Paul Min
07/19/18
CS-162 Summer 2018

Description: This is my design and reflection document for project 2: Zoo Tycoon. In it I outline my approach to coding the Zoo Tycoon game.

Design

Programming style and documentation

I will follow the coding style guidelines provided on Canvas. I will use comments to help document my code. I will check for memory leaks. I will make sure to de-allocate any dynamic memory that was allocated. I will validate input when necessary. I will make sure to submit all my files in a zip archive. I will also allow the user to exit the game. I will have separate .hpp and .cpp files for animal, tiger, turtle, and penguin classes.

Create the Animal class and object

The animal class will have the following data members:

- int age
  - Adult if age >= 3 days
  - Baby if age < 3 days
- int cost
  - Tiger cost $10,000
  - Penguin cost $1,000
  - Turtle cost $100
- int numberOfBabies
  - Tigers have 1 baby
  - Penguins have 5 babies
  - Turtles have 10 babies
- int baseFoodCost
  - I plan on setting this as a constant
  - Tigers will have a feeding cost that is 5 * baseFoodCost
  - Penguins will have a feeding cost that is the same as the baseFoodCost
  - Turtles will have a feeding cost that is ½ the baseFoodCost
- int payoff
  - Tiger's payoff is $2,000
  - Penguin's payoff is $100
  - Turtle's payoff is $5

Create the Penguin, Turtle, and Tiger class objects

The penguin, turtle, and tiger class objects will inherit the animal class data members and member functions. I plan on writing a default constructor for each that will set their cost, numberOfBabies, baseFoodCost, payoff, and age to what is required by the assignment specifications.

Implement the program to manage the feed cost and payoff at the end of each day

At the beginning of the day:

- Increase all animals ages by 1
- Get number of penguins * baseFoodCost = penguinFee
- Get number of turtles * ½ baseFoodCost = turtleFee
- Get number of tigers * 5 baseFoodCost = tigerFee
- penguinFee + turtleFee + tigerFee = dayFeedCost

At the end of the day:

- Get number of penguins * penguinPayoff
- Get number of turtles * turtlePayoff
- Get number of Tigers * tigerPayoff
- Check to see if there is a bonus, add to the profit.

Implement the random events

To implement the random events I will have a function called random events. It will generate a random number between 1 - 4. I will use a switch statement with 4 cases.

Case 1: Sickness occurs

- generate a random number (1-3)
- Tigers are 1, penguins are 2, turtles are 3
- Use switch statement with 3 cases
    - Case 1:
        - Remove 1 tiger
    - Case 2:
        - Remove 1 penguin
    - Case 3:
        - Remove 1 turtle

Case 2: **Boom in attendance occurs**

- Generate a random bonus per day, 250-500 dollars for each tiger in the zoo
    - Bonus = Generate random number from 1 - 251 and add 249 to it
    - numTigers * bonus = tigerBonus
    - getMoney = getMoney + tigerBonus

Case 3: **Baby animal is born**

- generate a random number (1-3)
- Tigers are 1, penguins are 2, turtles are 3, cheetahs are 4
- Use switch statement with 4 cases
    - Case 1:
        - Iterate through tiger array up to numTigers.
        - Get age of all the tigers you own.
        - If age >= 3, add 1 tiger with age 0.
        - Set bool flag to false.
        - else if no tiger is age >= 3
            - Pick penguin, iterate through penguins to see if there is penguin age >= 3.
                - If there is a penguin age >= 3
                    - Add 5 penguins with age 0
                - Else if there is no penguin age >= 3
                    - Pick turtle, iterate through turtles to see if there is turtle age >= 3.
                        - If there is turtle age >= 3
                            - Add 10 turtles with age 0
                        - Else if there is no turtle age >= 3
                            - Cout << Your zoo has no adult animals. No baby was born.
    - Case 2:
        - Follow similar logic as case 1
    - Case 3:
        - Follow similar logic as case 1
    - Case 4:
        - Follow similar logic as case 1

