Reference to Joy of Postfix

from 2024-11-07

Subset of Joy Programming Language
with some Modifications

Original:
https://www.kevinalbrecht.com/code/joy-mirror/html-manual.html

Definition of Identifiers

identifier1 **==** *word1 word2 word3* …
identifier2 **==** *word4 word5 word6* …

Example:

```
makelist == [ ] swap [cons] times                <CALC>
10 20 30 40 50 3 makelist .s                     <CALC>
… 10 20 [30 40 50]
```

```
quote  '
comment ()    #
```

Stack Notation

word (*input parameters* --> *output parameters*)
Description of the word's functionality.

Words for the Stack

The parameter stack is a linked list.

stack (--> *list*)
Pushes the stack as a list onto the stack.

unstack (*list* -->)
The *list* becomes the new stack.

clear (*...* --> *(null)*)
Clears the stack.

dup (*x* --> *x x*)
Pushes an extra copy of x onto the stack.

pop (*x* -->)
Removes *x* from the top of the stack.

swap (*x y* --> *y x*)
Swaps *x* and *y* at the top of the stack.

over (*x y* --> *x y x*)
Gets the second value from stack.

rotate (*x y z* --> *z y x*)
Swap *x* and *z*.

rollup (*x y z* --> *z x y*)

rolldown (*x y z* --> *y z x*)

dupd (*x y* --> *x x y*)

popd (*x y* --> *y*)

swapd (*x y z* --> *y x z*)

rotated (*x y z k* --> *z y x k*)

rollupd (*x y z k* --> *z x y k*)

rolldownd (*x y z k* --> *y z x k*)

index (*... n* --> *... n*th_stack_value)
Picks a copy of the stack value with position num relative to the stack top from the stack and pushes it onto the stack;
with *n* = 1 -> first value, *n* = 2 -> second value, ...

dip (*x [program]* --> *... x*)
Stores the *x*, executes the *program*, pushes *x* back onto the stack.

dip2 (*x y [program]* --> *... x y*)
Stores the *x* and *y*, executes the *program*, pushes *x* and *y* back onto the stack.

id (-->)
Identity function, does nothing; as a placeholder for a function.

The IOMonad for Pure Functional Programming

num [program2] **' !** (iomonad behavior)
[iomonad] [program2] **' !** (iomonad behavior)

First, the primitive monad *num* or the *[iomonad]* is executed
- i.e. a side effect is triggered. Then the *[program2]* is executed.
The iomonad is **at the end of** a sequence/program.
([program2] can also be an iomonad)

Words for Input/Output

. (*value* -->)
Prints the top value from the stack. (iomonad behavior)

.s (-->)
Prints the contents of the stack. (iomonad behavior)

print (*list* -->)
print (*string* -->)
Outputs the *list* without square brackets. (iomonad behavior)
Outputs the *string* without quotation marks. (iomonad behavior)

load (*"fname"* -->)
A program text from the file *fname* from the "joy/" folder is read into the display with the definitions. (iomonad behavior)

save (*"fname"* -->)
A program text from the display is saved under the name *fname* in the "joy/" folder (iomonad behavior)

loadtext (*"fname"* --> *string*)
Loads the contents of a text file and pushes it as a *string* on the stack. (iomonad behavior)

savetext (*"fname" string* -->)
Saves the *string* as text in a text file. (iomonad behavior)

files (--> *list*)
Outputs a *list* of all file names in the "joy/" folder (iomonad behavior)

fremove (*"fname"* --> *bool*)
Deletes the file named *fname* from the "joy/" folder. (iomonad behavior)

fcopyto (*"fname1" "fname2"* -->) (iomonad behavior)

timestamp (--> *num*) (iomonad behavior)

date (--> *string*) (iomonad behavior)

words (-->)
words == identlist print (iomonad behavior)

dump (-->)
dump == identdump print (iomonad behavior)

help (-->)
help == helpinfo print (iomonad behavior)

