SLOGO
Simple Line Oriented Graphical Output
Language Reference Manual

Phil G. Sphicas
pgs2111@columbia.edu

March 9, 2010

Abstract

This manual describes the SLOGO language.

1 Introduction

SLOGO is a drawing language, capable of producing PostScript output using a relative cursor (the "turtle") that can move about the canvas in straight lines.

A SLOGO program consists of function declarations and variable declarations. These are typically contained in a text file with a `.sl` extension that is fed to the interpreter via redirection to STDIN, but a program could also be entered interactively.

Program execution begins when the end-of-file marker is reached. The interpreter looks for a function called `main`, and executes its contents as described below. If no `main` function has been defined, the interpreter exits with an error.

Based on the movements of the turtle, the SLOGO interpreter produces PostScript-compatible output to STDOUT, which would typically be redirected to a file with a `.ps` extension. The resulting graphics files can be viewed with a PostScript viewer.

The `print` command, used primarily for debugging purposes, generates PostScript comments (also to STDOUT).

2 Lexical Conventions

2.1 Tokens

There are five classes of tokens: identifiers, keywords, constants, operators, and other separators. Spaces, tabs, newlines, formfeeds, and comments, collectively called "whitespace", are ignored, expect as to separate tokens.

2.2 Comments

The characters `//` introduce a comment, which terminates at the end of the current line. Comments are ignored, but have the effect of separating adjacent tokens in a similar manner as other whitespace.

```
// This is a comment
var a; // This is also a comment
```

A comment cannot occur within an identifier, or any other type of token.

```
var ThisIs// This comment will result in a syntax error
aLongIdentifier;
```

A comment cannot occur immediately following the division operator `/`, and must be separated from it by some other form of whitespace.

```
i = 42/// Syntax error - not the same as i = 42 / 6;
6;

i = 42/ // OK - this is the same as i = 42 / 6;
6;
```

2.3 Identifiers

An identifier is a sequence of one or more letters, where letters are the ASCII characters `a` through `z` or `A` through `Z`.

Identifiers are case sensitive. The following identifiers are different.

```
if  iF  If  IF
```

2.4 Keywords

The following identifiers are reserved for the use as keywords, and may not be used otherwise. Note that these keywords, like all identifiers, are case sensitive.

```
bk        lt     return  setws
if        pd     rt      setw
else      pu     seth    var
fd        print  setpc   while
function  rep    setpos
```

2.5 Constants

Only integer constants are supported. An integer constant consists of a sequence of one or more digits, where digits are `0` through `9`. Integer constants are always specified in decimal.

3 Program Structure

As outlined in the Introduction, a SLOGO program consists of function declarations and variable declarations. The interpreter looks for a function called `main` as its starting point for execution, and runs it as described below.

4 Function Declarations

Function declarations are prefixed with the keyword `function`. They are followed by an identifier specifying the name of the function. The function identifier is then followed by a mandatory set of parentheses enclosing an optional list of function arguments. Finally, this is followed by a mandatory set of braces containing an optional sequence of variable declarations, and an optional sequence of statements.

The following are sample function declarations, which and without function arguments.

```
function foo () {
// 0 or more variable declarations, followed by
// 0 or more statements
}

function bar (a,b,c) {
// 0 or more variable declarations, followed by
// 0 or more statements
}
```

Any variables specified in the parameter list are automatically accessible as local variables within the body of the function. Attempting to redeclare these variables within the function is not allowed.

The `main` function must be declared with no arguments, as follows.

```
function main () {
// 0 or more variable declarations, followed by
// 0 or more statements
}
```

5 Variable Declarations

Variable declarations are prefixed with the keyword `var`. They are followed by an identifier specifying the name of the variable, and end with a semicolon.

Variables are used only to hold integer values. When variables are initially declared, they hold the value `0`.

The following is are sample variable declarations.

```
var a;
var be;
var see;
```

6 Statements

Statements are executed in sequence, except as noted below. The following statement types are supported:

- expression statements
- compound statements
- conditional statements
- iteration statements
- return statements

6.1 Expression Statements

An expression statement simply consists of an expression, described below, followed by a semicolon (`;`). For example:

```
i = 10;
```

6.2 Compound Statements

An compound statement consists of the left brace { followed by zero or more statements, followed by the right brace }. For example:

```
{
 i = 10;
 j = i * 2;
}
```

6.3 Conditional Statements

A conditional statement consists of the keyword `if`, followed by a set of parentheses enclosing an optional expression, followed by a statement. This can be optionally followed by the keyword `else` and another statement. If the expression following the `if` evaluates to something other than 0, then the first statement is executed. If the expression evaluates to 0, then the statement following the `else` is executed (if present). If no `else` clause is present, the next statement is executed sequentially. If the expression is empty, it is as if it evaluated to 0.

