Justin Tromp
CS 162
Project 2 – Zoo Tycoon
01/21/2019

**Overview:** I will be developing a zoo simulation game using classes and inheritance for the various class types. Different animals will cost different prices throughout the game, cost varying amounts for maintenance, and return a different profit at the end of every day of the simulation. The animals available will be tigers, penguins, and turtules.

**Board/Visual Units**
There will be no visual units/board for this game.

**Game Start Values**
Bank: Players will start with 100,000 dollars in the bank.
Animals: Player must start game by buying at least 3 animals (one of each type). All newly purchased animals start at 1 day old.

**Turns**
Each turn is a single day.
At the start of a turn or day, all animals increase in age by 1 and the user pays the feeding/maintenance costs for all animals in the zoo.
Once feeding costs are subtracted, a random event takes place through a randomInt function that utilizes the Mersenne twister engine random number generation.

Each day, a random event must occur:
1 – One animal at random dies using dynamic array
2 – A boom in zoo attendance for the day (bonus 250-500 dollars for each tiger in the zoo for that day)
3 – A baby animal is born! Random animal has a baby. There only needs to be one animal in order for a baby to be born. Animal giving birth must be an adult, so >= 3 days old. If no animals are old enough, count this as nothing happening. Babies added are according to the animal class. Tigers have 1, penguins have 5, turtles have 10 babies.
4 – Nothing happens.

I will be utilizing a food type selection that will change probabilities of sickness based on how much the user is paying for food. I intend to do this by making sickness more or less probable by increasing the numbers that can be randomly generated. The other 3 random events will be more probable to occur due to a larger number selection or less probable by the same methodology. For example (for cheap food, numbers generated would be 1-10, generic would be 1-4, expensive would be 1-7 to give a clear example of the method):
Sickness occurs if 1, 8, 9, 10
Boom occurs if 2, 5
Birth occurs if 3, 6

No event occurs if 4, 7

After the random event, calculate the profit for the day. If there is a bonus pay event, add the bonus pay to the profit. Before the day ends, ask if the player wants to buy an adult animal. If they do, ask for the type of animal they would like and then add the animal to the zoo. Subtract the cost of that animal from the bank. The adult animal purchased will be 3 days old.
At the end of the day, prompt user whether to keep playing or end the game. If user has no money, print message informing the user that the game is over and end the game. I will additionally be adding the functionality of a loop that will re-prompt users for animal purchases if they go over budget as a sort of safe guard. It is not mentioned in the game instructions, however it makes sense that you cannot buy what you don't have money for.

**Program Outline**
**Main:** main.cpp
**Contains:** Menu selection and simulation setup
**Variables:** zoo object and integer menu selection.

- **Functions**: No functions based in main, only called.
- Uses startMenu function from zoo object to get menu selection input from user. Depending on input, determine which direction the program goes. Menu will have two selections, 1 to start the simulation and 2 to exit the simulation.
- If user selects to start simulation, startMenu will return integer value of 1.
- If menu selection is set to 1, call zoo setup function from zoo object.
- Loop while the menu selection is 1. Inside loop, run zoo play function from zoo object and at end of loop call endMenu function to allow user to continue playing or to quit at the end of their turn.

**Classes**
**Zoo:** zoo.cpp zoo.hpp
**Class Contains:** Dynamic array for each type of animal. Each Dynamic array should have a capacity of 10 spaces for animals to start and has the option to resize if more animals are added. If the array is resized, it should be doubled. Arrays will be formatted as animaltype **animaltypeArray = new animaltype *[10].
**Variables:**

- 4 arrays, 1 for each animal type
- Double variables for profit, zoo balance, revenue, expenditures, and revenue boom events.
- Int variable for each array for array size tracking.
- Int variable for day number tracking for user output.
- Int variable for each animal type for number of animals purchased.
- Int variable for each animal type to track total number of animals in zoo (how many animal objects are in each array).
- Functions:

