Homework 3 – Oregon Trail

Due Date – Wednesday 11/5

Project Statement

This assignment is meant to grant you further C experience inside a full Linux environment. You will be designing software that emulates the Oregon Trail text based video game from the mid 1970's. The project is developed such that you will implement linked-list functions and develop a software hierarchy mentality while dealing with file input/output, command line input/output, dynamic memory, and pointer implementation.

You will begin at the bottom right of an NxN grid and attempt to traverse to the top right without losing your entire food stockpile. File input will define the size of the grid, the map layout, and the amount of food you have in your stockpile. The map grid will be defined as an NxN 2-dimensional doubly linked list, meaning you can traverse from any square to the square above, below, to the left, and to the right of the square. At each square you may encounter a river, a disease, wild-animals, or nothing. Each of the encounters will create a loss on your food stockpile of some varying amount defined below. You will only be able to see an MxM viewport (This is where we will dynamically require them to free/create nodes from file IO) around your player during the game.

Deliverables

- All code you use to implement the game
- A report with the following
    - A clear statement on the completeness of your system. Report what testing procedures you performed.
    - Document design decisions
    - Provide a usage manual (make it clear what your system DOES not what it was intended to do but you never got working).

Grading

The grade distribution for this project will be as follows

- 10% Coding style
- 15% Reports
- 75% Functionality

The functionality will be broken down into components. Each working component will give you credit. A fully working system will give you full credit for this portion.

Components in the functionality section:

- Linked-List matrix implementation
- File I/O
- Command line I/O
- Dynamic memory
    - Creating
    - Freeing (we will be using Valgrind to determine memory leaks)
- Pointers

Coding style is based on the following guidelines

- All functions need to be commented with:
    - Input parameters
    - Return Value
    - Description of what the function is intended to perform
- Your code should be split into different files based on functionality
    - A sample makefile will be provided, but you will need a makefile for your program compilation
- All main code should be documented with its intended functionality

Game Description

The game will work as follows:

- The game will boot into a short "splash screen" asking for input to start, quit, or display instructions
    - Instructions will explain how to play the game
    - Starting the game will display the viewport, and ask for the first input
- The user will begin to traverse the map to attempt to successfully cross the Oregon Trail
    - The game will take input as "up/down/left/right" from the user to move to a new square, or "exit" to allow the user to quit the game
- The viewport is displayed to the player along with his current food status and coordinates on startup and after each action.
- Upon arriving on the square the user will encounter one of the following
    - River – must be crossed/retreated from
        - A river crossing will lose a constant amount of food (20)
    - Wild animals – may hunt/retreat
        - After an animal has been discovered, it will appear on the viewport until it is hunted
        - Retreating moves the player back to the previous square
        - Hunting the animal will remove the animal from the viewport, and the player moves into the new square

        - Hunting an animal requires a variable amount of days to heal, which will cost food, but food is also gained from the hunt. The following represents the total food change:
            - Grizzly: -10 food
            - Boar: -5 food
            - Elk: -0 food
            - Hare: +5 food
    - Disease – will take time to rest, costing food:
        - Cold: -5 food
        - Flu: -10 food
        - Dysentery: -15 food
- Winning:
    - The player wins the game if he successfully traverses the Oregon Trail
    - This should display some sort of win screen which displays how much food the player had remaining
    - The screen should require an enter press to send the user back to the splash screen and allow the user to start the game again
- Losing:
    - The player loses the game if he runs out of food at any point
    - This should display a lose screen, which requires an enter press to send the player back to the splash screen, and allow the user to start the game again

Implementation

Map

The map will be read in from a file, which will be passed into the program using a command line argument to the location of the file. (To run the program, you will execute something like "./hw3 inputFile.txt") The map will be read into 2-dimensional linked list of nodes, from which you will be deriving your viewport. Any changes made to the map (by hunting an animal) will be updated to the array map. A sample input file has been given which defines N, M, amount of initial food, and the map layout using the first letter of each danger as the marker (i.e. Grizzly is marked with G, Elk with E, River with R, etc…). The map is surrounded by O's, which is for an impassable ocean space. Anything beyond the map is more ocean.

Viewport

Your viewport (MxM as defined in input file, M must be an ODD number) can be thought of as a mini-map, or what the player can currently see. All squares except ocean, river, your starting point, and the end point are initially marked as undiscovered. Your viewport should reflect this by displaying a 'U' in that square. You will print the viewport after each action. Discovered squares will display the threat that is in the square.

The viewport will be an MxM 2-dimensional linked list. The viewport will be constructed by copying each node from the original map into dynamic memory. Each node will only write back to the map whenever the player will be moving to where that node is no longer displayed. At that point, the node will copy its changed contents back to the map, and then free itself from memory.

If they viewport is large enough to see beyond the limits of the map, those squares should be filled with a temporary ocean node which will not be written back to the map.

