Python-like Interpreter

Annie Prasanna Manoharan

***Overview*— This project involved the design and implementation of a simple command-line interpreter in C; mimicking core functionalities found in the Python 3 programming language. The primary objective was to build a system capable of handling dynamic variable types, performing basic arithmetic, and managing nested lists. The outcome is a functional interpreter that successfully evaluates all provided test cases, including complex scenarios with nested lists and floating-point arithmetic.**

I. Introduction

The goal of this project was to construct a functional, limited, python programming language interpreter from the ground up. The interpreter was designed to provide a dynamic and interactive environment, similar to Python. The project was pursued to gain a deeper understanding of fundamental computer science concepts, parsing, data structures, and dynamic memory management. This project provides a unique, hands-on perspective on the relationship between code and execution, without relying on external libraries or frameworks.

II. Application Design

A command-line interpreter in C that emulates some of the basic functionalities of the Python 3 interpreter is created. The interpreter supports five distinct data types: long integers, double-precision floating-point numbers, characters, strings, and lists. It can store variables by name in a symbol table and perform common operations, including:

- Variable Assignment: Assigning a value or the result of an expression to a variable.
- Arithmetic Operations: Performing addition, subtraction, multiplication, and division.
- List Manipulation: Creating lists and appending elements to them.
- Printing: Displaying the value of any variable, including printing all elements of a list.

A simple linked list that holds all the variables created. Each variable has a name and a value. The values themselves are designed to be flexible. They can hold a whole number, a decimal number, a single letter, some text, or even a list of other values. The program reads commands, like a = 5 or print(a). It's smart enough to figure out what to do. If you type an equal sign (=), it knows you want to store a value. If you type print(), it knows you want to display a value. The program can also understand simple math, like a + b, and it gets the right answer before storing it. It can also handle lists including lists inside other lists, and it can add new items to the end of a list.

III. Results

The interpreter successfully handles a wide range of commands and operations as specified. It was able to pass all of the provided example tests provided.

```
annieprasanna@mac Assignment1 % ./hw1
Python-like Interpreter (type 'exit' to quit)
>>> a = 5
>>> print(a)
5
>>> thisVar = a+3
>>> print(thisVar)
8
>>> floatVar = 5.4-2.1
>>> print(floatVar)
3.3
>>> multVar = thisVar * a
>>> print(multVar)
40
>>> divVar = multVar / 6
>>> print(divVar)
6
>>> myList = []
>>> append(myList,7)
>>> print(myList)
[7]
>>> append(myList,3.5)
>>> print(myList)
[7, 3.5]
>>> append(myList,"hello")
>>> print(myList)
[7, 3.5, "hello"]
>>> tempVar = myList[1] + 3.1
>>> print(tempVar)
6.6
>>> mySecondList = []
>>> append(mySecondList,'c')
>>> append(myList,mySecondList)
>>> print(myList)
[7, 3.5, "hello", ['c']]
>>> exit
Exiting interpreter.
```