CS 202 Spring 2022 - Assignment 3

Virtuals and Abstracts

Overview

Farming is hard work, but farming simulations can be fun. In this assignment, we'll create a simple automated farming simulation to grow and harvest several crops. All of the different kinds of crops will behave differently, with different growth rates and different behavior when harvested.

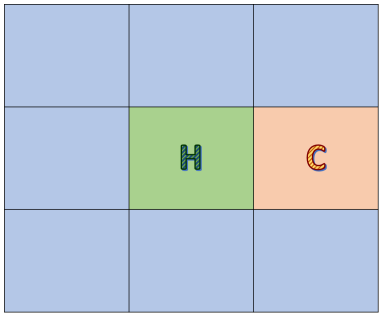
In our simulation, there will be a garden represented by a 2D grid of plots. Within each plot, we will either have an empty spot or a single crop that will grow over time. Our crops will be limited to just three types: Corn, Watermelon, and Sunflowers. Each of the three will inherit from an abstract Crop base class containing a few functions that must be redefined to work differently for each unique crop. This will include getting a character to use for displaying the Crop on the screen as well as a function called when the crop is harvested.

The Garden will be maintained for some number of days picked at the start of the program. On each day, the crops will grow by some percent towards maturity. Once a crop reaches maturity, it will be harvested and plant some new crops depdning on how it behaves. We will keep track of the total number of crops planted as well as how many of each crop was harvested.

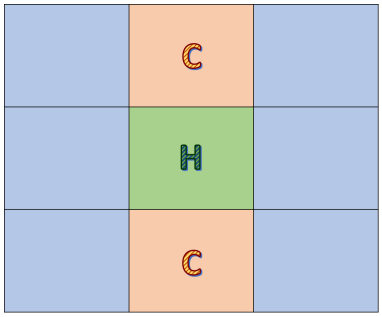
The Crops

When a Crop is harvested, it will drop seeds into adjacent tiles depending on the specific type of Crop. If a space is empty, a new Crop of the same type as the one harvested will be placed in the adjacent tile(s). Below is how each crop behaves, with H being the crop being harvested and C being the Crops that will be planted from the seeds:

