David An
Project 2: Design, Test, and Reflection

Design: The main goal for this program was to create a zoo simulation for a user to control. The program uses different classes in order to purchase animals (Tigers, Penguins, Turtles) that each have their own behavior and properties. Though these animals exhibit different characteristics such as cost and the number of babies they have, they are similar in that they share the main methods and characteristics. Because of this, I will need to create an abstract Animal class that contains the basic behavior of the animals (such as age, cost, and payoff) that can then be inherited by the specific animal classes. By doing so, I can avoid having to re-write the same functions for each animal and can instead write generic setter and getter methods in the Animal class. The next step is to create a Zoo class where all of the game's functions exist. This will make it easy to design the game flow and execute it in the main method. The Zoo class is where I will write the functions to initially set up the game by asking the user how of each animal they would like to purchase, incrementing the age of each animal, executing the random event for the day, and then calculating the profit and displaying it to the user. The Zoo class will also be useful since it is a way to keep track of various things going on in the zoo, such as the dynamic arrays representing the current number of each animal in the zoo, the total funds available to the user, and the daily profit that is calculated at the end of each day. An important design note is to make sure that whenever the user is adding animals to their zoo, if the array is not big enough to hold the desired final number of animals, I must create another array that has the capacity to hold everything and delete the old array. Because of this, I'll need to keep track of the number of animals (Tigers, Penguins, or Turtles) that are in the zoo, as well as the size of the array and check these two variables whenever animals are being added. Another important note is that since this program is interactive, I will need to incorporate some input validation method to check to see if the user's inputs are actually valid and will not crash the program.

Testing:

Test cases for binary options (1 or 2)

Test Case	Input Values	Functions	Expected	Observed
Input too low	Input < 1	Zoo.setup() Zoo.endOfDay() Zoo.playAgain() inputValidation()	re-prompt for new entry	re-prompt for new entry
Input in correct range	1 <= Input <= 2	^	inputValidation returns input back to variable	inputValidation returns input back to variable
Input extreme low	Input = 1	^	inputValidation returns input back to variable	inputValidation returns input back to variable
Input extreme high	Input = 2	^	inputValidation returns input back to variable	inputValidation returns input back to variable
Input too high	Input > 2	^	Re-prompt for new entry	Re-prompt for new entry
Input is incorrect type	Input = String	^	Re-prompt for new entry	Re-prompt for new entry

Beginning number of animals = [1 Tiger, 1 Penguin, 1 Turtle]

Starting Funds: $100,000
Random event: No event occurred
Profit: $2105 [Tiger(2000) + Penguin(100) + Turtle(5)]
Funds: $90,940 [100,000 – 1000 – 1000 – 100 - 50 – 10 – 5 + 2105]

Purchase an Adult Tiger:
Play Again:
Random Event: Tigers result in bonus of $842
Profit: $4947
Funds: $85823

Purchase an Adult Penguin:
Play Again:
Random event: 10 turtles have been born
Profit: $4255
Funds: $89013
Quit:

Beginning number of animals = [2 Tigers, 2 Penguins, 2 Turtles]

Starting funds: 100,000
Event: Bonus of 792
Profit: 5002
Funds: 82,672

Purchase an Adult Tiger:
Play Again:
Event: penguin dies
Profit: 6110
Funds: 78652

Purchase an Adult Tiger:
Play Again:
Event: Bonus of 1612 (4 tigers)
Profit: 9722
Funds: 78204

Purchase an Adult Penguin:
Play Again:
Event: 10 turtles born
Profit: 8260
Funds: 85284

Do not purchase an animal:
Quit:

Reflection: The most difficult part of this assignment was to figure out how all of the necessary classes were going to work together to make the simulation work. I initially designed the Animal class with a constructor, but realized that it would only be used as an abstract class to be inherited by the other, more specific animal classes. So the only functions I wrote in for the Animal.cpp class were basic setter and getter functions to be used by the Zoo class. Inside the Tiger, Penguin, and Turtle classes I created the constructors and a couple helper methods that would aid the constructor create the animal object according to the specifications. Once I finished those classes, the first issue I ran into while creating the Zoo.cpp file was how to keep track of the numbers of each animal that was currently in the Zoo. I initially had the idea of iterating through each array to find the number of current animals of that kind, but I found that to be too expensive in terms of time and resources. I decided to simply keep track of the number of each animal using a variable declared in the header files. By doing so, I needed to be diligent about making sure to increment the number correctly when the user is adding more of that specific animal, and decrementing the number correctly when an animal died. This specifically made it easier when I had to check to see if the current dynamic array was big enough to hold the desired number of animals. This also meant that I needed to keep track of the array size as a variable as well. To check, I just needed to see if the ending number of animals was larger than the current array size – if it was, then I created a new array that had double the capacity of the original and copied every element in the old array into the new one. I would then delete the old array to prevent memory leaks. Looking back, it may have helped to build this into a separate helper function, since I needed to use it at multiple instances throughout the program (basically whenever the user added more animals).

I also had trouble calculating the daily profit, since I initially had profit calculated over time instead of for each day. I solved this by simply resetting the daily profit to zero at the end of each day, but making sure that I added the daily profit to the total funds before doing so. The last thing that tripped me up was figuring out how to delete the dynamically allocated arrays once the user decided not to play anymore, or they ran out of money. I realized I couldn't delete them from the main program since that wasn't where they were created, so I decided to create a separate function with Zoo.cpp that was used specifically to delete the arrays in order to prevent memory leaks. This function was called from main right before the program terminates.

My greatest takeaway from this assignment was how to efficiently create classes so that shared functions could be reused by different objects. If we had not used inheritance for this, it would have been tedious to write the same functions for each class. That's not to say that it would have been difficult, but knowing how effective inheritance can be when used correctly has shown me how efficiency can exist in programming classes and not just in data structures.