- Zoo Default Constructor – Set zoo balance to 100,000 as a default for any objects created. Can easily be changed later if needed.
- Zoo Destructor – Deallocate memory allocated at end of object.
- Zoo Setup – Set up the zoo by prompting user and welcoming user to the simulation. States cost of animals and prompts user for initial animal purchases (1 or 2 of each type to start). Setup also calls user animal setup function to set up user animal class type based on user input. At end, zoo setup should call initialize array function. No return value.
- initializeArray – Initialize arrays by setting all values to nullptr at start. No return value.
- increaseArray – Increases array size by 10 by copying array to a new array variable and reassigning. Delete unused memory locations to prevent leaks. Adjust array size variable for animal type. No return values.
- decreaseArray – Delete animal and reduce counter for total number of animals for that type. No return value.
- removeAnimal – Remove animal from array. If the number of animals is equal to zero, return false for no removal of animal type. If one is in the zoo, remove by decreaseArray and return true.
- addAnimal – Add animal based on parameters received (age and number added, both integers). Increase array size if necessary by increaseArray function. Increase total animal counter for respective animal after adding turtle object. No return value.
- randomInt function – Generates random integer based on parameters for min and max values of output. Returns generated value as integer.
- Random Event function – Receive random integer value from randomInt function and select random event from possible choices. Depending on food type, probability changes for sickness event. No return value.
- EVENT FUNCTIONS – Triggered by random event function
  - sicknessEvent – Uses randomInt to return integer value for selection of animal to succumb to sickness and die. Use loop to enter series of if statements. If there are no animals of the type selected, move to next animal type and so on. Output message to file and read message to screen from file by functions. No return value.
  - revenueBoom – Calculate revenue boom based on number of tigers present. Revenue boom is based on random integer output from randomInt between 250 and 500 and is multiplied by number of tigers. If there are no tigers, no profit from boom. Output respective message to file and read from file from functions. No return value.
  - birthEvent – Animal gives birth. Use randomInt to select an animal type to give birth. If no animals are old enough, loop through other animal types to find one that is. If none are old enough, no animals give birth. Output message to file for birth and read to screen from input file that functions printed to. If none are old enough, print message saying none were old enough.

- setupUserAnimal function – Sets up the user selected animal by requesting values from the user, validating them as needed and storing them for later use in the user animal constructor. No return value.
- increaseAge – Increase age of all animals. Cycles through all animal arrays and increases age through class function in animal class. No return value.
- dayTurn Function – Set all changing class variable values to zero at start of day such as profit, boom, revenue, costs, and food type. Add one to number of days to keep track of the day number for user output. No output values. Day events called as previously mentioned above:
    - feedType – Request food type from user at start of day.
    - increaseAge – Increase age of all animals in zoo.
    - animalCost – Pay for costs of all animals for feeding.
    - randomEvent – Call random event
    - Calculate revenues
    - Calculate profit
    - Calculate zoo balance
- Added payMaintenance function - payMaintenance – Calculates expenditures for animal feeding in zoo. If food type is not generic, adjust costs accordingly. Use series of loops to loop through arrays based on number of animals in each type. Get cost of food from each animal object. No return value.
- zooPlay – Controls most of the game functionality. Function is called by zoo object in main. Starts by calling function to perform all aspects of day such as profit calculations. Print out zoo status to user. Check for bankruptcy, if there is no money, then notify user and exit game. If no bankruptcy, enter loop and request user for purchase amount of each animal type. If there is a purchase of any animals, then add animals 3 days old to respective arrays. Return false if user is not bankrupt, return true if user is bankrupt.
- startMenu and endMenu – Similar menu functionality with slightly different outputs to indicate game has not started versus has started already. Returns int value based on user selection and uses input validation.
- Turtle, penguin, tiger, and user animal birth functions – If called, adds respective animal at 0 day old parameter for the number of babies based on birth rate of animal type. No return value.
- feedType – Asks user for food type selection after outputting options to user. Validate input and store in food type variable. No return value.
- Input event message function – Reads message printed to file in output event message function and prints message to screen for user to read without opening file. If input file cannot be opened, inform user of error. No return value is generated.
- Output event message function – Prints message to file based on the event that occurred. Parameters accepted are int value for message type and string value for animal type. If output file cannot be opened, inform user of error. No return value is generated.

