

# Software Design Document

Michael Farghali

February 15, 2016

## Introduction

This is a preliminary Software Design Document (SSD) for my mini-Pascal compiler that I will be working on over the next two semesters. When completed it should accept a text file which represents a mini pascal language. The text file should then be able to be converted to assembly language if its syntax is correct according to our production rules.

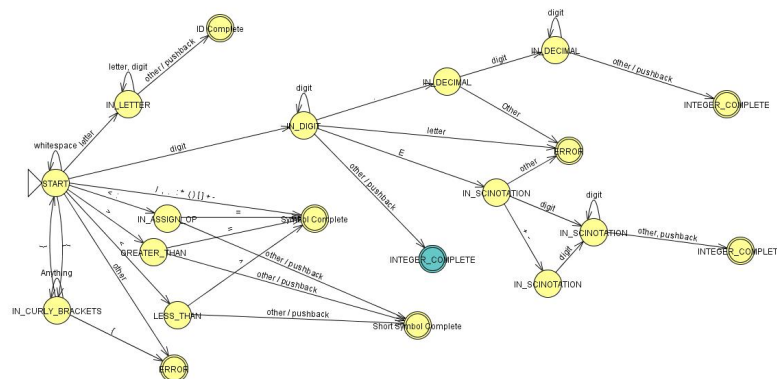
## Scanner

The Scanner class should read in a file and process it character by character. It is based on a Deterministic Finite Automata, Appendix A, and the given grammars. The Scanner reads in a file and attempts to match each string to a given keyword, symbol, or number. It returns a valid Token if the string is valid in the language or it will return false. See Appendix B for list of valid Tokens.

## Parser

The Parser class processes a text file token by token which are given by the Scanner class. It uses the grammar rules listed in Appendix C to match the tokens against expected tokens. It will eventually create a parse tree but for now it only checks that the production rules are followed.

## Appendix A Deterministic Finite Automata



## Appendix B List of Keywords

Symbols:  $\cdot, - + */()[] <<=>>=:$

Keywords: div, mod, and, program, id, var, array, of, num, integer, real, function, procedure, begin, end, if, then, else, while, do, not

## Appendix C Grammar Rules

## Production Rules

<i>program</i> ->	<b>program id ;</b> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i> <b>.</b>
<i>identifier_list</i> ->	<b>id</b>   <b>id , identifier_list</b>
<i>declarations</i> ->	<b>var identifier_list : type ; declarations</b>   $\lambda$
<i>type</i> ->	<i>standard_type</i>   <b>array [ num : num ] of standard_type</b>
<i>standard_type</i> ->	<b>integer</b>   <b>real</b>
<i>subprogram_declarations</i> ->	<i>subprogram_declaration ;</i> <i>subprogram_declarations</i>   $\lambda$
<i>subprogram_declaration</i> ->	<i>subprogram_head</i> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i>
<i>subprogram_head</i> ->	<b>function id arguments : standard_type ;</b>   <b>procedure id arguments ;</b>
<i>arguments</i> ->	<b>( parameter_list )</b>   $\lambda$
<i>parameter_list</i> ->	<i>identifier_list : type</i>   <i>identifier_list : type ; parameter_list</i>
<i>compound_statement</i> ->	<b>begin optional_statements end</b>
<i>optional_statements</i> ->	<i>statement_list</i>   $\lambda$

*statement\_list* ->     *statement* |  
                          *statement ; statement\_list*

*statement* ->         *variable assignop expression* |  
                      *procedure\_statement* |  
                      *compound\_statement* |  
                      **if** *expression* **then** *statement* **else** *statement* |  
                      **while** *expression* **do** *statement* |  
                      *read ( id )* |  
                      *write ( expression )*

*variable* ->         **id** |  
                      **id** [ *expression* ]

*procedure\_statement* ->         **id** |  
                                  **id** ( *expression\_list* )

*expression\_list* ->     *expression* |  
                          *expression , expression\_list*

*expression* ->         *simple\_expression* |  
                      *simple\_expression relop simple\_expression*

*simple\_expression* ->         *term simple\_part* |  
                                  *sign term simple\_part*

*simple\_part* ->         **addop** *term simple\_part* |  
                           $\lambda$

