Program name: Project2 Zoo Tycoon
Author: Xiaoying Li
Date: 7/20/2019
Description: Design & Test Plan & Reflection of Project2 Zoo Tycoon

- **Design**

1. Main

The main function controls the entire program.
Ask user whether he/she wants to start the game or not.
If the user decided not to start the game, quit the program.
Otherwise, start the game.
Create a Zoo object.
Introduce some basic information about the game to the user.
Ask user to buy animals to start the game.
Start a day:
increase all animal's age by 1 day;
pay the feeding cost for all animal;
a random event takes place.
End a day:
calculate the profit for the day;
ask the user if he/she would like to buy an adult animal;
prompt user whether to keep playing or end the game.
If the user decided not to keep playing, quit the program.
Otherwise, start another day.
Loop until the user has no money or decided to quit.
Check if the user has no money, print a message to tell the user the game is over, and end the game.
After the game ends, ask the user whether he/she wants to play again.
If the user decided to play again, start the game again.
Otherwise, quit the program.
Loop the game until the user decided to quit.

2. Menu

The menu function prints menu options to the screen for the user, and after the user makes a choice, verify the user's input, and return the value back to the program.

(1) startMenu:
When the project starts, ask user whether he/she wants to start the game or not.
If the user decided to start the game, return true.
Otherwise, return false.

(2) endMenu:
When the game ends, ask the user whether he/she wants to play the game again.
If the user decided to play the game again, return true.

Otherwise, return false.

3. Animal

The Animal class contains general information of all types of animals in the zoo.
Its private data members follow the requirement of the project:
Age, cost, number of babies, base food cost and payoff.
All data members are integers.
It has following public member functions:
Default constructor;
Get function for all data members;
Function increases an animal’s age by 1 day.

4. Tiger

The Tiger class inherits from Animal class, and specifies the information of a tiger.
Its private data members are inherited from Animal class.
It has following public member functions:
Default constructor;
Initialize all data members use a tiger’s information.
Overload constructor:
Take in an integer parameter as a tiger's age, and use it to initialize a tiger’s age when a Tiger object is created.

5. Penguin

The Penguin class inherits from Animal class, and specifies the information of a penguin.
Its private data members are inherited from Animal class.
It has following public member functions:
Default constructor;
Initialize all data members use a penguin’s information.
Overload constructor:
Take in an integer parameter as a penguins age, and use it to initialize a penguin’s age when a Penguin object is created.

6. Turtle

The Turtle class inherits from Animal class, and specifies the information of a turtle.
Its private data members are inherited from Animal class.
It has following public member functions:
Default constructor;
Initialize all data member use a turtle’s information.
Overload constructor:
Take in an integer parameter as a turtle's age, and use it to initialize a turtle’s age when a Turtle object is created.

7. Zoo

The Zoo class has a dynamic array for each type of animal and defines all events in the zoo.

It has following private data members:

(1) Integer holds tiger’s number;
(2) Integer holds penguin’s number;
(3) Integer holds turtle’s number;
(4) Integer holds size of tiger’s array;
(5) Integer holds size of penguin’s array;
(6) Integer holds size of turtle’s array;
(7) Dynamic array for Tiger;
(8) Dynamic array for Penguin;
(9) Dynamic array for Turtle;
(10) Integer holds number of day;
(11) Integer holds money in the bank account;
(12) Integer holds bonus.

It has following private help functions:

(1) A function adds a tiger to tiger’s dynamic array;
It takes in an integer parameter as a tiger's age, and uses it to initialize a tiger's age when a Tiger object is created. Then add the tiger to tiger's dynamic array.
If the tiger can fit in the current tiger array, add the tiger to tiger's dynamic array.
Else, double the size of the current tiger array to hold more tigers.
And add the tiger to the new tiger's dynamic array.

(2) A function adds a penguin to penguin’s dynamic array;
It takes in an integer parameter as a penguin's age, and uses it to initialize a penguin's age when a Penguin object is created. Then add the penguin to penguin's dynamic array.
If the penguin can fit in the current penguin array, add the penguin to penguin's dynamic array.
Else, double the size of the current penguin array to hold more penguins.
And add the penguin to the new penguin's dynamic array.

