SOFTWARE DESIGN DOCUMENT

MICHAEL FARGHALI

1. Introduction

This is a preliminary Software Design Document (SSD) for my mini-Pascal complier 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.

2. Scanner

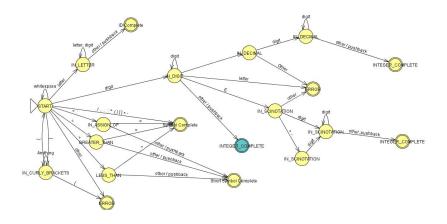
The Scanner class should read in a file and process it character by character. It is based on a Deterministic Finite Automata, Appendix A, and the given grammars. The Scanner reads in a file and attempts to match each string to a given keyword, symbol, or number. It returns a valid Token if the string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

3. 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.

- 4. Symbol Table
- 5. Syntax Tree
- 6. Semantic Analysis
- 7. Code Generator
 - 8. Conclusion

APPENDIX APPENDIX A DETERMINISTIC FINITE AUTOMATA



APPENDIX APPENDIX B LIST OF KEYWORDS

Symbols: ., -+*/()[] <<=>>=::=; Keywords: div, mod, and, program, id, var, array, of, num, integer, real, function, procedure, begin, end, if, then, else, while, do, not

APPENDIX APPENDIX C GRAMMAR RULES

CSC 450/451 The Grammar

Production Rules

```
program ->
                     program id;
                     declarations
                     subprogram_declarations
                     compound_statement
identifier_list ->
                     id
                     id , identifier_list
declarations ->
                     var identifier_list : type ; declarations |
type ->
                     standard_type |
                     array [ num : num ] of standard_type
standard_type ->
                     integer |
                     real
subprogram_declarations ->
                               subprogram_declaration ;
                               subprogram_declarations |
                               λ
subprogram_declaration ->
                               subprogram_head
                                declarations
                               subprogram_declarations
                                compound_statement
subprogram_head -> function id arguments : standard_type ; |
                     procedure id arguments;
arguments ->
                     ( parameter_list ) |
parameter_list ->
                     identifier_list : type |
                     identifier_list : type ; parameter_list
                                begin optional_statements end
compound_statement ->
optional_statements ->
                               statement_list |
```

```
CSC 450/451
```

The Grammar

```
statement_list ->
                     statement |
                     statement ; statement_list
                     variable assignop expression |
statement ->
                     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 ( expression_list )
expression_list ->
                     expression |
                     expression , expression_list
expression ->
                     simple_expression |
                     simple_expression relop simple_expression
                                term simple_part |
simple_expression ->
                                sign term simple_part
simple_part ->
                     addop term simple_part |
                     λ
term ->
                     factor term_part
                     mulop factor term_part |
term_part ->
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:

- 4. The **addop**'s are +, -, and **or**.
- 5. The mulop's are *, /, div, mod, and and.
- 6. The lexeme for token **assignop** is **:=**.

Scanner Scanner Scan

- 획 -TokenType type
- 획 String lexeme
- 획 PushbackReader input
- 획 LookupTable lookup
- +Scanner(File inputFile)
- -int getNextChar()
- -void pushBackChar(int aChar)
- +boolean nextToken()
- +TokenType getToken()
- +String getLexeme()

APPENDIX APPENDIX D UML DIAGRAMS

📤 Parser

- ካ Scanner scanner
- 획 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()

📤 SymbolTable

- 🖶 ~ Hashtable table
- + boolean addProgramName(String name)
- + boolean addFunctionName(String name)
- + boolean addProcName(String name)
- + boolean addVarName(String name)
- + boolean addArrayName(String name)
- + boolean isProgName(String name)
- + boolean isFuncName(String name)
- + boolean isProcName(String name)
- + boolean isVarName(String name)
- + boolean isArrayName(String name)