The Programming Language EGA Reference Manual

Written by Katayama Hirofumi MZ.

What is EGA?

EGA is a small programming language of a simple grammar, written in C++11. It is very tiny (< 500 KiB). You can extend its functions easily.

The source code of EGA will be found at https://github.com/katahiromz/EGA .

How to use

Please start up EGA. The following text will be displayed:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ EGA Version 4 by katahiromz                  @
@ Type 'exit' to exit. Type 'help' to see help. @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

EGA>
```

Enter an EGA expression (for example, `print(+(1, 2));`) and press `Enter` key. `3` will be shown.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ EGA Version 4 by katahiromz                  @
@ Type 'exit' to exit. Type 'help' to see help. @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

EGA> print(+(1, 2));
3
EGA>
```

To quit EGA, please enter " `exit` ".

The syntax of the EGA is similar to one or more function call(s) of C language. But, every operator in EGA is a function.

Enter " `help` " to see all the EGA functions:

```
EGA> help;
EGA has the following functions:
  !
  !=
  %
  &
  &&
  ...
  and
  array
  ...

EGA>
```

To see brief usage of `print` function, enter " `help print` ".

```
EGA> help print;
EGA function 'print':
  argument number: 0..256
  usage: print(value, ...)

EGA>
```

The detailed descriptions of the EGA functions will be described later.

Comments

An comment of EGA begins with the first atmark (`@`) of a line, and ends with newline. The comments are ignored in EGA execution.

Values

The EGA values are integers, strings, and/or arrays.

An EGA integer literal is a sequence of digit(s) (`0` , ..., `9`).

An EGA string literal is a string wrapped by double quotations (`" "`). If the string contains a double quotation, it will be doubled in the string literal.

The EGA array literal is a list of the EGA values it contains, separated by commas (`,`), and wrapped by braces (`{` and `}`).

Variables

You can set a value into a variable by using the `set` special function. For example, `set(A, 123);` will create a variable named `A` whose value is `123` .

The `define` special function can store the unevaluated expression into a variable.

Integers

Expression `+(1, 2)` is the sum of two integers `1` and `2` . Expression `*(3, 4)` is the multiplication of two integers `3` and `4` .

You can compare two integers by `==` , `!=` , `<` , `<=` , `>` , `>=` functions.

Strings

Expression `"This is a string."` is a string literal of length 17.

Expression `"This is a ""string""."` is a string literal of length 19.

You can compare two strings by `==` , `!=` , `<` , `<=` , `>` , `>=` functions.

See also `left` , `len` , `mid` , `right` , `replace` , `remove` and `str` functions.

Arrays

Expression `{1, 2, "string"}` is an array literal of length 3.

Expression `set(ary, {1, 2, "string"});` can store the array to the `ary` variable.

To get the 2nd element of `ary` , use `at(ary, 1)` . To set `999` to the 2nd element of `ary` , use `at(ary, 1, 999)` .

See also `left` , `len` , `mid` , `right` , `replace` , `remove` and `array` functions.

Booleans

In EGA, the boolean value is an integer value. Zero means false. Non-zero means true.

Conditional execution

The special function `if` can switch the execution by condition.

Loops

The special functions `for`, `foreach` and `while` can make an execution loop. The `break` special function can break the loop.

Normal functions vs. special functions

In a call of the normal function, the parameters will be evaluated in the order of parameters. The special functions can change the order of expression evaluations and can ignore some parameters.

Input and output

The `input` function prompts the user for a string and waits for input. If input is done, the function returns a string.

The `print`, `println`, `dump` and `dumpln` functions shows text of the specified values to the user.

Unlike `print` and `println`, the `dump` and `dumpln` functions add quotes to the string values. `println` and `dumpln` add a newline at the end of the output text.

Samples

Sample `break.ega`

```
for(i, 1, 10000, (println(i), if(>=(i, 10), break())));
```

This EGA program will outputs 0-to-10. Output:

```
1
2
3
4
5
```

```
6
7
8
9
10
```

The `break` special function cut the `for` loop.

