Software Design Document

Michael Farghali

December 18, 2015

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.

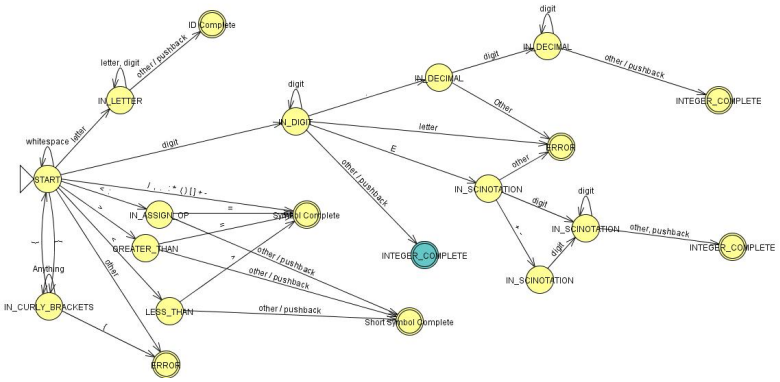
Scanner

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: $., -, +, *, /, (), [], <, <=, >, >=, :, :=, ;$
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

program -> **program id ;**
declarations
subprogram_declarations
compound_statement
.

identifier_list -> **id** |
id , *identifier_list*

declarations -> **var** *identifier_list* **:** *type* **;** *declarations* |
λ

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* **)** |
λ

parameter_list -> *identifier_list* **:** *type* |
identifier_list **:** *type* **;** *parameter_list*

compound_statement -> **begin** *optional_statements* **end**

optional_statements -> *statement_list* |
λ

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* |
λ

term -> *factor term_part*

term_part -> **mulop** *factor term_part* |
λ

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 | λ
   optional_exponent -> (E (+ | - | λ) digits) | λ
   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 **:=**.