Pascal to MIPs Compiler

Joseph Miller

February 17, 2018

1 Overview

This program will be able to compile Pascal code into MIPs assembly code. The compiler will be made divided into 4 major sections:

- Scanner

- Parser

- Semantic Analysis

- Code Generation

2 Design

2.1 Scanner

This program will be able to take in a file with Pascal code and output each token in the order that it was written. Each of the following keywords and symbols will be identified as tokens, as well as variable names (IDs).

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

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

This project is made of 3 Java/Class files:

- **Scanner**: the DFA scanner made by JFlex to analyze the Paacal code and identify token types.

- **Token**: the token object creator, this file also takes in what kind of token it is and adds a more specific distinct type by using a look-up table (switch function / if-else function). Each Token object contains the lexeme and the type.

```
Token: PROGRAM, Program
Token: ID, Lesson1
Token: ERROR, _
Token: ID, Program3
Token: SEMI, ;
Token: VAR, Var
Token: ID, Num1
Token: COMMA, ,
Token: ID, Num2
Token: COMMA, ,
Token: ID, Sum
Token: COLON, :
Token: INTEGER, Integer
Token: SEMI, ;
Token: BEGIN, Begin
Token: COMMENT, {no semicolon}
Token: POSEXP, 12686E20938

The List of Token Arrays:
[Token: PROGRAM, Program, Token: ID, Lesson1, Token: ERROR, _, Token: ID, Program3, Token: SEMI, ;
```

Figure 1: Token output to console

– **TokenType**: creates the specifications for a the TokenType attribute of the Token attribute

Illegal symbols and sequences not accepted by our simple pascal compiler will have the TokenType: ERROR. A visual of the tokens created can be seen in Figure 1, one of which produces an error.

2.2   Parser

The parser takes the tokens from the scanner and verifies whether it is correctly formatted Pascal code. It is also the Parsers job to create a syntax tree which has the correct, legal order of tokens, and provide feedback if an error does occur.

This project is made of 3 Java/Class files, only one of which (Parser) relates explicitly to the parser:

– **Parser**: The parser class takes the series of tokens created by the scanner and used functions for each production rule to either accept the Pascal code or reject it. If an error results it will also spot the location and reason for the error and end a message to the console informing the coder of the error. The parser adds identifiers to the symbol table with their proper type, and also uses the symbol table to resolves the ambiguity in the "Statement" production rule, where it calls either the "Procedure statement" or "Variable " production rule respectively.

– **SymbolTable**: The symbol table is independent of the parser but is put in the parser package since the parser using it most and its not a large

enough part of the compiler to be given its own package. The symbol table is used to take in all identifiers used in the code and associate them with the types of identifier (or symbol) they are. Other parts of the compilers will need to reference the symbol table.

SymbolType: This is an enum file for SymbolTable and has no relation to the parser itself. The five types of identifiers are: PROGRAMTYPE, VARIABLETYPE, FUNCTIONTYPE, PROCEDURETYPE, and AR-RAYTYPE

2.3   Semantic Analysis

Not written

2.4   Code Generation

Not written

3   Change Log

- 18 January 2019 - Miller - Created

- 3 Februrary 2018 - Miller - Added parser section

- 11 February 2018 - Miller - Added symbol table info under parser section