Sample `fact.ega`

```
define(fact, do(set(prod, 1), for(i, 2, n, set(prod, *(prod, i)))));

for(k, 1, 12, (
        set(n, k),
        fact,
        println("n = ", n, ": fact == ", prod)
    )
)
```

This EGA program will outputs the factorial numbers of 1-to-12. Output:

```
n = 1: fact == 1
n = 2: fact == 2
n = 3: fact == 6
n = 4: fact == 24
n = 5: fact == 120
n = 6: fact == 720
n = 7: fact == 5040
n = 8: fact == 40320
n = 9: fact == 362880
n = 10: fact == 3628800
n = 11: fact == 39916800
n = 12: fact == 479001600
```

The `define` special function defines a macro variable. This program is same as:

```
for(k, 1, 12, (
        set(n, k),
        do(set(prod, 1), for(i, 2, n, set(prod, *(prod, i)))),
        println("n = ", n, ": fact == ", prod)
    )
)
```

Sample `input.ega`

```
set(s, input("Type a string"));
for(i, 1, +(len(s), 4), print("#"));
println();
println(cat("# ", s, " #"));
for(i, 1, +(len(s), 4), print("#"));
println();
```

This program wraps the input string by `#`. Output:

```
Type a string? This is a test.
####################
# This is a test. #
####################
```

The `input` function prompts user input. The `cat` function concatnates strings.

Sample `plus.ega`

```
set(A, input("A="));
set(B, input("B="));
set(C, +(int(A), int(B)));
println(C);
```

This program calculates the sum of input `A` and `B`. Output:

```
A=? 3
B=? 5
8
```

The EGA Functions

The following sections are a list of the EGA functions.

EGA ' and ' Function

```
EGA function 'and':
  argument number: 2
  usage: and(value1, value2)
```

Calculates logical AND of two integers. Returns an integer.

Same as `&&`.

EGA 'array' Function

```
EGA function 'array':
  argument number: 0..256
  usage: array(value1[, ...])
```

Creates an array from specified parameters. Returns an array.

EGA 'at' Function

```
EGA function 'at':
  argument number: 2..3
  usage: at(ary_or_str, index[, value])
```

Gets or sets the item at the specified index.

`ary_or_str` must be an array or a string.

If the value is not specified, the function gets the value at the position of the specified index.

If the value is specified, the function sets the value at the position of the specified index.

Returns the value.

Same as `[]`.

EGA 'bitand' Function

```
EGA function 'bitand':
  argument number: 2
  usage: bitand(value1, value2)
```

Calculates bitwise AND of two integers. Returns an integer.

Same as `&`.

EGA ' `bitor` ' Function

```
argument number: 2
usage: bitor(value1, value2)
```

Calculates bitwise OR of two integers. Returns an integer.

Same as `|`.

EGA ' `break` ' Function

```
argument number: 0
usage: break()
```

Goes out of an EGA loop.

EGA ' `cat` ' Function

```
EGA function 'cat':
  argument number: 1..256
  usage: cat(ary_or_str_1, ary_or_str_2, ...)
```

Concatnates the specified arrays and/or strings. Returns an array or a string.

EGA ' `compare` ' Function

```
EGA function 'compare':
  argument number: 2
  usage: compare(value1, value2)
```

Compares two values. Returns 0 if `value1` and `value2` are equal, -1 if `value1` was less, or 1 if `value1` was greater.

EGA ' `compl` ' Function

```
EGA function 'compl':
  argument number: 1
  usage: compl(value)
```

Calculates bitwise NOT. Returns an integer.

Same as `~`.

EGA '`define`' Function

```
EGA function 'define':
  argument number: 1..2
  usage: define(var[, expr])
```

Defines an EGA macro variable. `var` is a variable. Unlike the `set` function, the `expr` argument will be not evaluated. If `expr` is omitted, `var` will be reset. Returns `expr`.

