Etap I

- W pliku Program.cs napisać statyczną funkcję RandomNumberGenerator, która przyjmuje parametry int seed, oraz int max i int min. Funkcja zwraca delegatę, generującą liczby losowe z podanego zakresu z zadanym ziarnem. Każde wywołanie funkcji RandomNumberGenerator powinno zwrócić nową delegatę, która będzie generować po kolei te same wartości.

- W pliku Program.cs napisać statyczną funkcję FibonacciNumbers, która przyjmuje jeden parametr i zwraca delegatę. Delegata generuje kolejne elementy ciągu fibonacciego. Każde wywołanie funkcji FibonacciNumbers powinno zwrócić nową delegatę, która będzie generować po kolei te same wartości.

Etap II

- W pliku ExtensionMethods.cs napisać funkcję rozszerzającą IEnumerable<T> o metodę MyEach. Funkcja przyjmuje delegatę przyjmującą jako parametr typ T i zwraca void. Funkcja wykonuje zadaną delegatę na każdym elemencie IEnumerable<T>

- W pliku ExtensionMethods.cs napisać funkcję rozszerzającą IEnumerable<T> o metodę MyWhere. Funkcja przyjmuje delegatę przyjmującą jako parametr typ T i zwracającą typ bool. Funkcja MyWhere filtruje zadaną kolekcję i zwraca nową kolekcję, której elementy spełniają zadaną delegatę.
- W pliku ExtensionMethods.cs napisać funkcję rozszerzającą IEnumerable<T> o metodę ToTransformedArray. Metoda ToTransformedArray zapisze i zwróci przetransformowane elementy z kolekcji IEnumerable<T> do tablicy typu T. Transformacja odbywa się za pomocą delegaty (Func<T,T>) przekazanej jako parametr do funkcji ToTransformedArray. Pamiętać o tym, aby tablica miała ten sam rozmiar co wejściowa kolekcja.

- W funkcji Main dopisać w odpowiednich (oznaczonych komentarzami) miejscach wyrażenia:
    - za pomocą metody MyWhere na kolekcji o nazwie fibonacciNumbers2, wybrać elementy podzielne z tej kolekcji przez k.
    - za pomocą funkcji ToTransformedArray na kolekcji fibonacciNumbers1, zwrócić tablicę, w której elementy są dwukrotnie zmniejszone
    - za pomocą funkcji ToTransformedArray na kolekcji fibonacciNumbers1, zwrócić tablicę, w której elementy są losowo wybrane za pomocą delegaty o nazwie rndGen (nazwa zmiennej w funkcji Main)
    - za pomocą funkcji MyEach na kolekcji values, zwrócić sumę elementów w tej tablicy.

Etap III

- w funkcji Main dodać fragment kodu, który wczyta tekst z pliku “TextToRead.txt” (funkcja File.ReadlAllLines) i zliczy ilość powtórzeń poszczególnych słów. Następnie wypisać słowa w kolejności alfabetycznej (użyć wbudowanej funkcji sort na odpowiedniej kolekcji). Wykorzystać funkcję MyEach na kolekcji zwróconej przez File.ReadAllLines. Można użyć funkcji string.Split, aby z danej linii wydobyć pojedyncze słowa. Podpowiedź: użyć Dictionary<string,int>. We wczytanym tekście oprócz słów nie ma znaków interpunkcyjnych (tylko spacje i znaki nowej linii)

Etap IV

- W pliku Polynomial.cs napisać statyczną funkcję CreatePolynomial, która przyjmuje argument “out Func<double,double> derivative” oraz dowolną ilość argumentów typu double. Funkcja CreatePolynomial zwróci typ Func<double,double>. Zwrócona funkcja pozwala na obliczenie wartości wielomianu zdefiniowanego przez argumenty typu double przekazane do funkcji CreatePolynomial. Pierwszy argument typu double zadany do funkcji CreatePolynomial to wyraz wolny wielomianu. Argument out Func<double,double> derivative pozwoli na obliczenie pochodnej wielomianu w danym punkcie. Pochodną wyznaczyć na podstawie tablicy współczynników przekazanych do funkcji CreatePolynomial.
- W pliku Polynomial.cs napisać statyczną funkcję CreatePolynomial, która przyjmuje argumenty “int degree”, Func<double> generatingFunction oraz “out Func<double,double> derivative”. Funkcja działa tak samo jak funkcja CreatePolynomial opisana w poprzednim punkcie z tym, że wyliczenie współczynników wielomianu odbywa się za pomocą funkcji “generatingFunction”
- W pliku Polynomial.cs napisać statyczną funkcję SumOfPolynomials, która przyjmuje dowolną ilość argumentów typu Func<double,double> i zwraca Func<double,double>. Funkcja oblicza sumę wielomianów w danym argumencie.

Punktacja:
Etap I - 1.0p
Etap II - 1.0p
Etap III - 1.0p
Etap IV - 2.0p