(3) A function adds a turtle to turtle’s dynamic array;
It takes in an integer parameter as a turtle's age, and uses it to initialize a turtle's age when a Turtle object is created. Then add the turtle to turtle's dynamic array.
If the turtle can fit in the current turtle array, add the turtle to turtle's dynamic array.
Else, double the size of the current turtle array to hold more turtles.
And add the turtle to the new turtle's dynamic array.

(4) A function that implements the random event that a sickness occurs to an animal in the zoo.
Generate an integer from 1, 2 and 3 representing 3 types of animals.

If the animal type picked at random is tiger, check if there is a tiger in the zoo.

If there is at least one tiger in the zoo, remove one tiger from tiger's dynamic array.

Else, let the animal type picked at random be penguin.

If the animal type picked at random is penguin, check if there is a penguin in the zoo.

If there is at least one penguin in the zoo, remove one penguin from penguin's dynamic array.

Else, let the animal type picked at random be turtle.

If the animal type picked at random is turtle, check if there is a turtle in the zoo.

If there is at least one turtle in the zoo, remove one turtle from turtle's dynamic array.

Else, let the animal type picked at random be tiger.

If all three types of animals are checked and there is no animal in the zoo, output a message to the player.

(5) A function that implements the random event that a boom in zoo attendance occurs.

Generate a random bonus between 250 and 500 dollars for each tiger in the zoo for the day.

(6) A function that implements the random event that a baby animal is born.

Generate an integer from 1, 2 and 3 representing 3 types of animals.

If the animal type picked at random is tiger, check if there is a tiger in the zoo.

If there is at least one tiger in the zoo, check if there is a tiger odd enough to be a parent.

If a tiger odd enough to be a parent is found, add 1 baby tiger of age 0 to tiger's dynamic array.

If there is no tiger odd enough to be a parent, let the animal type picked at random be penguin.

If there is no tiger in the zoo, let the animal type picked at random be penguin.

If the animal type picked at random is penguin, check if there is a penguin in the zoo.

If there is at least one penguin in the zoo, check if there is a penguin odd enough to be a parent.

If a penguin odd enough to be a parent is found, add 5 baby penguins of age 0 to penguin's dynamic array.

If there is no penguin odd enough to be a parent, let the animal type picked at random be turtle.

If there is no penguin in the zoo, let the animal type picked at random be turtle.

If the animal type picked at random is turtle, check if there is a turtle in

the zoo.

If there is at least one turtle in the zoo, check if there is a turtle odd enough to be a parent.

If a turtle odd enough to be a parent is found, add 10 baby turtles of age 0 to turtle's dynamic array.

If there is no turtle odd enough to be a parent, let the animal type picked at random be tiger.

If there is no turtle in the zoo, let the animal type picked at random be turtle.

If all three types of animals are checked and there is no animal odd enough to be a parent, output a message to the player.

(7) A function that check the player's bank account, if the player has no money, return false to end the game; else, return true to continue the game.

It has following public member functions:

(1) A constructor that initializes data members when a Zoo object is created.

(2) A function that outputs some information of the game at the start of the game, and asks the player to buy animals to start the business.
Get the number of tigers, penguins, and turtles the player would like to buy.
Add that many tigers, penguins and turtles to their array.
Subtract the cost from the bank.

(3) A function that increases all animals' age by 1 day, and subtracts the feeding cost of all animals from the bank at the beginning of the day.

(4) A function that generate an integer from 1, 2 and 3 to determine what event takes place during the day.

(5) A function that calculates the profit for the day and ask the player if they would like to buy an adult animal before the day ends.
If the player wants to buy an adult animal, ask for the type of animal they would like.
Then add the animal to the zoo and subtract that cost from the bank.

(6) A function that prompts the player whether to keep playing or end the game. If the player decides to keep playing, return true; if not, return false.

(7) A destructor.

8. Validation

The validation function takes in a string and pass it by reference. The function makes sure the user's input is an integer. Afterwards, it returns the integer the user inputted.

Loop until the user inputs an integer.

When a string inputted, detect whether an extraction has failed and fix it.

If an extraction has failed, ask the user to enter again.

If extraction succeeds, then make sure no extraneous input.

Then detect if every character of the string is digit.

If every character of the string is digit, then the string is an integer.

Otherwise, ask the user to enter again.