An `else` will bind to the most recent `if` that does not already have an `else` clause associated with it.

Some `if` examples:

```
if (a)
 foo();                         // Call foo() if a is non-zero
else
 bar();                         // Call bar() if a is zero

if (1)                          // This is always true
{
 print(10);                     // Debug print 10
}
```

6.4 Iteration Statements

SLOGO supports two looping constructs:

- while
- rep

6.4.1 while

The `while` construct is defined as follows:

`while (`*expression*`)` *statement*

The expression is evaluated initially. As long as the expression is not equal to zero, the statement is executed repeatedly. The expression is tested before each iteration of the loop, and any side effects occur before the execution of the statement.

For example:

```
function main()
{
 var i;
 i = 5;
 while (i > 3) {                // Prints % 5, % 4
  print(i);
  i = i - 1;
 }
}
```

6.4.2 rep

The `rep` construct is used to repeat a statement a fixed number of times.

`rep (`*expression*`)` *statement*

The expression is evaluated once. If the expression is less than or equal to 0, the substatement is skipped. If the expression is 1 or more, then the statement is executed that number of times.

For example:

```
function square(size)
{
 rep (4) {                      // Repeat 4 times
  fd (size);
  rt (90);
 }
}
```

6.5 Return Statements

The return statement uses the `return` keyword to terminate the execution of a function and pass a return value to the calling function.

`return` *expression*`;`

The syntax of the statement is the `return` keyword, followed by an expression, followed by a semicolon. If the expression is omitted, it is equivalent to returning `0`.

For example:

```
function add(x,y)
{
 return (x + y);
}

function main()
{
 var sum;
 sum = add(41 + 1);
 print(sum);                    // Should print % 42
}
```

7 Expressions

Expressions in SLOGO consist of constants, variables, variable assignment, unary or binary operations, or function calls. Expressions may also be surrounded by parentheses.

The following are valid expressions.

```
42
foo
24 + 18
a <= 10
bar() < 100
z = 10
(17)
```

7.1 Constants

As previously described, integer constants are valid expressions.

7.2 Variables

Variables are accessed simply by specifying their identifier.

7.3 Unary Operators

7.3.1 Unary Minus

The unary minus operator is supported.

`-`

7.4 Binary Operators

The following binary operators are listed in order of decreasing precedence.

7.4.1 Multiplicative Operators

The following binary operators are supported for multiplication and division. They are left associative.

`* /`

7.4.2 Additive Operators

The following binary operators are supported for addition and subtraction. They are left associative.

`+ -`

7.4.3 Relational Operators

The following binary operators are supported for less than, greater than, less than or equal to, and greater than or equal to. They return 1 for true and 0 for false.

`< > <= >=`

They are also left associative, but not usefully so. `a<b<c` is parsed as `(a<b)<c`, which evaluates `a<b` and returns `0` or `1`, then compares this result to `c`.

7.4.4 Equality Operators

The following binary operators are supported for equal to or not equal to. Again, they are left associative, but not in a useful way.

`== !=`

7.5 Assignment Expressions

The `=` operator is used for assignment. Specifically, it is used to assign an integer variable to a variable. This operator is right associative.

For example:

```
a = b = 5                       // Both a and b will get 5
```

7.6 Function Calls

Functions are called by specifying the function identifier, followed by a set of parentheses enclosing the necessary arguments. The number of arguments passed to a function must match the function declaration. Parentheses are required even if the function takes no arguments.

For example, the following are expressions representing function calls:

```
foo(1,1)
bar()
```

8 Lexical Scope

Variables and functions are statically scoped.

Variables declared outside a function can be accessed by any function, including `main` and others. These can be considered global variables. Declaring a new global variable with the same name as an existing global variable is not permitted.

Functions cannot be declared within other functions. Therefore, all functions are effectively global functions. Declaring a new function with the same name as an existing function is not allowed.

The lexical scope of a global function or global variable begins at program execution and ends at program termination, regardless of where exactly the variables or function is declared. In particular, functions can be called before they are declared, and variables accessed before they are declared, as long as they are declared eventually. This allows for recursion and mutual recursion.

Variables declared inside a function can only be accessed within that function. These can be considered local variables. If a local variable is declared within a function that has the same name as a global variable, the global variable is "masked" within that function.

The name space of function identifiers and variable identifiers is separate. Therefore it is possible to use the same name for a function and for a variable, although not recommended.

9 Predefined Functions

SLOGO provides a rich set of predefined drawing functions. These are provided as a combination of built-in and library functions.

