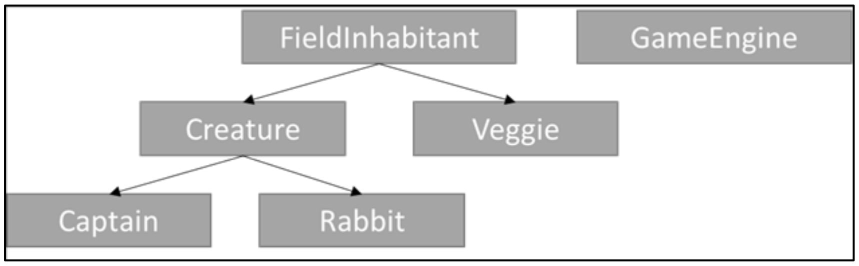
Project 2: Captain Veggie

Program Description:

For this project you will be creating the game Captain Veggie, in which rabbits have invaded the good captain's field and so the captain must harvest as many of their vegetables as possible before they are consumed by the leporine menace. In the game, Captain Veggie should be able to move about the field and attempt to harvest vegetables by moving on top of them. The captain scores points for each vegetable they harvest. At the same time, the rabbits should also be randomly hopping about consuming any vegetables they should land on. The game will continue until all of the vegetables have been removed from the field, after which the player's score will be displayed to the screen.

Program Requirements

- This a team-based project, as such, only one member of your group should create a private repository on GitHub to store the code and add the other group members to it
    - Each group member should make at least 3 substantial commits to the group repository during development
    - Consider making a commit to the repository after completing a one of the super or subclasses or after completing a major function
- The veggie file contains the comma delimited information regarding the initial configuration of the game:
    - The first line will always specify the field size, with the height first and the width second
    - All other lines will have the name of a vegetable, the letter representing that vegetable, and then the number of points that vegetable is worth
    - The vegetables will be in no particular order
    - See example input files
- Your program will need several classes that may utilize inheritance and polymorphism to properly function in the program:
    - `FieldInhabitant`
    - `Veggie`
    - `Creature`
    - `Captain`
    - `Rabbit`
    - `GameEngine`
- The class hierarchy should look like the following:



- All class scoped variables must be `private` or `protected` depending on how they are or are not used across the different classes in the program
- In a file named **FieldInhabitant.h**, define a class named `FieldInhabintant`, which should contain:
  - A string member variable to store a symbol
  - A declaration for a public constructor that takes in a parameter representing a string symbol for the field inhabitant (vegetable, rabbit, captain, etc)
  - Declarations for appropriate getter/setter functions
  - A declaration for a virtual deconstructor
  - Appropriate header guards
- In a file named **FieldInhabitant.cpp**:
  - Define the constructor so it stores the parameter value in the appropriate member variable
  - Define the getter/setter functions
  - Define the deconstructor so that is does nothing
- In a file named **Veggie.h**, define a class named `Veggie`, that is a `public` subclass of `FieldInhabitant`, and which should contain:
  - A string member variable to store the name of the vegetable
  - An integer member variable to store the point value of the vegetable
  - A declaration for public constructor that takes in two string parameters representing the name and symbol of the vegetable and an integer representing the number of points the vegetable is worth
  - Declarations for appropriate getter/setter functions
  - Appropriate header guards
- In a file named **Veggie.cpp**:
  - Define the constructor so it stores the parameter values in the appropriate member variables
    - The superclass's constructor should be called and the symbol should be passed to it
  - Define the getter/setter functions
- In a file named **Creature.h**, define a class named `Creature`, that is a `public` subclass of `FieldInhabitant`, and which should contain:
  - Two integer member variables to store the x and y coordinates of the creature
  - A declaration for public constructor that takes in two integer parameters representing the x and y coordinates of the creature and a string representing the symbol of the creature
  - Declarations for appropriate getter/setter functions
  - Appropriate header guards
- In a file named **Creature.cpp**:
  - Define the constructor so it stores the parameter values in the appropriate member variables
    - The superclass's constructor should be called and the symbol should be passed to it
  - Define the getter/setter functions

