

# MaSH Environment console

Andrew Rock  
School of Information and Communication Technology  
Griffith University  
Nathan, Queensland, 4111, Australia  
a.rock@griffith.edu.au

June 9, 2011

## Contents

<b>1 Purpose</b>	<b>1</b>
<b>2 Rewrites</b>	<b>2</b>
<b>3 Console input</b>	<b>2</b>
3.1 Purpose . . . . .	2
3.2 Methods . . . . .	2
<b>4 Console output</b>	<b>4</b>
4.1 Purpose . . . . .	4
4.2 Methods . . . . .	4
<b>5 Math</b>	<b>5</b>
5.1 Purpose . . . . .	5
5.2 Constants . . . . .	5
5.3 Methods . . . . .	5
<b>6 Strings</b>	<b>7</b>
6.1 Purpose . . . . .	7
6.2 Methods . . . . .	8
<b>7 Debugger</b>	<b>10</b>
7.1 Purpose . . . . .	10

## 1 Purpose

This environment supports programming for a console program that reads from standard input and writes to standard output.

## 2 Rewrites

*mandatory*

```
void main ()
```

*Purpose:* A program that is organised into methods must have a `main` method (a procedure with no arguments). This will be the first method to execute. `mashc` automatically rewrites this method to conform to standard Java.

## 3 Console input

### 3.1 Purpose

The following are methods for reading from standard input (which is usually the keyboard) and for checking to see if there is more input to read.

### 3.2 Methods

```
int readInt ()
```

*Purpose:* Returns the next integer from standard input.

*Precondition:* Will cause a run time error unless there is an integer in standard input to read. Use `isNextInt()` to check first.

```
long readLong ()
```

*Purpose:* Returns the next long from standard input.

*Precondition:* Will cause a run time error unless there is a long in standard input to read. Use `isNextLong()` to check first.

```
boolean readBoolean ()
```

*Purpose:* Returns the next boolean from standard input.

*Precondition:* Will cause a run time error unless there is a boolean in standard input to read. Use `isNextBoolean()` to check first.

```
double readDouble ()
```

*Purpose:* Returns the next double from standard input.

*Precondition:* Will cause a run time error unless there is a double in standard input to read. Use `isNextDouble()` to check first.

```
float readFloat ()
```

*Purpose:* Returns the next float from standard input.

*Precondition:* Will cause a run time error unless there is a float in standard input to read. Use `isNextFloat()` to check first.

`String readWord ()`

*Purpose:* Returns the next word as a `String`. A “word” is a sequence of one-or-more non-whitespace characters.

*Precondition:* Will cause a run time error unless there is a word in standard input to read. Use `isNextWord()` to check first.

`String readLine ()`

*Purpose:* Returns the next line of text as a `String`. A line is a sequence of zero-or-more characters terminated by the end of line, which is not returned as part of the line.

*Precondition:* Will cause a run time error unless there is a line in standard input to read. Use `isNextLine()` to check first.

`boolean isNextInt ()`

*Purpose:* Returns `true` if and only if there is an integer in standard input available to read, that is `readInt()` would succeed.

`boolean isNextLong ()`

*Purpose:* Returns `true` if and only if there is a long in standard input available to read, that is `readLong()` would succeed.

`boolean isNextBoolean ()`

*Purpose:* Returns `true` if and only if there is a boolean in standard input available to read, that is `readBoolean()` would succeed.

`boolean isNextDouble ()`

*Purpose:* Returns `true` if and only if there is a double in standard input available to read, that is `readDouble()` would succeed.

`boolean isNextFloat ()`

*Purpose:* Returns `true` if and only if there is a float in standard input available to read, that is `readFloat()` would succeed.

`boolean isNextWord ()`

*Purpose:* Returns `true` if and only if there is a word in standard input available to read, that is `readWord()` would succeed.

`boolean isNextLine ()`

*Purpose:* Returns `true` if and only if there is a line in standard input available to read, that is `readLine()` would succeed.

## 4 Console output

### 4.1 Purpose

The following are methods for printing to standard output (which is usually the terminal window).

### 4.2 Methods

`void print (char c)`

*Purpose:* Writes character `c` to standard output.

`void print (String s)`

*Purpose:* Writes string `s` to standard output.

`void print (long i)`

*Purpose:* Writes integral `i` to standard output.

`void print (boolean p)`

*Purpose:* Writes boolean `p` to standard output.

`void print (double x)`

*Purpose:* Writes floating point number `x` to standard output.

`void println (char c)`

*Purpose:* Writes character `c` and then a newline to standard output.