**Animal:** animal.cpp animal.hpp
**PARENT CLASS**
**Class Contains**:

- Member Variables:
    - Int animalAge – Holds age of animal
    - Int animalCost – Holds cost of animal
    - Int birthRate – Holds number of babies born for each birth event for animal (Changed to numberBabies)
    - Double foodCost – cost of food for animal object
    - Double animalRevenue – Revenue from animal object
- Functions:
    - Animal constructor – Sets all values to 0 for all variables in class.
    - getNumberBabies – Returns int value of birth rate
    - getAnimalRevenue – Returns double value for revenue for animal object
    - getFoodCost – Returns double value for food cost
    - getAnimalAge – Returns age of animal
    - increaseAge (changed to increaseAnimalAge) – Increases age of animal object

**Tiger Class:** tiger.hpp tiger.cpp
**CHILD CLASS**
**Class Contains:** Inherits functions and variables from animal class

- Functions:
    - Tiger default constructor – Inherits animal class variables, which sets all values to 0 from animal class.
    - Tiger constructor – Takes integer age input as parameter and sets that to animalAge. Animal cost is set to 10000 for tiger, foodCost is set to baseFoodCost in animal class multiplied by 5, revenue is set to .2 multiplied by cost of animal, and number of babies is set to 1.

**Penguin Class:** penguin.hpp penguin.cpp
**CHILD CLASS**
**Class Contains:** Inherits functions and variables from animal class

- Functions:
    - Penguin default constructor – Inherits animal class variables, which sets all values to 0 from animal class.
    - Penguin constructor – Takes integer age input as parameter and sets that to animalAge. Animal cost is set to 1000 for penguin, foodCost is set to baseFoodCost in animal class (no multiplier), revenue is set to .1 multiplied by cost of animal, and number of babies is set to 5.

**Turtle Class:** turtle.hpp turtle.cpp
**CHILD CLASS**
**Class Contains:** Inherits functions and variables from animal class

- Functions:

  - Turtle default constructor – Inherits animal class variables, which sets all values to 0 from animal class.
  - Turtle constructor – Takes integer age input as parameter and sets that to animalAge. Animal cost is set to 100 for turtle, foodCost is set to baseFoodCost in animal class multiplied by .5, revenue is set to .05 multiplied by cost of animal, and number of babies is set to 10.

**User Animal Class:** userAnimal.hpp userAnimal.cpp
**CHILD CLASS**
**Class Contains:** Inherits functions and variables from animal class.

- Member Variables:
  - Adds string variable for animal type – which will be filled by name of animal from user input.
- Functions:
  - User animal default constructor – Inherits animal class variables, which sets all values to 0 from animal class.
  - Turtle constructor – Takes integer age input as parameter and sets that to animalAge. Additional parameters taken are string value for user input animal name, int value for cost of animal, int value for birth rate, int value for animal revenue, and double value for foodCostMultiplier. Animal cost is set to user input for user animal, foodCost is set to baseFoodCost from animal class multiplied by multiplier input by user, string name input is set for animal type variable for user animal class, revenue is set to user input revenue, and birth rate is set to user input value of number of babies per birth event.
  - Set animal type – set the type of animal received as parameter (string value) to the animalType variable. No return value.
  - Set animal age – set the age of animal taken by parameter as integer value as animal age. No return value.
  - Set animal revenue – take integer value for animal revenue and set to animalRevenue variable. No return value.
  - Set animal cost – take integer value as parameter for cost of animal and set to animalCost variable. No return value.
  - Set food cost – Take double value as parameter for food cost multiplier and set foodCost variable equal to multiplier multiplied by the baseFoodCost value from animal class. No return value.
  - Set birth rate – Take integer value of birth rate as parameter and set to number of babies per birth event variable (numberBabies) in animal class. No return value.

**Input Validation Functions:** inputValidation.cpp inputValidation.hpp
**Contains**: Functions to validate values passed by the user to the program throughout the various steps.  All functions will validate the user input. If the user input does not meet the requirements, the function will re-prompt the user for input. When the user input does meet the requirements of input, the validated value will be returned.

**Functions**: I have not determined the exact functions to use for validation, but I know there will be one to validate a menu choice and one to validate board/simulation step entry. I will update in the reflection as needed.

**Random Number Generator Function:** randomNumGen.cpp randomNumGen.hpp (***CHANGED to randomInt function in zoo class***)
**Contains**: Takes in a minimum and maximum value, which will be based on the board size selected by the user. A value will be generated between these two values and returned.

