Elementy programowania funkcyjnego

31 października 2025

Instrukcja

Zadanie ćwiczy pisanie kodu z użyciem funkcji wyższych rzędów oraz iteratorów. Projekt utwórz jako bibliotekę i kod startowy zapisz jako lib.rs. Część zadań oznaczona jest przez (iter+loop). Są to zadania, w których do zaimplementowania są dwie funkcje o identycznym zachowaniu. Wersja *loop* to wersja z pętlami i warunkami (bez rekurencji). W wersji *iter* zabronione jest używanie pętli i standardowej kontroli przepływu (`for`, `loop`, `while`, `if`, `match`) oraz rekurencji. Należy tam używać tylko kombinacji metod iteratorów (bez jawnych pętli). Obie implementacje powinny wykorzystywać ten sam algorytm lub logikę obliczeń.

1. Pobierz plik z kodem startowym.
2. Zaimplementuj wszystkie algorytmy w kolejnej sekcji.
3. Napraw wszystkie ostrzeżenia kompilatora.
4. Sprawdź, czy program przechodzi testy (`cargo test`).
5. Znajdź powszechne problemy z kodem używając `cargo clippy` i je napraw.

Funkcjonalność

1. Napisz funkcję `make_counter(start: i64) -> ???`, która zwraca funkcję anonimową, która przy pierwszym wywołaniu zwraca `start`, przy drugim `start+1`, a przy i-tym `start+i-1`.
2. Odkomentuj funkcję `wrap_call`. Uzupełnij typy w definicji `wrap_call` tak, by `nasty_test` się kompilował. Nie zmieniaj niczego w ciele funkcji! Możesz użyć jawnych parametrów generycznych lub słowa kluczowego `impl`.
3. (iter+loop) `sum_squares_odd` — funkcja wybiera liczby nieparzyste z przekazanego wycinka, podnosi je do kwadratu i oblicza ich sumę.
4. Zaimplementuj algorytmy działające na grafach skierowanych. Grafy przechowywane są jako wektory par wierzchołków reprezentujących krawędzie `(u, v)`.
   (a) (iter+loop) `vertices` — zwraca posortowany rosnąco wektor wszystkich wierzchołków w grafie (bez duplikatów).
   (b) (iter+loop) `cycles_2` — zwraca posortowany rosnąco wektor wierzchołków należących do co najmniej jednego cyklu długości 2 (*podpowiedź*: `.cartesian_product` z `itertools`).
5. (iter+loop) `primes` — zwraca wektor wszystkich liczb pierwszych mniejszych niż `n`.
6. (iter+loop) `run_length_encode` — zwraca wektor par `(wartość, długość_ciągu)` dla kolejnych ciągów identycznych wartości w wejściowym wycinka. Np. dla [1, 1, 1, 2, 8, 8] zwrata `[(1, 3), (2, 1), (8, 2)]`. (*podpowiedź*: `.chunk_by` z `itertools`.
7. (iter+loop) `compose_all` — przyjmuje funkcję `fn(i32) -> i32` i zwraca funkcję, które stosuje wszystkie funkcje po kolei do argumentu wejściowego. (*podpowiedź*: `.fold`).

Wymagania

- Kod kompiluje się przy użyciu stabilnego kompilatora Rust.
- Brak ostrzeżeń z kompilatora i `clippy`, w tym brak ostrzeżeń o nieużywanych strukturach lub funkcjach.
- Wszystkie testy przechodzą.
- Pełna implementacja zadanej funkcjonalności.

Ocena (3 pkt)

- 1 pkt: Praca w trakcie laboratorium.
- 1 pkt: Pełna funkcjonalność (implementacja zgodna z wymaganiami).
- 1 pkt: Prezentacja rozwiązania i odpowiedzi na pytania prowadzącego.