Same as `:=`.

EGA '`div`' Function

```
EGA function 'div':
  argument number: 2
  usage: div(int1, int2)
```

Divides an integer value `int1` by another integer value `int2`. Returns an integer.

Same as `/`.

EGA '`do`' Function

```
EGA function 'do':
  argument number: 0..256
  usage: do(expr, ...)
```

Does loop while `expr` is non-zero. The arguments will be evaluated in order. Returns the last argument. You can break the execution by `break` function.

EGA '`dump`' Function

```
EGA function 'dump':
  argument number: 0..256
  usage: dump(value, ...)
```

Outputs the values with quotations if necessary. No return.

EGA ' `dumpln` ' Function

```
EGA function 'dumpln':
  argument number: 0..256
  usage: dumpln(value, ...)
```

Same as `dump` except `dumpln` adds a newline.

Same as `?` .

EGA ' `equal` ' Function

```
EGA function 'equal':
  argument number: 2
  usage: equal(value1, value2)
```

Compares two values. Returns 1 if two values are equal. zero if not equal.

Same as `==` .

EGA ' `exit` ' Function

```
  argument number: 0..1
  usage: exit([value])
```

Ends the program with a value.

EGA ' `find` ' Function

```
EGA function 'find':
  argument number: 2
  usage: find(ary_or_str, target)
```

Finds a target value from an array or a string. Returns the zero-based offset of the found target. Returns -1 if not found.

EGA ' `for` ' Function

```
EGA function 'for':
  argument number: 4
  usage: for(var, min, max, expr)
```

Does loop from `min` and `max` .

The `expr` argument will be evaluated repeatedly. The `min` and `max` values must be integers. The `var` is the name of a loop variable.

1. At first, `min` will be stored into the `var` variable.
2. Then, `expr` will be evaluated.
3. `var` will be incremented upto `max` .
4. If `var` is less than `max` , then back to 2.

You can break the loop by `break` function.

EGA ' `foreach` ' Function

```
EGA function 'foreach':
  argument number: 3
  usage: foreach(var, ary, expr)
```

Does loop using an array. `ary` is an array. The item in the `ary` array will be evaluated and stored into variable `var` repeatedly. You can break the loop by `break` function.

EGA ' `greater` ' Function

```
EGA function 'greater':
  argument number: 2
  usage: greater(value1, value2)
```

Compares two values. Returns 1 if `value1` was greater than `value2` . zero if not greater.

Same as `>` .

EGA ' `greater_equal` ' Function

```
EGA function 'greater_equal':
  argument number: 2
  usage: greater_equal(value1, value2)
```

Compares two values. Returns 1 if `value1` was greater than `value2` or equal. Otherwise returns zero.

Same as `>=` .

EGA ' `if` ' Function

```
EGA function 'if':
  argument number: 2..3
  usage: if(cond, true_case[, false_case])
```

Chooses the execution by the condition.

If the integer value `cond` was non-zero, then `true_case` will be evaluated. If `cond` was zero, then `false_case` will be evaluated if any. Returns the evaluated value of `true_case` or `false_case` .

EGA ' `input` ' Function

```
EGA function 'input':
  argument number: 0..1
  usage: input([message])
```

Gets a text string as input from EGA console. `message` will be shown if any. Returns the text string.

EGA ' `int` ' Function

```
EGA function 'int':
  argument number: 1
  usage: int(value)
```

Converts a value to an integer value. Returns an integer.

EGA ' `left` ' Function

```
EGA function 'left':
  argument number: 2
  usage: left(ary_or_str, count)
```

Returns an array or a string of `count` items at the left side of an array or a string.

EGA ' `len` ' Function

```
EGA function 'len':
  argument number: 1
  usage: len(ary_or_str)
```

Returns the length of an array or a string.