**Testing Plan/Execution – NOTE: String input is not validated as user is allowed to select any type of animal. Could have input validation for no digits or something of that spectrum, however this was not a requirement for the assignment.**

| Test Case | Input Value | Driver Functions | Expected Outcomes | Observed Outcome Initial |
|---|---|---|---|---|
| Start Menu | 0 | validateInputMenu – Executed by startMenu() from main() | Re-prompt/err | Re-prompt/err |
| | 3 | | Re-prompt/err | Re-prompt/err |
| | 12 | | Re-prompt/err | Re-prompt/err |
| | 21 | | Re-prompt/err | Re-prompt/err |
| | Enter | | Re-prompt/err | Re-prompt/err |
| | Space | | Re-prompt/err | Re-prompt/err |
| | 1a | | Re-prompt/err | Re-prompt/err |
| | a1 | | Re-prompt/err | Re-prompt/err |
| | 1.5 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 1 | | Start Simulation | Start Simulation |
| | 2 | | Quit Program | Quit Program |
| End Menu | 0 | validateInputMenu – Executed by endMenu() from main() | Re-prompt/err | Re-prompt/err |
| | 3 | | Re-prompt/err | Re-prompt/err |

| | | | | |
|---|---|---|---|---|
| | 12 | | Re-prompt/err | Re-prompt/err |
| | 21 | | Re-prompt/err | Re-prompt/err |
| | Enter | | Re-prompt/err | Re-prompt/err |
| | Space | | Re-prompt/err | Re-prompt/err |
| | 1a | | Re-prompt/err | Re-prompt/err |
| | a1 | | Re-prompt/err | Re-prompt/err |
| | 1.5 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 1 | | Keep Running Simulation | Keep Running Simulation |
| | 2 | | Quit Program | Quit Program |
| User Animal – Cost Multiplier | 101 | validateInputDoubleRangePt1_100 – Executed by setupUserAnimal() from zooSetup(). | Re-prompt/err | Re-prompt/err |
| .1-100 | .09 | | Re-prompt/err | Re-prompt/err |
| | 1a | | Re-prompt/err | Re-prompt/err |
| | a1 | | Re-prompt/err | Re-prompt/err |
| | .1 a | | Accept input | Accept input |
| | a .1 | | Re-prompt/err | Re-prompt/err |
| | a 1 | | Re-prompt/err | Re-prompt/err |
| | 1 a | | Re-prompt/err | Re-prompt/err |
| | 10 1 | | Re-prompt/err | Re-prompt/err |

| | | | | |
|---|---|---|---|---|
| | .1 | | Set cost multiplier to .1 | Set cost multiplier to .1 |
| | 100 | | Set cost multiplier to 100 | Set cost multiplier to 100 |
| Animal Cost | 0 | validateInputIntRange1_35000 – Executed by setupUserAnimal() from zooSetup(). | Re-prompt/err | Re-prompt/err |
| 1-35000 | 35001 | | Re-prompt/err | Re-prompt/err |
| | 1 .1 | | Re-prompt/err | Re-prompt/err |
| | Enter | | Re-prompt/err | Re-prompt/err |
| | Space | | Re-prompt/err | Re-prompt/err |
| | 10a | | Re-prompt/err | Re-prompt/err |
| | a10 | | Re-prompt/err | Re-prompt/err |
| | 10.5 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 1 | | Set to 1 | Set to 1 |
| | 35000 | | Set to 35000 | Set to 35000 |
| Birth Rate User Animal – 1 - 100 | 0 | validateInputIntRange1_100 – Executed by setupUserAnimal() from zooSetup(). | Re-prompt/err | Re-prompt/err |
| | 101 | | Re-prompt/err | Re-prompt/err |
| | 100 1 | | Re-prompt/err | Re-prompt/err |
| | Enter | | Re-prompt/err | Re-prompt/err |
| | Space | | Re-prompt/err | Re-prompt/err |
| | 10a | | Re-prompt/err | Re-prompt/err |

| | | | | |
|---|---|---|---|---|
| | a10 | | Re-prompt/err | Re-prompt/err |
| | 10.5 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 1 | | Set birth rate to 1 | Set birth rate to 1 |
| | 100 | | Set birth rate to 100 | Set birth rate to 100 |
| Revenue User Animal – 1 through 30000 | Enter | validateInputIntRange1_30000 – Executed by setupUserAnimal() from zooSetup(). | Re-prompt/err | Re-prompt/err |
| | Space | | Re-prompt/err | Re-prompt/err |
| | 0 | | Re-prompt/err | Re-prompt/err |
| | 1.1 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 30001 | | Re-prompt/err | Re-prompt/err |
| | a1 | | Re-prompt/err | Re-prompt/err |
| | 1a | | Set Player to regular die | Set Player to regular die |
| | 1 | | Set to 1 | Set to 1 |
| | 30000 | | Set to 30000 | Set to 30000 |
| Initial Animal Purchase | Enter | validateInputIntRange1_2 – Executed by zooSetup() from main(). | Re-prompt/err | Re-prompt/err |
| 1 or 2 Only | Space | | Re-prompt/err | Re-prompt/err |
| | .1 | | Re-prompt/err | Re-prompt/err |
| | 1.5 | | Re-prompt/err | Re-prompt/err |

