README.md

SimpL: The Simple Programming Language ©

Setup

Building SimpL Source

To build SimpL, navigate to the SimpL directory and run:

```
./build.sh
```

The above will generate antlr sources, compile all java into an `out` folder, and run various tests. The project is built with the only dependencies being the JVM assembler `Jasmin` and the parser generator `Antlr`.

Compiling a SimpL program

To compile a `.simpl` file, navigate to the SimpL directory and run:

```
./simplc.sh <source_filepath>
```

The above will generate .j (JVM assembly) and a .class (JVM bytecode) files of the same name. The output location defaults to the parent directory of the given sourcefile, but can be specified with the command line option `-d` (to be added in the near future).

Executing a SimpL program

To execute a program, run it by specifying its compiled class filepath by using:

```
./simplr.sh <class_filepath>
```

The above will run the compiled SimpL program with any output printed to console.

Troubleshooting and Tips

If executing `java` or `javac` directly, ensure classpath is set correctly by using:

```
export CLASSPATH="out:<jasmin2.4-jar-path>:<antlr4.7-jar-path>:$CLASSPATH"
```

If a `permission denied` error occurred while running a script, grant access by using:

```
chmod +x ./<script_filepath>.sh
```

If newline issues occur after modifying the scripts on windows, remove excess new line characters by using:

```
sed -i 's/\r$//' ./<script_filepath>
```

Delete any generated jasmin and class files by using:

```
find <output_directorypath> -maxdepth 1 -regex ".*\.\(j\|class\)" -type f -delete
```

Syntax

Overview

Only single programs are supported, of which consistent of multiple statements, each of which are terminated with a line break.

Statements include function definition, declaration, assignment, standalone expression, conditional, and while loop.

Blocks consist of a `{ <0 or more statements> }`, with the braces on their own lines. Blocks are expected following conditionals, function signatures, and loops. Again, curly braces MUST be on their own separate lines - this means egyptian/K & R style braces are NOT supported, sorry!

Expressions are any mix of enclosed parentheses, literals, identifiers, function calls, and operations.

Datatypes currently include `Number` and `Text`. The more elaborate constructs `Map`, `List`, `Struct`, `Func` are planned for the future.

Operators

Support for parenthetical, arithmetic, boolean, comparison operations

```
-------- Operator Precedence (HI to LO) --------
| order |  operator  |         meaning          |
|   0   |     ()     |       parenthesis        |
|   1   |     ^      |      exponentiation      |
|   2   |    * /     |   multiply and divide    |
|   3   |    + -     |     add and subtract     |
|   4   | < > <= >=  |       comparison         |
|   5   |   == !=    |        equality          |
|   6   |    not     |    logical negation      |
|   7   |    and     |   logical conjunction    |
|   8   |    or      |   logical disjunction    |
-------------------------------------------------
```

Conditionals

Syntax for conditionals is as follows:

```
if <expression>
{
    <0 or more statements>
}
elif <expression>
{
    <0 or more statements>
}
else
{

    <0 or more statements>
}
```