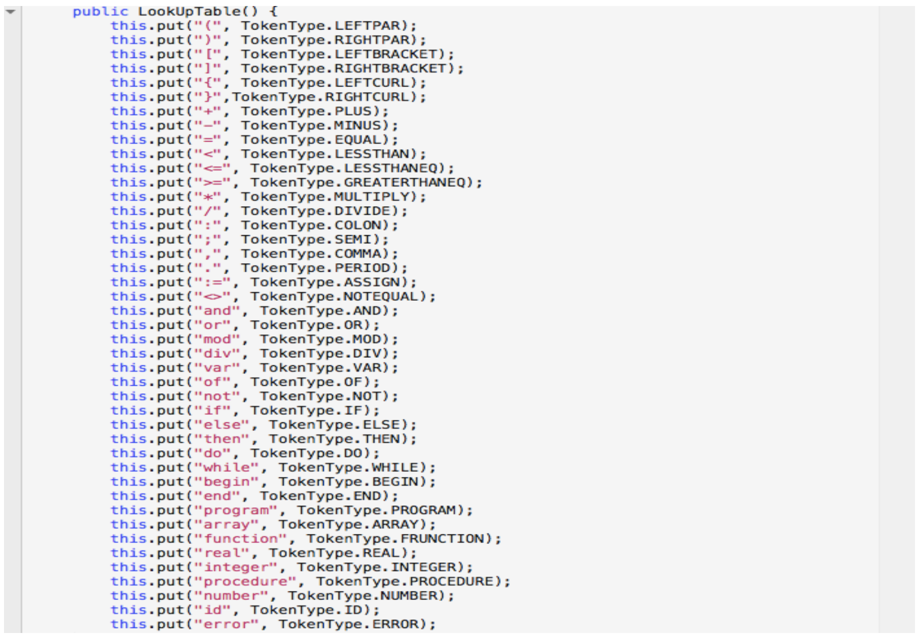
Software Design

Overview:

In this project, there will be a compiler. In order to build the compiler, there are few components that go into it. These components include building a scanner to scan strings or text, parser which will contain functions based on pascal grammar provided, and symbol table which will store information about identifiers found in a pascal program.

Scanner:

To create this scanner, the program will use Java and Jflex. It will use Jflex to create the scanner and compile that Scanner.java file. When the Scanner is fed a series of characters, then the scanner have to convert those character into tokens plus recognize if those tokens are part of the correct tokens of the pascal language.The application will be built from five modules ScannerMain, Token, TokenType, LookUpTable, and Scanner class. The ScannerMain class will be the default class which will contain the main method that run the application. The Token class will be defining the token object. Each token will consist of two sub-definition the content and the type. The TokenType, will be the enumeration class which will store the type for each reserve symbol and keyword. In LookupTable class I will be using a java predefined data structure which is HashMap to store the values for the Mini Pascal keywords



```
    public LookUpTable() {
        this.put("(", TokenType.LEFTPAR);
        this.put(")", TokenType.RIGHTPAR);
        this.put("[", TokenType.LEFTBRACKET);
        this.put("]", TokenType.RIGHTBRACKET);
        this.put("{", TokenType.LEFTCURL);
        this.put("}",TokenType.RIGHTCURL);
        this.put("+", TokenType.PLUS);
        this.put("-", TokenType.MINUS);
        this.put("=", TokenType.EQUAL);
        this.put("<", TokenType.LESSTHAN);
        this.put("<=", TokenType.LESSTHANEQ);
        this.put(">=", TokenType.GREATERTHANEQ);
        this.put("*", TokenType.MULTIPLY);
        this.put("/", TokenType.DIVIDE);
        this.put(":", TokenType.COLON);
        this.put(";", TokenType.SEMI);
        this.put(",", TokenType.COMMA);
        this.put(".", TokenType.PERIOD);
        this.put(":=", TokenType.ASSIGN);
        this.put("<>", TokenType.NOTEQUAL);
        this.put("and", TokenType.AND);
        this.put("or", TokenType.OR);
        this.put("mod", TokenType.MOD);
        this.put("div", TokenType.DIV);
        this.put("var", TokenType.VAR);
        this.put("of", TokenType.OF);
        this.put("not", TokenType.NOT);
        this.put("if", TokenType.IF);
        this.put("else", TokenType.ELSE);
        this.put("then", TokenType.THEN);
        this.put("do", TokenType.DO);
        this.put("while", TokenType.WHILE);
        this.put("begin", TokenType.BEGIN);
        this.put("end", TokenType.END);
        this.put("program", TokenType.PROGRAM);
        this.put("array", TokenType.ARRAY);
        this.put("function", TokenType.FRUNCTION);
        this.put("real", TokenType.REAL);
        this.put("integer", TokenType.INTEGER);
        this.put("procedure", TokenType.PROCEDURE);
        this.put("number", TokenType.NUMBER);
        this.put("id", TokenType.ID);
        this.put("error", TokenType.ERROR);
```