Words for List Processing

[*value1 value2 value3* ...**]**

first (*list* --> *value*)
value is the first value of the nonempty *list*.

rest (*list1* --> *list*)
list is the remainder of the nonempty *list1* without the first value.

cons (*value1 list1* --> *list*)
the *list* is created from *list1* with new first *value1*.

swons (*list1 value1* --> *list*)
the *list* is created from *list1* with new first *value1*.

uncons (*list1* --> *value list*)
Puts the *first* and the *rest* of the nonempty *list1* on the stack.

unswons (*list1* --> *list value*)
Puts the *rest* and the *first* of the nonempty *list1* on the stack.

reverse (*list1* --> *list*)
The order of the elements of *list1* is reversed in the new *list*.

size (*list* --> *num*)
num is the number of elements in the *list*.

make ( )

take (*list1 num* --> *list*)
A *list* with the first *num* elements of *list1*.

drop (*list1 num* --> *list*)
A *list* without the first *num* elements of *list1*.

concat (*list1 list2* --> *list*)
The *list* is the concatenation of *list1* and *list2*.

swoncat (*list1 list2* --> *list*)
The *list* is the concatenation of *list2* and *list1*.

enconcat ( )

last (*list1* --> *element*)

init (*list1* --> *list*)

iota (*num* --> *list*)
Generates a *list* of numbers from 1 to *num*.

fromto (*num1 num2* --> *list*)
Generates a *list* of numbers from *num1* to *num2*

at (*list num* --> *elementvnum*)
Picks the *elementvnum* from the *list*.

of (*num list* --> *elementvnum*)

set (*list1 num value* --> *list*)

find (*list key* --> *num*)

count (*list key* --> *num*)

pair (*value1 value2* --> *[value1 value2]*)

unpair (*[value1 value2]* --> *value1 value2*)

trans (*matrix1* --> *matrix*) (matrix = list of list)

Words for Processing Dict Lists

[*key1 value1 key2 value2 ... ...***]**

get (*dict key* --> *value*)
Gets the *value* for the *key* from the *dict*.

put (*dict1 key value* --> *dict*)
Creates a new *value* for the *key* in a *dict* with *dict1* as a copy.

.

.

.

Mathematical Functions

+ (*num1 num2* --> *num*)
num is the result of adding *num1* and *num2*.

- (*num1 num2* --> *num*)
num is the result of subtracting *num2* from *num1*.

***** (*num1 num2* --> *num*)
× (*num1 num2* --> *num*)
num is the product of *num1* and *num2*.

/ (*num1 num2* --> *num*)
÷ (*num1 num2* --> *num*)
num is the quotient of *num1* divided by *num2*.

mod (*num1 num2* --> *num*)
rem (*num1 num2* --> *num*)
Modulo or Remainder.

reci (*num1* --> *num*)
num is the reciprocal of *num1*

pow (*num1 num2* --> *num*)
Power to the Bauer

root (*num1 n* --> *num*)
*n*th root of *num1*

pred (*num1* --> *num*)
Predecessor function.

succ (*num1* --> *num*)
Successor function.

sign (*num1* --> *num*)
Signum function.

abs (*num1* --> *num*)
Absolute function.

neg (*num1* --> *num*)
num is the negative value of *num1*.

floor (*num1* --> *num*)
Rounding down the number.

ceil (*num1* --> *num*)
Round up the number.

trunc (*num1* --> *num*)
Integer value with truncation of the decimal places.

int (*num1* --> *num*)
num is the integer part of *num1*.

frac (*num1* --> *num*)
Fraction part of the number.

round (*num1* --> *num*)
Rounds to an integer value

roundto (*num1 fix* --> *num*)
Rounds to the *fix*-th decimal place.

exp (*num1* --> *num*)
Exponential function of the number.

log (*num1* --> *num*)
Natural logarithm of the number.

