Program: Zoo Tycoon
Description: Player owns a virtual zoo that can house tigers, penguins and turtles. Each day the play can earn money and must feed animals. Random events occur like birth, sickness and fluctuating zoo attendance.

Design Notes:

Zoo (game flow)

- Member variables
  - Zoo
    - Array of Animals
    - Dynamic array for each type of animal
    - Start with 10 animal capacity
    - Double if that is filled
  - Daily bonus
    - int
- Member functions
  - Start Menu
  - Day
  - Add age
  - Costs
  - Random event
  - Profits
  - Buy new animal
  - Play again menu
- Flow
  - Start
    - Player begins with $50K in the bank
    - Must buy 3 types of animals to start (1 of each)
      - Each has quantity of 1 or 2
    - Cost of each animal is subtracted from total
    - Each animal starts at 1 day old
  - Each Day
    - All animals age +1 day
    - User pays cost of feeding each animal
      - Animals die if not fed
    - After feeding, random event happens
      - A sickness occurs to an animal
        - Pick random animal
        - Remove animal from dynamic zoo array
      - Boom in attendance
        - Generate random bonus between 250 - 500 for each tiger in the zoo
        - Add bonus to pay off for each tiger for the day
      - A baby is born
        - Pick random animal to have 1 baby
        - Add baby to chosen animal if it is an adult
        - If no animal of the type is old enough, move on to different type
        - Babies start at age = 0
        - If no animals can give birth, then go to default
      - Nothing happens
        - Acts as default if baby can't be born
    - Calculate profits for the day
      - Use payoff % + any bonus
    - Ask if player wants to buy a new adult animal
      - Yes
        - Ask for animal type
        - Add animal to zoo
        - Subtract cost from bank
        - Animal age = 3
      - No
        - Keep going
    - Ask if user wants to keep playing
      - Yes
        - If user has no $, say so and end game
        - Otherwise, go to next day
      - No
- Notes:
  -

Animal

- Member Variables (no more than this!, keep names)
  - Int Age
    - Adult if age >= 3days
    - Baby is less than 3 days
  - Int cost
    - Tiger = $10,000
    - Penguin = $1,000
    - Turtle = $100
  - Int number of babies
    - Tiger = 1 baby
    - Penguins = 5 babies
    - Turtles = 10 babies
  - Int Base food cost
    - Constant or provided by user
    - Tigers = 5x base cost
    - Penguins = base cost
    - Turtles = 50% of base cost
  - Int Pay off
    - Per day & per animal
    - Tiger = 20% cost of animal
    - Penguins = 10% cost of animal
    - Turtle = 5% cost of animal

Tiger (inherits from Animal)

- Construct numbers
  - Adjust numbers

Penguin (inherits from Animal)

- Construct numbers
  - Adjust numbers
  - Should serve as closest to base

Turtle (inherits from Animal)

- Construct numbers
  - Adjust numbers

Input Validation

- Same as before
- Use what was set up in lab 3
- Maybe add isBelow function

Design Reflection

I originally intended to use array as I didn't feel comfortable with vectors. Early in the coding process, I was thinking off how to delete elements of dynamically generated arrays. It seemed like a big logical challenge to figure out how to do this without the code getting messy and unreadable. I was also concerned about doubling the dynamic arrays. After some trial and error, I got there. It wasn't quite as bad as I thought it would be.

For my Zoo, I intended to have a day function to do most of the work for each day. After coding, it seemed right to break up each major task into it's own function. I briefly considered getting rid of the day function entirely, since it is basically just pointing to a series of other functions. However, I need a sensical place to put the bank check, so I ended up keeping it. Given the chance to refactor, I think I would take the day function out entirely.

The random event function was particularly challenging. I intended to have a random event function that did all of the work, but actions like animals dying and being born were large enough to warrant their own dedicated functions. I think this helped things over all be more readable and digestible.

The inheritance wasn't too bad to deal with as a new concept. It was fairly quick an easy to set up after reviewing the lectures.

I think next time I will incorporate the input validation initially instead of waiting until the bulk of the code was done. I ended up having to almost rewrite my menu functions to accommodate the validation, so next time, I'll just do that from the start.

I found this project to be overall much easier than the last. I found myself breaking things down into smaller functions more than I expected and I feel that I am getting used to dealing with the C++ version of arrays. I encountered much fewer problems this time.

Test Table

Test	Input	Test Target	Expected Output	Actual Output
Player can exit or continue game on first prompt	1. 0 2. 1	Zoo::startMenu()	1. exit 2. Regular game play	1. exit 2. Regular game play
Player buys animals	1. 1 or 2	Zoo::makeTigers Zoo::makePenguins Zoo::makeTurtles	1. Remaining bank balance 2. Next prompt	1. Remaining bank balance 2. Next prompt
startMenu triggers game play	1. 1, 2, 2	Zoo::day • randomEvent • ageAll • calcCosts • calcProfits	1. Print age of all animals 2. Subtract feeding costs with message 3. Random event 4. Add profits 5. Ask to buy animal 6. Ask to keep playing	1. Print age of all animals 2. Subtract feeding costs with message 3. Random event 4. Add profits 5. Ask to buy animal 6. Ask to keep playing
Random event triggers appropriate functions	1. Animal dies 2. Baby is born 3. nothing 4. Bonus	Zoo::randomEvent Zoo::newBaby Zoo::animalDies	1. Animal is subtract and doesn't show up in print out 2. Animal is added at age 0; number of animals varies with animal type 3. Print message for nothing 4. Tiger bonus added to bank	1. Animal is subtract and doesn't show up in print out 2. Animal is added at age 0; number of animals varies with animal type 3. Print message for nothing 4. Tiger bonus added to bank
Check bank at start of each	1. Bank is less than 1 2. Bank is more than 1	Zoo::day	1. exit game 2. Keeping playing	1. exit game 2. Keeping playing
INPUT VALIDATION				
Each prompt rejects char (6 prompts)	1. hgjhjf	isInteger()	1. Error asking for integer	1. Error asking for integer
Each prompt rejects floats	1. 1.11	isInteger()	1. Error asking for integer	1. Error asking for integer
Each prompt rejects numbers out of range	1. 4 2. -1 3. 0	isBetween()	1-2. Error asking for value in range 3. Regular game play	1-2. Error asking for value in range 3. Regular game play
FLIP				
Valgrind: no segmentation faults	valgrind lab3	Whole program	1. No errors	ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Valgrind: No memory leaks or	valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all --track-origins=yes ./lab3	Whole program	1. No errors	ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Complies on school server	1. g++ -std=c++0x main.cpp ant.cpp menu.cpp -o zoo	Whole program	1. nothing 2. First prompt	1. nothing 2. First prompt (