Objective

The purpose of this assignment is to *write*, *load* and *display* a grid in LC-3 assembly language. The grid here is a 8x8 square of spaces where each space can hold Simba, a Friend, a Hyena, or Simba's Home. Unlike your previous two programs, in this programming assignment you will learn to write code that interfaces with pre-written code. This program is part of a sequence that will eventually result in a game called ***Save Simba***. You will use both an **Array** and a **Linked-list** data structure in the coding of this program.

Details

The *Save Simba game* involves a jungle made up of a grid of squares (aka *gridblocks*). For this assignment, the jungle is a 8x8 grid. The *Initial* and *Home* locations along with locations of *Friends* and *Hyenas* are loaded as input to the game. The input is structured as a **linked-list**. Your job is to complete 3 subroutines that load the jungle information (LOAD_JUNGLE), display the Jungle (DISPLAY_JUNGLE) and convert grid coordinates to an address (GRID_ADDRESS).

Input

The input is a linked list of records representing ***grid-blocks*** with 3 *fields*:

- a ***link*** to the next record
- coordinates **(row, col)** each ranging [0, 7]
- a character indicating ***gridblock*** **type**
    - **`I` (*Initial*)**
    - **`H` (*Home*)**
    - **`F` (*Friend*)**
    - **`#` (*Hyena*)**

  Note: Each record contains a field that is the address to the next record. This field is called a ***link***, which gives this data structure its name, **linked-list**. The last record will have a ***link*** field with an address of **`x0000`**. This is the sentinel that signals the end of the linked list.

An example layout of the Jungle is provided in the starter code. Note that, instead of scattering the linked list all over different parts of memory, all records are stored in a single file linked using labels instead of addresses. This is done for convenience; your code must work even if the linked list records are scattered across memory.

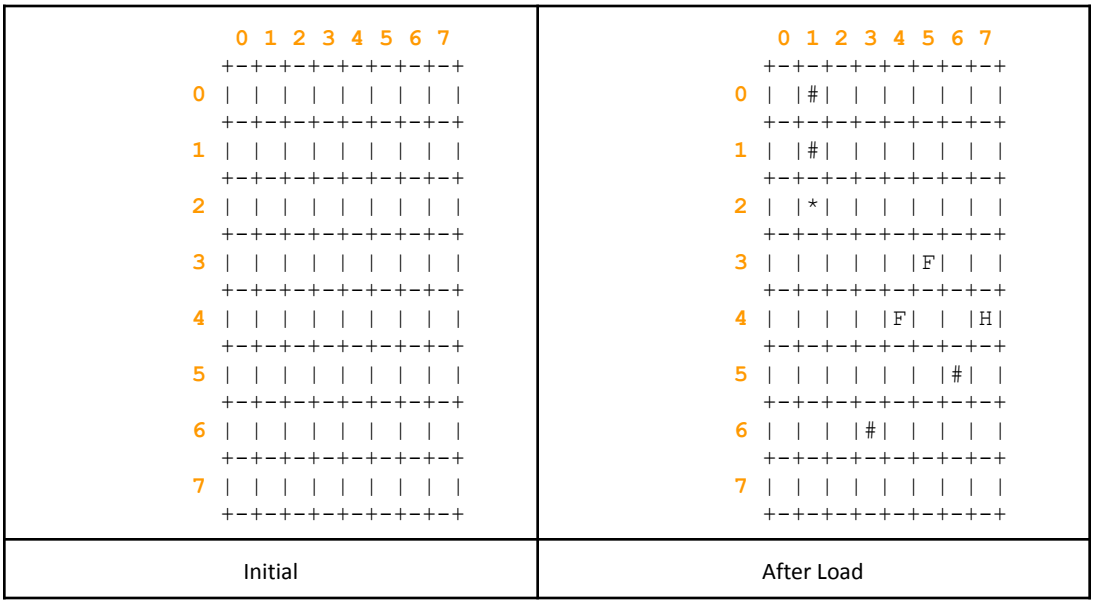
Notes:

- The **address** of the **first** record in the list will be stored in **x5500**. (*The first record is not at x5500*). This address is referred to as the **head** of the list.
- There is no defined order in which these records occur. For example, it is possible for the *Home* record to be at the head of the linked list.
- There will only be one ***Initial*** gridblock and one ***Home*** gridblock in the list. You do not have to do any error checking to check if there are multiple of these two types.
- There may be zero or more ***Friend*** and ***Hyena*** records. If there are no records with **F/#** in them, then that means there are no ***Friends/Hyenas*** in the jungle.
- When processing the list you need to pay special attention to the *Initial* and *Home* records:
    - For the ***Initial*** gridblock, you need to put a **`*`** in the corresponding entry in the **`GRID`** and set the global variables **`CURRENT_ROW`** and **`CURRENT_COL`** accordingly. (More on these variables below.)
    - For the ***Home*** gridblock, you need to put an **`H`** in the corresponding entry in the **`GRID`** and set the global variables **`HOME_ROW`** and **`HOME_COL`** accordingly. (More on these variables below.)