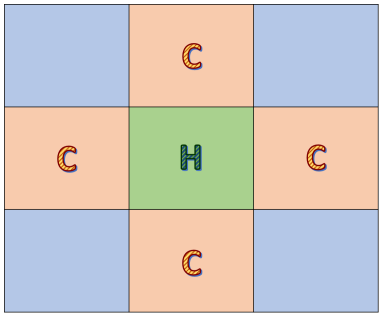
Corn - One to the right



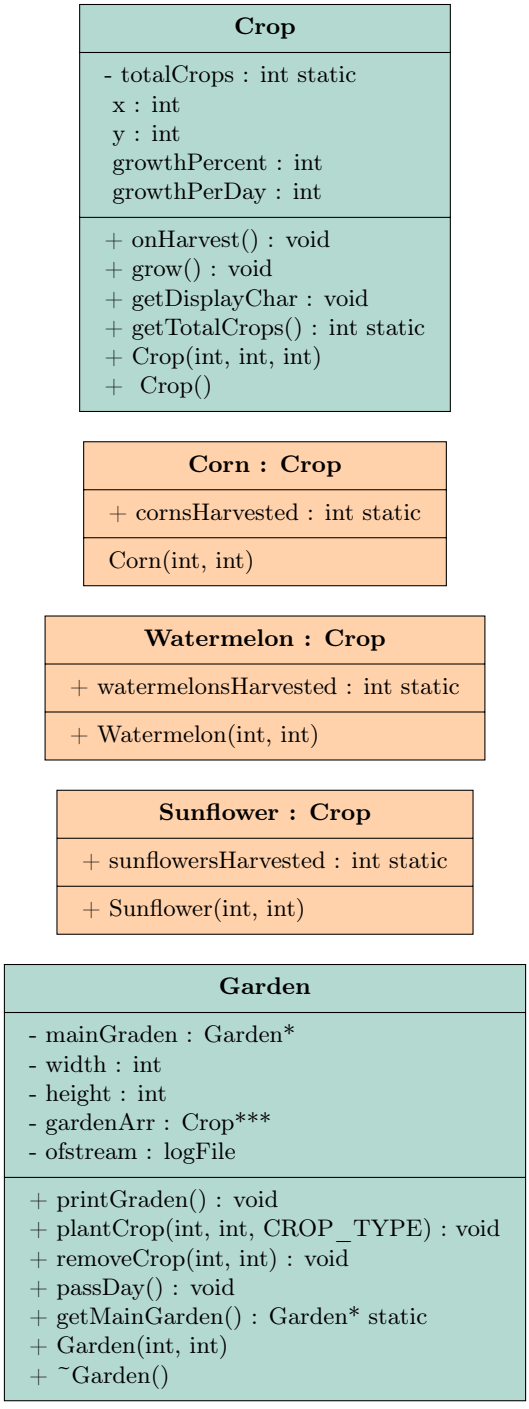
Watermelon - Directly above and below



Sunflower - Above, below, left, and right



UML Diagrams



Important Classes and Functions

Below is a list of functions and variables that are important for this assignment. ***Variables are in green***, **functions that you will need to write are in red**, **functions implemented for you already are in blue**, and **functions that are abstract that will be implemented later are in magenta**.

Global Functions and Constants

- **int main(int argc, char** argv)** -

Crop

Crop will act as an abstract base class for all of our more specific crops to be grown. Recall that an abstract class is one that can't be instantiated. In C++, this is because it contains one or more abstract functions that do not have defined code and therefore cannot be called. However, more generally we use abstract classes when we want to have a base class that is not specific enough to exist on its own. No crop we make can be just a Crop, it must be more specific (Corn, Potato, Tomato, etc.).

- ***static int totalCrops*** - The total number of crops that have been planted. This should be incremented whenever a new Crop is constructed and will be printed at the end of the main program
- ***int x, y*** - The x and y coordinate of our crop in the grid of land that is our Garden. We can use these to tell exactly where the Crop is planted and use these for later functions
- ***int growthPercent*** - The total percent this crop has grown, with 0 being just planted and 100 being 100% mature and ready to harvest
- ***int growthPerDay*** - By what percent this Crop grows in a single day
- **virtual void onHarvest()** - An abstract event function for when a crop has been harvested. Since it is abstract, that means it contains no code in the Crop class, and instead will be given different definitions in each of the classes that derive from Crop. Essentially, this means that different Crops will behave differently when harvested. This function is also considered an event handler. That is, it is only called in response to when a certain condition/event happens, here when a crop is harvested
- **void grow()** - Performs the action of growing this Crop by the amount it grows in a day. This will increase the ***growthPercent*** by the ***growthPerDay*** and then check if the crop is ready to be harvested. If it is, then the **onHarvest** event is triggered and the crop is removed from the Garden
- **virtual char getDisplayChar()** - Returns a character to display the Crop in the Garden. This will change depending on the crop and the percent growth it is at and will be implemented differently in different dervied classes. The derived versions of this function are implemented for you already
- **static int getTotalCrops()** - Accessor for the ***totalCrops*** variable
- **Crop(int growth_per_day, int x, int y)** - Crop constructor that initializes the ***growthPerDay***, ***x***, and ***y*** members to the given 3 parameters and then increments to the total count of crops. Also initializes the initial ***growthPercent*** to 0 to represent that the plan has not grown at all yet
- **~Crop()** - A virtual destructor for the Crop class. This doesn't do anything, but marking it as virtual will allow our derived class constructors to correctly be called without potentially segfaulting

Corn

The Corn class is a specific type of crop that grows by a total of 10% every day and will plant a new Corn in the tile to the right of where this one was harvested. The Corn class is already implemented for you as an example of how to implement virtual functions in a derived class and call the base class constructor from our derived class. You can use the written function as references for the other two Class that inherit from Crop.

- ***static int cornsHarvested*** - A static count of how many corns have been harvested in total. Should only be incremented when a Corn's **onHarvest** function is called
- **void onHarvest()** - This implements the abstract function from the Crop base class. While no crop has any define behavior when it is harvested, this will allow us to have the Corn be harvested in a way different from other Crops. When a Corn is harvested, it increments the total count of Corns that were harvested and then attempts to plant a Corn in the space to the right of where *this*Corn was harvested in our Garden

- **char getDisplayChar()** - Returns a char to display the Corn as based on its total growth. This implements the abstract function from the Crop class. This function exists to add a little bit of visual flair as Crops grow and will be implemented for you in the other two classes as well. Descriptions for those versions will be left out since they are similar to this version

