COMP 520 Compiler Design
Individual Assignment #1
Language Specifications

Overview:

Given the example program sqrt.min and the discussion in class, your first assignment is to implement the scanner and parser for the `minilang` programming language. Note that as you implement your compiler, some decisions may be more difficult to implement than you first thought - or we might have missed out a key detail you need. In these cases, bring them up in class and we can discuss possible changes.

sqrt.min

```
// Approximate the square root of x.

var x: float;
var guess: float;
var quot: float;
var iter: int;

read x;
guess = 1;
iter = 10;

while iter do
    quot = x / guess;
    guess = 0.5 * (guess + quot);
    iter = iter - 1;
done

print guess;
print guess * guess;
```

Specifications

A program in `minilang` consists of a list of variable declarations followed by a list of statements. Unlike many programming languages, declarations and statements may not be interwoven. Note though, the list of declarations or statements may be empty - that is, the empty program, programs with just declarations, and programs with just statements are valid.

General

Reserved words in the language are as follows:

```
var         while       do
done        if          then
else        endif
float       int         string
print       read
```

They may not be used in variable declarations as identifiers, and may not be reassigned (i.e. `while=3` is invalid). Keywords are case sensitive.

Comments in this language are single line, and start with the double forward slash. There are no block comments.

```
// this is a comment
```

Unidentified symbols (those not valid in any token) must cause the program to be rejected.

UPDATE (Jan 18th): For the purpose of this assignment, we will assume the only valid whitespace characters are `\n` `\r` `\t` and space.

Declarations

A variable declaration has the following form, where variables can either be of type `int`, `float`, or `string`. Note the terminating semicolon.

```
var a: float;
```

Types in this language are defined as:

- **int**: an integer with no leading zero (unless of course it is zero)
- **float**: a floating point (decimal) number with digits on both sides of the decimal. (i.e. `3.` or `.3` are not valid floating point numbers).

  UPDATE (Jan 18th): Floating points may not contain leading zeros on the LHS of the decimal (unless it is zero), however they may have trailing zeros. In other words, the LHS of the decimal must be a valid integer according to our specification.

  - **leading:** 000.01 and 01.3 are INVALID, while 0.01 is VALID
  - **trailing:** 0.00000, 0.01000, etc... all VALID

- **string**: a string of characters surrounded by quotation marks `"..."`. Note that in this language we will accept escaped quotation marks within the string. This means that `"derp\"derp"` should be treated as a valid string.

  UPDATE (Jan 18-20th): String literals may contain the following characters:

  - **Spaces**
  - **Alphanumerics:** `[a-zA-Z0-9]`
  - **Symbols:** `~ # $ % ^ & * - + / ‘ (i.e. backtick) < > = _ |`
    `’ (i.e. singlequote) ; : { } [ ] ( )`
  - **Escape sequences:** `\a \b \f \n \r \t \v \" \\`
    All other escape sequences are invalid
  - **Optional symbols:** You may optionally support the following symbols within strings (they were omitted by accident): `@ ! ? . ,`

Note that numeric literals do not have a sign.

An identifier must start with either: a letter (uppercase or lowercase) or an underscore. Subsequent characters can either be letters, underscores, or digits. Identifiers are case sensitive (this will matter for later phases of the compiler).

Statements

Statements in the language can either be one of the following. Note that the first 3 (read, print and assignment) are all terminated by semicolons. Statement lists (`<stmts>`) are a list of zero or more statements.

- Read into a variable

  ```
  read x;
  ```

- Print an expression

  ```
  print x * x;
  ```

- Assignment into a variable

  ```
  guess = 1;
  ```

- If statement, with optional else branch

```
if <expression> then
    <stmts>
[else
    <stmts>]
endif
```

- While loop

```
while <expression> do
    <stmts>
done
```

Expressions

Expressions follow the typical math notation found in modern programming languages, and consist of:

- Binary operations: +, -, *, /
- Unary operations: - (i.e. `-3` is a valid expression)
- Matched parentheses
- Left associativity
- Typical math precedence of operations