Sketchpad Graphics Language Reference Manual

Zhongyu Wang, zw2259
Yichen Liu, yl2904
Yan Peng, yp2321

October 20, 2013

1. Introduction

This manual provides reference information for using the SKL (Sketchpad Graphics Language). Sketchpad Graphics Language is a language designed for users to construct complex geometric graphics for mathematical analysis. By maintaining a specifically designed parent-children relationship between geometric objects, SKL enables users to change properties of the geometric objects without rewriting the whole graphics.

This manual describes the lexical conventions, basic types scoping rules, built-in functions and grammar of SGL.

2. Lexical conventions

2.1 Comments

The characters # introduce a comment.

2.2 Identifiers

An identifier is a sequence of letters and digits including '_' in alphabet. The first character must be alphabetic or '_'. Upper and lower case letters are considered no different.

2.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

integer	for	Object
string	while	Line
float	break	Circle
bool	continue	Point
if	true	
else	false	
elif	function	
end	return	

2.4 Constants

There are several kinds of constants, as follows:

2.4.1 Integer constants

An integer constant is a sequence of digits from 0 to 9. Only decimal integer is supported in our language. It takes 64 bit and is signed.

```
//Declaration and assignment of an Integer
Integer m = 99;
```

2.4.2 Floating constants

A floating constant consists of an integer part, a decimal point and a fraction part. The integer part and fraction part either consists of a sequence of digits and must have at least one digit. It is a 64-bit floating point number which is similar with double in C. Only decimal representations are allowed.

```
# Declaration and assignment of a Float
Float a = 0. ;
Float b = .13;
Float c = 26.40;
```

2.4.3 Bool constants

Bool type only includes two possible values: True and False.

2.4.4 String constants

A String is a sequence of characters surrounded by double quotes “ " ”. String are mainly used as labels in our program so they are unchangeable after assignment. The only operation of String is concatenation.

2.5 Operators

+	Addition
-	Substraction
*	Multiplication
/	Division
~	Not
&&	And
||	Or
==	Equality
>=	Greater or equal
<=	Less or equal
>	Greater than
<	Less than
!=	Not equal to
=	Assignment

2.6 Punctations

The following symbols are used to organize code and to specify different organizations of objects:

Symbol	Definition
;	Marks the end of a statement
{}	Initializes lists
[]	Array declaration and visiting
()	Function declaration and calling; and specifies operation priority
,	Separates elements in an array

3. Object Types

There are two broad classes of objects supported in SKL.

3.1 Basic types

There are three basic types defined by the Sketchpad Graphics Language. Type identifiers always begin with an upper-case letter followed by a sequence of one or more legal identifier characters. The built-in types include:

- Point
- Line
- Circle

These objects have properties and methods to modify their values.

3.1.1 Properties for Point

Type	Name	Definition
float	x	x-coordinate of the point
float	y	y-coordinate of the point
string	Label	name shown on the graph
integer [3]	Color	color shown on the graph: three channels RGB
point	ParentA	the first parent of the point
point	ParentB	the second point of the point
bool	Visible	whether the point can be seen on current graph

3.1.2 Properties for Line

Type	Name	Definition
string	Label	name shown on the graph
integer [3]	Color	color shown on the graph: three channels RGB
point	Start	the start point of the line
point	End	the end point of the line
bool	Visible	whether the point can be seen on current graph
integer	Type	1: Segment; 2: Ray; 3: Line

3.1.3 Properties for Circle

Type	Name	Definition
string	Label	name shown on the graph
integer [3]	Color	color shown on the graph: three channels RGB
point	Center	the center point of the circle
point	PointOnCircle	one point on the circle

bool	Visible	whether the point can be seen on current graph

3.2 Atom types

Legal atom-types are as follows:

- Integer: An integer constant is a sequence of digits from 0 to 9. Only decimal integer is supported in our language. It takes 64 bit and is signed. The default value of an integer is 0.

- Float: A floating constant consists of an integer part, a decimal point and a fraction part. The integer part and fraction part either consists of a sequence of digits and must have at least one digit. It is a 64-bit floating point number which is similar with double in C. Only decimal representations are allowed. The default value of a float is 0.0.

- Bool: Bool type only includes two possible values: True and False. The default value of a bool is false.

