tasks.md

1. Read the language definitionas carefully. Find all unclear points and send me you questionsand/or remarks. Write a few examples in F (e.g., sorting).

   --read: 0% --things to implement: 2/7 selection sort quicksort mergesort segment tree bst treap dijkstra algorithm

2. Start thinking about the whole compiler architecture. First of all, choose the target platform for your compiler. It could be one of the following: LLVM framework, Java, or .NET. Taking into account the limited time of your stay in Inno, I would recommend you to choose .NET because it's rather easy to write code generator and other compiler parts for this platform. However the final choice is yours.

   -- .NET -> we are not on windows, but it's cool -- Java -> might be a good training for me, but i don't like it -- LLVM -> might be some c++ magics, but maybe difficult

3. Think about the approach for lexical analyzer implementation. Read carefully the language definition and defined the set of tokens (primary language alphabet) of the language. Choose whether you implement lexical analyzer manually of use some tool like lex. I would recommend you to try to implement it by your own.

   -- self implementation? it's a nice automata imo -- use lex? probably easier to use and bugproof

4. Think about the approach for syntax analyzer (parser) implementation. The point is the same as for lexical analyzer: either implement it "by hand" or to use some tool. I could mention at least two good tools specifically oriented for parser development: bison (or its version for .NET called gppg, easy to find in internet), and ANTLR. You should look at the documentation and choose the tool - or perhaps you decide to implement parser by hand ("recirsive descent parser")

   --recursive descent parses -> might be really funny to implement, but again gonna take some time for sure

5. Think about the language run-time support system. First of all, consider data structures defined in the language - dynamic arrays, maps, and tuples, and design how they should be represented internally, together with operations on them.

   -- internally -> really low level? --dynamic arrays -> vector, allocate like in c++ ? -- maps -> implementation wise, i'd go with AVL Tree - but if we need persistency probably we should go with RB tree -- tuples -> vector of void* ? not efficient, idk much about this

6. Read carefully the section about functions and function types. Your first aim here is to make clear how you are going to implement these concepts. Draw special attention on the features like passing function as function parameters and returning functions as results. Also, think how capturing the function context can be implemented.

   --not even read yet