Assignment 1

Michael Hunsinger

September 11, 2014

This file contains all documentation for both parts of the first homework assignment. The first section contains all information pertaining to setting up and running the scanner operation on a file. The second section contains all information pertaining to definitions for the extended Micro language.

1 Scanner Documentation

This implentation of the scanner uses Google's new language Go. There are instructions on how to setup Go, a description of the file structure, and how to compile and run the program.

1.1 Install Go

Download the appropriate installation from Google's Go website, `http://golang.org/doc/install`, there is additional documentation located on website as well.

1.2 Go's Workspace

Extract files from the tarball into the desired location. Inside the root folder you will find four directories

- `bin` compiled executables, along with sample micro program files
- `doc` documentation
- `pkg` package objects (the compiler package is located in here)
- `src` source files
  - `compiler` source files pertaining to the compiler package
  - `main` source files pertaining to the main package (the driver file)

We must also setup the `GOPATH` to ensure proper compilation of the files. Follow the steps below to set `GOPATH` in a *unix environment.

```
$ cd ../01
$ export GOPATH=$HOME/your/path/here/01
```

1.3 Compiling Source Files

There are two steps to compile and the executable; building the compiler package and then build the executable.

```
$ cd ../01
$ go build compiler
$ go install main
```

Now there is an executable in the `bin` folder.

1.4 Running the Program

You can run the executable that was compiled. Ensure you are in the directory where the `sample.micro` file is located.

```
$ cd ../01/bin
$ ./main
```

This will run scan the `sample.micro` file. There is also a `sample2.micro` file in the `bin` folder that uses some of the tokens found in the extended Micro language. If you wish to scan this file, you will need to change file name in `../src/main/main.go` on line 18.

1.5 Sample Input and Output

Input file `sample.micro`

```
BEGIN --SOMETHING UNUSUAL
   READ(A1, New_A, D, B);
   C:= A1 +(New_A - D) - 75;
   New_C:=((B - (7) + (C + D))) - (3 - A1); -- STUPID FORMULA
   WRITE(C, A1 + New_C);
   -- WHAT ABOUT := B + D;
END
```

Output from `sample.micro`

```
BeginSym ReadSym LParen Id Comma Id Comma Id Comma Id RParen SemiColon Id
AssignOp Id PlusOp LParen Id MinusOp Id RParen MinusOp IntLiteral SemiColon
Id AssignOp LParen LParen Id MinusOp LParen IntLiteral RParen PlusOp LParen
Id PlusOp Id RParen RParen RParen MinusOp LParen IntLiteral MinusOp Id RParen
SemiColon WriteSym LParen Id Comma Id PlusOp Id RParen SemiColon EndSym
```

2 Extended Micro Grammer

Write an extended Micro Grammer to include the equality and exponentiation operators. The following grammers are meant to supplement the existing grammer explained in class.
We will need the following definitions that were already defined in class.

```
<expression> -> <primary> | <primary> <add op> <expression>
<primary>    -> LParen <expression> RParen
<primary>    -> Id
<primary>    -> IntLiteral
<add op>     -> PlusOp | MinusOp
```

2.1 Exponential

The exponentiation should have higher precedence than `PlusOp` and `MinusOp` and should group from right to left, that is, `A**B**C = A**(B**C)`.

```
<expression> -> <term> | <expression> PlusOp <term> |
                <expression> MinusOp <term>
<term>       -> <factor> | <term> ExpOp <factor>
<factor>     -> Id | IntLiteral
```

2.2 Equality

The equality operator should have lower precednce than plus and minus, and should not group, that is, `(A = B = C)` is not allowed.

```
<bool expression> -> <expression> <bool> <expression>
<bool>            -> EqualityOp
```