Case 4: **Nothing happens**

Implement the game loop for each “day”

- While (money > 0) run the loop
- Increase all animals age by 1
- Deduct feeding cost
- Random event
- Calculate profit based off each animal and their payoff.
- If there is a bonus, add it to the profit as well.
- Ask player if they want to buy an adult animal
    - If yes
        - Ask for type of animal they would like?
        - Add animal age 3 to zoo
        - Subtract cost from bank
- Prompt user to keep playing or end game. If the user has no money print message to tell the user game is over, and end the game.

Resize the dynamic array correctly when the number of items exceeds the list capacity

- I plan on creating an entirely new array of the same type and of the new size.
    - In this case, I will double the initial starting size of 10.
    - So 10 becomes 20, 20 becomes 40, and so forth
- Copy the data from the old array into the new array(keeping them in the same positions)
    - I will use a for loop for this
- Delete the old array
- Change the pointer to the new address.

Implement input validation functions

- I am going to reuse or slightly modify the input validation functions that I previously created for the project 1.

Extra Credit

- For the extra animal I created a function that prompted the user to enter values for cost, feedcost, payoff, and babies. I also created private data members to hold those values in my zoo class.
- I also decided to make my extra animal a mandatory purchase like all the other zoo animals.
- For the status messages, I created 4 separate methods that could called to open a text file and read and display output to the console.

Test table

Test Case	Input Values	Driver Function	Expected Outcomes	Observed Outcomes
Input too low	Input < 1	promptFirstTiger() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too high	Input > 2	promptFirstTiger() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too low	Input < 1	promptFirstPenguin() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too high	Input > 2	promptFirstPenguin() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too low	Input < 1	promptFirstTurtle() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too high	Input > 2	promptFirstTurtle() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too low	Input < 1	promptFirstCheetah() oneOrTwo()	Invalid input. Please choose between the	Invalid input. Please choose between the

			integer ‘1’ or the integer ‘2’	integer ‘1’ or the integer ‘2’
Input too high	Input > 2	promptFirstCheetah() oneOrTwo()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input in correct range	1	promptFirstTiger() promptFirstPenguin() promptFirstTurtle() promptFirstCheetah() oneOrTwo()	Add 1 animal of tiger, or penguin, or turtle, or cheetah depending on which driver function is called. Subtract cost of one of the animal type chosen from money.	Add 1 animal of tiger, or penguin, or turtle, or cheetah depending on which driver function is called. Subtract cost of one of the animal type chosen from money.
Input in correct range	2	promptFirstTiger() promptFirstPenguin() promptFirstTurtle() promptFirstCheetah() oneOrTwo()	Add 2 animal of tiger, or penguin, or turtle, or cheetah depending on which driver function is called. Subtract cost of one of the animal type chosen from money.	Add 2 animal of tiger, or penguin, or turtle, or cheetah depending on which driver function is called. Subtract cost of one of the animal type chosen from money.
Input too low Input too high	Input < 1 Or Input > 20000	promptCheetahInfo() cheetahCostValidation()	Invalid input. Please enter an integer between ‘1’ and ‘20000’	Invalid input. Please enter an integer between ‘1’ and ‘20000’