- String: A String is a sequence of characters surrounded by double quotes “ " ”. String are mainly used as labels in our program so they are unchangeable after assignment. The only operation of String is concatenation. String must be initialized by user before used.

- Array: An array is declared with bracket []. It is a contiguous region of storage for any given object types in SKL. All elements in an array are initialized with the default value o their types.

4. Expressions and Operators

In this section we describe the built-in operators for SKL and define what constitutes an expression in our language. Operators are listed in order of precedence. All operators associate left to right, except for assignment, which associates right to left.

4.1 Primary expressions

4.1.1 identifier

See section 2.2.

4.1.2 constant

See section 2.4.

4.1.3 (expression)

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression.

4.1.4 primary-expression [expression]

A primary expression followed by an expression in square brackets is a primary expression. The intuitive meaning is index into a list.

4.1.5 primary-expression (list of 0 or more expressions, comma separated)

A function call is a primary expression followed by parentheses containing a possibly empty, comma separated list of expressions which constitute the actual arguments to the function.

4.2 Unary operators

Expressions with unary operators group right-to-left.

4.2.1 ~ expression

The result of the logical negation operator ~ is a bool. It performs negation on the expression. This operator is applicable only to integers.

4.3 Multiplicative operators

The multiplicative operators *, and /, all group let-to-right.

4.3.1 expression * expression

The binary * operator indicates multiplication. It is valid between two integers, two floats, an integer and a float, or a Point and an integer or float. If one is an integer and the other is a float, the result is a float. If one is a Point, its x and y values are each multiplied by the other expression.

4.3.2 expression / expression

The binary / operator indicates division. The same type considerations as for multiplication apply. Division by zero is not allowed.

4.4 Additive operators

The additive operators + and - group left-to-right.

4.4.1 expression + expression

The result is the sum of the expressions. Addition is valid between two integers, two floats, an integer and a float, or two Points. In the case of addition of an integer and a float, the result is a float. When adding two points (x1, y1), (x2, y2), the result is (x1+y1, x2+y2).

4.4.2 expression – expression
The result is the difference of the operands. Subtraction is defined identically to addition.

4.5 Relational operators

The relational operators group left-to-right.

4.5.1 expression < expression

4.5.2 expression > expression

4.5.3 expression <= expression

4.5.4 expression >= expression

The operators < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to) all yield a bool, either true or false. The relational operators are valid for comparison between two integers, two doubles, or a float and an integer.

4.6 Equality operators

4.6.1 expression == expression

Equal to.

4.6.2 expression ~= expression

Not equal to.

The equality operators are valid for comparison between two integers, two floats, a float and an integer, or two Points. The result is a bool. When comparing the equality of two points, the result is true is the x and y values of the two points are identical.

4.7 Logical operators

4.7.1 expression && expression

Logical AND between two bool expressions.

4.7.2 expression || expression

Logical OR between two bool expressions.

4.8 Assignment operators

4.8.1 lvalue = expression

The value of the expression replaces the value of the object that the lvalue refers to. Types must match.

4.9 Conversions

A number of operators may, depending on their operands, cause conversion of the value o an operand from one type to another. Multiplication of an integer and a float will be a float. The rule also applies to division, addition, and subtraction.

5. Declarations

5.1 Variable declaration

Each variable must be declared before used.

Integer:

```
Integer a = 10;
```

Float:

```
Float a = 10;
Float a = 10. ;
Float a = .10;
Float a = 1.0;
```

Bool:

```
Bool a = true;
```

String:

```
String s = " Hello! ";
```

Array:

```
Integer a[5] = { 1,2,3,4,5};
Integer a[100] = 0; # assign 0 to all the elements of array a.
Float b[3] = {0.1, 0.2, 0.3};
```

Point:

```
float x1, x2, x3, y1,y2, y3;
Point A = getPoint ( x1, y1 );
Point B = getPoint ( x2, y2);
Point C = getPoint ( x3, y3);
```

Line:

```
Line L1 = getLine ( A, B );
```

Line L2 = getLine (B, C);

Line L3 = getLine (C, A);

Circle:

Circle O = getCircle (A, B, C) ;

5.2 Function declaration

The SKL language supports user-defined functions.

