CPSCI 323
Programming Languages and Translation
Fall 2014

Project #2 - Parser
Due Wed 10/15

Write a syntax analyzer (Parser) for the Compiler language described below. The analyzer should be designed as a top-down recursive-descent parser.

Requirements:

The parser program should call the Lexer to get a lexeme/token pair. The Parser should then use this information to check the grammar up to that point. When the Parser needs more information, it should call the Lexer for another lexeme/token pair. Repeat this sequence until the entire input file has been processed or until the first grammar error occurs.

The parser should print all of the production rules that are used in the reductions by the parser.

Errors:

If a syntax error occurs, the parser should generate a meaningful error message and stop. If there are no errors, the parser should be able to parse the entire program.

Description of syntax for TitanF14 language

| | | |
|---|---|---|
| Program | => | **program** {Decl} {Func} **begin** { Stmt} **end .** |
| Decl | => | Type VarList |
| Type | => | **int** \| **real** \| **bool** |
| Varlist | => | ident { **,** ident } |
| Func | => | **function** ident **(** [ ParamList ] **) :** Type **;** { Decl } **begin** { Stmt } **end ;** |
| ParamList | => | Param { **,** Param} |
| Param | => | Type ident |
| Stmt | => | Assign \| Read \| Write \| If \| While |
| Assign | => | ident **<–** Expr **;** |
| Read | => | **read (** VarList **) ;** |
| Write | => | **write (** Expr **) ;** |
| If | => | **if (** Condition **) begin** { Stmt } **end** { **elsif (** Condition **) begin** { Stmt } **end** } [ **else begin** { Stmt } **end** ] |
| While | => | **while (** Condition **) begin** { Stmt } **end** |
| Condition | => | Expr RelOp Expr |
| RelOp | => | **>** \| **<** \| **>=** \| **<=** \| **=** \| **< >** |
| Expr | => | Expr **+** Term \| Expr **–** Term \| Term |
| Term | => | Term ***** Factor \| Term **/** Factor \| Factor |
| Factor | => | ident \| intValue \| realValue \| **true** \| **false** \| **(** Expr **)** \| FuncCall |
| FuncCall | => | ident **(** [ Expr { **,** Expr } ] **)** |