Homework 4: Group Project in Functional Programming

Due:

The purpose of this homework is to explore some language design issues related to functional programming languages and in particular their use in semantics. The problem on this homework is a group problem. Please follow the policy in section 6 of the course policies document; in particular ensure that the group leadership is rotated from the last group homework.

For your Haskell programs, you must run your code with a Haskell 1.3 interpreter (such as `hugs`). You must also provide evidence that your program is correct (for example, test cases). Hand in a printout of your code and your testing, for questions that require code.

1. (400 points) Choose *one* of the following problems, or invent your own problem for this homework.

   - Write, in Haskell, a type checker for the typed subset of BeCecil. [1, Section 3 and Appendix C]. To reduce the amount of code to write, you may use "abstract syntax" for BeCecil; that is, do not write a lexer and parser. Give examples of at least five (5) BeCecil programs that work with your system.
   - Write, in Haskell, an interpreter for the untyped subset of BeCecil. [1, Section 2 and Appendix A]. To reduce the amount of code to write, you may use "abstract syntax" for BeCecil; that is, do not write a lexer and parser. Give examples of at least five (5) BeCecil programs that work with your system.
   - Design an extension to Haskell that will allow it to be object-oriented (OO). That is, allow for subtyping and dynamic (as opposed to static) overloading. Your extension should integrate smoothly with the language. Give a semantics for your extension that translates extended programs into standard haskell and implement the main parts of this translation in Haskell. To save work, use "abstract syntax" for the core features of your proposal and Haskell itself, don't write a parser.

     Write code in Haskell that for whatever "run-time system" is needed by your extension, and demonstrate your extension by: (a) programming at least five (5) examples of OO design patterns in your extension, (b) giving the code that would be produced by translation of your extension back into standard Haskell, and (c) showing how that code runs in Haskell.
   - Design a domain specific programming langauge for some domain that at least one of you has written several programs for, and implement it as an extension to Haskell. You should pick a limited domain where your language can be a "very high level" langauge that provides a lot of automation. Examples might include: music composition, multimedia synchronization, report generation, scientific data reduction, statistical analysis of patient records, or whatever. Be sure not to design a general purpose programming language. You should implement the langauge by simply writing code in Haskell. Be sure to include a description reference manual for your langauge, and at least five (5) substantial examples of how it works.

   After choosing your topic take the following steps.

   - Send the staff an email about what topic you've picked.
   - Your initial group meeting should try to plan the scope and extent of what you will do, as well as having some ideas for carrying out your plans. After doing this, meet with the staff and get agreement that it is okay to go ahead with your plans.
   - Carry out the plan, checking with us whenever anything unexpected comes up, or if the scope of the project seems too large.

References

[1] Craig Chambers and Gary T. Leavens. BeCecil, a core object-oriented language with block structure and multimethods: Semantics and typing. Technical Report 96-17, Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, Iowa 50011, December 1996. Available by anonymous ftp from ftp.cs.iastate.edu, and by e-mail from almanac@cs.iastate.edu. Also University of Washington Department of Computer Science and Engineering TR number UW-CSE-96-12-02.