| | | | | |
|---|---|---|---|---|
| | 1 2 | | Re-prompt/err | Re-prompt/err |
| | 2 1 | | Re-prompt/err | Re-prompt/err |
| | 11 | | Re-prompt/err | Re-prompt/err |
| | 21 | | Re-prompt/err | Re-prompt/err |
| | 1.1 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 1 | | Set to 1 | Set to 1 |
| | 2 | | Set to 2 | Set to 2 |
| Food Quality Selection | Enter | validateInputIntRange1_3 – Executed by dayTurn() from zooPlay(). | Re-prompt/err | Re-prompt/err |
| 1 Through 3 Only | Space | | Re-prompt/err | Re-prompt/err |
| | .1 | | Re-prompt/err | Re-prompt/err |
| | 1.5 | | Re-prompt/err | Re-prompt/err |
| | 1 2 | | Re-prompt/err | Re-prompt/err |
| | 2 1 | | Re-prompt/err | Re-prompt/err |
| | 11 | | Re-prompt/err | Re-prompt/err |
| | 21 | | Re-prompt/err | Re-prompt/err |
| | 1.1 | | Re-prompt/err | Re-prompt/err |
| | 1 1 | | Re-prompt/err | Re-prompt/err |
| | 4 | | Re-prompt/err | Re-prompt/err |
| | 0 | | Re-prompt/err | Re-prompt/err |
| | 1 | | Set to 1 | Set to 1 |
| | 3 | | Set to 3 | Set to 3 |
| Animal Purchase | Enter | validateInputIntRange0_1 – Executed by zooPlay() from main(). | Re-prompt/err | Re-prompt/err |

|  |  |  |  |  |
|---|---|---|---|---|
| Validation (Not Initial) |  |  |  |  |
| 0 Through 1 Only | Space |  | Re-prompt/err | Re-prompt/err |
|  | .1 |  | Re-prompt/err | Re-prompt/err |
|  | 1.5 |  | Re-prompt/err | Re-prompt/err |
|  | -1 |  | Re-prompt/err | Re-prompt/err |
|  | 2 1 |  | Re-prompt/err | Re-prompt/err |
|  | 11 |  | Re-prompt/err | Re-prompt/err |
|  | 21 |  | Re-prompt/err | Re-prompt/err |
|  | 1.1 |  | Re-prompt/err | Re-prompt/err |
|  | 1 1 |  | Re-prompt/err | Re-prompt/err |
|  | 2 |  | Re-prompt/err | Re-prompt/err |
|  | 0 |  | Set to 0 | Set to 0 |
|  | 1 |  | Set to 1 | Set to 1 |

**RANDOM NUMBER GENERATOR TEST - CHECK**
To test random number generator, I will create a loop within the program to output all possible values generated and print to screen. Numbers generated (from 0 through 9).

**<u>Numbers Generated</u>**
7 4 8 0 3 0 2 7 7 0 1 2 7 5 4 7 2 2 6 3 7 3 5 7 8 2 4 2 7 8 1 4 1 0 4 0 7 9 3 3
Min Generated: 0
Max generated: 9
Thus, the test passed as the values can be properly generated between the minimum and maximum range given.