- **Corn(int x, int y)** - The Corn constructor initializes the ***x***, ***y***, and ***growthPerDay*** members by calling the base Crop class constructor with the relevant arguments. This is for reference for the other two Crop classes. This specific function sets the ***growthPerDay*** to 10 and the x and y to the passed parameters

Watermelon

Similar to the Corn class, this is another type of Crop we can grow. The Watermelon grows at a rate of 8% per day and when harvested will try to plant new Watermelons in the tiles directly above and below the harvested Watermelon

- ***watermelonsHarvested*** - Total count of watermelons harvested

- **onHarvest()** - Called when a Watermelon is harvested and implements the abstract version of this function inherited from Crop. This should increment the count of watermelonsHarvested and then attempt to plant a Watermelon in the tiles above and below *this*one.

- **Watermelon(int x, int y)** - Like the Corn constructor, initializes all of the inherited crop members. The growth rate of the Watermelon is 8% per day. Use the Corn constructor code as reference

Sunflower

- ***sunflowersHarvested*** - Total count of sunflowers harvested

- **onHarvest()** - Called when a Sunflower is harvested and implements the abstract version of this function inherited from Crop. This should increment the count of sunflowersHarvested and then attempt to plant a Sunflower in the tiles adjacent to this one above, below, left, and right

- **Sunflower(int x, int y)** - Like the Corn constructor, initializes all of the inherited crop members. The growth rate of the Sunflower is 20% per day. Use the Corn constructor code as reference

Garden

The Garden is a 2D grid of Crops containing some combination of empty spaces, Corn, Watermelon, and Sunflowers. Each Crop can only be placed in one square. This is our focus class and will contain instances of all of the Crops to be grown, grow them each day, and allows us to plant or remove Crops

- ***static Garden* mainGarden*** - A static reference to the Garden being used by main. This is set for you in the constructor. Static references are helpful when a specific object is important to your program and needs to be used frequently

- ***int width, height*** - The width and height of the 2D array grid that makes up our Garden

- ***Crop*** gardenArr*** - A pointer to the 2D array of Crop*s that constitutes are grid. Don't be offput by the three levels of pointers, this is a 2D array of pointers, so the first two *s come from the two dimensions of the array and the final * comes from its contents, being pointers themselves. This is necessary to utilize abstracts in C++.

- ***ofstream logFile*** - The file to write error messages for the garden out to. This is opened in the constructor and closed in the destructor. All logs are already handled by the existing code, the messages generated will be a result of correctly planting crops in the right places from onHarvest() overrides

- **Garden(int height, int width)** - The constructor sets up the Garden by opening its log file, settings the member variables, and allocating a 2D array of Crop*s to represent our grid. Recall that in order to dynamically allocate an with two dimensions, we must first allocate the first dimension as a single column array of pointers, and then have each entry in that column point to an allocated row. Arrays are also generally done row major, so start by allocating an array of size height, then have each row be of size width (HINT do a single allocation and then do a loop for all the row allocations like our example from class). Make sure to also set *this*width and height to the parameter width and height. As a general hint for dynamic allocation, consider that the *new* keyword always returns a pointer, so the type of each allocation should have one less * than the thing it is being stored in. As an example, ***gardenArr*** is a Crop***, so when an array is allocated for it to point to, it should be an array of Crop**. Finally, fill the allocated 2D array with nullptrs to represent an empty Garden
- **~Garden()** - The destructor should clean up everything associated with the Garden. It should close the log file and then deallocate all of the still allocated memory. This means we will need to deallocate not only the ***gardenArr***, but also any Crop*s in that array. Recall that when doing multidimensional dynamic arrays, the deallocation should be in the reverse order of allocation and work from the last dimension up to the first. Wheras we allocated the column first, then the rows, then the contents of the array, we should first deallocate any non-null Crops in the array, then each row in a loop similar to the allocation, then finally deallocate the single column that held all the rows. Keep in mind what should be an array and what is an object when deciding whether to use delete or delete[]
- **void removeCrop(int x, int y)** - This should delete the crop at the given (x,y) location within the ***gardenArr***. As an error check, make sure the contents of that location is first not null, then delete the crop that was there and set that position to nullptr. One thing to consider is that when doing row major arrays, the height comes before the width, so y should be the first dimension accessed and then x second (So, access gardenArr[y][x])
- **void plantCrop(int x, int y, CROP_TYPE type)** - Plants a crop of the given type (CORN, WATERMELON, or SUNFLOWER) at the given (x, y) position in the ***gardenArr***. If planting a crop in this location would be out of bounds of our garden or would be planted on top of another Crop, an error message is logged instead and no Crop is planted. This uses the enum CROP_TYPE defined at the top of garden.h
- **void printGarden()** - Prints all of the Garden, using each Crop's display char to represent it visually
- **void passDay()** - Updates the Garden by a single day, making all Crops grow once
- **static Garden* getMainGarden()** - Returns a reference to the Garden being used