9.1 Window Functions

- `setws(`x`,`y`)`
  Sets the window size to width x and height y.

9.2 Turtle Functions

- `fd(`s`)`
  Move forward s steps.
- `bk(`s`)`
  Move backward s steps. Equivalent to `fd(-`s`)`.
- `lt(`d`)`
  Turn left d degrees.
- `rt(`d`)`
  Turn right d degrees. Equivalent to `lt(-`d`)`.
- `setpos(`x`,`y`)`
  Set the turtle's position to (x,y).
- `seth(`d`)`
  Set the turtle's heading to d degrees.

9.3 Pen Functions

- `pu()`
  Pick the pen up.
- `pd()`
  Set the pen down.

- `setpc(`r`,`g`,`b`)`
  Set the pen color to the RGB values specified.
- `setw(`w`)`
  Set the pen width.

9.4 Debug Printing

The `print` function, useful for testing and debugging, prints a PostScript comment containing the value of n, where n is an expression that evaluates to an integer.

For example,

```
print(4*10+2);
```

yields

```
% 42
```

10 Example

The following is a complete SLOGO program, followed by the postscript commands that are output, as well as the resulting graphic.

10.1 Input file: test-shapes.sl

```
// define a function that can draw a square
function square(size) {
  rep (4) {
    fd (size);
    rt (90);
  }
}

// define a function that can draw a rectangle
function rectangle(length, width) {
  rep (2) {
    fd (length);
    rt (90);
    fd (width);
    rt (90);
  }
}

// define a function that can draw a polygon
function polygon(size, sides) {
  rep (sides) {
    fd (size);
    rt (360/sides);
  }
}

// define a function that will draw several squares
function squares() {
  var i;
  i = 1;
  while (i <= 5) {
    square(10 * i);
    i = i + 1;
  }
}

// main
function main()
{
```

```
  // set the turtle heading (to UP)
  seth(90);

  // put the pen down
  pd();

  // move turtle to position (125,125)
  setpos(125,125);

  // draw a square of size 50x50
  square(50);

  // move turtle to position (50,50)
  setpos(50,50);

  // draw a 75 x 25 rectangle
  rectangle(75,25);

  // move turtle to position (150,150)
  setpos(150,150);

  // draw an octagon of size 50
  polygon(50,8);

  // draw several squares at position (25,200)
  setpos(25,200);
  squares();
}
```

10.2 Output file: test-shapes.ps

```
newpath
125 125 moveto
125 175 lineto
stroke
newpath
125 175 moveto
175 175 lineto
stroke
newpath
175 175 moveto
175 125 lineto
stroke
newpath
175 125 moveto
125 125 lineto
stroke
newpath
50 50 moveto
50 125 lineto
stroke
newpath
50 125 moveto
75 125 lineto
stroke
newpath
75 125 moveto
75 50 lineto
stroke
newpath
75 50 moveto
50 50 lineto
```

```
stroke
newpath
150 150 moveto
150 200 lineto
stroke
newpath
150 200 moveto
185 235 lineto
stroke
newpath
185 235 moveto
235 235 lineto
stroke
newpath
235 235 moveto
270 200 lineto
stroke
newpath
270 200 moveto
270 150 lineto
stroke
newpath
270 150 moveto
235 115 lineto
stroke
newpath
235 115 moveto
185 115 lineto
stroke
newpath
185 115 moveto
150 150 lineto
stroke
newpath
25 200 moveto
25 210 lineto
stroke
newpath
25 210 moveto
35 210 lineto
stroke
newpath
35 210 moveto
35 200 lineto
stroke
newpath
35 200 moveto
25 200 lineto
stroke
newpath
25 200 moveto
25 220 lineto
stroke
newpath
25 220 moveto
45 220 lineto
stroke
newpath
45 220 moveto
45 200 lineto
stroke
newpath
```

```
45 200 moveto
25 200 lineto
stroke
newpath
25 200 moveto
25 230 lineto
stroke
newpath
25 230 moveto
55 230 lineto
stroke
newpath
55 230 moveto
55 200 lineto
stroke
newpath
55 200 moveto
25 200 lineto
stroke
newpath
25 200 moveto
25 240 lineto
stroke
newpath
25 240 moveto
65 240 lineto
stroke
newpath
65 240 moveto
65 200 lineto
stroke
newpath
65 200 moveto
25 200 lineto
stroke
newpath
25 200 moveto
25 250 lineto
stroke
newpath
25 250 moveto
75 250 lineto
stroke
newpath
75 250 moveto
75 200 lineto
stroke
newpath
75 200 moveto
25 200 lineto
stroke
```

10.3 Graphical Output