CST8234 – C Programming
Assignment 01: SMS Simple Language Simulator

Data dictionary
Some of the variables that I recommend for you to use. You may decide to use different ones, or with different names.

Name	Data Type	Size	Scope / location	Allocation	Purpose
`memory`	int array	1000*sizeof(int)	Local to main / Available to all functions	Stack/ Heap	**Memory** An array of size 1000 which represents the memory This array can be statically defined in the stack, or dynamically allocated in the heap (using `malloc( )`)
`opCode`	int	sizeof(int)	Local to main	Stack	**Operation Code** 2 digit that shows which operation is going to be done (READ, WRITE, …)
`op`	int	sizeof(int)	Local to main	Stack	**Operand** 3 digit that shows memory location
`instreg`	int	sizeof(int)	Local to main	Stack	**Instruction Register** Holds the instruction that is currently being executed
`instctr`	int	sizeof(int)	Local to main	Stack	**Instruction Counter** Holds the memory location of the next instruction to be executed
`acc`	int	sizeof(int)	Local to main	Stack	**Accumulator** Holds a single word a in order to do arithmetic and branching
`ints`	int	sizeof(int)	Local to main	Stack	**Valid Instruction** Counts the number of Instruction that has been executed
`maxmemo`	int	sizeof(int)	Local to main	Stack	**Max Memory** used to track maximum address of memory used
`mem`	int *	sizeof(int *)	Local to main	Stack	**Memory Pointer** counter used for storing instructions to memory
`on`	int	sizeof(int)	Local to main	Stack	**Machine ON** Flag to signal execution of instruction in memory

What to do

Step 0: Read the assignment at least 2 times.
Think about the way a computer works, use your notes from CST8216.
Be sure you understand the overall idea.

Step 1: Initialize all your variables
Start in your `main( )` function.
Declare the main 6 variables, your register and you memory array. Use a define SIZE for your array

Initialize all the registers to 0 – This should not be a problem.
Initialize all the memory locations to 50505:

(1) Define a toke INIT_MEMO as 50505
(2) Write a function `void init_memo( int a[ ] );`
Your function should initialize a with INIT_MEMO
(3) In your main call your function to initialize the memory array.

Step 2: Write your function dump
Your function dump should printout all the registers and the memory (array). The function does not need to return anything and it is not changing any values, so a valid prototype is:

```
void dump( int a[ ], int accumulator, int instcounter,
     int instregister, int oprationcode, int operand );
```

I would recommend you to write a sub function:

```
void dump_memo( int a[ ], int max );
```

This function should print the array, since not all the memory is used, you would print until `max` (this value is the `instcounter`) In all the labs up to now you have work with printf to create tables. Use that knowledge here!

At this point your main( ) has just a couple of instructions, the initialization and the dump.

The output of your program should look like this – I have called the dump_memo( ) with max 20.

```
root@luna:~ ./sms

REGISTERS:
accumulator            +00000
instructioncounter        000
instructionregister    +00000
operationcode              00
operand                   000
ValidInstructions           1
```

```
MEMORY:

   0 +50505 +50505 +50505 +50505 +50505 +50505 +50505 +50505 +50505 +50505
  10 +50505 +50505 +50505 +50505 +50505 +50505 +50505 +50505 +50505 +50505
```

Pay special attention to the width I'm using for the numbers.

Step 3: Load information from a file into your memory (array)
This is an easy one! Think back to lab 1, where you read from a file using fscanf( )

Write a function with function prototype `void load( int mem[ ] );`

Remember that the end of the instructions are demarcated with the number -999999, define END as -999999

Read from stdin until you reach the END.
As you read, place the instruction (the number you just read) into the memory.

Use the file load_memory.txt to test your program. At this point the output should look like:

```
root@luna:~# ./sms < load_memory.txt

REGISTERS:
accumulator            +00000
instructioncounter        000
instructionregister    +00000
operationcode              00
operand                   000
ValidInstructions           1


MEMORY:

   0 +10007 +10008 +10009 +10010 +10011 +10012 +10013 +10014 +10015 +10016
  10 +10017 +10018 +10019 +10020 +43000 +50505 +50505 +50505 +50505 +50505
```

Still using max as 20 in my dump_memory( )

Do not continue if you are not able to load information from the file into the array! Your load( ) is not more than 5 lines of code, so review them carefully. Look into your lab01, part b if you are not sure about reading with fscanf( )

If you are reading properly from the file, it is time for you to start thinking about possible errors that you may encounter, and handle them appropriately. There are two possible `abend` conditions when loading the program (see page 4 of the assignment).

When you encounter an error, it means that you can't continue with the execution of your program. Your program them should print a message to the user, and then terminate with a failure value.

General logic:

function encounter an error,
function return an error code
calling function (in this case, the main) prints an error message
calling function terminates with a failure value.

Define two error code (I usually use negative values for error codes)
TOO_BIG -10
INVALID_WORD -20

change the function so it now returns an integer instead of void, return 0 if everything goes OK, and an error value if there was a problem.

Lets work first with the TOO_BIG error. You encounter a TOO_BIG error when there are too many instructions to fit into the memory (in other words, when your instructions are more than SIZE of the array). As you are reading, you are loading into the array, if the position that you are in is more than SIZE then you should return TOO_BIG. To test this properly, the easy way is to change SIZE to 10 an run your program with the same file as before.

I will recommend for you to write a function to handle all the error messages. For example:
void error_message(int errorcode);

Your main( ) would call your error_message( ) , then dump( ) and exit( ) with EXIT_FAILURE.

I hope that this help you to start your assignment and from here. Now it is up to you to keep up with the same logic for the rest of the program.