**Game Tests – Pricing/Profits/Numbers (FIRST PURCHASE)**
**BASE FOOD COST IS 80 AND ASSUMING 1 OF EACH ANIMAL – LATER CHANGED TO 10 BASE FOOD COST DUE TO TURTLES BEING SO LOW. EVEN 10 FROM ASSIGNMENT ONLY RESULTS IN BREAK EVEN. MAKES IT HARD TO LOSE, BUT IT IS WHAT IS GIVEN IN THE ASSIGNMENT - CHECK**

| Animal | Expected Price | Actual Price | Expected Maintenance | Actual Maintenance | Zoo Balance |
|---|---|---|---|---|---|
| Turtle | 100 | 100 | 40 | 40 | |
| Penguin | 1000 | 1000 | 80 | 80 | |
| Tiger | 10000 | 10000 | 400 | 400 | |
| userChoice | 500 Input | 500 | 160 | 160 | |
| TOTAL | | | | Before Profits/Costs | 88400 |

Array Sizes: All 10 at start. **CHECK**
Initial Ages: All animals started at 1 day old when purchased at start. **CHECK**
**THE ABOVE TESTS ARE RIGHT AFTER PURCHASE. AFTER WHICH, THE FIRST DAY BEGINS I.E. zooPlay() BEGINS, STARTING WITH dayTurn().**

**DAY END PURCHASE**
**BASE FOOD COST IS 80 AND ASSUMING 1 OF EACH ANIMAL - CHECK**

| Animal | Expected Price | Actual Price | Expected Maintenance | Actual Maintenance | Zoo Balance |
|---|---|---|---|---|---|
| Turtle | 100 | 100 | 40 | 40 | |
| Penguin | 1000 | 1000 | 80 | 80 | |
| Tiger | 10000 | 10000 | 400 | 400 | |
| userChoice | 500 Input | 500 | 160 | 160 | |
| TOTAL ZOO BALANCE | | | | | 90325 |

Ages After First Day: Ages increased by 1 in all animal types. **CHECK**

**PROFIT/REVENUE/EXPENDITURES/BUDGET (1 OF EACH ANIMAL) - CHECK**

| | Expected Revenue / Actual | Expected Expenditures / Actual | Expected Profit / Actual | Expected Budget / Actual |
|---|---|---|---|---|
| Day 0 | 0 / 0 | 0 / 0 | 0 / 0 | 88400 / 88400 |
| Day 1 | 2605 / 2605 | 680 / 680 | 1925 / 1925 | 90325 / 90325 |
| Day 2 – lost 1 penguin | 2505 / 2505 | 680 / 680 | 1825 / 1825 | 92150 / 92150 |
| Day 3 | 2505 / 2505 | 600 / 600 | 1905 / 1905 | 94055 / 94055 |

**NOTE:** I ended up converting all integer values to double type values regarding revenue as user input for user animal would cause rounding issues.

**ANIMAL BIRTHS – ALL EVENTS WORKED PROPERLY, CHECK**

| Animal | Expected Birth | 1st Event | 2nd Event | 3rd Event |
|---|---|---|---|---|
| Turtle | 10 | 10 | 10 | 10 |
| Penguin | 5 | 5 | 5 | 5 |
| Tiger | 1 | 1 | 1 | 1 |
| userChoice | 5 Input | 5 | 5 | 5 |

**EVENTS CHECK**

Penguin died – Penguin was removed and wrote to output file and read from input file to screen.

Penguin died – Penguin was removed and wrote to output file and read from input file to screen.

Tiger gave birth – Tiger was added and write to ouput file and read from input file to screen.

Quiet day at the zoo – Nothing occurred and wrote to output file and read from input file to screen.

Quiet day at the zoo – Nothing occurred and wrote to output file and read from input file to screen.

Today is National Tiger Day! Tigers generate extra money today! You made: 379 extra dollars for each tiger you own! – 1516 added to profit (correct amount for 4 tigers) and printed to output file and read from input file to screen.

One of the tigers gave birth today at the zoo! – Tiger was added and properly printed and received from .txt file.

Today is National Tiger Day! Tigers generate extra money today! You made: 302 extra dollars for each tiger you own! – Random generation of bonus worked properly and properly read/output from file.

There was a sickness at the zoo and one of your animals did not recover! One tiger has died. – Properly output and input from file. Animal was removed.

There was a sickness at the zoo and one of your animals did not recover! One koala has died. A koala has been removed from the zoo. – Worked properly with input/output and removed koala.

