Homework 7

Version 0.1

COS125 - Instructor: Zachary Hutchinson

Due: Friday November 17th @12AM (midnight)

Submission Instructions

These must be followed or you will lose points.

1. Submit your coded solution as a .py file. Code in any other file format will be rejected.
2. Your .py file should be named: yourlastname_hwX.py. The 'X' in the filename is replaced by the homework number.
3. Your code must have as a comment at the top of the file: your name, the homework and the names of anyone from whom you received help. This includes students in the course as well as course staff. For example, if you went to Boardman 138 and received help from an MLA, put their name on your file. If a classmate helped you debug a bit of code, put their name down. This is for your benefit.
4. If you do not manage to squash all the bugs in your program, include a comment at the top of the file detailing the outstanding bugs. Acknowledging bugs is a sign of a mature programmer. Doing so will not eliminate point deductions but it might mitigate them. It shows you care about your work.

Learning Objectives

- Practice while-loops, file reading and lists.
- Understand how programs are used to create other programs.
- Understand the basics of creating and implementing a formal language.
- Practice reading command line arguments.

Problem Description

You are going to implement a simple interpreter for a new assembly programming language called *Blah*. Python is considered a "high-level" language. An assembly language is the lowest-level language one can write in without actually writing lines of 0s and 1s. The instructions are therefore very basic. The following sections give the specifications of the Blah language. The Blah langauge consists of only 5 instructions.

Requirements

Your Blah language interpreter will load a program file (text file). The name of the program file must be given as a command line argument. For example, typing the following into the command line should invoke the Blah interpreter and run a program called prog1:

```
python3 blah.py prog1
```

We will cover reading from the command line in class on Monday Nov 5th.

You interpreter will run on a pretend computer that has only 4 memory locations, or registers. For reference, a register is memory inside the CPU. What does this means for you? Your interpreter will need four places to store integers (HINT: list). When the interpreter starts it will set all four registers to zero.

Your interpreter does not need to account for bugs in a Blah program. You can assume that any program run on the interpreter is perfect.

The Blah Language

The Blah language consists of five different instructions: *add*, *set*, *print*, *goto-on-less-than*, and *end*. The first part of an instruction is called the *opcode*. The opcode specifies the type of the instruction. The contents of the rest of the instruction depends on the opcode. Some instructions, such as *end*, only consist of an opcode. Other instructions consist of registers, integers or program line numbers.

ADD instruction

The ADD instruction consists of four parts, the ADD opcode and three registers. *The ADD instruction adds together the values in registers **A** and **B** and stores the result in register **C**.* In this definition, the letters A, B and C are placeholders for actual register numbers 0 to 3 (since there are only four registers in our pretend computer).

```
ADD A B C
```

Example: the line of Blah code, `ADD 0 1 0`, will add the contents of registers 0 and 1 together and store the result in register 0.

SET instruction

The SET instruction consists of three parts, the SET opcode, an integer value and a register. *The SET instruction places the value **X** into register **A**.* X is an integer value, not a register.

```
SET X A
```

Example: `SET 55 2` will set the contents of register 2 to the integer 55.

PRT instruction

The PRT instruction causes the interpreter to print the contents of a register. It consists of two parts, the PTR opcode and a register. *The PRT instruction prints (to the command line) the value in register **A**.*

```
PRT A
```

Example: `PRT 3` will print out the value of register three to the command line. Assume register three contained the value 99, then 99 would be printed to the command line.

GLT instruction

The GLT instruction consists of four parts, the GLT opcode, two registers and a line number. *The GLT will cause the program to jump to line number given in **M** if the contents of register **A** is less than the contents of register **B**.* Line numbers start at 0 and count upward from the beginning of the program. In other words, the first line of the program is line 0. The fourth line is line 3.

```
GLT A B M
```

Example: Assume register zero contains 4 and register one contains 6. `GLT 0 1 8` will cause the program to jump to line 8 because 4 < 6.

Important: If the contents of A is equal to or greater than B, the program just continues on to the next instruction. No jump takes place.

END instruction

The END instruction causes the program to finish. All Blah programs must contain at least one END instruction. The END instruction does not have any additional information.

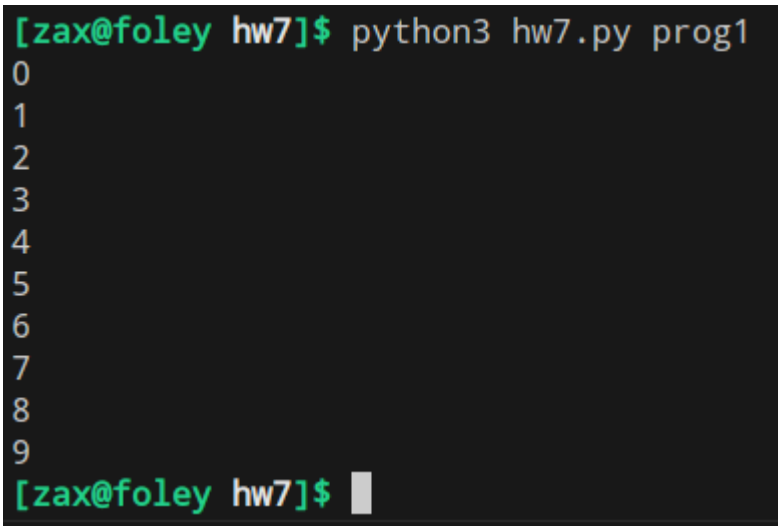
```
END
```

Example

I have uploaded with this pdf an example Blah program. The program uses all five types of instructions. The program implements a loop that counts from 0 to 9. Use this to test your program. When grading we will use a very similar program with just a few specifics changed, i.e. which registers are used and the start and end of the range. The Blah language does not support comments in the file so I am commenting the code below. Do not copy and paste this code (use the file). It will not work with the comments in it.

```
SET 1 1     Stores 1 in register one.
SET 10 3    Stores 10 in register three.
PRT 0       Prints the contents of register zero.
ADD 0 1 0   Reg. zero plus reg. one. Store in reg. zero.
GLT 0 3 2   If reg. zero < reg. three, goto line 2
END         End program
```

Here is the output:



Things to consider (HINTS)

1. If you read each line of a Blah program into a list, then a line's index in the list is also its line number.
2. readlines() is your friend.
3. To keep track of the current line number, use a *program counter* variable. To move to the next line, you add one to the program counter. To jump to another line number, you simply alter the program counter.
4. The pretend computer and the Blah language can only store and manipulate integers. No strings. No floats. No bools.
5. Remember the pretend computer only has 4 registers - 0, 1, 2, and 3 (hint: list).
6. This assignment does not require you to "parse" a Blah program or check for syntax errors. Again, assume only correct programs are used.
7. Since instructions carry different information, you must first examine the opcode before extracting the rest of the information.
8. Remember you need to *import sys* to read command line arguments. Command line arguments are stored in *sys.argv*. They are always strings. You will need to cast if you need an int.
9. For those of you using VSCode, since you need to use command line arguments, using the execution button in VSCode will be difficult. I suggest reverting to the command line to run the program.
10. Remember: programs are read sequentially from the top (hint: while-loop). But they can jump forward or backward if they encounter a GLT (double hint: for-loop won't work).
11. Start early. Ask questions.