In this lab exercise, you will extend the forest fire simulation program 1 you started last week. If you missed that lab, you will need to make sure you understand it before starting the exercises in this lab. A sample solution is posted on moodle if you did not complete last week’s lab.

Since the underlying material in this lab is complicated, expect this lab to take some time and be challenging. If you get stuck, review (and discuss with your partner) the relevant sections of the textbook, and the relevant examples from lecture. And feel free to ask for help from the TAs.

Workout (continued from last week...)

1) Bring Some Water!
How fast a fire spreads depends on how wet the trees are. Use the wetness member of the Tree class to indicate how wet a tree is: 1.0 (dry) to 100.0 (soaked). You can set the wetness member using the Tree class setWetness() function.

In the Forest method nextStatus(), divide each Tree object's probCatch value by its wetness value when deciding if the tree will catch fire. For wet trees, this will reduce the probability of catching fire.

Modify your test program (from Challenge Problem 1 or 2 from the last lab) to set each tree in a Forest object to have a high wetness value, and observe the effect when you run your test program (remember to recompile the class files and the test program). If you set the value high enough, the fire will not spread.

2) Regrowth
Nature uses forest fires to get rid of old trees to make way for new ones. Add a member function to the Forest class called regrowth to cause trees to spring up in cells where there are no trees. The regrowth function will have a parameter growProb of type double, where growProb is the probability of a tree starting to grow in an empty cell (a value between 0.0 and 1.0). You will have to add status value to represent new trees. Also, modify the Tree.draw member function to color such new trees blue.

Test your function and then call it in the while loop that performs the simulation applyNextStatus().

3) Inheritance
Using the Tree class as the base class, create derived class Oak. Class Oak will not have any new data members. It will have as public member functions a default constructor, a constructor with parameters, and draw(GraphX &) that uses GraphX methods to draw an oak tree at its x,y location. Oak trees should be displayed as solid circles regardless the symbol in the base class. Put the interface of the Oak class in file Oak.hpp and the implementation in file Oak.cpp.

Write a short test program that creates and draws both oak trees and regular trees. Make sure each tree is drawn using the correct symbol.
4) Inheritance and Polymorphism

After testing the Oak class, derive a similar Pine class. The Pine class draw() method should display a solid triangle to represent a pine tree regardless the symbol in the base class. (Or try adding a "trunk" to your oak and pine trees!)

With the Tree base class and derived classes Oak and Pine, you can explore dynamic polymorphism. You will first need to modify the Tree class to make the draw method a virtual function. Now write a main() program that instantiates a GraphX object and creates a Tree object, an Oak object, and a Pine object. Then create a base class pointer named treePtr, i.e., one that points to a Tree object. Then, in turn, assign the address of each object to treePtr and call draw() using either of the following methods:

\[ (*\text{treePtr}).\text{draw}(\text{graphXobject}); \]
\[ \text{... or ...} \]
\[ \text{treePtr->draw}(\text{graphXobject}); \]

Did you get the correct symbol for each tree? If not, modify the declaration part of the Tree class so that the draw method draws the correct symbol for each object, illustrating polymorphism.

5) Tree Pointers in the Forest Class

In order to use the new Tree, Oak, and Pine classes in the Forest class, the grid array must be changed to a two-dimensional array of Tree pointers. Make this change in the declaration part of the Forest class.

Having done that, all of the function calls in the Forest.cpp file must be changed. For example, a call to setStatus() must be changed from:

\[ \text{grid}[i][j].\text{setStatus}( \text{stat} ); \]
\[ \text{... to ...} \]
\[ \text{grid}[i][j]->\text{setStatus}( \text{stat} ); \]

The Forest class constructor must be modified to assign addresses of objects to grid elements as follows. Initialize each grid array element using the new operator. For example, to assign a Pine tree object to grid location 1,1, do:

\[ \text{grid}[1][1] = \text{new Pine}("pine", .8, 1, 1.0, 1, 25, 25, "solidtriangle"); \]
\[ \text{... or ...} \]
\[ \text{grid}[1][1] = \text{new Pine}; // uses default constructor \]

Initialize all the elements of the grid using this technique. (Think what a reasonable way is to do this while still getting a good variety of trees.) Populate your forest with both pine trees and oak trees. The probCatch for a pine tree should be 0.8 and the probCatch for an oak tree should be 0.5.

Add a destructor to the Forest class to de-allocate all dynamically allocated trees in the grid.
In the applyNextStatus() function, the next_grid array can be an array of Tree objects as before. However, when you copy grid (an array of pointers) to next_grid (an array of Tree objects) you must use the following:

```c++
next_grid[i][j] = *grid[i][j]; // dereference grid
```

Then, to copy next_grid back to grid, you must use:

```c++
*grid[i][j] = next_grid[i][j]; // dereference grid
```

Now test your program to simulate a fire in a forest populated with oak and pine trees.

**Challenge**

Congratulations! You now have a forest fire simulator that can model an interesting forest with different kinds of trees and simulate lightning strikes, wetness, and new growth. Here are some additional things you can do to make your simulator even more interesting:

- In addition to oak and pine trees, use regular trees in your simulation.
- Using the burnTime data member of the Tree class, change the Forest class to have trees that take more than one time step to burn. Use the simulator to explore the effect of this feature.
- Add wind. Wind blowing from a particular direction will determine which way the forest burns.
- Make a game. For example, you could have one player set a fire to a few trees. The other player is given 20 buckets of water to put out the fire. He or she can pour a bucket on a grid location by changing its wetness to 100. Develop the rules for each player’s turn to make it interesting.
- Experiment further with inheritance and/or polymorphism.