If the input string is an integer, transfer it to an integer, and return the integer.

The validation of the range of every integer returned should be validated after the integer returns to the program.

- **Test Plan**

This test plan does not include test cases for input validation, startMunu() and endMenu(), since they have been tested at Project1. This test plan only verifies if the program meets all given specifications and works well at all cases.

Test Scope	Description	Expected Results
Zoo.zoo()	Make sure a Zoo object is correctly created.	All the data members can be properly initialized, and member function in Zoo can be called in following procedures.
Zoo.menu()	Input different purchase strategy.	Zoo.addTiger(), Zoo.addPenguin(), and Zoo.addTurtle() can be called, number and purchase cost of animals can be correctly displayed. And bank account balance will deduct purchase cost correctly.
Zoo.dayBegin()	Input different purchase strategy in Zoo.menu() when game begins. Purchase different animals in the end of every day. Play game longer to keep number of each animal above 10.	Output correct number of animals, feed cost, and bank account balance. Each animal's information, including age and number, should be output correctly after all animal's age increased 1 day.
Zoo.randomEvent()	User plays longer than 30 days, the daily event occurs randomly on every day.	When random event is a sickness occurs to an animal in the zoo: Zoo.sick() should be called properly, generates sick animal species randomly. Program should be able to handle situations that there's no animal in the zoo or lack the generated type of animals. The last animal of the corresponding species should be removed from dynamic array when program output all the animal information on the following day.

		When random event is a boom in zoo attendance occurs: Zoo.boom() should be properly called and generate bonus between 250-500 randomly. Output profit on that day should plus bonus.
		When random event is a baby animal is born: Zoo.baby() should be properly called, generates the species of baby animal randomly. Program should be able to handle situations that this animal species doesn't exist, this animal species doesn't have adult animal, and there's no adult animal in the zoo. Output of all animal information on the second day should increase baby animal number for corresponding species with correct age.
Zoo.dayEnd()	Input different purchase strategy in Zoo.menu() when game begins. Purchase different animal before the end of every day.	Output correct profit of that day. Profit should plus bonus on that day if random event is boom(). Output animal information on following day should properly display purchased animals. Bank account balance should be displayed correctly.
Zoo.keepPlay()	Determine if bank account balance>0. If balance>0, select keep playing or end game.	checkMoney() should be properly called to determine if user's balance>0. If balance <= 0, notices user and ends game. If balance>0, game should proceed to next day if keep playing is selected; Output endMenu() if user selects to end game.
Zoo.~Zoo()	Use Valgrind: valgrind --leak-check=yes statements to check memory leak	All heap blocks were freed, no leaks are possible.

- **Reflection**

1. Design changes I have made:

In my initial design, Zoo class was separated into two parts: one only had a dynamic array for each animal species, with increase and decrease of animal number for each species; all the events happen in the zoo were placed in the other class Game. In the beginning I thought this design could prevent a file become too big and make my code more organized. However, when I started coding class Game, I found I have to access all the animal information through Zoo class, which made my code extremely complicated. Sometimes I had to use many “dot” to access variables or functions I needed. This increased the chance of having errors in code, and it’s an obstacle to read my code. In this case, I decided to not using Game class and placed all the variables and functions in Zoo class.

I made another minor modification during test. Initially the output of my code didn’t include the information of each animal in the zoo. This flaw compromised the clarity of my test because I was not able to know the actual situation when animals increase or decrease. Therefore, I added statements in Zoo.dayBegin() to output animal’s information when its age increased.

2. How I solved problems encountered:

The biggest challenge of this project was inaccuracy of project description. I noticed many classmates also have this kind of problem of having difficulty to understand the requirement of project. I posted questions on Piazza, and luckily received a lot of help from TAs and classmates. Also, I found many coding websites like learncpp.com, cplusplus.com, and geeksforgeeks.com very helpful since they answered a lot of knowledge related questions.

3. What I learned for this project:

This time I had an in-depth understanding of inheritance. Comparing with Lab3, I applied inheritance in my code more confidently. In order to fulfill some goals in my code, I also learned friend function/class related materials. Although I did not use friend function/class in my code this time, the accumulation of knowledge is still a big progress for me. Also, I sharpened my skills to organize my code thanks to the size and complexity of this project. It’s a valuable training for me to handle big project.