Домашнє завдання №25_2

Одною з альтернатив Haskell при використанні парадигми функційного програмування є Erlang/Elixir.

Загалом, якщо Haskell можна вважати базовою стандартизованою мовою при застосуванні парадигми функційного програмування, то Erlang та Elixir володіють значною практичною цінністю. Це пов'язано з тим, що для них доступні потужні Web-фреймворки, які необхідні для швидкого написання сучасного масового програмного забезпечення. Також отриманий байт-код після трансляції коду мовою Elixir виконується на віртуальній машині Erlang (BEAM), тому Elixir має сумісність з фреймворком Erlang/OTP та іншими бібліотеками і фреймворками мови Erlang.

Пропонується виконати домашнє завдання №25_1 повторно за допомогою Erlang або Elixir без використання готових бібліотечних реалізацій.

** **коментар:** це завдання тотожне до завдань №25_1, №26_1, №26_2 та лабораторної роботи №5; таким чином можна порівняти різні засоби програмування; далі наводиться приклад повністю виконаного завдання; для компіляції і запуску коду мовою Erlang можна використати https://repl.it/languages/erlang , а для компіляції коду мовою Elixir https://repl.it/languages/elixir .*

Вибір варіанту

Варіант завдання відповідає варіанту домашнього завдання №25_1

Приклад коду

Наведений зразок коду у першому лістингу реалізовує завдання мовою Erlang для даних, що подані в наступній таблиці:

Дані для прикладу	`5, 7, 3, 4, 1, 9, 2, 8, 10, 6`
Макровизначення	`-define (INPUT_DATA, [5, 7, 3, 4, 1, 9, 2, 8, 10, 6]).`

Лістинг 1

```
-module(main).
-export([start/0]).

-define (INPUT_DATA, [5, 7, 3, 4, 1, 9, 2, 8, 10, 6]).

qsort([]) ->
    [];
qsort([Pivot | L]) ->
    LesserPart = [X || X <- L, X < Pivot],
    GreaterPart = [X || X <- L, X >= Pivot],
    qsort(LesserPart) ++ [Pivot] ++ qsort(GreaterPart).

print([]) ->
```

```
    [];
print([CurrElement | Others]) ->
    io:fwrite("~w ", [CurrElement]), %io:fwrite("~w" ++ " ", [CurrElement]),
    print(Others).

start() ->
    Input = ?INPUT_DATA,
    Output = qsort(Input),
    io:fwrite("input:\n"),
    print(Input),
    io:fwrite("\n"),
    io:fwrite("output:\n"),
    print(Output),
    io:fwrite("\n").
```

Наведений зразок коду у другому лістингу реалізовує завдання мовою Elixir для даних, що подані в наступній таблиці:

Дані для прикладу	5, 7, 3, 4, 1, 9, 2, 8, 10, 6
Оголошення	input_data = [5, 7, 3, 4, 1, 9, 2, 8, 10, 6]

Лістинг 2

```
input_data = [5, 7, 3, 4, 1, 9, 2, 8, 10, 6]

defmodule QuickSort do
  def sort([]), do: []
  def sort([head|tail]) do
    {lesserPart, greaterPart} = Enum.partition(tail, &(&1 < head))
    sort(lesserPart) ++ [head] ++ sort(greaterPart)
  end
end

defmodule Printer do
  def print([]), do: []
  def print([currElement | others]) do
    #IO.write(inspect(currElement) <> " ")
    ((currElement |> inspect()) <> " ") |> IO.write()
    print(others)
  end
end

#IO.inspect QuickSort.sort(input_data)
input = input_data
output = QuickSort.sort(input)

IO.puts "input:"
Printer.print(input)
IO.write "\n"
IO.puts "output:"
Printer.print(output)
IO.write "\n"
```