Assignment 1

Microproject

Write a small C program which reads an input file, line by line, into an array and allows the user to retrieve the elements by index. The input file contains, on the first line, an integer number of lines to read after that first line. Each line after that will contain some character string.

For example, consider the following input:

```
3
My first love
csc344
My favorite
```

When loaded, your program will make an array of length 3, and populate the cells with the lines of the program. After loading the file, your program should ente a command loop where the user can enter an index number and the program should output the corresponding string.

Main Project

In this project you will implement a stack machine in C. A stack machine is a simple model of computation in which the primary memory construct is a stack. There is a set of instructions which allow adding/removing integers from the stack, as well as performing various mathematical operations. Stack machine programs are lists of these instructions. Instructions are executed one after another, except for branching instructions which allow jumping to a specific instruction number in the program.

You must implement a stack of integers as a linked data structure (think linked list). Be sure you implement this yourself and don't use code from the internet — it's not hard and is a useful task. Be careful to properly allocate memory when you need to push and to free memory when you pop. Stack machine programs will be read from a file — you may either provide an argument or have your program ask for a file name. The files will begin with the number of instructions, followed by the list of instructions. You should store the instructions in some data structure for easy access by index.

To execute your program, you will need some sort of counter to keep track of where you are in the list of instructions. Decode and execute instructions one after another until you've reached the end of the program.

Instruction Set

```
// Input / Output
- IN   -- Reads the next integer from the input stream, and pushes it on the stack.
- OUT  -- Pops the top integer from the stack, and prints it.

// Stack operations
- LIT [NUMBER] -- Pushes NUMBER on the stack.
- DROP         -- Pops the top element of the stack.
- DUP [N]      -- Duplicates the top N elements of the stack, in order (DUP 2: ... A B becomes ... A B A B).
- SWAP         -- Swaps the top two elements of the stack (SWAP: ... A B becomes ... B A).

// Integer operations
- ADD/SUB/MUL/DIV/MOD -- Pops the top two integers from the stack, Adds/.../Mods them, and pushes the result.
- AND/OR              -- Pops the top two integers from the stack, ANDs/ORs them, and pushes the result.

// Branching
- IFEQ [INSTRUCTION] -- checks if the two elements at the top of the stack are equal. If so, jump to the given instruction.
- IFLT [INSTRUCTION] -- checks if the first element of the stack is less than the second element. If so, jump to the given instruction.
- JUMP [INSTRUCTION] -- jumps to the given instruction.
```

Example

Below is an example input file for your stack machine which computes Euclid's algorithm (https://en.wikipedia.org/wiki/Euclidean_algorithm).

```
13
IN
IN
DUP 1
LIT 0
IFEQ 12
SWAP
DUP 2
DROP
SWAP
MOD
```

```
12 JUMP 3
13 DROP
14 OUT
```

What follows is a trace of the computation:

Instruction Number	Instruction	Input	Stack	Output
1	IN	20	20	
2	IN	15	15, 20	
3	DUP 1		15, 15, 20	
4	LIT 0		0, 15, 15, 20	
5	IFEQ 12		15, 20	
6	SWAP		20, 15	
7	DUP 2		20, 15, 20, 15	
8	DROP		15, 20, 15	
9	SWAP		20, 15, 15	
10	MOD		5, 15	
11	JUMP 3		5, 15	
3	DUP 1		5, 5, 15	
4	LIT 0		0, 5, 5, 15	
5	IFEQ 12		5, 15	
6	SWAP		15, 5	
7	DUP 2		15, 5, 15, 5	
8	DROP		5, 15, 5	
9	SWAP		15, 5, 5	
10	MOD		0, 5	
11	JUMP 3		0, 5	
3	DUP 1		0, 0, 5	
4	LIT 0		0, 0, 0, 5	
5	IFEQ 12		0, 5	
12	DROP		5	
13	OUT			5

Extra Credit (+10% to your score)

Augment your machine with 256 int-sized addressable memory locations which may be accessed by two new instructions, LOAD and STOR, which take as an argument a memory address $00 through $FF.

```
1 - STOR [ADDRESS] -- Pops from the stack, and puts the value at the given address.
2 - LOAD [ADDRESS] -- Pushes the value stored in the given memory address onto the stack.
```

Write a stack machine program which demonstrates the use of these in some interesting way.