**ALL EVENTS OCCURRED AND WORKED PROPERLY, CHECK**

**MEMORY LEAK CHECK**

| # Animals | 0 | 9 | 10 | 11 | 20 | 21 (AND 31) |
|---|---|---|---|---|---|---|
| Turtle | None | None | None | None | None | None |
| Penguin | None | None | None | None | None | None |
| Tiger | None | None | None | None | None | None |
| UserAnimal | None | None | None | None | None | None |

Select quit at start menu – 4 memory leaks were found. Fixed by moving delete functions from testing location in code to destructor. **CHECK**

**ANIMAL PURCHASE CHECK – Prior to changing purchase limits to 1 animal per turn**

| # Animals | Expected | Observed | Expected | Observed |
|---|---|---|---|---|
| Turtle | 0 | 0 | 25 | 25 |
| Penguin | 0 | 0 | 25 | 25 |
| Tiger | 0 | 0 | 25 | 25 |
| UserAnimal | 0 | 0 | 25 | 25 |

**ANIMAL PURCHASE CHECK – After changing purchase limits to 1 animal per turn**

| # Animals | Expected | Observed | Expected | Observed |
|---|---|---|---|---|
| Turtle | 0 | 0 | 1 | 1 |
| Penguin | 0 | 0 | 1 | 1 |
| Tiger | 0 | 0 | 1 | 1 |
| UserAnimal | 0 | 0 | 1 | 1 |

Purchase only allowed one total animal to be purchased per turn as required. If an animal was selected for purchase, the remaining animal options were not presented to the user. **CHECK**

**TEST OVERPURCHASE OF ANIMALS OUTSIDE OF BUDGET**
Purchase input was rejected if over budget and looped until user selected a value within budget. **CHECK**

**TEST BANKRUPTCY**
Bankruptcy was tested and once zoo budget went below 0 dollars at the end of the day (prior to selecting purchases for the next day) the game ended with a notification as intended. **CHECK**

**TEST SICKNESS UNTIL FAIL (0 ANIMALS OF ALL TYPES FROM DEATH)**
No errors or memory leaks! **CHECK**

**REFLECTION:**
**DAY TURNS/OPERATIONAL VALUES EXPLAINED**
As per instructor suggestion I am creating this section to explain the actual day turns and how things are calculated.

Day 0: This is the day the zoo is set up and the first animals are purchased. No other events occur this day as there are no visitors and the animals are assumed to be fed prior to purchase. Therefore, profit reflects just the cost of the animals for that day.

Day 1+:
- User selects feed type to start day

- Ages of all animals in zoo are increased (for day 1, this includes the animals purchased on day 0)
- Pay food costs for all animals in zoo currently
- Random event occurs
- Revenue/Payoff is calculated for all animals currently in zoo (this includes baby animals if this random event occurred, this is because baby animals attract a lot of visitors, so they should increase revenue for that day). If an animal dies, they do not create revenue, but they do cost money to feed that day. It is figured that if they are sick, you feed them and do not display them to public.
- Zoo profit is updated (revenue from animals minus food costs)
- Zoo profit is updated again (if there was a revenue boom, add to profit)
- Once day essentials are done, user is allowed to purchase one animal or no animals as per guidelines for assignment. If user selects to buy an animal, they will not be displayed the other animal options after purchase since they reached their limit.
- Add the animal user selected to zoo (if they selected one) and then adjust zoo profit for day by subtracting cost of that animal.
- Update zoo balance for day by adding the zoo profit for the day (if zoo balance is less than 0, then the user receives a message that they went bankrupt and the game ends and displays ending stats). If bankruptcy occurs, the ages of animals are not printed as it is unnecessary information.
- If user did not go bankrupt, continue on. Print status of zoo for the day and print the ages of all animals in the zoo.
- Allow user to select to quit or continue by menu.

Tiger revenue boom events – Revenue boom is randomly generated and applied to all tigers in zoo as per assignment example. Some forum talk brought up using random values for each tiger, however I chose to follow the single value example on the assignment. This is based on the idea that in large quantities, randomly assigning values to every tiger vs all tigers each boom event will essentially be the same over time. Plus, allows for better output/user experiences and less drain on resources.

**CHANGES**
Adjusted user input animal ranges for cost and payoff values. Since I did not want the user to be able to go into a bankruptcy situation at start, I reduced these values to make it impossible to bankrupt at the start of the simulation. You must wait until after day 0 to go bankrupt. Therefore, I chose 35000 as the maximum cost of a user selected animal and 30000 as the maximum revenue for a user selected animal. I also felt these were more reasonable values from what I had previously.
Added userBankrupt boolean variable to main – If user goes bankrupt, Boolean variable is changed to true and causes exit from loop to end game.
Added revBoomTF Boolean variable – If set to true, variable initiates a message to zoo status print to screen message that shows funds earned from boom.