You need to follow all these rules (listed above) for processing the input data when writing your subroutine called **LOAD_JUNGLE**.

GRID

The Jungle GRID is a 8x8 grid of spaces defined initially as shown in the left figure below (after **DISPLAY_JUNGLE** is called)



The GRID itself is a set of .STRINGZ declarations without the row and column numbers (numbers in orange, above). For example, after loading an input linked list with a *Initial* location at (2, 1); a *Home* location at (4, 7); two Friends at (3,5), (4,4) and four Hyenas at (0, 1), (1, 1), (5,6) and (6, 3); the GRID looks like the right figure above.

Pre-written Code

The main program that is the engine of your game is provided to you along with the declaration of the GRID and several variables.

Your Task / Subroutines

Implement the following **three** subroutines. Two of these subroutines are called by my engine, the other (GRID_ADDRESS) is a utility routine that is called by your code. We will test it though.

- **DISPLAY_JUNGLE**: This subroutine displays the state of the GRID to the Console.
  - Inputs: None
  - Outputs: None

  The jungle GRID is located in memory at the label GRID. It is simply a set of .STRINGZ declarations:

```
GRID .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
     .STRINGZ "| | | | | | | | |"
     .STRINGZ "+-+-+-+-+-+-+-+-+"
```

  Note that each string is null-terminated. The output (to the console) is not just a simple print of these strings as can be seen in the figure above. Each column and row number are indicated (shown in **orange** above) in the display of the grid on the console. You can create your own .STRINGZ for column numbers, if you need; row numbers/blanks must be printed in front of each row. **You are NOT allowed to edit the GRID or any code provided to you**.(*See starter file for the boundary above which you are not allowed to edit*)
- **LOAD_JUNGLE**: This subroutine reads the input linked list and appropriately populates (modifies) the contents of the jungle GRID. See above for the description of the input format.
  - Input: R0 - the address of the head of the linked list of gridblock records
  - Output: None
  - Purpose: Populates the GRID with the appropriate characters (*, H, F, #)
    Additionally, sets the following variables:
    - CURRENT_ROW - initialize with the row of the *Initial* gridblock
    - CURRENT_COL - initialize with the column of the *Initial* gridblock
    - HOME_ROW - initialize with the row of the *Home* gridblock
    - HOME_COL - initialize with the column of the *Home* gridblock
- **GRID_ADDRESS**: This subroutine is a key subroutine that both your code and eventually the game itself depends on. It translates the (row, col) logical GRID coordinates of a gridblock to the physical address in the GRID memory.
  - Inputs: R1 - row number [0, 7]; R2 - column number [0, 7]
  - Output: R0 - corresponding memory address of the gridblock in the GRID

  Hint: There are 17 physical rows of characters and only 8 logical rows.

Sample Run

Here is the console output from a sample run of a working solution:

```
   0 1 2 3 4 5 6 7
  +-+-+-+-+-+-+-+-+
0 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
1 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
2 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
3 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
4 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
5 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
6 | | | | | | | | |
  +-+-+-+-+-+-+-+-+
7 | | | | | | | | |
  +-+-+-+-+-+-+-+-+

Jungle Initial

   0 1 2 3 4 5 6 7
  +-+-+-+-+-+-+-+-+
0 | |#| | | | | | |
  +-+-+-+-+-+-+-+-+
1 | |#| | | | | | |
  +-+-+-+-+-+-+-+-+
2 | |*| | | | | | |
  +-+-+-+-+-+-+-+-+
3 | | | | | |F| | |
  +-+-+-+-+-+-+-+-+
4 | | | | |F| | |H|
  +-+-+-+-+-+-+-+-+
5 | | | | | | |#| |
  +-+-+-+-+-+-+-+-+
6 | | | |#| | | | |
  +-+-+-+-+-+-+-+-+
7 | | | | | | | | |
  +-+-+-+-+-+-+-+-+

Jungle Loaded

----- Halting the processor -----
```

The top of the output shows the console with the initial state of the jungle before it is populated. The second part shows the console after the jungle is populated and the program halted.

Submission Instructions

- You must do the programming assignment by yourself. You are permitted to get help from ONLY the TAs and the instructor.
- The file you submit should be a LC-3 assembly language file named **Program3.asm**. A starter version of this file is provided for you. This is the **only** file you need to submit.
- Do not edit parts of the code you have been told explicitly to not modify.
- Remember, subroutines have contracts and Callee-saves.
- Submit your completed **Program3.asm** on Gradescope.