Reference to Joy of Postfix

from 2024-10-27

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 '
notation -- erklären

Words for the Stack

The parameter stack is a linked list.

stack -- *list*
Pushes the stack as a list onto the stack.

list **unstack**
The *list* becomes the new stack.

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

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

x **pop** --
Removes *x* from the top of the stack.

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

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

x y z **rotate** -- *z y x*
Swap *x* and *z*.

x y z **rollup** -- *z x y*

x y z **rolldown** -- *y z x*

dupd --

popd --

swapd --

rotated --

rollupd --

rolldownd --

... *n* **index** -- *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, ...

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

x y [program] **dip2** -- ... *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 Monad for Pure Functional Programming

num [program] **' !** (monad behavior)
[monad] [program] **' !**

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

Words for Input/Output

value **.**	--		
Prints the top value from the stack.			(monad behavior)
.s	--		
Prints the contents of the stack.			(monad behavior)
list **print**	--		
string **print**	--		
Outputs the *list* without square brackets.			(monad behavior)
Outputs the *string* without quotation marks.			(monad behavior)
fname **load**	--		
fname **save**	--		
fname **loadtext**	--	*string*	
Loads the contents of a text file and pushes it as a *string* on the stack.			(monad behavior)
fname string **savetext**	--		
Saves the *string* as text in a text file.			(monad behavior)
files	--	*list*	(monad behavior)
fname **fremove**	--	*bool*	(monad behavior)
fname1 fname2 **fcopyto**	--		(monad behavior)
timestamp	--	*num*	(monad behavior)
date	--	*string*	(monad behavior)
words	--	identlist print	(monad behavior)
dump	--	identdump print	(monad behavior)
help	--	helpinfo print	(monad behavior)

Words for List Processing

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

list **first** -- *value*
value is the first value of the nonempty *list*.

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

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

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

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

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

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

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

make

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

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

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

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

enconcat

list1 **last** -- element

list1 **init** -- *list*

num **iota**	--	*list*

Generates a *list* of numbers from 1 to *num*.

fromto

list num **at**	--	*elementvnum*

Picks the *elementvnum* from the *list*.

of

.

find

count

value1 value2 **pair**	--	*[value1 value2]*

[value1 value2] **unpair**	--	*value1 value2*

Words for Processing Dict Lists

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

dict key **get**	--	*value*

Gets the *value* for the *key* from the *dict*.

dict1 key value **put**	--	*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*.

num1 num2 **mod** -- *num*
num1 num2 **rem** -- *num*

num1 **reci** -- *num*

num1 num2 **pow** -- *num*

num1 num2 **root** -- *num*

num1 **pred** -- *num*

num1 **succ** -- *num*

num1 **sign** -- *num*

num1 **abs** -- *num*

num1 **neg** -- *num*
num is the negative value of *num1*.

num1 **floor** -- *num*

num1 **ceil** -- *num*

num1 **trunc** -- *num*

num1 **int** -- *num*
num is the integer part of *num1*.

num1 **frac** -- num

num1 **round** -- *num*

num1 *fix* **roundto** -- *num*

num1 **exp** -- *num*

num1 **log** -- *num*

num1 **log10** -- *num*

num1 **log2** -- *num*

num1 **fact** -- *num*

pi -- 3.141592653589793

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

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

num1 **tan** -- *num*

num1 **asin** -- *num*

num1 **acos** -- *num*

num1 **atan** -- *num*

numy numx **atan2** -- *num*

num1 **sinh** -- *num*

num1 **cosh** -- *num*

num1 **tanh** -- *num*

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*

data1 data2 **<** -- *bool*

data1 data2 **>** -- *bool*

data1 data2 **<=** -- *bool*

data1 data2 **>=** -- *bool*

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*
data1 data2 **max**	--	*data*
list **qsort**	--	*list*
Recursive Quicksort.		

String Functions

string1 num1 num2 **substr**	--	*string*
string1 num **leftstr**	--	*string*
string1 num **rightstr**	--	*string*
string1 sub **indexof**	--	*num*
string1 **upper**	--	*string*
string1 **lower**	--	*string*
string1 **capitalize**	--	*string*
string1 **trim**	--	*string*
string1 **triml**	--	*string*
string1 **trimr**	--	*string*
string1 pre **trimpre**	--	*string*
num **chr**	--	*string*
string **ord**	--	*num*
string1 old new **replace**	--	*string*
string1 old new **replace1**	--	*string*

string sep **split** -- *list*

list sep **join** -- *string*

string **unpack** -- *list*

list **pack** -- *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*

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*
ident **body**	--	*num* | *list* | undef
ident **info**	--	*string*
intern	-	
ident **user**	-	
ident **bound**	-	
identlist	--	*list*

List of identifiers used.

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
string **error**	>>>	exception
undefined	>>>	exception

eof