Added cons tint animalTypes = 4 – Allowed for more streamlined program, since I could use constant variable instead of typing 4 in loops and if statements. That way if another animal class was added, it would be an easier addition.
Added addAnimals to initializeArrays function – Added initial animal add to function for animals selected during zooSetup.
Added zooAnimalRevenue function – By adding revenue calculation function I was able to shorten the dayTurn function and make it more modular.
Added printZooStatus function – By adding printZooStatus function, I could print the status of the zoo anywhere without copy and pasting it multiple times or squeezing it into a function.
decreaseArray function – I chose to separate into 4 different functions with no return value. Each function handled removal of animal object from respective array and reduced the animal type counter for number of animals.
addAnimal function – Separated into four functions, one for each animal type with same functionality. Also set animaltypePurchase to 0 at end.
removeAnimal function – Separated into four functions, one for each animal type with same functionality.
randomEvent function – Added no event portion directly to randomEvent function.
sicknessEvent function – Added message and if statement for possibility that no animals are in the zoo and the zoo still has money. Thus, print message with this in mind.

**WOULD HAVE CHANGED**
increaseArray function – Could have separated into multiple functions to have a smaller function layout based on animal type or if I was able to adjust the array style to be able to combine multiple animal functions into one single function without if statements to separate by animal type.

All in all, I would have changed the array structure if I was to make this program again or modify it in any way. By changing the array types to a more polymorphic structure, I could have made four triple pointer animal class arrays, where each would hold one animal type. This way I could have shortened much of the coding by creating more generalized functions instead of 4 separate functions or 4 separate if statements within a function to perform the same task for each different array type variable. Some of the problems I encountered in this project were related to memory leak errors when increasing the size of the array and tracking issues related to moving out of bounds on arrays when the number of animals reached 0. The memory leak errors were fixed through the error checking in flip with my makefile calls. It turned out that it was not necessarily a leak, but rather I was trying to free more memory allocations than were reserved. The error checking/leak results allowed me to narrow down the function of allocation and I was able to see where I was incorrectly implementing delete and removing more than I actually dynamically allocated through new. I dealt with the array out of bounds issues by modifying the code causing it. I was trying to create a failsafe loop when users buy animals at the end of their turn that re-prompted for purchase if they spent more money than they had. This was not working however if there were no animals of the type they were trying to purchase. This is because the program was attempting to find the cost of that object when there was nothing there. I fixed this by just adding the prices manually in the failsafe, which works for this scenario.

Towards the end of development, I decided to remove the loop check to ensure the user does not go over budget from purchases. I misread the requirement that users can only purchase one animal at the end of each turn total, so having a budget check seemed unnecessary since we already check for bankruptcy in the zoo after purchase. I added an additional test table for the new values to double check animal purchases after the changes. I also changed many of the integer variables for profit, cost, and such to double values. This is mostly due to the extra credit user class, as the multiplier for food cost could be any fractional value from .1 to 100 (in the case of say an elephant that may eat a lot). Since a user may select values such as .133, I needed to account for this by allowing decimal values in the budget related variables. This made sense anyways seeing as finances are usually specified in double type values rather than straight integers.

In addition, I added a print age to screen function to the zoo class. This function acted to print all the ages in days of the animals in each array so that the user (and grader/myself) could see the animals ages as the days progressed. I also formatted the output columns for easier viewing, which created a lot of trial and error issues as I had not done that sort of output formatting as of yet.

**For the future:**
One of the largest problems that I encountered in this assignment was figuring out how to manage everything occurring in the program and what was specifically required for the assignment. In the future, I would like to take steps in my design to reduce the amount of coding needed to perform each function. I found throughout my zoo class that I would create repetitive functions or functions that would require if statements to really separate out the various animal type arrays. A big part of this was due to not incorporating a polymorphic structure. In lab 3, I was able to create a single variable that could use one of two classes based on user selection without actually making a separate player 1 variable for each class. This is a feature that may have been greatly beneficial in this program and could have reduced the lines of code and the structure by half, if not more. By reducing the overall structure, I could have spent less time writing code and more time focusing on debugging and creating a fully functional design. In fact, less time would have been needed for debugging since there would be a significant reduction in necessary if statements and loop structures to keep track of. It is my belief that the program works well and is fully functional while meeting the assignment needs, but it could use some optimization and I am glad I experienced these issues in this assignment for use in future assignments in this class and in my career.