log10 (*num1* --> *num*)
Ten logarithm of the number.

log2 (*num1* --> *num*)
Dual logarithm of the number.

fact (*num1* --> *num*)
num is the Factorial of *num1*.

pi (--> 3.141592653589793)
Ludolf number (Circle number).

sin (*num1* --> *num*)
num is the sine of *num1* angle in radians.

cos (*num1* --> *num*)
num is the cosine of *num1* angle in radians.

tan (*num1* --> *num*)
Tangent function of the number in radians.

asin (*num1* --> *num*)
Arcsine function.

acos (*num1* --> *num*)
Arccosine function.

atan (*num1* --> *num*)
Arc tangent function.

y x **atan2** -- *num*
Phase (or Arg) to (*x*,*y*)

num1 **sinh** -- *num*
Hyperbolic sine function.

num1 **cosh** -- *num*
Hyperbolic cosine function.

num1 **tanh** -- *num*
Hyperbolic tangent function.

num1 **sq** -- *num*
num is the square of *num1*.

num1 **sqrt** -- *num*
num is the square root of *num1*.

num1 **cbrt** -- *num*
num is the cube root of *num1*.

num1 **deg** -- *num*
Radiant value is converted to degree value.

num1 **rad** -- *num*
Degree value is converted to radian value.

[num1 num2 ... numn] **sum** -- *num*
Sum of all elements of the list.

[num1 num2 ... numn] **prod** -- *num*
Product of all elements of the list.

Logical Functions

true and false are of type bool

true -- *true*
Pushes the value *true* onto the stack.

false -- *false*
Pushes the value *false* onto the stack.

bool1 **not** -- *bool*
Logical negation for truth values.

bool1 bool2 **and** -- *bool*
Logical conjunction for truth values.

bool1 bool2 **or** -- *bool*
Logical disjunction for truth values.

bool1 bool2 **xor** -- *bool*
Exclusive-OR operation for truth values.

data1 data2 **=** -- *bool*
Checks if *data1* is equal to *data2* and pushes the *bool* value onto the stack.

data1 data2 **<>** -- *bool*
data1 data2 **!=** -- *bool*
Checks for inequality.

data1 data2 **<** -- *bool*
Compare to less than.

data1 data2 **>** -- *bool*
Compare to greater-than.

data1 data2 **<=** -- *bool*
Comparison on less than or equal.

data1 data2 **>=** -- *bool*
Greater-equal comparison.

num **small** -- *bool*
list **small** -- *bool*

data1 **null** -- *bool*

data1 **list** -- *bool*

data1 **leaf** -- *bool*

data1 **consp** -- *bool*

data1 **bool** -- *bool*

data1 **ident** -- *bool*

data1 **float** -- *bool*

data1 **string** -- *bool*

data1 **undef** -- *bool*

ident1 **user** -- *bool*

data1 **type** -- *ident*
?

x list **in** -- *bool*

list x **has** -- *bool*

data1 data2 **min** -- *data*
Minimum of *data1* and *data2*.

data1 data2 **max** -- *data*
Maximum of *data1* and *data2*.

list **qsort** -- *list*
Recursive Quicksort.

String Functions

string1 num1 num2 **substr** -- *string*
Copies a sub*string* from *string1*.

string1 num **leftstr** -- *string*

string1 num **rightstr** -- *string*

string sub **indexof** -- *num*
Searches the position of *sub*str in the *string* from the left.

string **size** -- *num*
Specifies the length of the *string*.

string1 **upper** -- *string*
Converts the *string* to uppercase.

string1 **lower** -- *string*
Converts the *string* to lowercase.

string1 **capitalize** -- *string*
Converts the *string* into a capital word.

string1 **trim** -- *string*
Cuts off the spaces left and right.

string1 **triml** -- *string*
Cuts off the spaces on the left.

string1 **trimr** -- *string*
Cuts off the spaces on the right.