EGA ' `less` ' Function

```
EGA function 'less':
  argument number: 2
  usage: less(value1, value2)
```

Compares two values. Returns 1 if `value1` was less than `value2` . zero if not less.

Same as `<` .

EGA ' `less_equal` ' Function

```
EGA function 'less':
  argument number: 2
  usage: less_equal(value1, value2)
```

Compares two values. Returns 1 if `value1` was less than `value2` or equal. Otherwise returns zero.

Same as `<=` .

EGA ' `mid` ' Function

```
EGA function 'mid':
  argument number: 3..4
  usage: mid(ary_or_str, index, count[, value])
```

Returns the sequance of the specified range of an array or a string. `ary_or_str` must be an array or a string. The range starts from offset `index`. The length of the range is `count`. If `value` is specified, the range will be replaced with a value of `value`.

EGA ' `minus` ' Function

```
EGA function 'minus':
  argument number: 1..2
  usage: minus(int1[, int2])
```

Negates or subtract. `int1` and `int2` must be integers. Returns an integer.

Same as `-`.

EGA ' `mod` ' Function

```
EGA function 'mod':
  argument number: 2
  usage: mod(int1, int2)
```

Calculates the remainder of division of two integers. `int2` must be non-zero. Returns an integer. Same as `%`.

EGA ' `mul` ' Function

```
EGA function 'mul':
  argument number: 2
  usage: mul(int1, int2)
```

Calculates multiplication of two integers. Returns an integer.

Same as `*`.

EGA ' `not` ' Function

```
EGA function 'not':
  argument number: 1
  usage: not(value)
```

Calculates logical NOT of the value. Returns an integer. Same as `!`.

EGA ' `not_equal` ' Function

```
EGA function 'not_equal':
  argument number: 2
  usage: not_equal(value1, value2)
```

Compares two values. Returns 1 if `value1` was different from `value2`. Otherwise returns zero. Same as `!=`.

EGA ' `or` ' Function

```
EGA function 'or':
  argument number: 2
  usage: or(value1, value2)
```

Calculates logical OR of two values. Returns an integer.

Same as `||`.

EGA ' `plus` ' Function

```
EGA function 'plus':
  argument number: 2
  usage: plus(int1, int2)
```

Calculates sum of two integer values. Returns an integer.

Same as `+`.

EGA ' `print` ' Function

```
EGA function 'print':
  argument number: 0..256
  usage: print(value, ...)
```

Outputs the values without quotation. No return.

EGA ' `println` ' Function

```
EGA function 'println':
  argument number: 0..256
  usage: println(value, ...)
```

Outputs the values without quotation with a newline. No return.

EGA ' `remove` ' Function

```
EGA function 'remove':
  argument number: 2
  usage: remove(ary_or_str, target)
```

Returns an array or a string, whose parts are removed.

If `ary_or_str` is an array, the items with the same value as `target` are removed. If `ary_or_str` is a string, the substrings `target` are removed. Returns the array or the string of the results. This function doesn't change `ary_or_str` .

EGA ' `replace` ' Function

```
EGA function 'replace':
  argument number: 3
  usage: replace(ary_or_str, from, to)
```

If `ary_or_str` is an array, every item with the same value as the `from` value are replaced with the `to` value. If `ary_or_str` is a string, the substrings `from` are replaced with the `to` string. Returns the array or the string of the results. This function doesn't change `ary_or_str` .

EGA ' `right` ' Function

```
EGA function 'right':
  argument number: 2
  usage: right(ary_or_str, count)
```

Returns an array or a string of `count` items at the right side of an array or a string.

EGA ' `set` ' Function

```
EGA function 'set':
  argument number: 1..2
  usage: set(var[, value])
```

Creates a variable whose value is `value`. If `value` is not specified, the variable is cleared. Returns the value.

Same as `=`.

EGA ' `str` ' Function

```
EGA function 'str':
  argument number: 1
  usage: str(value)
```

Converts the value to a string. Returns a string.