`void println ()`

*Purpose:* Writes a newline to standard output.

`void println (String s)`

*Purpose:* Writes string `s` and then a newline to standard output.

`void println (long i)`

*Purpose:* Writes integral `i` and then a newline to standard output.

`void println (boolean p)`

*Purpose:* Writes boolean `p` and then a newline to standard output.

`void println (double x)`

*Purpose:* Writes floating point number `x` and then a newline to standard output.

## 5 Math

### 5.1 Purpose

The following are some commonly used numeric constants and functions.

### 5.2 Constants

```
final int MAX_INT
```

*Purpose:* A constant holding the maximum value an `int` can have,  $2^{31} - 1$ .

```
final int MIN_INT
```

*Purpose:* A constant holding the minimum value an `int` can have,  $-2^{31}$ .

```
final long MAX_LONG
```

*Purpose:* A constant holding the maximum value a `long` can have,  $2^{63} - 1$ .

```
final long MIN_LONG
```

*Purpose:* A constant holding the minimum value a `long` can have,  $-2^{63}$ .

```
final double PI
```

*Purpose:* The closest `double` approximation to  $\pi$ .

### 5.3 Methods

```
double abs (double a)
```

*Purpose:* Returns the absolute value of `a`.

```
float abs (float a)
```

*Purpose:* Returns the absolute value of `a`.

```
long abs (long a)
```

*Purpose:* Returns the absolute value of `a`.

```
int abs (int a)
```

*Purpose:* Returns the absolute value of `a`.

`double ceil (double a)`

*Purpose:* Returns the least double value that is greater than or equal to `a` and equal to an integer.

`double exp (double x)`

*Purpose:* Returns  $e^x$ , that is Euler's constant  $e$  raised to power  $x$ .

`double floor (double a)`

*Purpose:* Returns the greatest double value that is less than or equal to `a` and equal to an integer.

`double log (double x)`

*Purpose:* Returns the natural logarithm of  $x$ .

`long round (double a)`

*Purpose:* Returns the closest `long` to `a`.

`int round (float a)`

*Purpose:* Returns the closest `int` to `a`.

`double sqrt (double a)`

*Purpose:* Returns the square root of `a`.

*Precondition:*  $a \geq 0.0$ .

`double pow (double a, double b)`

*Purpose:* Returns `a` raised to the power `b`,  $a^b$ .

`double sin (double a)`

*Purpose:* Returns the trigonometric sine of `a` radians.

`double cos (double a)`

*Purpose:* Returns the trigonometric cosine of `a` radians.

`double tan (double a)`

*Purpose:* Returns the trigonometric tangent of `a` radians.

`double asin (double a)`

*Purpose:* Returns the trigonometric arc sine of **a** in radians.

`double acos (double a)`

*Purpose:* Returns the trigonometric arc cosine of **a** in radians.

`double atan (double a)`

*Purpose:* Returns the trigonometric arc tangent of **a** in radians.

`double max (double a, double b)`

*Purpose:* Returns the greater of **a** and **b**.

`float max (float a, float b)`

*Purpose:* Returns the greater of **a** and **b**.

`int max (int a, int b)`

*Purpose:* Returns the greater of **a** and **b**.

`long max (long a, long b)`

*Purpose:* Returns the greater of **a** and **b**.

`double min (double a, double b)`

*Purpose:* Returns the lesser of **a** and **b**.

`float min (float a, float b)`

*Purpose:* Returns the lesser of **a** and **b**.

`int min (int a, int b)`

*Purpose:* Returns the lesser of **a** and **b**.

`long min (long a, long b)`

*Purpose:* Returns the lesser of **a** and **b**.

`double random ()`

*Purpose:* Returns a random value  $x$  such that  $0.0 \leq x < 1.0$ .

## **6 Strings**

### **6.1 Purpose**

The following are methods for working with **Strings**.

## 6.2 Methods

`int length (String s)`

*Purpose:* Returns the length of `s`.

`char charAt (String s, int i)`

*Purpose:* Returns the character at position `i` in `s`.

*Precondition:*  $0 \leq i < \text{length}(s)$ .

`boolean equals (String a, String b)`

*Purpose:* Returns `true` if and only if `a` contains the same sequence of characters as in `b`.

`String format (boolean p, int w)`

*Purpose:* Returns `p` converted to a string, padded with spaces to a minimum width  $|w|$ . If `w` is negative, the result is left-justified, otherwise right-justified.

Examples:

`format(true, 10)` returns " true"; and

`format(true, -10)` returns "true".

