[[1]](#footnote-1)

Embedded Systems Assignment #1

Reid Sutherland

# INTRODUCTION

T

he goal of this assignment was to create a Python interpreter from scratch using C. The learning objective here was to learn how to use a number of different C methods to read inputs from the user endlessly, parse those inputs into variables and operations, and then store those variables in simple data structures, using minimal memory.

# Application Design

My interpreter operates like any other standard command line interface, which acts on an endless loop taking in inputs from the keyboard. The interpreter then uses a parsing function to split up the input string into operands and assignment variables, which are simply c-strings. That parsing function also works to determine what type of operation that command is meaning to do. The next action the program takes is to parse those c-string operands into actual values, and categorizes those values into types.

For variables, my interpreter uses a dictionary-like approach, which stores each variable in a linked list of simple variable data structs, with containers for the variable name, element, element type, and a pointer to the next variable in the list. Lists were also implemented here, and the lists were a simple carbon copy of the linked lists that we produced in lab3. That resource made the production of the interpreter significantly easier, since I did not have to figure out how I wanted to store lists.

Finally, after each command is read and parsed, the interpreter decides what to do based on what kind of operation is being utilized. I created a simple math function which takes care of all the necessary checks and conditions for doing math operations, including accessing and assigning variables, and deciding whether or not a math operation is valid. Then it simply computes the operation based on which type of operation is specified. Other operations like assignment, print, and append were made easier with a very simple syntax.

# Results

Overall, I think my compiler turned out very well, except that I simply did not have time to implement the dynamic accessing and adjustment of list elements, so the bracket syntax is not fully functional. Despite this, my interpreter passes all known tests except for test #9. However, I can say that my interpreter does quite a bit more than what was specified only by the assignment, in terms of heavy error checking and safe memory management. While I didn’t fully finish the bracket functionality, I can at least say that I think it is very hard to break my interpreter, because I have not yet found a way to do so. Looking back on it, perhaps I should have spent less time making it a good interpreter, and more time making sure I had every single test case finished.

Frankly, this project took an absurd amount of time. While it may be because I am not the fastest coder, and I am too much of a perfectionist to leave things not working properly under any possible stimulation, I still think I must have spent at least 40+ hours on this assignment over the last week and a half, which was nearly all of my free time during that time. I am sincerely hoping that this practice will make me more apt to finish future assignments on time.

1. [↑](#footnote-ref-1)