Project 3

Due Tuesday by 11:59pm **Points** 100 **Available** Mar 10 at 12am - Apr 5 at 12:15am 26 days

Project 3

This is our 3rd of 5 projects. We are turning our attention to arrays and in particular to 2D arrays. In this project we are going to simulate a fire spreading through an area.

Details

A 2D array allows us to represent rectangular data. In this case the data is going to represent an area that is on fire. Each of the grid locations can be in 1 of 4 distinct states. The states will be described below. We will load in our initial grid state and then start our simulation. The simulation will end once all of the fires are out.

Each grid in our world is in 1 of the following 4 states:

n - not burning. An n in our grid represents a grid square that is not burning. A grid that is not burning will have a corresponding moisture value. If there is a grid that is burning next to it, then the moisture level will decrease. Once the moisture level reaches 0 and there is a burning square next to it, then the grid starts burning.

b - burning. A b in our grid represents a grid square that is currently burning. A grid that is burning will have a corresponding burn level. Each iteration of the simulation reduces the burn level by one. Each of the grid squares that are next to a burning grid square will have their moisture level reduced by 1 until they potentially catch fire.

t - burnt. A t in our grid represents a grid that has burnt out. It was on fire, now it's not. It's essentially empty, but is different from empty in that at one point it was not empty and also was once burning.

e - empty. An e in our grid represents a grid that is empty. It cannot catch fire. It is different from t in that it was never on fire nor not burning.

The grid will be of max size 30x30. This means that you can only store this much data at most. Sometimes we will have less than that and sometimes there will be more data in the input files.

We will use a simulation file of the form:

```
Moisture: ww
Burn: xx
Grid: filename
```

ww represents an integer that is the moisture level. All not burning grid locations will have ww as their initial level. xx represents an integer that is the burn level. For the squares that are burning, the burning squares will all have xx as their initial level. The moisture level and burning level will almost always be different numbers.
Empty and burnt grid locations have a moisture and burn level of 0.

I recommend having 2 more 2D arrays of type integer to represent the moisture levels and burning levels.

filename is a file with the grid data in it. This is how the filename file will start:

```
Rows: yy
Columns: zz
```

The entire contents of filename are shown in the next section.
yy and zz are integers. They can be of any size. They may be larger than 30. If they larger than 30, only read up to the 30. In the case where the rows is larger than 30, once you've read 30 integers on a line, then you need to ignore the rest of the line and start the next row on the next line. If there is less than 30, then read just what's there.

The data will be of the form

e b t n

e = empty
b = burning
t = burnt
n = not burning
So we end up with 1) a grid of what the squares are, 2) a grid of the moisture levels, and 3) a grid of the burning levels.
* e and t have moisture level 0
* e and t have burn level 0
* b has moisture level 0 and burn level with whatever it is
* n has moisture level with whatever it has and burn level 0

Input

Your program will be given an initialization file. You are to open that file and read it. It will be in the form of the example below.

Example ini file: simulation.ini

```
Moisture: 3
Burn: 5
Grid: sample.grid
```

So you read the information and initialize the respective grids. You have to load the grid data before you can initialize the moisture and burn grids. The 3rd line is the name of the grid file to open and store.

Sample grid file: sample.grid

```
Rows: 5
Cols: 5

n n n n n
e n n n e
n n b n n
e n n n e
n n n n n
```

So you can read this data. The first two lines will tell you how many rows and columns in the data below. So if the rows or columns is larger than 30, you can adjust the variables you read in to be 30. The characters will be space delimited to make this data easier to read than the last project.

On Web-CAT, I'll make sure the files end in .txt so you can see them for testing. The file extension is arbitrary. I used .ini to represent an initialization file and .grid to represent the grid data. The program doesn't need any particular extension but it helps us the programmers recognize what they are for. You can open these files on your computer on an editor. You should make some of your own and test.

