HYD

File Format

Description

The HYD file format is a compiled version of HYD code. This compiled code is used by the hydreigon interpreter to execute the HYD code.

Header

Every HYD file starts with a header, it is 4 bytes with the values as follows:

In decimal: 127, 72, 89, 68

In hexadecimal: 7F, 48, 59, 44

In char: DEL, H, Y, D

Hexdump View:

```
$ hexdump -C file.hyd
00000000  7f 48 59 44
```

After the header, the HYD file goes straight to the code.

Variable types

All variables start with a byte detailing their size (byte detailing the size of the byte not included).

If the variable is a pointer to another variable, the first bit of its size will be one to tell the interpreter to go to that location to find the wanted value.

Characters are stored as a byte, so we store them as their ascii value, so ‘a’ will be 61 in hexadecimal or 97 in decimal.

Integers are stored as 4 bytes, we do not take unsigned integers into consideration, so we take integers from -2,147,483,648 to 2,147,483,647. Instead of putting the length of an int in the byte before the int,

instead put the 126 decimal value in the byte to indicate that it's an integer and not a 4 character long string, this is important for printing integers correctly.

Booleans are represented just by one bit; however, they take a byte of space. A True has a value of 1

whereas False has a value of 0.

Functions

To call a function, just put the function call keyword followed by the function's pointer.

The function call keyword is as follows:

In decimal: 13, 14, 15

In hexadecimal: 0d, 0e, 0f

If it has any parameters, put them after the function's pointer, then put an empty byte so the interpreter can know that was the last parameter.

Note that before the function's pointer as well as in front of all the variables you need to put a byte in front to signify their size as well as whether they are a pointer (as described in variable types).

Creating functions

If your function takes parameters, then your function will start with a byte giving the number of parameters your function takes followed by the size of the parameter for each parameter. Each size is then followed by the byte length of bytes as empty bytes. To call a parameter, just point to where you gave its size at the beginning of the function.

Inside you can add variable data and call functions freely.

To signify the end of a function, just put the following keyword:

In decimal: 127, 10, 13

In hexadecimal: 7F, 0a, 0d

Once this has been called, the function will end, and the program will go back to its previous function.

If you wish to return a value, put it right after the end of function keyword, this can also be a pointer. If the function doesn't return anything, put an empty byte after the end of function keyword.

Starting the program

The program will start in the main function, this function starts with the “main” keyword and does not take any arguments, therefore it is just a “main” string followed by the contents of the function.

Pointers

We talk about pointers to values, pointing to a value is just the area in the file where it starts from the beginning of the file. Pointing to a value points to the place where we specify its size.

Built-ins

We have some built in functions that you can call by putting their name as a string instead of their addresses. They are the following:

- **if** -> Takes 3 args, the first is a condition, the second is the return value if the condition is true, the third is the return value if the condition is false

- **+** -> Takes 2 args, returns the first plus the second as an int value

- **-** -> Takes 2 args, returns the first minus the second as an int value

- ***** -> Takes 2 args, returns the first multiplied by the second as an int value

- **/** -> Takes 2 args, returns the first divided by the second as an int value

- **mod** -> Takes 2 args, returns the first modulo the second as an int value

- **>** -> Takes 2 args, checks if the first is greater than the second, returns a Boolean value

- **<** -> Takes 2 args, checks if the first is less than the second, returns a Boolean value

- **=** -> Takes 2 args, checks if they are equal, returns a Boolean value

- **print** –> Prints all args given

Documentation by Adrien THIBAULT

Contact at: adrien.thibault@epitech.eu