string1 pre **trimpre** -- *string*

num **chr** -- *string*
Produces a character according to the Unicode value.

string **ord** -- *num*
Specifies the Unicode value of the first character.

string1 old new **replace** -- *string*

string1 old new **replace1** -- *string*

string sep **split** -- *list*
Breaks the *string* into a *list* of strings without *sep*.

list sep **join** -- *string*
Connects the strings of the *list* with *sep* in between.

string **unpack** -- *list*
Breaks the *string* into a *list* of individual characters.

list **pack** -- *string*
Concatenates the strings of the *list* into a total *string*.

string **parse** -- *list*
Converts the string representation into a list of internal representations.

data **tostr** -- *string*
Converts the *data* value into a *string* representation.

string **toval** -- *data1*
Converts numbers, words, lists in the *string* into *data1*.

string **trytoval**

string **strtod** -- *num*

num **timeformat** -- *string*

Words for Flow Control and Combinators

' *identifier* -- *identifier*
The identifier following the quote is pushed onto the stack.

[program] **i** -- ...
Executes the program.

x [program] **dip** -- ... *x*
Stores the value *x*, executes the program, pushes value *x* back onto the stack.

x y [program] **dip2** -- ... *x y*
Stores the *x* and *y*, executes the program, pushes the *x* and *y* back onto the stack.

nullary

<stack> [... x ***return*** *... y]* **do** -- *<stack> x*
<stack> [... y] **do** -- *<stack> y*

bool [then] [else] **if** -- ...
If *bool* = true -> *then* is executed;
if *bool* = false -> *else* is executed.

bool [then] [else] **branch** -- ... *like **if**

[bool] [then] [else] **ifte** -- ...
If *bool* = true -> *then* is executed;
if *bool* = false -> *else* is executed.

*bool value**t** value**e*** **choice** -- *value*

valuei [[value1 rest1...] [value2 rest2...] ... [valuen restn...]] **case** -- *[resti...]* **i**

*[[[bool1] then1...] [[bool2] then2...] ... [[booln] thenn...] [**true** else...]]* **cond** -- ...

num [program] **times** -- ...
The *program* is executed *num* times.

[test] [program] **while** -- ...
If executing test evaluates to true, the program is executed and repeated
until test evaluates to false.

[... ***break*** *...]* **loop** -- ...

list [program] **step** -- *...*

list1 [program] **map** -- *list*

list zero [program] **fold** -- *cross-result*

list [predicate] **filter** -- *list*

list [predicate] **split2** -- *list1 list2*

x [program1] [program2] **cleave** -- *result1 result2*

x [init] [operand] **primrec** -- *result*

tailrec

genrec

linrec

binrec

[program] **Y** -- ...
Y-Combinator in Joy

[program] **try**

x [then] [else] **ifnull** -- ...
x [then] [else] **iflist** -- ...
x [then] [else] **ifcons** -- ...
x [then] [else] **ifbool** -- ...
x [then] [else] **ifident** -- ...
x [then] [else] **iffloat** -- ...
x [then] [else] **ifstring** -- ...
x [then] [else] **ifundef** -- ...

Misc Functions

data1 **type** -- *ident*
?

ident **name** -- *string*
Extracts the *string* of the *ident*.

ident **body** -- *num* | *list* | undef
Extracts the definition value of the *ident*.

ident **info** -- *string*
Extracts the compiler-*string* of the *ident*.

string **intern** -- *ident*
Pushes the *ident* whose name is *string*.

ident **user** -

ident **bound** -

identlist -- *list*
list of all used identifiers.

identdump -- *string*

helpinfo -- *string*
Information on where to find help on the Internet.

gc --
Forces a garbage collection that otherwise only occurs spontaneously.

abort >>> *exception*
Aborts the execution of the current Joy program with an *exception*.

string **error** >>> exception

undefined >>> exception

eof