RecursiveParsing

Consider the following EBNF grammar for a very simple programming language:

function ::= B {statemt}

statemt ::= assnmt | ifstmt | loop | read | output | funcall

assnmt ::= ident ~ exprsn ;

ifstmt ::= I comprsn @ {statemt} [% {statemt}] &

loop ::= W comprsn L {statemt} T

read ::= R ident {, ident } ;

output ::= O ident {, ident } ;

funcall ::= C function E

comprsn ::= ( oprnd opratr oprnd )

exprsn ::= factor {+ factor}

factor ::= oprnd {\* oprnd}

oprnd ::= integer | ident | ( exprsn )

opratr ::= < | = | > | !

ident ::= letter {char}

char ::= letter | digit

integer ::= digit {digit}

letter ::= X | Y | Z

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

The tokens are: B I W L T R O C E X Y Z 0 1 2 3 4 5 6 7 ; ~ @ % & , ( ) + \* < = > !

Nonterminals are shown as lowercase words.

The following characters are NOT tokens (they are EBNF metasymbols): | { } [ ]

Note that parentheses are TOKENS, not EBNF metasymbols in this particular grammar.

1. Draw Syntax Diagrams for the above grammar.

2. Show that the grammar satisfies the two requirements for predictive parsing.

(it does, you just need to prove it).

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 \*function\* (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

- syntax diagrams (may be handwritten/scanned, or computer drawn)

- proof of predictive parsing (may be handwritten/scanned, or computer drawn)

5. ZIP the one folder into a single .zip file.

6. Submit your ONE .zip file to the dropbox in Canvas.