MDraw Language Reference Manual

Jingyu Shi

Huimin Sun

Dongxiang Yan

July 24, 2013

Contents

1. Introduction

MDraw means “my draw”, which is designed as a 2D graphics drawing and manipulation language. It enables users to automatically draw and manipulate numerous graphics by taking advantage of this programming language.

Now days, although there are many popular graphics drawing and manipulation software solutions, there is no dedicated text-based language to facilitate and automate this process. Some repeating works are boring and time consuming. The case is even worse for complex graphics. Some graphics are hard to be repeated and scaled manually. By using MDraw, these issues could be resolved with ease.

2. Lexical conventions

2.1 Comments

In MDraw, comments are represented inside of parentheses and asterisks. Comments usually start with /* and end with */.

2.2 Identifiers

Identifiers are sequences of letters, digits and underscores (_). Uppercase letters and lowercase letters are considered the same in MDraw because it is not case sensitive. The first character of an identifier must be a letter.

2.3 Keywords

The following keywords are reserved for MDraw:

int	point
float	line
string	triangle
boolean	rectangle
if	move
else	copy
elseif	rotate
for	combine
do	remove
while	scale
switch	arc

2.4 Constants

Constants in MDraw include Integer, float, string and boolean.

2.4.1 Integer constants

An integer constant is a sequence of digits without a decimal point.

```
/* Here is an example of integer constant. */
int a = 8;
```

2.4.2 String constants

A string constant is a sequence of characters surrounded by single or double quotes. Escaped sequence cannot be recognized in MDraw.

```
/* Here is an example of string constant. */
string a = “This is a string constant.”
```

2.4.3 Floating constants

A floating constant includes an integer part, decimal point, fraction part, e, and an integer exponent (which is optional). Integer and fraction part, both of which consist of a sequence of digits, are separated by one decimal point. The fraction part can be missing. Either the decimal point or the e and exponent (not both) can be missing.

```
/* Here is an example of float constants. */
 float a = 0.3;
 float b = 1e-5
/* Here is an illegal representation of float constants. */
 float a = .3;  /* integer part missing */
 float b =1..2 /* more than one decimal point */
```

2.4.4 Boolean constants

A boolean constant represents true or false (case sensitive).

```
/* Here is an example of Boolean constants. */
 boolean a = True;
 boolean b = False;
```

3. Objects

An object is a manipulatable region of storage.

4. Expressions

4.1 Primary expressions

Primary expressions group left to right.

4.1.1 identifier

An identifier is a primary expression. Its type is specified by its declaration. It follows the rules in Section 2.2.

4.1.2 constant

A constant is a primary expression. It follows the rules Section 2.4.

4.1.3 string

A string is a primary expression. It follows the rules in Section 2.5

4.1.4 (expression)

A parenthesized expression is a primary expression, which gives the expression high priority in calculation.

4.1.5 [expression]

The expression in square brackets is a primary expression. It indicates index into a list.

4.1.6 (expression-list)

The expression list consists of 0 or more expressions, which are the arguments, separated by comma.

```
/* Here is an example of expression-list. */
move (line_01, layer_02);
```

4.1.7 expression −> expression

The arrow -> stands for a function type. The type before the arrow is the type of the function's argument, and the type after the arrow is the type of the result.

4.2 Unary operators

Expressions with unary operators group left to right.

4.2.1 − expression

- expression gives the negative of the expression and has the same type. It is only valid for integers and floats.

4.2.2 ! expression

! expression gives the logical negation of the expression, which must be boolean type. ! expression returns 1if the value of the expression is 0, 0 if the value is not zero.

4.2.3 ++ expression

The object referred to by the expression is incremented by one, which is the result. The value is the new value of expression and the type is the type of the expression. It is only valid for integers.

4.2.4 expression ++

The result is the value of the object referred to by the expression. After the result is calculated, the object referred to by the expression is incremented by one. It is only valid for integers.

4.2.5 −− expression

It is similar to the ++ expression. But it is decrement instead of increment.

4.2.6 expression −−

It is similar to the expression ++. But it is decrement instead of increment.

4.3 Multiplicative operators

4.3.1 expression * expression

Multiplication is valid between integers, floats and Points, but the two expressions cannot be Points at the same time. If both expressions are integers, the result is an integer. If both expressions are float, the result is float. If one expression is integer and the other one is float, then the integer is converted into float and the result is float. If one of the expression is a Point, like (a, b), each of a and b is multiplied by the other expression and it returns a Point.

4.3.2 expression / expression

Division is similar to multiplication. The only difference is that the second expression cannot have a value of zero.

4.3.3 expression % expression

Mod gives the remainder of expression / expression. Both expressions must be an integer.

4.4 Additive operators

The additive operators + and – group left to right.

4.4.1 expression + expression

Addition gives the sum of the two expressions. It is valid between integers and floats. It is also valid between two Points. If both expressions are float, the result is float. If one expression is integer and the other one is float, then the integer is converted into float and the result is float. While adding two Points, like (a, b) and (c, d), the result is (a+c, b+d), which is also a Point.

4.4.2 expression − expression

Subtraction is similar to addition.

4.5 Relational operators
The relational operators group left to right. They are valid between integers and floats. They are also valid between two Points (a, b) and (c, d). It returns a boolean type.

4.5.1 expression < expression
Less than. While comparing two Points, it returns 1 if a<c and b<d, otherwise returns 0.

4.5.2 expression > expression
Great than. It is similar to less than.

4.5.3 expression <= expression
Less than or equal. It is similar to less than.

4.5.4 expression >= expression
Great than or equal. It is similar to less than.