Input in correct range	Input >= 1 And Input <= 20000 I.e 10000	promptCheetahInfo() cheetahCostValidation()	cheetahCost = cheetahCostValidation() Cheetahs cost $10000	cheetahCost = cheetahCostValidation() Cheetahs cost $10000
Input too low Input too high	Input < 1 Or Input > 10	promptCheetahInfo() cheetahBabyValidation()	Invalid input. Please enter an integer between ‘1’ and ‘10’	Invalid input. Please enter an integer between ‘1’ and ‘10’
Input in correct range	Input >= 1 and Input <= 10 I.e. 3	promptCheetahInfo() cheetahBabyValidation()	cheetahBabies = cheetahBabyValidation() Cheetahs have 3 babies	cheetahBabies = cheetahBabyValidation() Cheetahs have 3 babies
Input too low Input too high	Input < 2 or Input > 200 or input is odd	promptCheetahInfo() cheetahFoodCostValidation()	Invalid input. Please enter an even integer between ‘2’ and ‘200’	Invalid input. Please enter an even integer between ‘2’ and ‘200’
Input in correct range	Input >= 2 and Input <= 200 and input is even I.e. 100	promptCheetahInfo() cheetahFoodCostValidation()	cheetahFeedCost = cheetahFoodCostValidation() Cheetah food costs $100 per cheetah.	cheetahFeedCost = cheetahFoodCostValidation() Cheetah food costs $100 per cheetah.
Input too low Input too high	Input < 1 or Input > 5000	promptCheetahInfo() cheetahPayoffValidation()	Invalid input. Please enter an even integer between ‘1’ and ‘5000’	Invalid input. Please enter an even integer between ‘1’ and ‘5000’

Input in correct range	Input >= 1 and Input <= 5000 I.e. 2000	promptCheetahInfo() cheetahPayoffValidation()	cheetahPayoff = cheetahPayoffValidation() Cheetahs generate $2000 per cheetah Call promptFirstCheetah()	cheetahPayoff = cheetahPayoffValidation() Cheetahs generate $2000 per cheetah Call promptFirstCheetah()
Input too low Input too high	Input < 0 Or Input > 2	promptAdultAnimal()	Invalid input. Please enter an integer between ‘1’ and ‘2’	Invalid input. Please enter an integer between ‘1’ and ‘2’
Input in correct range	2	promptAdultAnimal()	Displays Menu with choice to buy an animal. You can choose from 1 - 4. 1) Tiger 2) Penguin 3) Turtle 4) Cheetah	Displays Menu with choice to buy an animal. You can choose from 1 - 4. 5) Tiger 6) Penguin 7) Turtle 8) Cheetah
Input too low Input too high	Input < 0 Or Input > 2	promptKeepPlaying()	Invalid input. Please enter an integer between ‘1’ and ‘2’	Invalid input. Please enter an integer between ‘1’ and ‘2’
Input in correct range	2	promptKeepPlaying()	Game over! The end!	Game over! The end!
Input in correct range	1	promptKeepPlaying()	Calls checkMoney(); If money > 0 Loops through gameLoop	Calls checkMoney(); If money > 0 Loops through gameLoop

Reflection

Zoo was a big project. My Zoo.cpp file was over 1300 lines of code. I think the biggest challenge of this project was figuring out how to resize dynamic arrays and also using anonymous objects to add animals.

I found a great resource online that showed me how to dynamically resize arrays. I referenced that resource to help me implement my addTiger, addPenguin, addTurtle, and addCheetah methods.

I also realized I was breaking up this program into smaller pieces. From a first glance at the project specifications, I felt overwhelmed at just how much work I had to do. But once I realized that I could break this project into smaller pieces, the work became much more manageable.

One problem that I ran into while testing was a read size 8 error. I was accessing an array out of bounds whenever the number of animals reached the size of the array. The error was that I was resizing my array to double the size and then I was copying all elements of my old array into my new array. As I was doing that, I was accessing elements in my old array that were out of bounds. I fixed this by making sure I wasn't accessing the array out of bounds.

The biggest thing that this lab taught me was inheritance. My tiger, penguin, turtle, and cheetah classes all inherited from animal. Inheritance saved me a lot of work from having to write all the member variables and member functions again.

Lastly, I realized that there was a more efficient way to structure my program than I had done so in previous assignments. Instead of creating a ton of local variables and then passing them as arguments to my various member functions, I decided to create private data members that would allow me to have a somewhat efficient zoo class. These private data members kept track of things like the money, number of various animals, the day, etc.