*term* ->             *factor term\_part*

*term\_part* ->         **mulop** *factor term\_part* |  
                           $\lambda$

*factor* ->             **id** |  
                      **id** [ *expression* ] |  
                      **id** ( *expression\_list* ) |  
                      **num** |  
                      ( *expression* ) |  
                      **not** *factor*

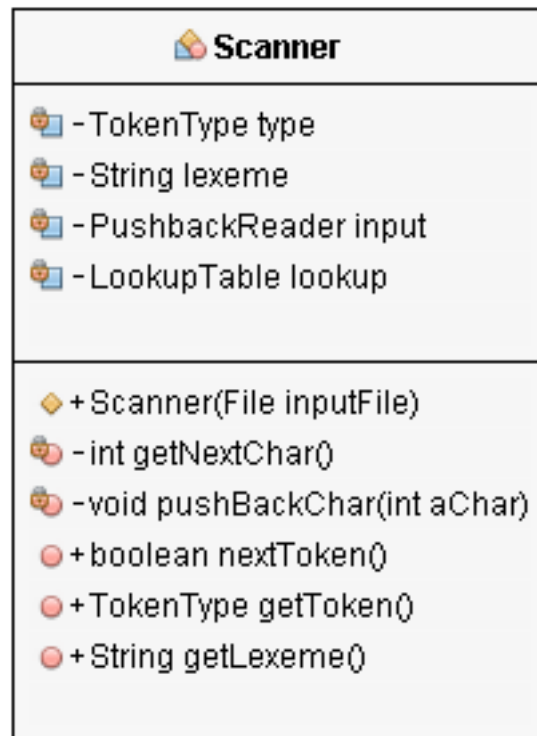
*sign* ->             **+** |  
                      **-**

## Lexical Conventions

1. Comments are surrounded by **{** and **}**. They may not contain a **{**. Comments may appear after any token.
2. Blanks between tokens are optional.
3. Token **id** for identifiers matches a letter followed by letter or digits:  
**letter** -> **[a-zA-Z]**  
**digit** -> **[0-9]**  
**id** -> **letter (letter | digit)\***



The **\*** indicates that the choice in the parentheses may be made as many times as you wish.











1. Token **num** matches numbers as follows:  
**digits** -> **digit digit\***  
**optional\_fraction** -> **. digits |  $\lambda$**   
**optional\_exponent** -> **(E (+ | - |  $\lambda$ ) digits) |  $\lambda$**   
**num** -> **digits optional\_fraction optional\_exponent**
2. Keywords are reserved.
3. The relational operators (**relop**'s) are:  
**=, <>, <, <=, >=, and >.**
4. The **addop**'s are **+, -, and or.**
5. The **mulop**'s are **\*, /, div, mod, and and.**
6. The lexeme for token **assignop** is **:=.**





## Appendix D UML Diagrams

## Parser

-  - Scanner scanner
-  - TokenType currentToken

-  +Parser(String filename)
-  +void match(TokenType expectedToken)
-  +void program()
-  +void identifier\_list()
-  +void declarations()
-  +void type()
-  +void standard\_type()
-  +void subprogram\_declarations()
-  +void subprogram\_declaration()
-  +void subprogram\_head()
-  +void arguments()
-  +void parameter\_list()
-  +void compound\_statement()
-  +void optional\_statements()
-  +void statement\_list()
-  +void statement()
-  +void variable()
-  +void expression()
-  +void simple\_expression()
-  +void simple\_part()
-  +void term()
-  +void term\_part()
-  +void factor()
-  +void expression\_list()
-  +void sign()
-  +void mulop()
-  +void error()

 <b>SymbolTable</b>
 ~Hashtable table
<ul style="list-style-type: none"> <li>+ boolean addProgramName(String name)</li> <li>+ boolean addFunctionName(String name)</li> <li>+ boolean addProcName(String name)</li> <li>+ boolean addVarName(String name)</li> <li>+ boolean addArrayName(String name)</li> <li>+ boolean isProgName(String name)</li> <li>+ boolean isFuncName(String name)</li> <li>+ boolean isProcName(String name)</li> <li>+ boolean isVarName(String name)</li> <li>+ boolean isArrayName(String name)</li> </ul>