Node

A node will be a struct object which contains the following variables:

Threat: A union which contains either a disease or an animal.

ThreatType: An enum which contains either “DISEASE” or “ANIMAL”

XLocation : An integer of the x location in the map

YLocation: An integer of the y location in the map

Discovered: A Boolean value as to whether or not the node has been discovered

Node*Up, *Down, *Left, *Right: 4 pointers to the nodes to the top, left, right, and below the node

A few notes

You should approach the design of this assignment using a hierarchical design methodology. You should begin with the pieces closest to memory, the smallest blocks, and make their functionality correct before moving up. Create unit-tests to make sure that each portion works before moving further up. Your main loop should not have to focus on anything in memory, but rather game states provided by functions from your hierarchy. I suggest having: a topfile which only consists of your main driver which takes command input, and prints game output; a game.h and game.c file which provides all game specific functions like “move right”, “print viewport”, etc…; a map.h and map.c which contains the map, and functions to create, update, and finally free the map, this file may also be leveraged to create the viewport if the size is abstract and variable; and a threat.h and threat.c file which contains the structures and union needed to create all the possible threats on the map, and any specific helper functions which you may wish to write. You do not have to have these specific files, but this is an idea of a structure which may help you. You should, however, have some file organization which makes sense for consistent coding practices.

Lastly, go to your TAs and instructor early with questions regarding the homework. Office hours are a great resource for code specific questions. Piazza is a very good resource for general questions. **DO NOT POST CODE ANYWHERE ONLINE. DO NOT SHARE CODE WITH CLASSMATES**. If you use any code you get from the web, you must **CITE** it according to citation policy. We will be using cheat check software, if you attempt to plagiarize, you will be caught. It is better to turn in nothing than to turn in plagiarized work.

Game Scenarios

This section is used to detail what the input/output should look like at various inputs. Assume the initial map is as follows (assume all spaces are filled with E for elk) Map size: 8x8, viewport 3x3:

O	O	O	O	O	O	O	O	O	O
O	Z								O
O	R	R							O
O		R							O
O		R	R			H	H	H	O
O			R	R	R	R	R	H	O
O						E	R	R	O
O						E	C	E	O
O						E	G	X	O
O	O	O	O	O	O	O	O	O	O

When the user enters the game they will be presented with the current viewport and food status:

Food: 50

U	U	O
U	X	O
O	O	O

This viewport says that the user is surrounded above and to the left by unknown squares, and to the right and below by ocean tiles. As the user cannot pass ocean tiles, a “right” or “down” command would report “Cannot pass ocean tiles” and display the current viewport again. Let’s assume that the user chooses to move left. They will be presented by a screen which reads:

“You have encountered a Grizzly bear, would you choose to hunt (h) or retreat (r)”

If the user chooses to hunt the Grizzly Bear, they would lose 15 food. Let’s assume that the user chooses that this would be unwise, and chooses to retreat instead. The user would be sent back to the current viewport, but it would be updated with the Grizzly Bear instead of U for undiscovered:

Food: 50

U	U	O
G	X	O
O	O	O

At this point, the user chooses to go up. At this point, the user will be presented by as screen which reads:

“You have encountered an Elk, would you choose to hunt (h) or retreat(r)”

If the user chooses to hunt the Elk, they would end up losing 0 food as the elk provides food some food, but requires maintaining camp for some amount of time to repair wounds. We will assume that the user chooses to hunt the Elk. The user will be presented with the new viewport:

Food: 50

R	R	O
U	X	O
G	N	O

The user sees that there is a river above him, and so chooses to move left into the undiscovered square, hoping to find a hole somewhere so he doesn't have to attempt to ford the river. He will be presented with the following message:

"You have contracted the common cold, your food storage drops by 5 as you take a day off to recuperate" At which point, the user will be presented with the new viewport:

Food: 45

U	R	R
U	X	N
U	G	N

After being hit by the cold, the user chooses to ford the river, and hope to make it safely. The user will be presented with the following prompt:

"You are attempting to ford a river; you may either choose to ford (f) or retreat(r)"

The user presses on and fords the river, losing 20 food in the process. The next viewport is displayed:

Food: 25

R	R	U
U	X	R
U	C	N

At this point, the user realizes that they cannot get to the other side of the river in a single motion. They must continue fording to the right or above to cross the river. They choose to move up, knowing that they can escape to the right if they cannot move up after the river, and knowing if they move back down they will get the cold again. They are again presented with:

"You are attempting to ford a river; you may either choose to ford (f) or retreat(r)"

The user presses on and fords the river, losing another 20 food in the process, and presented with the next viewport:

Food: 5

U	U	U
R	X	U
U	R	R

At this point, they can move up or to the right to finish fording the river, and will have to deal with whatever dangers are left with only 5 food.