Recognizer:

The parser package contains a recognizer class for Pascal based on the grammar provided. This class contains methods including, program, addop, statement, mulop, declarations, relop. The package also contains a test file called RecognizerTest which has a Junit testing cases for statement, simple_expression,factor, program, declarations, subprogram_declarations classes.

Parser:

The symbol table main functionality is to store information about identifiers found in a pascal program. Hence, each entry for an identifier in the Symbol Table will contain appropriate information about the identifier: its lexeme, the kind of identifier (program, variable, array, or function) and other information appropriate to the kind of identifier such as, a variable should have its type information. The Symbol Table module will be implemented by using two classes a symbol table class which will define the structure of the table such as, adding identifier to the table and verifying if the identifier exists in the table and a Kind class which will be an enum for defining the kind of the identifier.

Symbol Table integration I will integrate my Symbol Table with the Recognizer by having it add symbols to the table when declarations are made. I will use the symbol table to resolve the ambiguity in the ’statement()’ function in the recognizer class which I couldn’t solve previously without the help of the symbol table. This should allow me to now properly decide between an assignment statement and a procedure state-ment. In addition to that, I will build a CompilerMain class; This class will have a main which runs the parser as a compiler: taking in a filename and outputting a yes or no along with printing out the contents of the symbol table.

Syntax Tree

In this iteration I will convert my recognizer into a parser by having my top level program() function return a ProgramNode object as the root of a syntax tree. I will be doing that by building a syntax tree package, most of the classes are provided by my professor, but I will be adding more to complete the the syntax tree. Then I will refactor my recognizer methods to return nodes instead of been void. Finally, I will be modifying my CompilerMain class to write the indentedToString() of the program to a file, and write out the contents of the symbol table to a separate file.

Semantic Analysis.

In this iteration, I will implement in the Parser Module a three kinds of Semantic Analysis. My implementation will include the following: I will make sure all variables are declared before they are used, I will assign a type, integer or real, to each expression. The type of any expression node in the tree will be printed in the tree's indentedToString and I will make sure that types match across assignment. So, each of these errors can be noticed by the compiler without halting the compilation process.

Code Generation.

This code generation will be a standalone module in a separate package. I will create a new package and name it codegeneration. The Code Generation will takes the syn- tax tree as its input, and returns a string with the MIPS assembly language code as its output.

Product:

The final version of this project includes the Jar file which will contain a .jar executable in a new product folder. Alongside the User Manual to use the final product.

Change Log:

28 January 2019: -Ahmed: Created

28 January 2019 - Ahmed: Modified Overview and Scanner section

14 February 2019- Ahmed: Added Recognizer section

14 February 2019 – Ahmed: Added the Symbol Table section

4 March 2019 - Ahmed : Added the Symbol Table integration section

10 April 2019 - Ahmed : Added the syntax tree section

20 April 2019 - Ahmed : Added the analyzer section

21 April 2019 - Ahmed : Added the codegeneration section

30 April 2019 - Ahmed : Added the product section