CSc 135 -- Fall 2016
Programming Languages
Homework 1
due Friday October 7 8 at 11:59:00pm

```
block   ::= B {statemt} E [D]
 statemt ::= asignmt | ifstmt | while | inpout | block
 asignmt ::= A ident ~ exprsn
 ifstmt  ::= I comprsn T block [L block]
 while   ::= W comprsn block
 inpout  ::= iosym ident {, ident}
 comprsn ::= ( oprnd opratr oprnd )
 exprsn  ::= factor {sumop factor}
 factor  ::= oprnd {prodop oprnd}
 oprnd   ::= integer | ident | ( exprsn )
 ident   ::= letter {char}
 char    ::= letter | digit
 integer ::= digit {digit}
 iosym   ::= R | O
 opratr  ::= < | = | > | !
 sumop   ::= + | -
 prodop  ::= * | /
 letter  ::= X | Y | Z
 digit   ::= 0 | 1
```

The tokens are: B E D A ~ I T L W , ( ) R O < + > ! + - * / X Y Z 0 1
Nonterminals are shown as lowercase words.

Note that the following characters are NOT tokens (they are EBNF metasymbols): **| { }**

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 read the supplement regarding the rules for an EBNF grammar below.**

3. Implement a recursive-descent recognizer.
   - Prompt the user for an input stream.
   - 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 "block" (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 for me 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 and scanned, or done on computer. You may use syntax diagrams if you prefer.

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.

The Rules of Predictive Parsing for an EBNF Grammar

When using the EBNF notation the formal rules of recursive parsing need to be adapted.

1. For every production $A ::= \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots \mid \alpha_n$, we must have

   $$\text{FIRST} (\alpha_i) \cap \text{FIRST} (\alpha_j) = \emptyset \quad \text{for each pair } i, j, i \neq j$$

   This rule applies in a EBNF situation as well.

2. For every nonterminal A such that FIRST (A) contains λ, we must have

   $$\text{FIRST} (A) \cap \text{FOLLOW} (A) = \emptyset$$

   Since one of the effects of using EBNF is to eliminate most λ we need to modify this rule as follows: For every production containing an optional portion, such as:

   $$A ::= \beta \, [\, \alpha \,] \, \gamma$$

   we must have $\text{FIRST} (\alpha) \cap \text{FOLLOW} (\alpha) = \emptyset$

   In addition, for every production containing a repetition such as

   $$A ::= \alpha \, \{ \, \beta \, \}$$

   we must have $\text{FIRST} (\beta) \cap \text{FOLLOW} (\alpha) = \emptyset$