Reference to Joy of Postfix

from 2024-11-22

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.

collect (*value1 value2 ... valuen n* --> *[value1 value2 ... valuen]*)

copy (*value1 value2 ... valuen n* --> *value1 value2 ... valuen value1 value2 ... valuen*)

newdict (*x1 x2 ... xn [i1 i2 ... in]* --> *dict*)

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)

viewurl (*string* -->) (iomonad behavior)
Displays the web page for the URL in the web browser.

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

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

help (-->)
help == helpinfo viewurl (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 (*data num* --> *list*)

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 lists)

zip (*list1 list2* --> *matrix*) (matrix = list of pairs)

unlist (*[x1 x2 ... xn]* --> *x1 x2 ... xn*)

cat (*list1 list2* --> *list*)
*like concat

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.

newdict (*x1 x2 ... xn [i1 i2 ... in]* --> *dict*)

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.

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

sinh (*num1* --> *num*)
Hyperbolic sine function.

cosh (*num1* --> *num*)
Hyperbolic cosine function.

tanh (*num1* --> *num*)
Hyperbolic tangent function.

sq (*num1* --> *num*)
num is the square of *num1*.

sqrt (*num1* --> *num*)
num is the square root of *num1*.

cbrt (*num1* --> *num*)
num is the cube root of *num1*.

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

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

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

prod (*[num1 num2 ... numn]* --> *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.

not (*bool1* --> *bool*)
Logical negation for truth values.

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

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

xor (*bool1 bool2* --> *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.

small (*num* --> *bool*)
small (*list* --> *bool*)

null (*data1* --> *bool*)

list (*data1* --> *bool*)

leaf (*data1* --> *bool*)

consp (*data1* --> *bool*)

bool (*data1* --> *bool*)

ident (*data1* --> *bool*)

float (*data1* --> *bool*)

string (*data1* --> *bool*)

undef (*data1* --> *bool*)

user (*ident1* --> *bool*)

type (*data1* --> *ident*)
?

in (*x list* --> *bool*)

has (*list x* --> *bool*)

min (*data1 data2* --> *data*)
Minimum of *data1* and *data2*.

max (*data1 data2* --> *data*)
Maximum of *data1* and *data2*.

qsort (*list1* --> *list*)
Recursive Quicksort.

String Functions

concat (*string1 string2* --> *string*)

cat (*string1 string2* --> *string*)
*like concat

midstr (*string1 num1 num2* --> *string*)
Copies a sub*string* from *string1*.

leftstr (*string1 num* --> *string*)

rightstr (*string1 num* --> *string*)

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

size (*string* --> *num*)
Specifies the length of the *string*.

upper (*string1* --> *string*)
Converts the *string* to uppercase.

lower (*string1* --> *string*)
Converts the *string* to lowercase.

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

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

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

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

trimpre (*string1 pre* --> *string*)

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

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

replace (*string1 old new --> string*)

replace1 (*string1 old new --> string*)

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

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

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

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

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

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

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

trytoval (*string*)

strtod (*string --> num*)

timeformat (*num --> string*)

Words for Flow Control and Combinators

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

i (*[program]* --> *...*)
Executes the program.

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

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

nullary ()

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

infra (*list1 [program]* --> *list2*)

unary (*x1 [program]* --> *r1*)
unary2 (*x1 x2 [program]* --> *r1 r2*)
unary3 (*x1 x2 x3 [program]* --> *r1 r2 r3*)
unary4 (*x1 x2 x3 x4 [program]* --> *r1 r2 r3 r4*)

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

branch (*bool [then] [else]* --> *...*)
*like **if**

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

choice (*bool value-t value-e* --> *value*)

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

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

times (*num [program] --> ...*)
The *program* is executed *num* times.

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

loop (*[... **break** ...] --> ...*)

step (*list [program] --> ...*)

map (*list1 [program] --> list*)

fold (*list zero [program] --> cross-result*)

filter (*list [predicate] --> list*)

split2 (*list [predicate] --> list1 list2*)

constr1 (*x [[p1] [p2] ... [pn]] --> list*)
?

cleave (*x [program1] [program2] --> result1 result2*)

primrec (*x [init] [oprand] --> result*)

tailrec ()

genrec ()

linrec ()

binrec ()

Y (*[program] --> ...*)
Y-Combinator in Joy

try (*[program]*)

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 (*x [then] [else] --> ...*)

Misc Functions

type (*data1* --> cons | ident | float | string | bool | null | "Int" | "Long" | undef)

name (*ident* --> *string*)
Extracts the *string* of the *ident*.

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

info (*ident* --> *string*)
Extracts the compiler-*string* of the *ident*.

intern (*string* --> *ident*)
Pushes the *ident* whose name is *string*.

user (*ident* --> *bool*)

bound (*ident* --> *bool*)

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*.

error (*string* >>> *exception*)

undefined (>>> *exception*)