All functions must start with the " function " keyword, followed by the return type and function name. A function can have any numbers of parameters, whose types and names should also be declared. Note that the parameters are passed by VALUE.

Functions must be declared and implemented meanwhile, before they are called by users.

function return type function name (parameter type parameter name, …) # function declaration

{

function implementation

}

6. Statements

Except as indicated, statements are executed in sequence.

6.1 Expression statement

Most statements are expression statements, which have the form
expression ;
Usually expression statements are assignments or function calls.

6.2 Conditional statement

The two forms of the conditional statement are
if (*expression*) *statement* elif *(expression) statement* else *statement*
if (*expression*) *statement* else *statement*
In both cases the expression is evaluated and if it is true, the substatement is executed.

6.3 While statement

The while statement has the form
while (*expression*) *statement*
The substatement is executed repeatedly so long as the value of the expression remains true.
The test takes place before each execution of the statement.

6.4 For statement

The for statement has the form

for (*expression1* ; *expression2*; *expression3*) *statement*

Thus the first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the expression becomes true; the third expression typically specifies an incrementation which is performed after each iteration. Any or all of the expressions may be dropped.

6.5 Break statement

The statement

break ;

causes termination of the smallest enclosing while, or for statement; control passes to the statement following the terminated statement.

6.6 Continue statement

The statement

continue ;

causes control to pass to the loop continuation portion of the smallest enclosing while, or for statement; that is to the end of the loop.

6.7 Return statement

A function returns to its caller by means of the return statement, which has one of the forms

return ;

return (*expression*) ;

In the first case no value is returned. In the second case, the value of the expression is returned to the caller of the function. If required, the expression is converted, as if by assignment, to the type of the function in which it appears.

7. System Functions

7.1 Arithmetic functions

Float sin (Float): return the sin value of the parameter.

Float cos (Float): return the cos value of the parameter.

Float tan (Float): return the tan value of the parameter.

Float exp (Float): return the exp value of the parameter.

Float log (Float): return the log value of the parameter.

Float sqrt (Float): return the square of the parameter.

Integer floor (Float): return the greatest integer that is less than the parameter.

Integer ceil (Float): return the least integer that is greater than the parameter.

Integer round (Float): return the integer that is closest to the parameter.

7.2 Get object functions

getPoint (float x, float y): create a new point at (x, y)

create three new points A, B, C

```
float x1, x2, x3, y1,y2, y3;
Point A = getPoint ( x1, y1 );
Point B = getPoint ( x2, y2);
Point C = getPoint ( x3, y3);
```

getLine (point X, point Y, integer t) : create a new line XY with type t, 1 for segment ; 2 for ray ; 3 for line

create a new segment L1, a new ray L2, a new line L3

```
Line L1 = getLine ( A, B, 1 );
Line L2 = getLine ( B, C, 2 );
Line L3 = getLine ( C, A, 3 );
```

getCircle (point X, point Y, point Z) : create a new circle with three points X, Y, Z on the circle

getCircle (point Center, point X) : create a new circle with given center and point X on the circle

create a new circle O with three points A, B,C on the circle

```
Circle O = getCircle ( A, B, C) ;
```

create a new circle Q with center A and point B on the circle

```
Circle Q = getCircle ( A, B ) ;
```

7.3 Object operation functions

delete (object)	Delete an object
move (point a, float x, float y)	Move the point a toward new position (x, y)
intersect (line x, line y)	Create the intersection point of line x and line y
bisect (point a, point b, point c)	Create the angular bisector of angel ABC

perpendicular (point a, point b, point c) midpoint (line x) unhook (point a, integer index) hook (point a, integer index)	Create a perpendicular line of BC through A Create the middle point of line x when it is a segment

8. Scoping Rules

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable can not be accessed. There are three places where variables can be declared in SKL:

- Inside a function or a block which is called local variables,
- Outside of all functions which is called global variables.
- In the definition of function parameters which is called formal parameters.

8.1 Local variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

8.2 Global variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout the entire program after its declaration.

A program can have same name for local and global variables but value of local variable inside a function will take preference.

8.3 Formal parameters

Function parameters, formal parameters, are treated as local variables with-in that function and they will take preference over the global variables.