Output

After you've initialized the grids, I want you to show me the initial state of all the data. Here's what the beginning of the output would look like for the samples that are given:

```
Moisture: 3
Burn: 5
Grid_file: sample.grid
Initial State:
nnnnn       33333       00000
ennne       03330       00000
nnbnn       33033       00500
ennne       03330       00000
nnnnn       33333       00000
```

So we start our output out with repeating the moisture and burn levels and the name of the grid file. Then below that, output the grid, the moisture and burn grids. I used two tabs between my grids, but you do not need to match that. Make sure not to space the grid data.

After the initial state is shown, you start the simulation. For each step in the simulation, you output the state. Here's what the first step looks like:

```
Step: 1
nnnnn       33333       00000
ennne       03230       00000
```

```
nnbnn       32023       00400
ennne       03230       00000
nnnnn       33333       00000
```

Once nothing is burning you will show the final state:

```
Final State:
ttttt       00000       00000
ettte       00000       00000
ttttt       00000       00000
ettte       00000       00000
ttttt       00000       00000
```

The final state will show what it looks like at the end.

Preview the output.txt file in samples folder to see the entire output.

Simulation

So a simulation is just something that models a situation. In this case, we are simulating a fire burning. It's a fairly crude simulation in that this isn't really how fire spreads but it allows us to work with 2D arrays. The basic algorithm for how to run the simulation is given below:

While anything is still burning:

- For each grid location, if the location is not on fire check up, down, left and right
- For each of the locations next to it that are on fire, decrement the moisture count for this grid location.
- If that causes the location's moisture content to go below 0, then it becomes on fire
- Else if the location is already on fire, then decrement it's burn level by 1
- If the burn level reaches 0, then it becomes a burnt grid location
- So we are only looking in grid locations in the 4 directions. We are not looking in diagonal directions.

Note Index location 0,0 is the top left corner. When you add 1 to the rows you go down. When you subtract 1 to the rows you go up. When you add 1 to the columns you go right. When you subtract 1 from the columns you go left.

One thing you need to make sure is that when you are at the top of the grid, you don't go up. When you are at the bottom of the grid you don't go down. The same applies to the left and right edges.

This is one of the major focuses of this project, making sure that the index values we use in our array stay within bounds.

Requirements

1. You must declare the function void fire(string input, string output); in a header file fire.h
2. You must write your code in a cpp file.
3. You must use a 2D array of either char or string to store the grid data.
4. You should use at least 2 more 2d arrays for the moisture and burning levels. There may be other ways to do this, but I think this is the easiest.

5. You may not use vector or any other container class from the STL.
6. You may not use structs or classes.
7. You may not use pointers.
8. The size of the array is 30X30.
9. You must write functions for the initialization of the grids and the simulation.
10. Other function should be written for major parts of the functionality. This will help make this easier to do.
11. You may not use global variables. Global constants are fine.
12. You may not work with any other students on this project.
13. You must follow the code style guidelines.

Grading

Zip up your fire.h and fire.cpp and turn them in to Web-CAT. You have 10 submissions.

You will be awarded **bonus points for early submission** : At most 5 points in increments of 1 for every 1 day early. (5 points if 5 or more days early, 4 points if 4 days early and so on)

Late submission: Upto 2 days (-10 for every 1 day late)

Reminder: **Projects are individual work.** Your project code submission will be run through a system know as Moss that was developed at Berkeley. It's purpose is to compare code for similarity and attempts to detect cheating. Cases of Cheating will be reported to the Honor Court.

Due

See Canvas.

Files

- **Project3samples.zip**
- Test Samples: **Project3TestSamples.zip**

Videos

1) This video gives a better understanding on how you could deal with 2D arrays.

-Reading 2D data (in Helpful Videos ->**Reading Data**)

2)This video briefly discusses some points in this project, including running simulations.

-**Project3 ExplainSimulation.mp4**

3) This video talks about 2D array designs to be considered for this Project

Array design.mp4