CSCI 2240
Program 3

Virtual Computer

Concepts Covered

- Pointers
- Memory Concepts
- String Manipulations
- Formatted I/O

This assignment has you creating your own virtual computer with its own limited instruction set and language. The program you write will accept as input programs written in this language, "compile" it into the computer's instruction set and execute the program.

The computer has a 100 word memory, where each word is a signed 4 digit integer. Instructions and stored values share the same 100 word memory. It also contains an accumulator, which acts as a register that operations are performed upon. The current instruction to be executed is kept in the instructionCounter and the current instruction is kept in the instructionRegister. There is also two additional registers used for splitting up the data in the instruction, an operationCode and operand. When the program in memory is executed, it starts with the instruction at memory location 0, and stops when the halt instruction is reached.

The computer is capable of handling instructions, 4 digit signed integer values, and strings. Each word when interpreted as an instruction can be broken down into two, 2 digit chunks. The first 2 digits represents the instruction (operationCode), the second represents the memory address that instruction uses (operand). When a word is interpreted as a value, the entire 4 digits is the value. When interpreted as a string each 2 digits represents the ASCII character of the string. The only characters that are understood are NULL (00), newline (10) and A-Z (65-90). Each instruction is followed by a newline.

The language consists of a number of commands, each representing an instruction in the computer. Each command is on a line of its own, and is preceded by the 2 digit address in memory it is to be placed and followed by a single value, typically an address in memory.

Listing of commands and their instruction code

Input/Output

READ 10 – Retrieves a value from the user and places it in the given address
WRIT 11 – Outputs a word from the given address to the terminal
PRNT 12 – Outputs a string starting at the given address, will continue outputting consecutive words as strings until NULL is reached

Load/Store

LOAD 20 – Load a word from the given memory address into the accumulator
STOR 21 – Store the word in the accumulator into the given address
SET 22 – Stores the given word into the preceding address (Note: The operation code value of 22 will never appear in a compile program, is only included for completeness)

Arithmetic Operations

ADD 30 – Adds the word at the given memory address to the accumulator
SUB 31 – Subtracts the word at the given memory address to the accumulator
DIV 32 – Divides the word at the given memory address to the accumulator
MULT 33 – Multiplies the word at the given memory address to the accumulator
MOD 34 – Mods the word at the given memory address to the accumulator

Control Operations

BRAN 40 – Execution jumps to the given memory location
BRNG 41 – Execution jumps to the given memory location if the accumulator is negative
BRZR 42 – Execution jumps to the given memory location if the accumulator is zero
HALT 99 – Terminates execution, no address given, value of 99 is standard, also prints out the state of memory in a tabular format like shown below. Notice the spacing and justification of the output.

```
REGISTERS:
accumulator                +0000
instructionCounter            00
instructionRegister        +0000
operationCode                 00
operand                       00
MEMORY:
          0     1     2     3     4     5     6     7     8     9
  0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 50 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
 90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
```

Example Program:

The following reads two values from the user and outputs the larger of the two

00 READ 9 ← Read From keyboard, location to store
01 READ 10
02 LOAD 9
03 SUB 10
04 BRNG 7
05 WRIT 9
06 HALT 99
07 WRIT 10
08 HALT 99
09 SET 0
10 SET 0

The "compiled" version of this in memory would look like:
1009
1010
2009
3110
4107
1109
9999
1110
9999
0000
0000

Where the first instruction is stored in the 0^{th} address in memory and each following instruction is stored in subsequent addresses. However, instructions do not have to be listed in order. The address the instruction is stored in is determined by the 2 digit number that precedes the command.

The following simply outputs "HELLO" to the terminal, followed by a newline

00 PRNT 02
01 HALT 99
02 SET 7269
03 SET 7676
04 SET 7910
05 SET 0000

The PRNT command points to where the string starts, and each character is printed starting at that location until NULL is reached. Notice the words used need to be set in when the program is written so the words are in memory for execution.