- In a file named **Captain.h**, define a class named `Captain`, that is a `public` subclass of `Creature`, and which should contain:
  - A vector of Veggie pointers storing all of the `Veggie` objects the captain has collected
  - A declaration for public constructor that takes in two integer parameters representing the x and y coordinates of the captain
  - A declaration for a public function named `addVeggie()` that takes in pointer to a `Veggie` object as a parameter, returns nothing, and adds the object to the vector of `Veggie` objects
  - Declarations for appropriate getter/setter functions
  - Appropriate header guards
- In a file named **Captain.cpp**:
  - Define the constructor so it stores the parameter values in the appropriate member variables
    - The superclass's constructor should be called and the x and y coordinates as well as the string "V" should be passed to it
  - Define `addVeggie()` so that it adds the `Veggie` pointer to the captain's vector
  - Define the getter/setter functions
- In a file named **Rabbit.h**, define a class named `Rabbit`, that is a `public` subclass of `Creature`, and which should contain:
  - A declaration for public constructor that takes in two integer parameters representing the x and y coordinates of the captain
  - Declarations for appropriate getter/setter functions
  - Appropriate header guards
- In a file named **Rabbit.cpp**:
  - Define the constructor so it stores the parameter values in the appropriate member variables
    - The superclass's constructor should be called and the x and y coordinates as well as the string "R" should be passed to it
  - Define the getter/setter functions
- In a file named **GameEngine.h**, define a class named `GameEngine`, which should contain:
  - A `FieldInhabitant` triple pointer for storing a 2D dynamic array of `FieldInhabitant` pointers
  - Integers to store the height and width of the field, and the player's score
  - Constant integers to store the initial number of vegetables in the game named `NUMBEROFVEGGIES`, initialized to 30, and the number of rabbits in the game named `NUMBEROFRABBITS`, initialized to 5
  - A `Captain` pointer to store the captain object
  - A vector of `Rabbit` pointers to store the rabbit objects
  - A vector of `Veggie` pointers to store all of the possible vegetable objects
  - Declarations for private functions named `initVeggies()`, `initCaptain()`, and `initRabbits()` that take in no parameters and return nothing
  - Declarations for private functions named `moveCptVertical()` and `moveCptHorizontal()` that take in an integer representing the movement of the captain as a parameter and return nothing

  - Declarations for public functions named `initializeGame(), intro(), printField(), moveRabbits(), moveCaptain(),` and `gameOver()` that take in no parameters and return nothing
  - Declarations for public functions named `getScore() and remainingVeggies()` that take in no parameters and return integers representing the player's score, and the number of vegetables still remaining on the field, respectively
  - Appropriate header guards
- In a file named **GameEngine.cpp**:
  - Define the function `initializeGame()` such that:
    - The `initVeggies()` method is called
    - The `initCaptain()` method is called
    - The `initRabbits()` method is called
    - `score` is initialized to 0
  - Define the function `initVeggies()` such that:
    - The user is prompted for the name of the veggie file, and if the user's file name doesn't exist, repeatedly prompts for a new file name until a file that does exist is provided
    - The height and width of the field should be read in and stored in the appropriate member variables
    - The remaining lines in the files should be used to create new `Veggie` objects that are added to the vector of possible vegetables
    - Generate the 2D, dynamic array of `FieldInhabitant` pointers of the dimensions specified in the file
      - All slots should be initialized to `nullptr`
    - The field should be populated with `NUMBEROFVEGGIES` number of new `Veggie` objects, located at random locations in the field
      - If a chosen random location is occupied by another `Veggie` object, repeatedly choose a new location until an empty location is found
      - Make sure you seed your random number generator to have new random fields each time you play the game
    - Do not forget to close your file after you are done reading from it!
  - Define the function `initCaptain()` such that:
    - A random location is chosen for the `Captain` object
      - If a chosen random location is occupied by another object, repeatedly choose a new location until an empty location is found
    - A new `Captain` object is created using the random location and the object is stored in the appropriate member variable and to the random location the field

