Bosley's High Level Language (BHL)

Comments:

BHL comments are indicated by the '#' character.

Functions:

Entry point of BHL is the function "main." All functions must have a return statement. Example :

```
def main            # Function definition
                    # Function body
    ret             # Function return
```

All functions called within a function body must be declared prior to the caller function. Example:

```
def func1
    # Func1 body
    ret

def caller
    do func1            # func1 must be above caller
    ret
```

Functions cannot be nested. To call a function use 'do.' To obtain function results, use a global-set variable. To do this, have the top-most function within the program host variables so that all proceeding functions. An example of this could be:

```
def _globalSet
    _globalVar1 = 0             # Var now able to be globally accessed
    _globalVar2 = 1
    ret

def main
    do _globalSet
    return
```

If you want to terminate a function early, use the instruction 'done.' This will jump to the very end of the current function. Ignoring all instructions.

Variables:

Declaration can happen in one of two ways. The first being the explicit var declaration, and the second being without the declaration. Example:

```
var myVar = 0        # With
myVar1 = 0           # Without
```

The interpreter handles variable type, thus making operations on different types available without explicit casting. Example:

```
myInt = 2
myDouble = 3.14
myString = “A String”
myNewVar = myInt + myDouble + 14 + 3.2
```

Currently available mathematical operations :

Parenthesis, Multiplication, Division, Addition, Subtraction

Exponents, and modulus will be added in the future.

If a raw string, or a variable that is a string is used within an addition, the set variable will be of string type.

Printing:

The same rules apply to printing as with variable declaration, except its only allowed operator is ‘+’ for concatenation. To print, use ‘put.’ Example:

```
def main
    x = 2
    y = 3.14
    z = “ put command”
    put “Example of ” + z + “ Vars : ” + x + “ “ + y + “ Raw : “ + 4  + “\n”
    ret
```

This will display:

Example of put command Vars : 2 3.14 Raw : 4

If statements:

Currently if-statements only work with variables, and not with raw values. The allowed operators are: > , < , <= , >= , == , !=
The syntax of an if statement is as follows:

```
If var1 operator var2          # If
        # Instructions
fi                             # End if
```

For Loops:

For loops have the items 'start,' 'end,' and 'step n.' Each item can be a raw value, or a variable. The item 'step n' is optional. The 'n' in 'step n' is the value to increment the for-loop by. If 'step n' is not given, the loop is incremented by 1. Example Syntax:

```
for var from 'start' to 'end' 'step n'
        # For loop body
ef                             # End for loop
```

The for-loop will use the 'var' within instruction to access the current increment. If the variable 'var' already exists, it will be overwritten. If 'var' has not yet been made, it will be set to the 'start' integer. One more example:

```
forEnd = 10
for X from 0 to forEnd step 2
        put X + "\n"                    # Output : 0 2 4 6 8
ef
```

Read from user:

To read from a user, use the instruction 'uinput varName.' If the variable given in the instruction exists, it is overwritten. If it does not, it is created. Example Syntax:

```
uinput myVariable
```

File Output:

There are two file output modes. Write, and append. The filename, and data source must be given by a variable. The Instruction to output is as follows:

```
fileName = "Output.test"
fileData = "Example Data"

fout fileName write fileData        # Overwrite existing
fout fileName append fileData       # Append to existing
```

Go / Labels:

Labels can be used with go to for program flow. To create a label, create an instruction starting with the '@' character. A label can become before, or after a go instruction that calls it. Example:

```
def main
        x = 3
        y = 5
        if x < y
                go @_A_NEW_LABEL                # Go instruction
        fi

        put "This will not be executed!\n"

        @_A_NEW_LABEL                           # Label referenced

        ret
```