Your Program

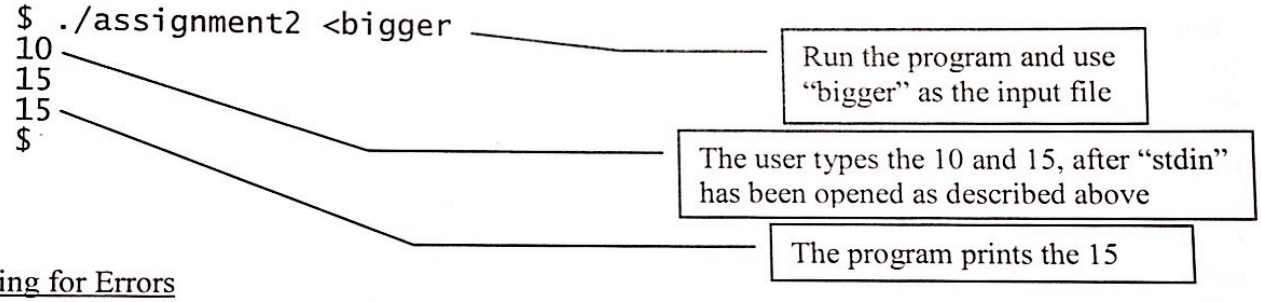
Your program will accept a program of this language (redirected from a file), parse the commands and load the instructions into the 100 word memory. If and only if this is successful, the loaded program will then be executed.

A Note About Files

Your program can read the "machine program" using the usual "fgets" or "getchar" or any of the standard ways. However, after reading in the program, your code will then have to execute it and take input from the keyboard. To do this, you will need to reassign the "stdin" file to the keyboard after reading in the machine program but before executing it; use this:

```
stdin = fopen("/dev/tty", "r");
```

For example, suppose you have the above machine program in a file called "bigger". Once the program is read in, stored in the memory, and is ready to execute, it is necessary to reassign the input to the terminal. The result would look something like:



Checking for Errors

There are a number of errors that can occur, at both compile and runtime on the virtual computer. You must catch these, and report them with the address they were encountered at and a message. A compile error will stop the compiling process and not attempt to execute the program, a runtime error will cause a HALT, terminating execution and output the state of the registers and memory.

Compile Errors

- Unknown command – Unrecognized command word, they are case sensitive
- Word overflow – attempts to place a word in memory that is larger than 4 digits
- Undefined use – Command is not in the proper format
- No HALT – No HALT command is ever given

Runtime Errors

- Unknown command – Unrecognized command code
- Word overflow – attempts to place a word in memory or alter the accumulator so that it is larger than 4 digits
- Segmentation fault – attempts to access an unknown address
- Divide 0 – Division by 0 was attempted
- Unknown Character – When printing a string, and unknown character was reached (only understands NULL, newline, and A-Z)

Representing your computer

The memory of your computer is very limited, and this needs to be reflected in your virtual solution of the computer. As discussed, the *only* memory that is available is:
memory of 100 words– can be represented by an int array
accumulator – represented by an int
instructionCounter – represented by an int
instructionRegister – represented by an int
operationCode – stores the instruction code of the instruction, represented by an int
operand – stores the operand of the instruction, represented by an int

This is the only memory you have, therefore the only memory you can use for your entire solution (with the exception of creating pointers for function parameters). These must only be created once, that is no duplicates. All data must be passed by reference.

Your solution needs at least two functions in addition to main. One to accept the written program called compile() and one to execute what is in memory, called execute(). You can create additional functions that are called upon by these two, but must at least have these two separate functions. Your main function will simply call upon these two functions, compile() and execute(), execute only running if appropriate (the inputted program compiles). **You may not allocate any other memory other than what was discussed above in any of your functions.**

There is one and only one exception. You are allowed to allocate memory for a string and create a pointer to it within the compile function in order to read in the lines and parse them out. You cannot use this memory for the execution function.

Virtual Computer Programs

Once you have your computer written, you must also write some programs in the new language. Within your assignment folder save these in files titled prog1, prog2, and prog3.

prog1 – Write a program that uses a counter controlled loop to read in positive and negative values and output their sum. The first piece of data entered will be how many numbers are going to be summed.

prog2 – Write a program that uses a sentinel controlled loop to read in only non-negative numbers (a negative value will end the loop, make sure this value is not part of the computation) and outputs the average, do not worry about catching a divide by zero, this should give a runtime error.

prog3 – Write a program that reads in two numbers and if the **second** number is a multiple of the **first** outputs the string "MULT" otherwise outputs the string "NOT"

Make sure that for all programs after they are executed your command prompt is on a line of its own.