EGA ' `typeid` ' Function

```
EGA function 'typeid':
  argument number: 1
  usage: typeid(value)
```

Returns the type ID of the value.

If the value is NULL, then returns `-1`. If the value is an integer, then returns zero. If the value is a string, then returns `1`. If the value is an array, then returns `2`.

EGA ' `while` ' Function

```
EGA function 'while':
  argument number: 2
  usage: while(cond, expr)
```

Does loop while the specified condition is non-zero. The `expr` argument will be evaluated repeatedly. The `cond` is the condition.

1. At first `cond` will be evaluated. If it was zero, then loop will be ended.
2. `expr` will be evaluated. Back to 1.

You can break the loop by `break` function.

EGA ' `xor` ' Function

```
EGA function 'xor':
  argument number: 2
  usage: xor(value1, value2)
```

Calculates bitwise XOR of two integers. Returns an integer.

Same as `^`.

RisohEditor EGA extension

RisohEditor EGA has the following functions as EGA extension:

- `RES_search`
- `RES_delete`
- `RES_clone_by_name`
- `RES_clone_by_lang`
- `RES_unload_resh`

EGA ' `RES_search` ' Function

```
EGA function 'RES_search':
  argument number: 0..3
  usage: RES_search([type[, name[, lang]]])
```

`RES_search` returns an array of the resource items.

`type` must be an integer or a string of a resource type. If `type` is zero or omitted, then search all resource types. `name` must be an integer or a string of a resource name. If `name` is zero or omitted, then search all resource names. `lang` must be an integer that specifies the language ID. If `lang` is `-1` or omitted, then search all resource languages.

EGA ' `RES_delete` ' Function

```
EGA function 'RES_delete':
  argument number: 0..3
  usage: RES_delete([type[, name[, lang]]])
```

`RES_delete` deletes the resource items.

`type` must be an integer or a string of a resource type. If `type` is zero or omitted, then search all resource types. `name` must be an integer or a string of a resource name. If `name` is zero or omitted, then search all resource names. `lang` must be an integer that specifies the language ID. If `lang` is `-1` or omitted, then search all resource languages.

Returns `1` if deleted. Otherwise returns zero.

EGA ' `RES_clone_by_name` ' Function

```
EGA function 'RES_clone_by_name':
  argument number: 3
  usage: RES_clone_by_name(type, src_name, dest_name)
```

`RES_clone_by_name` clones the resource data as another resource name.

`type` must be an integer or a string of a resource type. If `type` is zero, then search all resource types. `src_name` must be an integer or a string of a resource name. If `src_name` is zero, then search all resource names. `dest_name` must be an integer or a string of a new resource name.

Returns `1` if cloned. Otherwise returns zero.

EGA ' `RES_clone_by_lang` ' Function

```
EGA function 'RES_clone_by_lang':
  argument number: 4
```

```
usage: RES_clone_by_name(type, name, src_lang, dest_lang)
```

`RES_clone_by_name` clones the resource data as another resource language.

`type` must be an integer or a string of a resource type. If `type` is zero, then search all resource types. `name` must be an integer or a string of a resource name. If `name` is zero, then search all resource names. `src_lang` must be an integer that specifies the source language ID. If `lang` is `-1`, then search all resource languages. `dest_lang` must be an integer that specifies the destination language ID.

Returns `1` if cloned. Otherwise returns zero.

EGA '`RES_unload_resh`' Function

```
EGA function 'RES_unload_resh':
  argument number: 0
  usage: RES_unload_resh()
```

`RES_unload_resh` unloads the "`resource.h`" file. Always returns `1`.

How can I extend EGA?

1. Import `libega`.
2. Include `ega.hpp`.
3. Call the following EGA C++ functions: `EGA_init`, `EGA_set_input_fn` and `EGA_set_print_fn`.
4. Add your EGA functions by `EGA_add_fn` C++ functions.

Special thanks

- `md2pdf` : https://md2pdf.netlify.com/