LVA 185.A05 Advanced Functional Programming (SS 21)

Central and Control Questions I

Thursday, 4 March 2021

Topic: Part I, Chapter 1; Part II, Chapter 2

Why Functional Programming Matters, Programming with Streams

(No submission, no grading; for self-assessment only)

Part I, Chapter 1 'Why Functional Programming Matters?'

1. What does John Hughes consider *glue* of a programming language? What is glue good for?
2. What kinds of new glue do functional programming languages offer compared to other, especially the imperative programming paradigm?
3. What is considered *conventional* glue by John Hughes? Why does he consider conventional glue insufficient to push the limits of powerful programming significantly forward?
4. How does John Hughes demonstrate the superiority of the new glue offered by functional languages? What are new programming opportunities rendered possible by this 'functional' glue? What are examples he presents to make his point?
5. How does folk knowledge wisdom argue a possible superiority of functional programming compared to programming in other, especially the imperative programming paradigm?
6. Why does John Hughes consider the reasoning along the lines of folk knowledge wisdom to not stand the test?
7. 'Go to statement considered harmful.' Who did coin this statement? Why does it play a role in the reasoning of John Hughes?
8. Shall functional programming languages be lazy or eager? The eternal question of functional programming. How does John Hughes answer it? What is the reasoning underlying his answer?
9. How does the generator/prune pattern support the reasoning of John Hughes?
10. What are characteristics of a 'good' functional program? What shall a functional programmer strive for when programming? What shall (s)he expect to be the relevant 'tools?'

Part II, Chapter 2 'Programming with Streams'

1. What is the generator/prune paradigm, pattern, or principle? What are instances of this paradigm?
2. Illustrate each of the instances of the generator/prune paradigm by means of a concrete example.
3. How can we achieve performance gains using stream programming? Illustrate your answer by means of appropriate examples.
4. What is memoization? How is it related to stream programming?
5. Illustrate memoization by means of a concrete example.
6. What is the difference between recursion and corecursion? Illustrate your answer also in terms of appropriate examples.
7. Is memoization an idea that only recently came up? Who did invent it?
8. From a pragmatical point of view, what are possible problems a programmer will face when implementing a memoization-inspired approach?
9. What is a *stream diagram*? What does it serve for? Illustrate your answer by means of an example.
10. Stream programming and eager evaluation. Do they go well along with each other? What is the reasoning leading to your answer?