ENGR3690U Programming Languages and Compilers

Assignment 3

***Minifun* Language**

A Compiler

For this assignment, add code generation to your compiler. When the finished compiler is executed, it should open a *Minifun* source program file, compile it, and produce a **.asm** assembler text file containing a MIPS assembly language version of the *Minifun* program.

Most of the work in this assignment consists of adding new code to the existing parser. While you might find it useful to create some extra functions or data structures, the bulk of the changes will be additions to existing parser functions.

The easiest way to run the MIPS code is to use SPIM. It is a self-contained simulator that runs MIPS32 programs. It reads and executes assembly language programs written for this processor.

You must implement function and variable definitions (define), conditional statement (if or cond), function calls, basic relational operators (==,>,<, <=, >=) and basic arithmetical operators (+,-,*,/).

You must hand in to Blackboard a .zip archive containing your source code. Your main program must be called `Mfun`. And Submit to Blackboard a PDF document of no more than five pages explaining the design of this phase of your compiler. The document should be organized to enable someone unfamiliar with your code to understand the structure of (this phase of) your compiler. In the document, discuss challenges that you encountered and how you tried to overcome them in your design and implementation. Also explain the testing that you did before submitting it.

NOTE: If you didn't complete assignment 1 you can use the parser `Minifun.jar`. This is not a complete parser but you can use it for this assignment. This is an example of a `Mfun` class and the `main` method.

```
import java.io.File;
import java.util.ArrayList;
import java.util.Scanner;

// The package in the jar file
import Minifun.Parser;

public class Mfun {

      public static void main(String args[]) throws Exception{
            if ( args.length < 1 ) {
                  System.out.println("Please pass in the filename.");
                  System.exit(1);
            }
            // Loading the file as a string
             String myProgram = loadFile(args[0]);
            // The parser returns a Java Collection of type ArrayList
            // It can store Java Objects. In this case, it stores ArrayLists
```

```
            // In other words, it is an ArrayList of ArrayLists
            // This is the representation of an AST
            ArrayList<Object> ast= Parser.parse(myProgram);
            // Print the AST
            System.out.println(ast);
      }

      public static String loadFile(String name) throws Exception {
            Scanner s = new Scanner(new File(name));
            String file = "";
            while (s.hasNext()) {
                  file += s.nextLine() + "\n";
            }
            s.close();
            return file;
      }
}
```

If you want to use this parser you have to load the jar file into the environment variable CLASSPATH. You can do it from the command line using the -cp or -classpath option of the JVM.

```
c:\java -cp Minifun.jar;. <YourProgram> <testfile>
```

If your testfile is named `program1.mfun` and it has the following code

```
(define x 5)
(if (< x 6) 1 0)
```

And you run the interpreter with the command

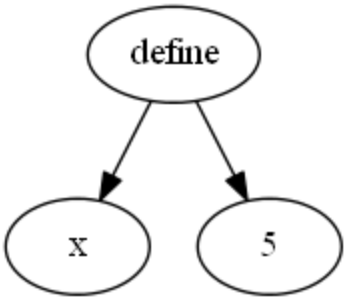
```
c:\java -cp Minifun.jar;. Mfun program1.mfun
```

The output is

```
Parse Completed
[[define, x, 5], [if, [<, x, 6], 1, 0]]
```

The program is represented as an array. Each element in the array is a s-expression <s-exp> (see Minifun grammar) that it is represented as an array as well. The first element represents the root of the tree and it is the operator. The next elements represent the children of the root and are the operands.

`[define, x, 5]` represents the AST



`[if, [<, x, 6], 1, 0]` represents the AST