LAB 12:
Linked Lists

Provided Files

- `main.c`
- `gba.c, gba.h`
- `spritesheet.bmp, spritesheet.c, spritesheet.h`
- `list.c, list.h`
- `tiles.c, tiles.h`
- `print.c, print.h`

Files to Edit

- `list.c`

Files to Add from Lab00

- `Makefile`
- `.vscode`
    - `tasks.json`

Instructions

In this lab, you will be implementing a linked list in order to complete the classic game *Snake*! Each linked list node represents a segment of the snake, therefore we must be able to create a linked list, create nodes, push them to the back, and update node positions.

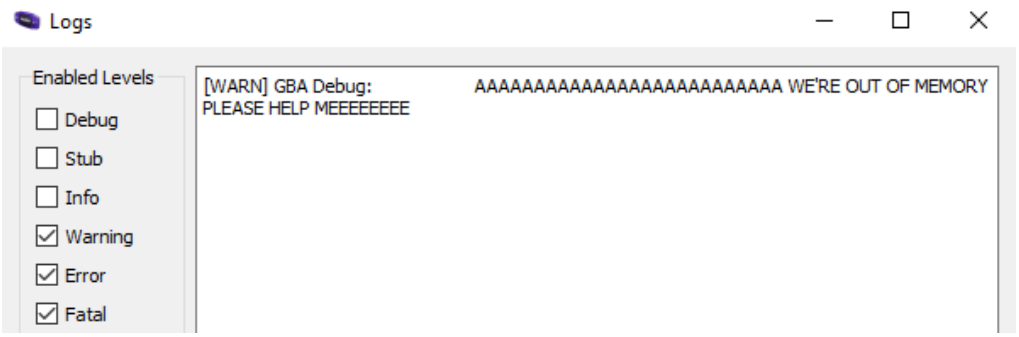
TODO 1 - Setting Up the Linked List

First, let's set up our linked list functions! Since our functions to create a linked list are not complete, if you build and run at this time you will only see the border of the map. Navigate to `list.c`.

TODO 1.0

- Complete the `createList` function. This function should create a doubly-linked list, using the `LIST` struct. This new list should have both head and tail initialized to `NULL`.
    - You need to handle the case where malloc cannot allocate enough

memory for the list. You should use `mgba_printf` to log an error to the console and the error message should be entertaining in some **school-appropriate** (please don't get us in trouble) way. For example:



You may receive 0.25pts of extra credit for an especially creative or funny message :)

If malloc returns null, you **should not** set the head and tail pointers (as that would be dereferencing a null pointer and would crash the program) and instead should just return null.

TODO 1.1

- Complete the `createNode` helper function. This function should create a node of type `NODE` by allocating memory for it on the heap. The parameter position, of type `POSITION`, should be stored in the node. Be sure to set the node's next and previous pointers to `NULL`.
    - Just like for `createList`, make sure to print a funny log message if malloc cannot allocate enough memory and not dereference the pointer (just return null.)

TODO 1.2

- Complete the `pushBack` function. This function should add data to the end of the linked list. To do this, it should create a node storing the `position` parameter using `createNode` and then add that to the front of the list.
    - **Note**: Make sure to handle the case where the `dllist` parameter is an empty list.
    - **Hint**: Feel free to reference the `pushFront` function, which adds the data to the front of the linked list.

Build and run. At this point, you should be able to see the moving snake head and four snake segments that are not moving, as well as the food object. The moving snake part

should be controlled by using the UP, DOWN, LEFT, and RIGHT buttons. If the moving part collides with any of the body segments or with the edge of the screen, the screen should be filled with tiles. At this point, you can click the START button to restart the game.

TODO 2 – Updating Node Positions

Now you'll focus on an essential function for our game: `updateNodePositions`. This is what is going to make the snake's body segments move with the snake's head.

TODO 2.0

- Complete the `updateNodePositions` function. This function takes in a pointer to the list which contains the nodes we want to update. The purpose of this function is to start at the end of the list and set each node's position to the previous node's position.

Build and run. You should be able to play the game in its entirety at this point. The snake's body should be fully connected, and eating a food object should add another node to the end of the linked list, expanding the snake by one tile.

TODO 3 – Clearing the List

When the player loses, we want to free up all of the memory we were using for the list so we don't leak memory.

TODO 3.0

- Complete the `deleteList` function which frees all of the memory used by the list.
    - You will need to call `popFront` repeatedly until all nodes are removed from the list and freed.
    - Then, free the memory for the list struct itself.
- This function is already called for you in `main.c`.

If everything still works correctly, then…

You've finished all the labs for CS 2261!!!

Thank you for all your hard work and best of luck with your final project! :)

Tips

- Follow each TODO in order, and only move forward if everything is correct.
- Reference `list.h` to see the structs you should be using for this!

Submission Instructions

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of previous assignments for instructions on how to perform a "clean" command.**

Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission Lab11_LastnameFirstname, for example:

> “Lab12_BittelTravis.zip”

It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs as expected.