

COMP 520 Compiler Design

Individual Assignment #1

Language Specifications

Updated January 17th

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 design your compiler, some language features may be more difficult to support than you first thought - or we might have missed out on a key detail that 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 = 0.0;
var guess: float = 1.0;
var quot: float = 0.0;
var iter: int = 10;

read x;

while iter {
    quot = x / guess;
    guess = 0.5 * (guess + quot);
    iter = iter - 1;
}

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. However, 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, and may not be used as identifiers. Reserved words are case-sensitive.

<code>var</code>	<code>while</code>	<code>boolean</code>
<code>float</code>	<code>if</code>	<code>TRUE</code>
<code>int</code>	<code>print</code>	<code>FALSE</code>
<code>string</code>	<code>read</code>	

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. Whitespace (spaces, tabs, newlines) is ignored.

Declarations

A variable declaration consists of the keyword `var`, an identifier, the variable type, and an initial value (may be compound). The identifier and type are separated by a colon, and the declaration ends with a semicolon.

```
var a: float = 0.0;
```

Types supported by MiniLang are:

- **boolean**: either `TRUE` or `FALSE`
- **int**: integer with no leading zero (unless it is zero)
- **float**: floating point number with digits on *both* sides of the decimal. (i.e. `3.` or `.3` are not valid floating point numbers). They may not contain any leading zeros on the LHS of the decimal, but may have any number of trailing zeros. In other words, the LHS of the decimal must be a valid integer according to our specification.
 - **leading**: `0.01` is OK, while `000.01` and `01.3` are all errors
 - **trailing**: `0.00000`, `0.01000`, etc... all OK
- **string**: a sequence of characters surrounded by quotation marks (i.e. `"..."`). 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)

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.

Note that numeric literals (integers and floating point numbers) do not have a sign - the sign is part of a unary expression.

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

- Read into a variable

```
read <variable>;
```

- Print an expression

```
print <expression>;
```

- Assignment into a variable

```
<variable> = 1;
```

- If statement, with optional else branch

```
if <expression> {  
    <stmts>  
} [else {  
    <stmts>  
}]
```

- While loop

```
while <expression> {  
    <stmts>  
}
```

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 for all operators
- Precedence (highest to lowest)
 - Unary operators
 - Typical math precedence of operations for math operators
 - Relational operators (`==` and `!=`)
 - Logical and (`&&`)
 - Logical or (`||`)