Obfuscated Weird Language

From Wikipedia, the free encyclopedia

Obfuscated Weird Language, or **OWL**, is an esoteric programming language developed by Antonio Maschio between 2005 and 2006. It is based on Forth and FALSE and is an interpreted, stack-based language. An OWL program is a sequence of ASCII characters, hereafter called the *code stream*.

Contents

Language summary

This summary applies to version 0.7.2 (February 11, 2006) of the OWL language.

OWL has the following datatypes: 32-bit integer; string of 8-bit characters; and OWL function. Each data type is entered and stored differently.

- 32-bit integers are stored on the integer stack, which can hold up to 1024 integers. In addition, there are 26 integer variables, named *A* through *Z*. Upon being encountered in the OWL code stream, integers are pushed on the integer stack.
- Strings are stored in the PAD, which is a buffer which can store up to 1024 8-bit characters. Upon being encountered in the OWL code stream, a string is stored in the PAD, and may optionally be printed.
- OWL functions are stored in a function buffer which can hold up to 2 functions. In addition, there are 26 function variables, named *a* through *z*. An OWL function consists of a sequence of up to 1024 ASCII characters. Upon being encountered in the OWL code stream, OWL functions are entered into the function buffer. The contents of the PAD may also be treated as a function. Finally, there is a *function index*, which may hold the name of any of the 26 function variables.

Also, each data type admits a different set of operations.

- For the integer stack and integer variables, the following operations are supported:
  - Pushing integers on the stack, as above.
  - Integer arithmetic operations on the stack, including powers and roots.
  - Bitwise logical operations on the stack.
  - Integer comparisons on the stack; true comparisons return -1, while false comparisons return 0.
  - Stack manipulation operations.
  - Reading in an integer from standard input to the stack, and printing an integer from the stack to standard output.
  - Reading 8-bit characters from standard input to the stack, and writing 8-bit characters from the stack to standard output.
  - Reading the value of an given integer variable to the stack, and writing the value of an given integer variable from the stack.
- For the PAD, the following operations are supported:
  - Storing strings in the PAD, as above, and possibly printing them while doing so.

    - Reading a string from standard input into the PAD, and writing a string from the PAD to standard output.
    - Reading 8-bit characters from the PAD to the integer stack, and writing 8-bit characters to the PAD from the integer stack.
    - Treating the contents of the PAD as a function and writing it into a given function variable.
    - Treating the contents of the PAD as a function and executing it.
- For the function buffer, function variables and function index, the following operations are supported:
    - Entering functions into the buffer, as above.
    - If-then, for a function in the buffer; the condition is taken from the integer stack.
    - If-then-else, for a pair of functions in the buffer; the condition is taken from the integer stack.
    - Repeat until true, for a function in the buffer.
    - While-do, for a pair of functions in the buffer.
    - Writing the value of a given function variable from the buffer.
    - Execution of the value of a given function variable.
    - Setting the value of the function index to the name of a given function variable.
    - Execution of the value of the function variable denoted by the function index.

As well as the above operations, OWL implements various system commands.

The function buffer is cleared after each use.

Integer variables are initially 0; function variables are initially the empty string.

Language syntax

In the following table of syntax examples, we will write, for example, C B A → D E for an operation which first pops operands A, B and C from the stack and then pushes results D and E onto the stack. If an operation assumes that the function buffer contains an operand X (or two operands, X and Y), we will write {X} (or {X Y}.) If an operation requires one operand in the function buffer, but two are provided, the second is used and the first is discarded. Recall that the function buffer is cleared after each use.

This table applies to version 0.7.2 of the OWL language.

Data entry.	
0x100	These all push the number *256* onto the stack. Numbers can be written in hexadecimal, decimal, octal, or binary, as shown.
256	
O400	
B100000000	
D	This will push *68*, the ASCII value of *D*, onto the stack. This can be done for any uppercase or lowercase letter.
"ab\ncd"	Prints the characters *a*, *b*, linefeed, *c*, and *d*, and stores *a*, *b*, linefeed, *c*, and *d* into the first five characters of the PAD, in that order.
"abcd""	Stores *a*, *b*, *c*, and *d* into the first four characters of the PAD, in that order, and prints nothing.
[10 [3 +] +]	Places the function *10 [3 +]* + into the function buffer.
[10 [3 +] +] [20 +]	Places the functions *10 [3 +]* + and *20* + into the function buffer.
Arithmetic and logical operations.	
+	Addition: a b → a+b.
-	Subtraction: a b → a-b.
*	Multiplication: a b → ab.

