Language Is Simply Perfect

- Category: rev
- Suggested Points: ???
- Distribute: ???

Description

A source code reversing challenge written in ANSI Common LISP. The solver needs to figure out what the code does, which two integers should take the place of two symbols and then reverse/decipher and/or run the corrected code in a LISP Interpreter such as SBCL or CLISP.

Challenge code to show

```
(defun &e(n)(if(< n |)n(+ (&e(- n 1))(&e(- n |)))))(&e(*(let((% |)(@ $))(* % @))(let((^ |)(! $))(+ ^ !))))
```

Challenge public text

> "During our CS archaeology quest we stumbled across this partially corrupted magnetic tape that seems to contain some code of the ancients. Apparently during the read-out two numbers have been replaced with the non-executing characters '|" and '$'. Which integers do these two need to become to make the code return the value 832040?"
>
> Write the answer as flag{a,b} where “a” is the integer representing '|' and “b” is the integer representing '$'.

Deployment

n/a, just show the source code and challenge text

Flag

flag{2,3}. (I guess this is too simple, not?)

Solution

Challenge explanation:

A simple code reversing challenge written in ANSI Common LISP.

The code has two parts:

1. the definition of a function that determines the nth number of the fibonacci series
2. calling of the defined function with inline declaration of two variable via simple calculation results.

Code breakdown:

1. Function definition for the nth fibonacci number:

```
;nth fibonacci number
(defun fib (n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

In LISP variable or function names can "consist of any number of alphanumeric characters other than whitespace, open and closing parentheses, double and single quotes, backslash, comma, colon, semicolon and vertical bar."

Thus we can write the above function as:

```
(defun & (n)
  (if (< n 2)
      n
      (+ (& (- n 1))
         (& (- n 2)))))
```

with the function name 'fib' replaced with '&'

In compressed form, removing unnecessary whitespace it becomes:

```
(defun &(n)(if(< n 2)n(+ (&(- n 1))(&(- n 2)))))
```

Calling the function with the value '30' gives:

```
(& 30) => 832040
```

which is the 30th number in the fibonacci series.

2. We can then further breakdown the number 30 into the product of two integers 5 and 6. So calling the function with:

`(& (* 5 6))` also gives 832040.

[Note that LISP uses prefix notation for calculations, so 5*6=30 becomes '* 5 6' in LISP.]

In LISP we can declare variables with the let command like this:

```
(let (
      (x 1)
      (y 5)
     )
     (+ x y)
)
```

which, when compressed becomes this:

```
(let((x 1)(y 5))(+ x y))
```

and executes as the result '6'. Or we can get 6 by writing

```
(let((% 2)(@ 3))(* % @))
```

again replacing the clearly readable x and y variable names with more obscured characters '%' and '@'. We can then do the same for the number 5 also with obscure characters as variable names:

```
(let((^ 2)(! 3))(+ ^ !))
```

So with these representations of the numbers 5 and 6 we can write our 5*6=30 {or in LISP (*5 6)} as:

```
( * (let((% 2)(@ 3))(* % @)) (let((^ 2)(! 3)) (+ ^ !)) ) => 30
```

We can then plug this into the ‘`(& (* 5 6))`’ function call such as:

```
(& (* (let((% 2)(@ 3))(* % @)) (let((^ 2)(! 3))(+ ^ !))))
```

Putting the two parts together we get:

1. the compressed function definition:

```
(defun &(n)(if(< n 2)n(+ (&(- n 1))(&(- n 2)))))
```

2. the compressed code calling of the function with the obscured number 30:

```
(&(*(let((% 2)(@ 3))(* % @))(let((^ 2)(! 3))(+ ^ !))))
```

resulting in the code string:

```
(defun &(n)(if(< n 2)n(+ (&(- n 1))(&(- n 2)))))(&(*(let((% 2)(@ 3))(* % @))(let((^ 2)(! 3))(+ ^ !))))
```

In accordance with the challenge question we then replace the number '2' with '|' and the number '3' with '$' we get as the challenge code:

```
(defun &(n)(if(< n |)n(+ (&(- n 1))(&(- n |)))))(&(*(let((% |)(@ $))(* % @))(let((^ |)(! $))(+ ^ !))))
```

Of course this code does not run as is since the two masking symbols '|' and '$' will have to be replaced with the correct integers first in order to run the code.

The above code executes with SBCL (http://www.sbcl.org) on MacOS as shown below. It should be tested with CLISP as well to assure accuracy. SBCL can be installed on MacOS ((https://lisp-lang.org/learn/getting-started/)) with

```
$ brew install sbcl
```

or on Ubuntu with

```
$ sudo apt-get install sbcl
```

As an alternative, or add-on, the challenge code could also be shown in hexadecimal representation of the ASCII characters.

Learnings from this challenge:
- about the fibonacci number series
- about the LISP Language syntax and the LISP Language in general
- about hexadecimal representation of ASCII code (if alternative is used)
- about prefix notation (https://en.wikipedia.org/wiki/Polish_notation)
- about recursive function calls

Author
Wolfgang R. von Stuermer, OSIRIS Lab, NYU Tandon, wvs215@nyu.edu, September 2020.

Proof
>>> Terminal output (* is the LISP command prompt):

```
Aeons-MacBook-Pro:LISP aeonflux$ sbcl
This is SBCL 2.0.8, an implementation of ANSI Common Lisp.
More information about SBCL is available at <http://www.sbcl.org/>.

SBCL is free software, provided as is, with absolutely no warranty.
It is mostly in the public domain; some portions are provided under
BSD-style licenses.  See the CREDITS and COPYING files in the
distribution for more information.
* (defun &(n)(if(< n 2)n(+ (&(- n 1))(&(- n 2)))))(&(*(let((% 2)(@ 3))(* % @))(let((^ 2)(! 3))(+
^ !))))
&
* 832040
*
```

```
Aeons-MacBook-Pro:LISP aeonflux$ sbcl
This is SBCL 2.0.8, an implementation of ANSI Common Lisp.
More information about SBCL is available at <http://www.sbcl.org/>.

SBCL is free software, provided as is, with absolutely no warranty.
It is mostly in the public domain; some portions are provided under
BSD-style licenses.  See the CREDITS and COPYING files in the
distribution for more information.
* (defun &(n)(if(< n 2)n(+ (&(- n 1))(&(- n 2)))))
&
* (& 30)
832040
* (& (* 5 6))
832040
* (let((x 1)(y 5))(+ x y))
6
* (let((% 2)(@ 3))(* % @))
6
* (let((^ 2)(! 3))(+ ^ !))
5
* ( * (let((% 2)(@ 3))(* % @)) (let((^ 2)(! 3)) (+ ^ !)) )
30
* (& (* (let((% 2)(@ 3))(* % @)) (let((^ 2)(! 3))(+ ^ !))))
832040
*
```

SCREENSHOTS

Full code run:

Code breakdown: