Assignment 1 - Python-like interpreter
CSCE 4114 – Embedded Systems – Fall 2018
Due Date – Wednesday, October 3rd, 11:59 PM CT

Description: The Python programming language has gained popularity in recent years for its various abstractions that reduce development overhead. One of the more visible of these abstractions is the removal of explicit variable type. In Python, it is perfectly valid to define a variable 'A' as an integer value (say 8), redefine it as a float value (3.14), and then as a string value ("hello!"). The python list is similarly abstract in that a list can hold heterogeneous data types at each location. In this assignment, you will develop a version of the Python interpreter that can 1) define and redefine variables as various data types, 2) define a variable as an abstract list that can hold various data types, 3) perform simple operations on these variables.
Expected Operation: The following will define the behavior of the program you will develop:

1. The interpreter will print three right angle brackets (>>>) to indicate that the interpreter is ready for a new input from the user
2. The interpreter will scan a maximum of 100 characters to find a newline character as the command string
3. The interpreter will parse the command string into one of three operations
   - Assignment operation – Has target variable name, followed by an assignment operator, followed by a dependent clause (e.g. a = 3, a = b+3)
   - Print operation – Calls the function "print". As we will not be developing functions in the interpreter, this can strictly look for "print()" with a dependent variable inside the parentheses that will be printed.
   - List appending – Calls the appendItem function. As we will not be developing functions in this interpreter, this can strictly look for "append()" with a dependent list and item as comma separated arguments e.g. append(myList,5)
4. The interpreter will accept as types of variables a:
   - Long Integer – In this case, we are assuming plenty of available memory. Therefore, all integers are long integers
   - Double Floating Point – Same as above. When printed, will maximally show 5 fractional digits
   - Character – Delimited by single ticks ('), can hold a single ASCII character
   - String – Delimited by quotes (""), can hold up to 50 ASCII characters.
   - List – Linked list of abstract variables from the above (including a list). Lists are only initialized as empty lists by passing open and close square brackets ([]);

5. The interpreter will allow for the following operations on exactly two operands for like data types for integral and floating point types: (i.e. integer + integer, but not float + integer, or char + char, or int + int + int)

   - Addition (+)
   - Subtraction (-)
   - Multiplication (∗)
   - Division (/)

6. Variables that the interpreter stores can be recalled and accessed by their name (up to 15 characters). **Variables must start with an alphabetic character, and cannot have spaces or an escape character**. This will require a data structure that hold the variable name, as well as the variable itself. The implementation is left up to the developer, but a suggestion would be to have a linked list hold "variable" structures, that have a name (up to 15 characters), and a structure called element that itself holds a type and a value (which may be a pointer). Example output:

```
>>> a = 5
>>> b = 6
>>> c = a+b
>>> print(c)
11
```

7. The assignment operation will either be a single value, or have an arithmetic operation. If it has an arithmetic operation, one of the 5 above operands will separate the two variables/values. The interpreter should parse the operands and the operator, evaluate the arithmetic operation (if exists), and then assign the new value to the variable. If the variable type is different than the previous type, the type will need to change as well. It may be easier to free the entire variable (even all list members), and then reassign new dynamic memory regardless of if type changed.

8. List assignment and access will make use of the bracket operator ([]). A bracket next to a variable name on the left will assign a new value to the list at that index (if exists). (e.g. a[5] = 4 will assign the integral value 4 to the 6th index in the array 'a'). A bracket operator on a variable on the right will access the value of the element at the given index as part of the right side operands. (e.g. b = a[4] + 3 will get the value at a[4] (if exists) and add 3 to the value (if integral value), assigning the resulting value to b.)

9. The interpreter can print any variable, including a list. A list will recursively call print for each of the variables in the list. This means that a list that has a list as one of its elements will print the inner list in its entirety as well. Example from Python3 interpreter.

```
>>> mylist = [1,2,[3,4,5],6]
>>> print(mylist)
[1, 2, [3, 4, 5], 6]
```

Rubric: The project will be graded according to the following rubric:

Category	Description	Percentage
Pass Given Tests	There are a set of ten example tests in this document which you must implement. Each test is worth 5% of the grade for this project.	50%
Pass Hidden Tests	I will run a set of five hidden tests on your executable to test operation. Four will be straightforward, one will attempt to trip you up. Each is worth 4% of the project grade.	20%
Coding Comments & Style	Use appropriate coding styles. Comment functions with a description about their behavior, any parameters, any return values, and any shared variables which it manipulates.	15%
Report	A simple, one- or two-page report. The report should have the project name (e.g. Python-like Interpreter), a short description of what you did, and the outcomes (e.g. Did it pass all example tests, if not, why not, etc...).	15%

Table 1: Grading Rubric

Given Tests: The known tests will evaluate the following (in this sequence):

1.
   ```
   >>> a = 5
   >>> print(a)
   5
   ```

2.
   ```
   >>> thisVar = a+3
   >>> print(thisVar)
   8
   ```

3.
   ```
   >>> floatVar = 5.4-2.1
   >>> print(floatVar)
   3.3(ish. Rounding errors are okay)
   ```

4.
   ```
   >>> multVar = thisVar * a
   >>> print(multVar)
   40
   ```

5.
   ```
   >>> divVar = multVar / 6
   >>> print(divVar)
   6
   ```

6.
   ```
   >>> myList = []
   >>> append(myList,7)
   >>> print(myList)
   [7]
   ```

7.
   ```
   >>> append(myList,3.5)
   >>> print(myList)
   [7, 3.5]
   ```

8.
   ```
   >>> append(myList,“hello”)
   >>> print(myList)
   [7, 3.5,“hello”]
   ```

9.
   ```
   >>> tempVar = myList[1] + 3.1
   >>> print(tempVar)
   6.6
   ```

10.
    ```
    >>> mySecondList = []
    >>> append(mySecondList,’c’)
    >>> append(myList,mySecondList)
    >>> print(myList)
    [7, 3.5,“hello”,[‘c’]]
    ```

These documents should be uploaded through Blackboard to the TA before the beginning of the next lab. Documents turned in after this deadline are subject to the course late policy.