/	Division : a b → a/b. The quotient is always an integer.
^	Exponentiation: a b → a^b. The result is always an integer.
:	Root extraction: a b → $a^{1/b}$. The result is rounded to an integer.
\	Negate: a → -a.
=	Equality comparison: a b → c, where c is -1 if a and b are equal and 0 if a and b are unequal.
>	Greater-than comparison: a b → c, where c is -1 if a is greater than b and 0 if it is not.
~	Logical NOT: a → b, where b is -1 if a is zero and 0 if a is nonzero.
&	Bitwise AND: a b → $a \wedge b$.
|	Bitwise OR: a b → $a \vee b$.
Stack manipulation.	
$	Swap: a b → b a.
%	Dup: a → a a.
;	Drop: a →.
'	Roll: $a_n … a_1\ a_0\ b → a_{n-1} … a_0\ a_n$, if b has value n.
`	Pick: $a_n … a_1\ a_0\ b → a_n … a_0\ a_n$, if b has value n.
Variable manipulation.	
D,	Store into integer variable *D*: a →, and a is stored in *D*.
G@	Read from integer variable *G*: → a, where a was stored in *G*.
d,	Store function buffer into function variable *d*: {X}, and X is stored in *d*.
e@,	Set the function index to the name of function variable *e*.
@@	Execute the function in the function variable named by the function index.
f@	Execute the function in function variable *f*.
d_,	Copy the PAD into function variable *d*.
Control flow manipulation.	
! *(with one function in the function buffer)*	Repeat…until true: {X}, and X is repeatedly executed and a value a popped from the stack until a is nonzero.
! *(with two functions in the function buffer)*	While…do: {X Y}; X is executed, and a value a is then popped from the stack; while a is nonzero, Y is executed, X is executed again, and a is popped again.
? *(with one function in the function buffer)*	If…then: {X}, a →, and if a was nonzero, X is executed.
? *(with two functions in the function buffer)*	If…then…else: {X Y}, a →; X is executed if a is nonzero, and Y is executed if a is zero.
?!	These both terminate execution.
!?	
Input and output.	
(	Getchar: → a, where a is the value of a character read from standard input.
)	Putchar: a →, and the character with value a is printed to standard output.

<	Read: → a, where a is an integer parsed from a line which is read from standard input. The input format is the same as for integers which occur in the OWL code stream.
.	Print: a →, and a is printed to standard output in the current output base. Possible output bases are binary, octal, decimal, and hexadecimal.
{	Read string: one line of input is read from standard input into the PAD. When stored into the PAD, the line is terminated with a null character.
}	Print string: the contents of the PAD is treated as a null-terminated string and printed on standard output.
Miscellaneous functions.	
,	Store into PAD: a b →, and a character with value a is stored into the bth entry of the PAD.
@	Load from PAD: a → b, where b is the value of the ath character in the PAD.
_@	Execute the function stored in the PAD.
System commands.	
_OS	Push operating system type: → a, where a is 0 if the OWL interpreter is running on Unix and 1 if it is running on Windows.
_v	Push version number: → c b a, if the OWL interpreter has version number a.b.c.
_b	Set output base to binary.
_o	Set output base to octal.
_d	Set output base to decimal.
_h	These both set the output base to hexadecimal.
_x	
_t	*(Unix only)* Time: → y m d s m h, where y, m, and d are the year, month, and day of the current local date, and h, m, and s are the hour, minute, and second of the current local time.
_ty	*(Unix only)* Time components: These seven commands push the year, month, day, hour, minute, second, and millisecond of the current local time, respectively.
_tM	
_td	
_th	
_tm	
_ts	
_tn	
Metalinguistic constructions.	
# This is a comment.	This is a comment which extends to the end of the line.
(* comment starts comment, line 1 comment, line 2 … comment ends *)	This is a multiline comment.
]inc.owl[	Read and execute the OWL source file *inc.owl*.

Language examples

- 2 2 + .
  - Prints the value of 2+2 (which is 4.)

- "Hello, world!\n"
  - Prints *Hello, world!*.
- 0A,1B,"Powers of 2:\n"["2^"A@."="B@."\n"B@2*B,A@1+A,A@20=]!
  - Prints out the first 20 powers of 2 (starting at 2^0.)

Implementation

OWL is written in C, using GNU extensions.

See also

- Forth programming language
- FALSE programming language
- Esoteric programming language
- Stack-oriented programming language

External links

- OWL manual (http://it.geocities.com/tonibin/owl/doc/manual.txt)
- OWL web page (http://it.geocities.com/tonibin/owl/owl.html)

Retrieved from "http://en.wikipedia.org/wiki/Obfuscated_Weird_Language"

Categories: Esoteric programming languages | Stack-oriented programming languages

---