CSc 135 – Fall 2017
Programming Languages
Homework 1
due Saturday October 8 at 11:59:00pm

You are given the following grammar:

piece ::= {stmnt **;** } [laststmnt **;**] **Note ';' no longer optional**

block ::= piece

stmnt ::= assignst | whilst | ifst | forst

assignst ::= varlist **=** explst

whilst ::= **W** expr **D** block **E**

ifst ::= **I** expr **T** block [**S** block] **E**

forst ::= **P** varname **=** expr **,** expr [**,** expr] **D** block **E**

laststmnt ::= **R** [explst] | **K**

varlist ::= varname { **,** varname}

explst ::= expr **,** {expr **,**}

expr ::= term [binop expr] | unop expr

term ::= **N** | **F** | **V** | num | varname | **(** expr **)**

binop ::= **+** | **-** | ***** | **/** | **<** | **>** | **A** | **O** **Note ' before A removed**

unop ::= **-** | **&** | **#**

varname ::= letter { letter | digit }

num ::= digit { digit }

letter ::= **U** | **X** | **Y**

digit ::= **0** | **1** | **2** | **3** | **4** | **5**

The tokens (which are bolded in the grammar) are: **E F R W I D T S P K N U V X A O Y 0 1 2 3 4 5** and, in quotes, **; , = # & ( ) < > - + * /**
~~The terminal ',' has been quoted to distinguish it from the meta language symbol~~.
Non-terminals are shown as lowercase words.

Note that the following characters are NOT tokens (they are EBNF meta-symbols):

[ ] | { }

1. Compute the FIRST and FOLLOW for all the non-terminal in the above grammar.

2. Show that the grammar satisfies the two requirements for predictive parsing (it does, you just need to prove it). **Make sure that you use the supplement rules for EBNF grammars found in the notes and the example provided.**

3. Implement a recursive-descent recognizer.
   - Prompt the user for an input stream (e.g. a file).
   - The user enters an input stream of legal tokens, followed by a $.
   - You can assume:
     - the user enters no whitespace,
     - the user only enters legal tokens listed above,
     - the user enters one and only one $, at the end.
   - The start symbol is "piece" (as defined above)
   - Your program should output "**legal**" or "**errors found**" (not both!).
     You can report additional information as well, if you want.
     For example, knowing where your program finds an error could be helpful to assign partial credit if it's wrong.
   - Assume the input stream is the TOKEN stream. Assume that any whitespace has already been stripped out by the lexical scanner (i.e., each token is one character - lexical scanning has been completed)
   - Since the incoming token stream is terminated with a $, you will need to add the $ to the grammar and incorporate it in your answers to questions #1 and #2 above, where appropriate.
   - Use Java, C, or C++, or ask your instructor if you wish to use another language.
   - Limit your source code to ONE file.
   - Make sure your program works on ATHENA before submitting it.
   - INCLUDE INSTRUCTIONS FOR COMPILING AND RUNNING YOUR PROGRAM ON ATHENA IN A COMMENT BLOCK AT THE TOP OF YOUR PROGRAM. Also, explain any input formatting that your program requires of the user entry.

4. Collect the following submission materials into ONE folder:
   - your source code file
   - the FIRST and FOLLOW
   - a proof indicating that predictive parsing can be performed. (you can hand write it and scan it, or you can do it on the computer). Make sure that it is clear and readable.

Note: you may use the example parser as long as you acknowledge properly the source and adapt it to the assigned grammar (i.e. modify the comments appropriately). Obviously, it is not the only thing you have to modify.