- Define the function `initRabbits()` such that:
  - For `NUMBEROFRABBITS`, a random location is chosen for a `Rabbit` object
    - If a chosen random location is occupied by another object, repeatedly choose a new location until an empty location is found
    - A new `Rabbit` object is created using the random location and the object is added to the member variable vector of rabbits and assigned to the random location in the field
- Define the function `remainingVeggies()` such that it examines the field and returns the number of vegetables still in the game
  - Remember that you only want to count `Veggie` objects, so make sure to test what class of object is in a particular slot in the field, before you increment the count of vegetables
- Define the function `intro()` such that:
  - The player is welcomed to the game
  - The premise and goal of the game are explained
  - The list of possible vegetables is output including each vegetable's symbol, name, and point value
  - Captain Veggie and the rabbit's symbols are output
  - Remember that you are informing the user about the game, so be sure to include appropriate messages and descriptions
- Define the function `printField()` such that:
  - The contents of the field are output in a pleasing 2D grid format with a border around the entire grid
- Define the function `getScore()` that returns the player's current score
- Define the function `moveRabbits()` such that:
  - Each `Rabbit` object in the vector of rabbits is moved up to 1 space a random x,y direction
    - Thus, the rabbit could move 1 space up, down, left, right, any diagonal direction, or possibly not move at all
    - If a `Rabbit` object attempts to move outside the boundaries of `field` it will forfeit its move
    - If a `Rabbit` object attempts to move into a space occupied by another `Rabbit` object or a `Captain` object it will forfeit its move
    - If a `Rabbit` object moves into a space occupied by a `Veggie` object, that `Veggie` object is removed from `field`, and the `Rabbit` object will take its place in `field`
    - Note that `Rabbit` object's appropriate member variables should be updated with the new location as well
    - Make sure you set the `Rabbit` object's previous location in the field to `nullptr` if it has moved to a new location

- Define the function `moveCptVertical()` such that:
    - If the `Captain` object's current position plus the movement would move them into an empty slot in the field
        - Update their appropriate member variable
        - Assign them to the new location in the field
    - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Veggie` object
        - Update the `Captain` object's appropriate member variable
        - Output that a delicious vegetable, using the `Veggie` object's name, has been found
        - Add the `Veggie` object to the `Captain` object's vector of `Veggies` using the appropriate function
        - Increment the player's `score` using the `Veggie` object's point value
        - Assign the `Captain` object to the new location in the field
    - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Rabbit` object
        - Inform the player that they should not step on the rabbits
        - Do not move the `Captain` object
    - Make sure you set the `Captain` object's previous location in the field to `nullptr` if it has moved to a new location
- Define the function `moveCptHorizontal()` such that:
    - If the `Captain` object's current position plus the movement would move them into an empty slot in `field`
        - Update their appropriate member variable
        - Assign them to the new location in the field
    - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Veggie` object
        - Update the `Captain` object's appropriate member variable
        - Output that a delicious vegetable, using the `Veggie` object's name, has been found
        - Add the `Veggie` object to the `Captain` object's vector of `Veggies` using the appropriate function
        - Increment the `score` by the `Veggie` object's point value
        - Assign the `Captain` object to the new location in the field
    - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Rabbit` object
        - Inform the player that they should not step on the rabbits
        - Do not move the `Captain` object
    - Make sure you set the `Captain` object's previous location in the field to `nullptr` if it has moved to a new location

- Define the function `moveCaptain()` such that:
  - The user is prompted for which direction to move the `Captain` object in, Up(W), Down(S), Left(A), or Right(D)
    - You should accept both uppercase and lowercase W,A,S,D for the movement
  - Check the player's input using a `switch` such that:
    - In the case of W or w:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptVertical()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - In the case of S or s:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptVertical()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - In the case of A or a:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptHorizontal()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - In the case of D or d:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptHorizontal()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
  - Otherwise, inform the user that their input is not a valid option and do not move the `Captain` object
- Define the function `gameOver()` such that:
  - The player is informed the game is over
  - The names of all of the vegetables the `Captain` object harvested are output
  - The player's score is output
  - Remember that you are informing the user about the game, so be sure to include appropriate messages and descriptions

- In a file named **main.cpp,** you should have:
  - Your `main` function in which:
    - You instantiate and store a `GameEngine` object
    - Initialize the game using the appropriate `GameEngine` function
    - Display the game's introduction using the appropriate `GameEngine` function
    - Create an integer variable to store the number of remaining vegetables in the game, initialized using the appropriate `GameEngine` function
    - While there are still vegetables left in the game
      - Output the number of remaining vegetables and the player's score
      - Print out the field using the appropriate `GameEngine` function
      - Move the rabbits using the appropriate `GameEngine` function
      - Move the captain using the appropriate `GameEngine` function
      - Determine the new number of remaining vegetables using the appropriate `GameEngine` function
    - Display the Game Over information using the appropriate `GameEngine` function
- No class or globally scoped variables, other than those specified are allowed for this project
- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, date, and brief description), comments for each variable, and commented blocks of code

