tablicy `distrib` – na miejscu danych wejściowych.

Test 3.4
Test wywołuje najpierw funkcję `two_dice_probab_distrib(distrib, throws_num)`, która oblicza rozkład prawdopodobieństwa, a następnie wywołuje funkcję `cum_discret_distrib(distrib, n)`, która oblicza wartości dystrybuanty tego rozkładu. Obliczone wartości są wypisywane w postaci liczbowej.

- **Wejście**
  Numer testu, `seed`, liczba prób (rzutów) `throws_num`.
- **Wyjście**
  Wartości dystrybuanty obliczone dla kolejnych wartości zmiennej losowej.
- **Przykład:**
  Wejście: 4 20 1000
  Wyjście: 0.024 0.087 0.178 0.273 0.419 0.578 0.724 0.831 0.920 0.976 1.000

3.5 Histogram

Szablon programu należy uzupełnić o definicję funkcji
`histogram(double tab[], size_t n, int x_start, double y_scale, char mark)`, która w trybie znakowym przedstawia histogram funkcji o `n` wartościach zapisanych w tablicy `tab` o długości `n`. Należy przyjąć założenia:

1. Oś zmiennej niezależnej jest pionowa, skierowana w dół. Oś zmiennej zależnej jest pozioma, skierowana w prawo (nie jest rysowana).
2. Wartości zmiennej niezależnej są kolejnymi liczbami naturalnymi, począwszy od `x_start`. Są one pisane od pierwszej lewej kolumny, w polu o szerokości 2 znaków z wyrównaniem w prawo.
3. W trzeciej kolumnie są spacje, a w czwartej – znaki |, które tworzą oś x.
4. Począwszy od 5. kolumny pisane są znaki `mark`. Liczba znaków jest przeskalowaną i zaokrągloną wartością funkcji. Parametr `y_scale` jest wartością zmiennej zależnej odpowiadającej szerokości jednego znaku na wykresie.

[1] Dla przypomnienia definicji i własności dystrybuanty: `https://en.wikipedia.org/wiki/Cumulative_distribution_function`, w szczególności wykresy w części Properties ilustrujące punkty skokowe i prawostronną ciągłość dystrybuanty dyskretnej.

5. Wartości funkcji (liczby nieujemne typu `double`) są wyprowadzane z dokładnością do 3 cyfr po przecinku w każdym wierszu, po jednej spacji na prawo od ostatniego znaku `mark`.

Test 3.5

Test jest modyfikacją testu 3.3 sprowadzającą się do zmiany sposobu wyprowadzenia wyniku obliczeń - postać liczbowa jest zastąpiona histogramem.
Znak `mark` jest wczytywany ze strumienia wejściowego. Wartości skali `y_scale` jest ustalona w funkcji main() (podana w postaci stałej).

- **Wejście** – Numer testu, `seed`, liczba prób (rzutów) `throws_num` (jak w teście 3.3) oraz – po dowolnej liczbie białych znaków – znak `mark`.

- **Wyjście** – histogram rozkładu prawdopodobieństwa dla rzutu dwoma kostkami

- **Przykład**
  Wejście: 5 20 1000 *
  Wyjście:
  2 | ***** 0.024
  3 | ************* 0.063
  4 | ****************** 0.091
  5 | ******************* 0.095
  6 | ***************************** 0.146
  7 | ******************************** 0.159
  8 | ***************************** 0.146
  9 | ********************* 0.107
  10 | ****************** 0.089
  11 | *********** 0.056
  12 | ***** 0.024

Test 3.6

Test 3.6 jest analogiczny do testu 3.5 – jest modyfikacją testu 3.4 sprowadzającą się do zmiany sposobu wyprowadzenia wyniku obliczeń - postać liczbowa jest zastąpiona histogramem.

- **Wejście** – Numer testu, `seed`, liczba prób (rzutów) `throws_num` (jak w teście 3.4) oraz znak `mark`.

- **Wyjście** – histogram dystrybuanty dla rzutu dwoma kostkami.

- **Przykład**
  Wejście: 6 20 1000 #
  Wyjście:
  2 | # 0.024
  3 | #### 0.087
  4 | ######### 0.178
  5 | ############## 0.273
  6 | ##################### 0.419

```
 7 | ############################ 0.578
 8 | ################################### 0.724
 9 | ######################################### 0.831
10 | ############################################# 0.920
11 | ################################################ 0.976
12 | ################################################# 1.000
```

3.7 Monty Hall problem, czyli jak wybierać „drzwi”, aby zwiększyć prawdopodobieństwo wygranej

Paradoks Monty'ego Halla, w przypadku trojga drzwi (bramek) do wyboru, polega na tym, że intuicyjnie przypisujemy równe szanse dwóm sytuacjom — wskazanie wygranej w jednej z dwóch zakrytych ciągle bramek wydaje się równie prawdopodobne jak wskazanie bramki pustej, bo przecież „nic nie wiadomo”. Tymczasem układ jest warunkowany przez początkowy wybór zawodnika i obie sytuacje nie pojawiają się równie często.
Opis problemu: `https://pl.wikipedia.org/wiki/Paradoks_Monty'ego_Halla`.
Szablon programu należy uzupełnić o definicję funkcji
`monty_hall(int *p_switch_wins, int *p_nonswitch_wins, int n)`,
która symuluje `n` rozgrywek. Założenia:

1. W każdej rozgrywce funkcja wywołuje `rand()` dokładnie 2 razy.
2. W pierwszym losowaniu jest wybierany numer drzwi, za którymi jest nagroda.
3. W drugim losowaniu – numer drzwi, które gracz wybiera na początku gry.

Funkcja oblicza (i przekazuje przez adresy przekazane do parametrów `p_switch_wins` i `p_nonswitch_wins`) ile razy w ciągu `n` rozgrywek wygrywał gracz, który po otwarciu jednych drzwi zmieniał pierwotną decyzję i ile razy wygrywał gracz pozostający przy początkowym wyborze.

Test 3.7
Test wczytuje liczbę prób (rozgrywek), wywołuje funkcję `monty_hall(...)` i wypisuje wyniki symulacji.

- **Wejście**
  Numer testu, `seed`, liczba prób (rozgrywek). `n`

- **Wyjście**
  Liczba wygrywających decyzji „zmień wybór” i liczba wygrywających decyzji „nie zmieniaj”.
- **Przykład**
  Wejście: 7 15 1000
  Wyjście: 656 344