TO-DO / HINTS

In terms of difficulty, it would be recommended to do the class that derive from Crop first and then move on to the Garden functions, however, if you want to work in order of what main needs, it might be recommended to start with Garden, make sure the allocations work, and then move on to the Crops. The Watermelon and Sunflower constructors do need to be finished before the program will compile.
There are a lot of functions, but don't be overwhelmed, you only need to work on a few that call the existing written functions. Slowly break down each function and consider what just that one function should do when working.
A makefile is again provided and the program can be compiled with **make** on a Unix system like Linux or Mac. The program can be run one of two ways, either with no arguments or a single int that will seed the random number generation. Ex:
./garden
./garden 180

Sample Run

Below is a sample run from Sally with the random seed 1200. This generates two sunflowers and a corn that grow over 12 days. On day 4, each of the sunflowers reach maturity and are harvested, planting more sunflowers in adjacent tiles. Following the sample run is the output logged to the *garden_ log.txt* file, which logs any issues encountered while running. Please note that RNG functions may work differently on different systems, so you **may get a different output on Windows or Mac even with the same seed**.

```
blacks1@sally:~/CS202/Summer2022$ ./garden 1200
Please enter the width of the Garden plot
4
Please enter the height of the Garden plot
5
How many days should the Garden go for?
12
Populating Garden...
-------------- Garden Day 0--------------
----------
| | | |.|
| | | | |
| | |'| |
| | | |'|
| | | | |
----------

-------------- Garden Day 1--------------
----------
| | | |.|
| | | | |
| | |'| |
| | | |'|
| | | | |
----------

-------------- Garden Day 2--------------
----------
| | | |c|
| | | | |
| | ||| |
| | | |||
| | | | |
----------

-------------- Garden Day 3--------------
----------
| | | |c|
| | | | |
| | ||| |
| | | |||
| | | | |
----------

-------------- Garden Day 4--------------
----------
| | | |c|
```

```
| | | | |
| | |O| |
| | | |O|
| | | | |
----------

---------- Garden Day 5----------
----------
| | | |c|
| | |'| |
| |'| |'|
| | |'| |
| | | |'|
----------

---------- Garden Day 6----------
----------
| | | |c|
| | |'| |
| |'| |||
| | ||| |
| | | |||
----------

---------- Garden Day 7----------
----------
| | | |c|
| | ||| |
| ||| |||
| | ||| |
| | | |||
----------

---------- Garden Day 8----------
----------
| | | |C|
| | ||| |
| ||| |O|
| | |O| |
| | | |O|
----------

---------- Garden Day 9----------
----------
| | | |C|
| | |O|'|
| |O|'| |
| |'| |'|
| | |'| |
----------

---------- Garden Day 10----------
----------
| | |'| |
```

```
| |'| |'|
|'| |'| |
| |'| |||
| | ||| |
----------

-------------- Garden Day 11--------------
----------
| | |'| |
| |'| |||
|'| ||| |
| ||| |||
| | ||| |
----------

--------------Final Garden--------------
----------
| | ||| |
| ||| |||
||| ||| |
| ||| |O|
| | |O| |
----------

Total number of crops planted: 16
Corn harvested: 1
Watermelons harvested: 0
Sunflowers harvested: 7
Crops unharvested: 8
```

Garden Log from this Run

```
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
Tried to plant a crop out of bounds at (4, 3)
Tried to plant a crop out of bounds at (4, 2)
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
Tried to plant a crop out of bounds at (3, 5)
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
Tried to plant a crop out of bounds at (4, 4)
Tried to plant a crop out of bounds at (4, 0)
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
Could not plant crop on top of existing crop
```