better call **SOL**

SHAPE ORIENTED LANGUAGE REFERENCE MANUAL

Aditya Naraynamoorthy
an2753

Erik Dyer
ead2174

Gergana Alteva
gla2112

Kunal Baweja
kb2896

October 14, 2017

Contents

1 Introduction

SOL is a simple language that allows programmers to create 2D animations with ease. Programmers will have the ability to define and create objects, known as shapes, and dictate where they appear, and how they move. As a lightweight object-oriented language, SOL allows for unlimited design opportunities and eases the burden of animation. In addition, SOLs simplicity saves programmers the trouble of learning complicated third-party animation tools, without sacrificing control over behavior of objects.

2 Conventions

The following conventions are followed throughout this SOL Reference Manual.

1. `literal` - Fixed space font for literals such as commands, functions, keywords, and programming language structures.
2. *variable* - The variables for SOL progamming language and words or concept being defined are denoted in italics.

3 Lexical Conventions

This section describes the complete lexical conventions followed for a syntactically correct SOL program, forming various parts of the language.

3.1 *Comments*

Comments in SOL start with character sequence `/*` and end at character sequence `*/`. They may extend over multiple lines and all characters following `/*` are ignored until an ending `*/` is encountered.

3.2 *Identifiers*

In SOL, an identifier is a sequence of characters from the set of english alphabet, arabic numerals and underscore (_). The first character cannot be a digit. Identifiers are case sensitive. Identifiers cannot be any of the reserved keywords mentioned in section 3.3.

3.3 *Keywords*

Keywords in SOL include data types, built-in functions, and control statements, and may not be used as identifiers as they are reserved.

int	if	main	shape
float	while	setFramerate	parent
char	func	translate	extends
string	construct	rotate	
	return	render	
		wait	
		drawPoint	
		drawCurve	
		print	
		length	
		consolePrint	

3.4 *Integer Constants*

A sequence of one or more digits representing a number in *base-10*
Eg: `1234`

3.5 *Float Constants*

Similar to an integer, a float has an *integer*, a decimal point (`.`), and a fractional part. Both the integer and fractional part are a sequence of one or more digits.
Eg: `0.55 10.2`

3.6 *Character Constants*

An ASCII character within single quotation marks.
Eg: `'x' 'a'`

3.7 *Escape Sequences*

The following are special characters represented by escape sequences.

Name	**Escape**
newline	\n
tab	\t
backslash	\\
single quote	\'
double quote	\"
ASCII NUL character	\0

3.8 *String constants*

A SOL *string* is a series of *characters* within double quotation marks. Its type is an array (defined in) of characters. The compiler places a null byte (`\0`) at the end of a string literal to mark its end.
Eg: `cat`

3.9 *Operators*

SOL has mainly four categories of operators defined below:

3.9.1 *Assignment Operator*

An *assignment operator* is denoted by the `=` symbol having a variable identifier to its left and a valid expression on its right. The *assignment operator* of the expression on the right to the variable on the left.

3.9.2 *Arithmetic Operators*

SOL has following *binary arithmetic operators*. A *binary arithmetic operator* operates on two *arithmetic expressions* specified before and after the operator respectively. The said expressions must be of type *int* or *float*

Operator	Definition
+	Addition
-	Subtraction
	Multiplication
/	Division
%	Modulo

3.9.3 *Comparison Operators*

The comparison operators are binary operators for comparing values of operands defined as expressions.

Operator	Definition
==	Equality
!=	Not Equals
¡	Less than
¿	Greater than
¡=	Less than or equals
¿=	Greter than or equals

3.9.4 *Logical Operators*

The logical operators evaluate boolean expressions and return an integer as the result - with 0 as False and 1 as True. The AND (&&) and OR (||) operators are binary, while the NOT (!) operator is unary.

Operator	Definition
&&	AND
||	OR
!	NOT

3.10 *Punctuators*

The following symbols are used for semantic organization in SOL:

Punctuator	Usage
{}	Used to denote a block of code. Must be present as a pair.
()	Specifies conditions for statements before the subsequent code, or denotes the arguments of a function. Must be present as a pair.
[]	Indicates an array. Must be present as a pair.
;	Signals the end of a line of code.
,	Used to separate arguments for a function, or elements in an array definition.

4 Identifier Scope

4.1 *Block Scope*

Identifier scope is a specific area of code wherein an identifier exists. A scope of an identifier is from its declaration until the end of the code block within which it is declared.

4.2 *File Scope*

Any identifier (such as a variable or a function) that is defined outside a code block has file scope i.e. it exists throughout the file.

If an identifier with file scope has the same name as an identifier with block scope, the block-scope identifier gets precedence.

5 Expressions and Operators

5.1 *Typecasting*

A typecast is the conversion a variable from one type to another. SOL supports explicit casting of ints to floats and floats to ints. To cast a variable to a different type, place the desired type in parentheses in front of the variable.
Eg: `(int) myFloat`

5.2 *Precedence and Associativity*

SOL expressions are evaluated with the following rules:

- Expressions are evaluated from left to right
- Multiplication, division and modulo operations take precedence over addition and subtraction
- Parentheses override all precedence rules
- Logical NOT has precedence over logical AND, which has precedence over logical OR

5.3 *Dot Accessor*

To access members of a declared `shape` (further described in section 7), use the dot accessor ..
Eg: `shape_object.point1 /* This accesses the variable point1 within the object shape_object */`

6 Declarations

6.1 *Array*

7 Classes