CS 3240 : Languages and Computation
Course Project (Phase II)

Due date: April 28, 11:59 pm (No extensions!)

Overview:

1. The goal of the project is to use the tools we learned in class to implement a compiler. Phase II asks you to implement an LL(1) parser. More specifically, you are to generate an LL(1) parser generator (see details below).
2. Do **not** use automatic tools for your parser-- such solutions will get ZERO points. However, if you are not yet done with Phase I, for now you can use some tools (see details below) to implement the scanner, which is needed for the parser. But at the end you will only be given credit for parts which do not use automated tools.
3. Code should be properly documented with meaningful variable and function names. Short elegant code is preferred.
4. You will find the course slides on LL(1) parser useful along with the relevant chapters from the Louden book.
5. Provide instructions about how to compile and execute your files along with test inputs.
6. Any re-use of code (from internet or any other source) is prohibited and will constitute act of plagiarism as per Georgia Tech honor code.
7. If you run into road-blocks or have any questions, seek help from the TAs (please use Piazza to clarify the doubts– we check those round the clock and will respond expediently). Please get started on this project as soon as possible.
8. You have to set up a 20 min appointment with the TAs for the Finals week to do a demonstration.

Goal for Phase II:

As an example of a grammar you will be given a grammar for small language that we call **MiniRE** (as in "mini reg-ex"). You can think of it as a mini Awk or mini Perl. Scripts in MiniRE search and replace strings in **text** files, can save them in string lists, can perform some operations on the string-list and also print to the output. Scripts written using Mini-RE could be very helpful for document analysis and processing. (One could extend the parser to do the semantic analysis of the input and execute the script. But this is not a requrement for the project.)

You will develop an LL(1) parser generator that can generate a parser from *any* LL(1) grammar using the specified syntax and convert it into a LL(1) parsing table which can be used by a LL(1) driver to parse the input, such as e.g., MiniRE script.

Designing a scanner generator: In this phase you will use the scanner developed for REGEX in phase I of the project. If you are not yet done with Phase I, but want to start working on Phase II, you can use Java's regex class to scan and get tokens. See http://docs.oracle.com/javase/tutorial/essential/regex/

As mentioned above, you will first implement the parser generator for LL(1) grammars and then use it to generate the LL(1) parser for the given grammar.

If the scanner or parser finds an error, it stops and reports the error type and location in the script file. The error type for the scanner is the start location of the unrecognized token and the unrecognized token itself, and for the parser, it is the token (including location) for which no grammar rule was found.

The grammar for MiniRE is provided in file grammar.txt. It contains a non-left-recursive, left-factored rule for MiniRE based on BNF. Also it uses a special notation for epsilon, <epsilon>.

In addition, there is a token specification file (token_spec.txt) for the MiniRE tokens.

And finally you will be given a MiniRE script to parse. An example is in script.txt.

Submission Guidelines:

You need to submit the following with your project submission:

1) A report containing:

   a) A brief description of how you have implemented each part of the project.

   b) Any assumptions you have made.

   c) Any problems you faced, and any module you were not able to implement (if at all) and why.

   d) Test Cases.

2) A tar ball containing:

   a) Source code of all the files along with the makefile or equivalent compilation/execution instructions.

   b) Test cases - input files tested on.

   c) Output for the test cases used.

   d) Some documentation of the organization of the code and its functionality with respect to different modules etc.

   e) Instructions on compiling and testing.