4.6 Equality operators
The equality operators group left to right. They are valid among integers, floats. They are also valid between two Points. It returns a Boolean type.

4.6.1 expression == expression
Equal to. Two points (a, b) and (c, d) are equal to each other if and only if a = c and b = d.

4.6.2 expression != expression
Not equal to. It is similar to equal.

4.7 Logical operators
Logical operators group left to right. They are valid between two boolean expressions and return a boolean type.

4.7.1 expression & expression
AND. If the first expression is zero, the second expression still needs to be evaluated.

4.7.2 expression && expression
AND. The second expression is not evaluated if the first expression is zero.

4.7.3 expression | expression
OR. If the first expression is not zero, the second expression still needs to be evaluated.

4.7.4 expression || expression
OR. The second expression is not evaluated if the first expression is not zero.

4.7.5 ^ expression
The operator ^ performs string concatenation and return a string type. For example, “This is” ^ “a string” returns a string: “This is a string”.

4.7.6 expression ? expression : expression
Conditional expression. It returns the second expression if the first expression is not zero, otherwise returns the third expression.

4.8 Assignment operators
Assignment operators group right to left. They are valid between two expressions that have the same type: integer, float and Point.

4.8.1 expression = expression
It gives the value of the second expression to the first expression.

4.8.2 expression =+ expression
It replaces the first expression with the sum of first and second expression.

4.8.3 expression =* expression
It replaces the first expression with the product of first and second expression.

4.8.4 expression =− expression
It uses the result of first expression subtracting the second expression to replace the first expression.

4.8.5 expression =/ expression
It uses the result of first expression dividing by the second expression to replace the first expression.

5. Declarations

5.1 Variable declaration
Variable declaration includes the type and the value. See section 2.4 for more details.

5.2 Function declaration
Function declaration includes function key word and optional expressions as arguments. See more details in section 8. Examples.

6. Statements

Except as indicated, statements are executed in normal sequence from top to low and from left to right.

6.1 Expression statement
Most of expression statements are assignments or function calls, which are all ended with semicolon ";".

6.2 Conditional statement

6.2.1 if (expression) statement
The expression will be evaluated first. If it is true, then the following statement will be executed. Otherwise, the statement will be ignored.

6.2.2 if (expression) statement else statement
The expression will be evaluated first. If it is true, then the first statement will be executed. Otherwise, the second statement will be executed.

6.2.3 if (expression) statement elseif (expression) statement else statement
The first expression will be evaluated first. If it is true, then the first statement will be executed. If not, the second expression will be evaluated. If it is true, the second statement will be executed. If not, the last statement will be executed. There are maybe more than one elseif. The expression will be evaluated in the order of sequence.

6.3 while (expression) statement
The expression will be evaluated before each execution of statement. This is a loop, which will repeatedly execute statement when expression is evaluated as true.

6.4 for (expression; expression; expression) statement
The first expression is used to initialize this loop. The second expression is evaluated as true or false. If it is true, the statement will be executed. Otherwise, this loop will be ended. The third expression is used to change the value in second expression after each execution of the statement.

7. Functions

```
/* Draw a point with coordinate (x, y) */
point (int x, int y) ;

/*Draw a line with a starting point and an end point */
line (point_1name, point_2name);

/*Draw an arc with the point as its center, and stangle as start angle, and endangle as the end
angle, as well as the radius of the arc.*/
arc ( point_name , int stangle, int endangle, int radius );

/*Use a point as the center, and make the coordinate */
concenter (point_name);

/* Draw a polygon */
/* Draw a triangle (lines between three points) */
triangle_01 = ~((1, 2), (3, 4), (5, 6), (1, 2));

/* Draw a rectangle */
rectangle_01 = ~((0, 0), (4, 0), (4, 3), (0, 3), (0, 0));

/* Get the coordinate of the point */
getPoint ( point_name );
/* Get the center, radius and angle of the arc */
getArc (arc_name);

/* Move a line to right by x units and up for y points at current layer*/
move((x, y), line_01);

/* Rotate a line clock wisely 90 degrees at point (x,y)*/
rotate(90, (x, y), line_01);
/* Rotate a line anti-clock wisely 90 degrees at point (x, y)*/
rotate(-90, (x, y), line_01);

/* Draw the object that is constructed*/ draw(objects);
/* Remove line_01 from graph_01*/ remove(object);
/* Scale a square to 5 times of original area with the same central point*/ scale(square_01, 5);
```

8. Examples

8.1 A first example using MDraw

```
/* To draw an arc on the coordinate */
void main(){
/*To draw a point as the center of the arc*/
        point_01 = point(0,0);
/*To draw a point as the center, and the coordinate is made */
```

```
        center = concenter(point_01);
/*To draw an arc with the center, and stangle as start angle, and endangle as the end angle, as well
as the radius of the arc.*/
        arc_01=arc( center, int stangle, int endangle, int radius );
/*Draw the arc */
        draw(arc_01);
}
```

8.2 Draw concentric circles

```
void main(){

point_01 = point(0,0);

/*To build a function called circle */
        fun arc circle ( center, radius ){
                if radius >=0;
                circle = arc(center, 0, 360, radius);
                return circle;
        }
/* To draw concentric circles*/
        for ( radius = 25; radius < =125; radius = radius +20)
                {
                circle_d = circle( point, radius);
                draw (circle_d);
}
```

8.3 Draw a 3d rectangle

```
void main(){

/* Draw a rectangle, with the starting point and the end point given*/
        rectangle_01 = rectangle((0, 0), (4, 0), (4, 3), (0, 3));

/*Draw a 3d rectangle*/

        rec3d ( rectangle_01, depth, topflag );

}
```