

Branch: complete-docum... ▾

CS153-Assignments / HW06 / docs / Language Overview.md

Find file

Copy path

JeffPersons Docs: Simplified naming d0e59a1 2 minutes ago

1 contributor

187 lines (157 sloc) 5.23 KB

# SimpL ©: Language Overview ▾

## Motivation and Summary ▾

SimpL is a *simple* language for *simple* people, who are tired of overbearing syntax or are simply just learning to program. Elements of SimpL's syntax were influenced by Python, with all statements ending in a line break instead of the more traditional semicolon.

Unlike Python, however, SimpL is a statically typed, dynamically scoped language that compiles and runs on the JVM. It was built using `Jasmin` (JVM assembly) for intermediate code generation, and `ANTLR4` for parser source code generation.

SimpL programs are single files ending with a `.simp1` extension, that when compiled generates respective `.j` (Jasmin) and `.class` (bytecode) files.

## General Syntactic Elements ▾

Only *single programs* are supported, of which consist of multiple statements, each of which are terminated with a line break. Any curly braces must be on their own separate lines -- no egyptian style braces, sorry!

### Expressions ▾

An `expression` is any mix of parenthetical expressions, datatype literals, identifiers, function calls, and operations.

### Statements ▾

A `name` is an underscore or letter followed by any combination of underscores, letters, or numbers. This applies to both function and variable names.

A `statement` is a declaration, assignment, standalone expression, function definition, conditional, and while loop. Of those, the last three are multiline statements with the following syntax:

```
<keyword> <expression>
{
    <0 or more statements>
    <optional return statement>
}
```

Note that the above braces and their enclosed statements form a block. Thus, blocks follow all function signatures, as well as all while, if, else if, else conditions.

### Comments ▾

Any `comment` is ignored by the SimpL parser, along with tabs and excess newlines.

Single line comments have the following syntax:

```
## <some single line comment>
```

Multiline comments have the following syntax:

```
##
<some multiline comment>
##
```

## Datatypes

### Number

An integer or decimal number (internally stored as a *double*), with the following syntax for literals:

```
<1 or more digits>
<1 or more digits>.<1 or more digits>
```

### Text

A character sequence (internally stored as a *String*), with the following syntax for literals:

```
'<0 more characters>'
```

Note that quotes and slashes can be escaped with a slash (eg ' \').

### Boolean

A true or false token (internally stored as a *boolean*), with the following syntax for literals:

```
True
False
```

## Operators

Support for parenthetical, arithmetic, boolean, comparison operations, ordered from high to low precedence:

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

Equality, parenthesis, and assignment operators apply to *any* expression and *any* datatype. Comparison and arithmetic operators apply only to `Numbers`. Logical operators apply only to `Booleans`.

## Variables, Declarations, and Assignments

Support for variables is restricted to datatypes `Number` and `Text`. Variables can be declared with or without an initial value, as follows:

```
<datatype> <name> = <expression>
<datatype> <name>
```

## Control Flow

Syntax for conditionals is as follows:

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

Syntax for loops is as follows:

```
while <expression>
{
    <0 or more statements>
}
```

## Functions

### Function Definitions

A function definition consisting of a signature and a body, with the following syntax:

```
<void or datatype> <name>(<parameter list>)
{
    <0 or more statements>
}
```

If the return type is void, then nothing is returned within the function. Above, `<parameter list>` is a comma-separated list of one or more `<datatype> <name>`.

### Function Calls

A function defined within the current scope is invoked, with the following syntax:

```
<name>(<argument list>)
```

Above, `<argument list>` is defined as a comma-separated list of one or more `<name>`. When calling functions, arguments are passed by value, and must correspond to the parameters specified in the function definition.

### Builtin Functions

Predefined functions `print` and `println` are available, and take any number of arguments. Just like in Java, `print` writes the expression to standard out,

## Error Handling



Errors and exceptions encountered during compilation are written to standard error.

### Type Checking



Errors are raised to ensure operators are between appropriate type(s), as specified in the *Operators* section from before.

### Error Recovery



Like most compilers, `simpl` will continue parsing the rest of the file even if an error occurs.