User manual

Intro

Our language is imperative with each statement terminated with a semi-colon `;`. Arguments passed to functions are separated by spaces similar to functional languages such as Ocaml/Scheme.

Below is a simple example program which takes an integer input from stdin and prints to stdout:

```
int value = console.read_int;
console.print_int value;
```

Curly brackets `{ }` are used to denote scope in loops, conditionals and lambda expressions.

Variables

Variables are used to store dynamic values, with assignment syntax similar to most programming languages:

```
int variable = 0;
```

The value of 'variable' is now the integer 0.

Math

Basic arithmetic

All basic arithmetic operations are supported:

- `int + int` or `int math.plus int` - addition
- `int - int` or `int math.minus int` - subtraction
- `int * int` or `int math.mul int` - multiplication
- `int / int` or `int math.div int` - division
- `int % int` or `int math.mod int` - modulus
- `int ^ int` or `int math.pow int` - power/index

Other math functions

- `math.sqrt <int>` - square root function
- `math.log <int>` - logarithm function
- `math.fact <int>` - factorial function
- `math.sign <int>` - integer sign function (returns +1/-1)
- `math.max <int> <int>` - bigger number function
- `math.min <int> <int>` - smaller number function

Input/Output

All I/O operations interact with stdin/stdout and all built in functions are part of `console` similar to JavaScript.

Input

- `console.read_int` - read an integer from stdin
- `console.read_string` - read a string from stdin
- `console.read_bool` - read a boolean from stdin

Output

- `console.print_int` - print an integer to stdout
- `console.print_string` - print a string to stdout
- `console.print_bool` - print a boolean to stdout
- `console.println_int` - print an integer to stdout with new line terminator
- `console.println_string` - print a string to stdout with new line terminator
- `console.println_bool` - print a boolean to stdout with new line terminator

Error output

- `console.error_int` - print an error to stdout as an integer
- `console.error_string` - print an error to stdout as a string
- `console.error_bool` - print an error to stdout as a boolean
- `console.errorln_int` - print an error to stdout as an integer with new line terminator

- `console.errorln_string` - print an error to stdout as a string with new line terminator
- `console.errorln_bool` - print an error to stdout as a boolean with new line terminator

Loops

Basic loop

Loops are important when operating on streams of continuous data. For this reason, loops are simple in our language:

```
loop {
  console.println_int console.read_int;
}
```

The above program will loop printing integers from stdin to stdout with a new line terminator. The loop will continue until `EOF` is encountered.

While/do and do/while

It is also possible to loop based on any boolean condition in a while/do or do/while loop.

While/do

```
while (someValue < someOtherValue) do {
  console.println_string "Hello world!";
}
```

Do/while

```
do {
  console.println_string "Hello world!";
} while (someValue < someOtherValue);
```

`Hello world!` will be printed to stdout for as long as `someValue < someOtherValue` evaluates to true.

A do/while will *always execute at least once* even if `someValue < someOtherValue` always evaluates to false. Contrastingly, a while/do will not print to stdout if `someValue < someOtherValue` is never true.