Submission

- Your program will be graded largely upon whether it works correctly
- Your program will also be graded based upon your program style. This means that you should use comments (as directed) and meaningful variable names
- You must submit the **FieldInhabitant.h, FieldInhabitant.cpp, Creature.h, Creature.cpp, Captain.h, Captain.cpp, Rabbit.h, Rabbit.cpp, Veggie.h, Veggie.cpp, GameEngine.h, GameEngine.cpp,** and **main.cpp** files
- You must submit the URL link to your repository and set the repository from private to public three days after the due date (i.e. the day after the second late day)
- You must work in teams of 2 or 3 students for this project. You are not allowed to work with individuals outside of your team, other than the instructor and TA. Any discovered instances of this will be considered cheating and appropriate actions will the taken according to the course syllabus
- Additionally, you are not allowed to download code off of the internet or use generative AI for this project. Any discovered instances of this will be considered cheating and appropriate actions will the taken according to the course syllabus
- Be sure that you have tested the version of the program you wish to submit to make sure it works correctly. You will not be allowed to resubmit work after the deadline
- All students are expected to contribute relatively equally with respect to the coding for this project. If it is determined that one or more members of the team provided little to no substantive effort with respect to the coding, those member's project grades will be significantly penalized. If you are having issues with a teammate, please contact your instructor as soon as possible

Rubric

The entire assignment is worth 100 points and partial credit is possible. No credit will be given for portions of the program that cannot be tested due to the program crashing.

- **Program Executes Successfully**
  - If your program fails to compile 10 points will be deducted from the project, but I will try to fix minor issues (incorrect indentations, stray character, missing import) so that I can execute and test the program. I will not fix major issues that would require functionality to be further implemented, or a reorganization of logic in your code.
- **Data Storage (10 points)**
  - Each of the data storage classes `FieldInhabitant`, `Creature`, `Captain`, `Rabbit`, and `Veggie` are setup as specified and use inheritance as appropriate
- **Game Initialization (25 points)**
  - Requested functions are used to initialize the game (5 points)
  - Vegetable data is read in and stored (5 points)
  - `Veggie` objects are stored at random locations in `field` (5 points)
  - A `Captain` object is stored in a variable and at a random location in `field` (5 points)
  - `Rabbit` objects are stored both in the `vector` and at random locations in `field` (5 points)
- **Game Play (35 points)**
  - Program correctly moves all `Rabbit` objects adhering to requirements (10 points)
  - Program correctly prompts the player for which direction the captain should move and handles all inputs as requested (5 points)
  - Program correctly moves the `Captain` object vertically, adhering to requirements (10 points)
  - Program correctly moves the `Captain` object horizontally, adhering to requirements (10 points)
- **Program Output (20 points)**
  - Program correctly outputs all required welcome information at the beginning of the game (5 points)
  - Program correctly outputs the number of remaining vegetable and player's score before asking where the captain wants to move (5 points)
  - Program correctly prints the field before asking where the captain wants to move (5 points)
  - Program correctly outputs all required game over information (5 points)
- **Each group member made at least three substantial commits to GitHub (5 points)**
- **Program contains sufficient comments (5 points)**

Bonus

For 10 bonus points, implement a new class named `Snake` in a **Snake.h** and **Snake.cpp** file. The `Snake` class should inherit from the `Creature` class, and should contains a constructor that takes in `x` and `y` parameter variables and provides them to the superclass's constructor along with the letter `"S"` for the symbol.

In your `GameEngine` class, you should:

- Declare a new private member variable to store a pointers `Snake` object
- Declare and separately define a function named `initSnake()` that instantiates a new `Snake` object in a random, unoccupied slot in the field. Be sure to store the object in the appropriate `GameEngine` member variable This function should be called by your `initializeGame()` function after you have initialized the rabbits
- Declare and separately define a function named `moveSnake()` that attempts to move the snake on the field. The snake can only move up, down, left, or right (not diagonally), cannot move out of the field, and cannot move into a space occupied by a vegetable or a rabbit. When the snake moves, it must always try to move closer to the captain object's position. If the snake ever attempts to move into the same position as the captain, the captain loses the last five vegetables that were added to their basket and the snake is reset to a new random, unoccupied position on the field. `moveSnake()` should be called in your **main.cpp** file after the captain has moved