`String format (char c, int w)`

*Purpose:* Returns `c` converted to a string, padded with spaces to a minimum width  $|w|$ . If `w` is negative, the result is left-justified, otherwise right-justified.

Examples:

`format('a', 3)` returns " a"; and

`format('a', -3)` returns "a".

`String format (double d, int w, char f, int p)`

*Purpose:* Returns `d` converted to a string, padded with spaces to a minimum width  $|w|$ . If `w` is negative, the result is left-justified, otherwise right-justified. `f` controls the format: `'e'` selects scientific notation; `'f'` selects fixed point; or `'g'` selects the best format depending on the number and `p`. For `'e'` and `'f'`, `p` is the number of decimal digits to display after the decimal point, but for `'g'` it is the total number of digits.

Examples:

`format(1234.56789, 12, 'e', 4)` returns " 1.2346e+03";

`format(1234.56789, 12, 'f', 4)` returns " 1234.5679";

```
format(1234.56789, 12, 'g', 4) returns "      1235";
format(0.000001234567, 12, 'e', 4) returns " 1.2346e-06";
format(0.000001234567, 12, 'f', 4) returns "      0.0000";
and
format(0.000001234567, 12, 'g', 4) returns " 1.235e-06".
```

String format (long l, int w)

*Purpose:* Returns l converted to a string, padded with spaces to a minimum width |w|. If w is negative, the result is left-justified, otherwise right-justified.

Examples:

```
format(42, 5) returns "   42"; and
format(42, -5) returns "42   ".
```

String format (String s, int w)

*Purpose:* Returns s padded with spaces to a minimum width |w|. If w is negative, the result is left-justified, otherwise right-justified.

Examples:

```
format("aaa", 5) returns "  aaa"; and
format("aaa", -5) returns "aaa  ".
```

boolean parseBoolean (String s)

*Purpose:* Returns s converted to a boolean.

int parseInt (String s)

*Purpose:* Returns s converted to an int.

long parseLong (String s)

*Purpose:* Returns s converted to a long.

float parseFloat (String s)

*Purpose:* Returns s converted to a float.

double parseDouble (String s)

*Purpose:* Returns s converted to a double.

# 7 Debugger

## 7.1 Purpose

This environment supports a debugger with a graphical user interface. It allows the user to slow down the execution of a program, so that the flow of control through the program may be observed and the values in variables monitored.

To activate the debugger, compile your program with the `mashc +debug` option.

While the program is running under the control of the debugger, the original MaSH program's source code is displayed on the left, with a pointer showing the statement that is about to execute. Sometimes it will point to a closing brace, indicating that the current method is about to be exited, or a loop guard is about to be tested again, depending on the context.

The contents of all the program's variables are displayed on the right in a table. The entries in the *scope* column, consist of a number and a word. The number indicates which method invocation the variable belongs to. For no-method MaSH programs the number is always 0. For method MaSH programs, 0 indicates a global variable, 1 indicates the `main` method, 2 indicates a method called from `main`, 3 indicates a method called from 2, etc. The word is blank for variables in a no-method program, unless the variable is declared in a block, which makes it *local*. In method programs, the word can be: *global*; *local*; or *param*. Variables are added to the table as they come into scope and removed again when their enclosing block exits. Anything changed on the last program step is displayed in green.

Limitations: The debugger can not display values of arrays with 3 or more dimensions. The debugger can not step through methods that run in the programs event dispatch thread (for example, the `paintWindow` method in the `graphics` environment). This GUI debugger can not work in environments without a big screen (for example, Lego Mindstorms).

# Index

abs, 5  
acos, 7  
asin, 6  
atan, 7  
  
ceil, 6  
charAt, 8  
cos, 6  
  
equals, 8  
exp, 6  
  
floor, 6  
format, 8, 9  
  
isNextBoolean, 3  
isNextDouble, 3  
isNextFloat, 3  
isNextInt, 3  
isNextLine, 3  
isNextLong, 3  
isNextWord, 3  
  
length, 8  
log, 6  
  
main, 2  
max, 7  
MAX\_INT, 5  
MAX\_LONG, 5  
min, 7  
MIN\_INT, 5  
MIN\_LONG, 5  
  
parseBoolean, 9  
parseDouble, 9  
parseFloat, 9  
parseInt, 9  
parseLong, 9  
PI, 5  
pow, 6  
print, 4  
println, 4  
  
random, 7  
readBoolean, 2  
readDouble, 2  
readFloat, 2  
readInt, 2  
readLine, 3  
readLong, 2  
readWord, 3  
round, 6  
  
sin, 6  
sqrt, 6  
  
tan, 6