CS 540 Programming Fall 2014
Assignment 2: Recursive Descent Parsing

Due: Sun, October at end of day (midnight)

Computing First sets

In class we computed first sets by hand but the process is clearly automatable. For this assignment, you are going to take as input a CFG and output the first sets for the non-terminals associated with that grammar.

Creating this assignment

You will need to do this assignment in several steps. BE SURE the previous steps work before going on to the next step.

1. Using a lex tool, write REs for the tokens needed for your input language. You will need to set up a different constant to return for each token type.
2. Once this works correctly, you will want to use this starting lexer to build your recursive descent parser. Using the grammar given in this document as a starting point, you need to create a LL grammar for the same language.
3. Once you are happy with your grammar, you need to write it as a recursive descent parser and test it on multiple inputs so that you know it works as intended (i.e. it accepts all inputs that it should and rejects incorrect inputs). It is important to get this step correct as I can't grade assignments that can't process the input files. I will put some input files online for you to try parsing.
4. Figure out what data structure you want to use to store your grammar and add code to the RDP of step 3 to build this as you parse the input. Remember that if you want to use some part of yytext, you need to do this before you call 'match() ' for the relevant token. I suggest you use this data structure to output the grammar once you have completed this step to be sure you have done what you intended to do.
5. Once you have stored your grammar, you can process it to determine the first sets. You will need to do this iteratively – keep processing your grammar until your computed first sets no longer change.

Be sure to test your program on different inputs. TRY to break your tool; this is preferable to having me break it ☺

Lexical Elements

- NT – nonterminals will be single character {'A' – 'Z'}
- T – terminals will be a single character {'a'-'z'}
- GOES - '--> ' characters

- SEMI – ';'
- OR – '|'
- EOL – end of line (\n)
- You will want to discard other white space (blanks, tabs) as well as blank lines. Don't make any assumptions about white space EXCEPT that there will be EOL marker in the places described in the grammar below.

Starting grammar

```
prod_list          :    prod_list production
                   |    production
production         :    NT GOES production_body SEMI EOL
production_body    :    production_body OR rule
                   |    rule
rule               :    NT rule
                   |    T rule
                   |    EOL
```

Note:

1. There will always be at least one production in the file.
2. Note that epsilon is not an explicit character in the input. If the rule portion of a production body is empty (i.e. only has EOL), this production goes to epsilon.
3. As noted above, there may be blank lines in the file.

Submitting this assignment

Electronic submissions on blackboard. Be sure to include information in the comments

- on what language you used (C, C++ or Java),
- build instructions and
- the system (osf1, zeus) where you want me to build/test your assignment.

For this and all other assignments, but sure you give me ALL files needed to build your executable. The TA will not test your program on any executables you include.