**Лабораторна робота №2**

**Рекурсивні функції. Рекурсивна обробка списків**

**Цілі роботи**: отримання навичок роботи з рекурсивними функціями у мові F#; рекурсивної обробки списків.

**Завдання:**



Короткі теоретичні відомості

Одним з найбільш важливих в програмуванні є можливість здійснити циклічні операції. В мові F# є оператори циклу, однак вони відносяться до імперативних властивостей мови. У функціональному програмуванні циклічні операції виконуються за допомогою рекурсії, що відповідає і математичному, і функціональному сенсу рекурсії. У якості прикладу розглянемо функцію розрахунку факторіала.

Математично факторіал задається функцією:



при обмеженнях: 

Тоді на мові F# функція визначення факторіалу в функціональному стилі має наступний вигляд:

let rec fact n =

if n <= 1 then 1

else n \* fact (n-1)

Ключове слово rec використовується для вказівки компілятору на рекурсивність процесу.

Однак, рекурсивні функції працюють на основі стеку. Тому вони потребують більше пам’яті (а також вони можуть працювати повільніше). У деяких мовах програмування розмір стеку обмежений (наприклад у JavaScript ~40000).

У багатьох мовах програмування присутні оптимізації деяких рекурсивних функцій. Наприклад, якщо виклик рекурсивної фукнції – це сама остання операція, то зберігати контекст виклику не потрібно. Тому компілятор (або інтерпретатор) може оптимізувати рекурсивну функцію, і вона буде виконуватись подібніше до циклу. При цьому стек переповнюватись не буде. Така оптимізація називається **tail-call optimization**.

Тому при можливості завжди є сенс писати функції у вигляді tail-recursion. У мові F# для цього зазвичай використовують спеціальний шаблон **Accumulator Pattern.** Наприклад, функція знаходження факторіалу у вигляді tail recursion може виглядати так:

let accFactorial x =

    let rec accTailRecursiveFactorial x acc =

        if x <= 1 then acc

        else accTailRecursiveFactorial (x - 1) (acc \* x)

    accTailRecursiveFactorial x 1

**Рішення**

Аналіз задачі

Необхідно знайти добуток від’ємних чисел списку.

Але перше, що треба зробити – проаналізувати, що треба зробити насправді.

Очевидно, що для списку [-1,-2,-3] результатом буде -6, а для списку

[1,2,-3,-4] результат становить 12

Але чому має дорівнювати значення добутку для списків [1,2,3], []? Логіка підказує, що 0... або 1.

По-перше список, у якому немає від’ємних елементів для даної задачі – це теж саме що і пустий список. І справді, задачу можна реалізувати двома способами – послідовно перебрати и множити від’ємні елементи, або спочатку виконати фільтрацію, залишивши таким чином лише від’ємні елементи, а потім їх перемножити. Тобто списки [1,2,3], [] – однакові.

А згідно з <https://en.wikipedia.org/wiki/Empty_product> та <https://math.stackexchange.com/questions/110546/what-is-the-product-of-the-empty-set>

добуток елементів пустої множини дорівнює 1.

Таким чином маємо два випадки:

* Коли в списку є негативні елементи – результат – їх добуток
* У всіх інших випадках – 1.

Напишемо рекурсивну реалізацію даної задачі у вигляді рекурсивної функції (а також рекурсивної функції у вигляді tail recursion). Функція записується досить прямолінійно.

Текс програми

let rec productOfNegs = function

   | [] -> 1

   | x::t when x < 0 -> x \* productOfNegs t

   | \_::t -> productOfNegs t

// With Accumulator Pattern

let productOfNegsTail x =

   let rec \_productOfNegs acc = function

      | [] -> acc

      | x::tail when x < 0 -> \_productOfNegs (acc \* x) tail

      | \_::tail -> \_productOfNegs acc tail

   \_productOfNegs 1 x

productOfNegs [-1;-2;-3]

productOfNegsTail [-1;-2;-3]

productOfNegs [2;3;-2;-5]

productOfNegsTail [2;3;-2;-5]

productOfNegs [1;2;3;-5;4]

productOfNegsTail [1;2;3;-5;4]

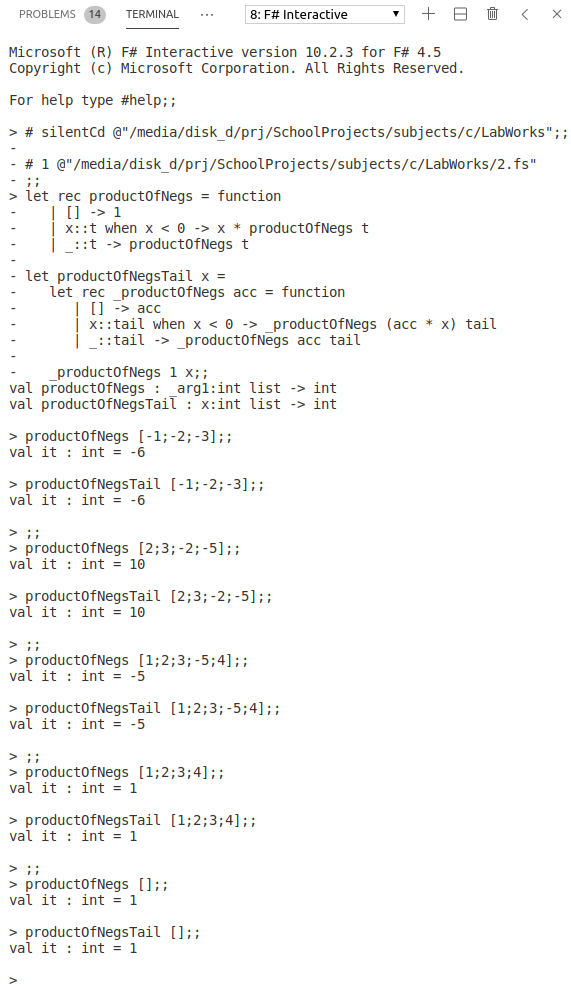
productOfNegs [1;2;3;4]

productOfNegsTail [1;2;3;4]

productOfNegs []

productOfNegsTail []

Результат виконання програми



**Висновок:** на цій лабораторній роботі я отримав навички роботи з рекурсивними функціями у мові F#; рекурсивної обробки списків.