Huy Nguyen

CS 162 – Intro to CS II

Section 400 – Spring 2019

Project 2 – Zoo Tycoon

Design

Animal class

1. Description:
   1. Animal class represents an animal of the Kingdom Animalia, that will live in a Zoo in our Zoo Tycoon game. All animals have an age, cost, number of babies each animal produces at a time, base food cost, and profitable payoff. Animal class functions include getting its age, cost, number of babies, base food cost, payoff, incrementing its age, and determine if they are an adult.
2. Protected data members:
   1. Int Age
   2. Int Cost
   3. Int Number of babies
   4. Int Base food cost (static const) = $10
   5. Double payoff
3. Public member functions:
   1. Constructor/Default constructor:
      1. Arguments:
         1. Int age
         2. Int cost
         3. Int number of babies
         4. double payoff
      2. Initializes age, cost, number of babies, payoff to arguments
         1. Defaults = zero
   2. incrementAge
      1. increases age of animal by 1
   3. isAdult (const)
      1. returns true if age is 3 or more, false otherwise
   4. get age (const)
      1. returns age
   5. get cost (const)
      1. returns cost
   6. get number of babies (const)
      1. returns number of babies
   7. get base food cost (const)
      1. returns base food cost
   8. get payoff (const)
      1. returns payoff

Tiger class

1. Description:
   1. Tiger is a derived class of Animal. Tigers represents a large, predatory species of Cat with red-orange fur, and large black stripes. Tigers are one of the more popular exhibits at our Zoo Tycoon simulation. They have a $10,000 purchasing cost, have one baby at a time, costs 5 times as much as the base cost to feed, and generates 20% of their purchasing cost as payoff. They inherit all data members and functions of Animal, with the addition of a string for their name, “Tiger”, and a function to get their name.
2. Protected data members:
   1. Inherits all from Animal class
   2. String name = “Tiger” (const static default initialized into variable)
   3. Double food cost multiplier = 5 (const default) (DEPRECATED)
   4. Double payoff percentage = 0.2 (20%) (const default) (DEPRECATED)
   5. Double feeding cost = base food cost \* food cost multiplier = $50
3. Public member functions:
   1. Inherits all from animal class
   2. Get name – returns a string of the animal’s name (“Tiger”) (const)
   3. Get Feeding cost – returns int of this animals’s feeding cost (const)
   4. Constructor/default constructor
      1. Arguments
         1. Int age – default = 0
      2. Sets age
      3. Sets purchase cost to default
      4. Sets number of babies to default
      5. Calculates and sets feeding cost
      6. Calculates and sets payoff
      7. Sets name to default

Penguin class

1. Description:
   1. Penguin is a derived class of Animal. Penguins represent an aquatic, flightless bird that can typically be found in cold climates. They are typically white feathered, with black wings and head. They have a $1,000 purchasing cost, have 5 babies at a time, have the same feeding cost as the base food cost, and generate 10% of their purchasing cost as payoff. They inherit all data members and functions of Animal, with the addition of a string for their name, “Penguin”, and a function to get their name.
2. Protected data members:
   1. Inherits all from Animal class
   2. String name = “Penguin” (const static default initialized into variable)
   3. Double food cost multiplier = 1 (const default)
   4. Double payoff percentage = 0.1 (10%) (const default)
   5. Double feeding cost = base food cost \* food cost multiplier = $10
3. Public member functions:
   1. Inherits all from animal class
   2. Get name – returns a string of the animal’s name (“Penguin”) (const)
   3. Get Feeding cost – returns int of this animals’s feeding cost (const)
   4. Constructor/default constructor
      1. Arguments
         1. Int age – default = 0
      2. Sets age
      3. Sets purchase cost to default
      4. Sets number of babies to default
      5. Calculates and sets feeding cost
      6. Calculates and sets payoff
      7. Sets name to default

Turtle class

1. Description:
   1. Turtle is a derived class of Animal. Turtles represent a reptile with a large bony shell in which it uses as a shield to protect itself, or to hide away from predators. They are typically greenish and brown animals. They have a $100 purchasing cost, have 10 babies at a time, feeding costs are half that of the base feeding cost, and generate 5% of their purchasing cost as payoff. They inherit all data members and functions of Animal, with the addition of a string for their name, “Turtle”, and a function to get their name.
2. Protected data members:
   1. Inherits all from Animal class
   2. String name = “Turtle” (const static default initialized into variable)
   3. Double food cost multiplier = 0.5 (const default)
   4. Double payoff percentage = 0.05 (5%) (const default)
   5. Double feeding cost = base food cost \* food cost multiplier = $5
3. Public member functions:
   1. Inherits all from animal class
   2. Get name – returns a string of the animal’s name (“Turtle”) (const)
   3. Get Feeding cost – returns int of this animals’s feeding cost (const)
   4. Constructor/default constructor
      1. Arguments
         1. Int age – default = 0
      2. Sets age
      3. Sets purchase cost to default
      4. Sets number of babies to default
      5. Calculates and sets feeding cost
      6. Calculates and sets payoff
      7. Sets name to default

Custom Animal class (Extra credit)

1. Description:
   1. Custom Animal class is a derived class of Animal. It allows the user to create a custom animal in the Zoo Tycoon simulation. The user will determine the animal’s purchase cost, number of babies, feeding costs, and payoff. They inherit all data members and functions of Animal, with the addition of a string for their name, and a function to get their name.
2. Protected data members
   1. Inherits all from Animal class
   2. String name – custom name provided by user
   3. Double feeding cost
3. Public member functions:
   1. Inherits all from animal class
   2. Get name – returns a string of the animal’s name (const)
   3. Get Feeding cost – returns int of this animals’s feeding cost (const)
   4. Default constructor
      1. Sets name to empty string “”
      2. Age = 0
      3. Cost = 0
      4. Number of babies = 0
      5. Feeding cost = 0
      6. Payoff = 0
   5. Constructor
      1. Arguments:
         1. String name
         2. Int age
         3. Int cost
         4. Int number of babies
         5. Double Feeding cost
         6. Double payoff
      2. Sets name, age, purchase cost, number of babies, feeding cost, and payoff

Animal Constants class

1. Description:
   1. Class to hold a lookup table of animal class constant configurations and defaults
2. Enum AnimalType
   1. Tiger
   2. Penguin
   3. Turtle
   4. Custom
3. Structure animal
   1. String default\_name
   2. Int default\_age
   3. Int default\_cost
   4. Int default\_num\_babies
   5. Double default\_food\_cost\_multiplier
   6. Double default\_payoff\_percentage
4. Defaults (lookup table)
   1. Static Array of animal structures
   2. Accessed such as:
      1. AnimalConstants::defaults[TIGER].name
      2. AnimalConstants::defaults[TIGER].cost
5. ZERO
   1. Used to initialize zero values
   2. Static constant int ZERO = 0;
6. Use typdefs to set aliases for
   1. Enum type
   2. Enum values
   3. ZERO

Zoo class

1. Description:
   1. Zoo class represents a zoo establishment in our Zoo Tycoon game. A Zoo maintains a collection of animals and allows visitors to pay an entrance fee to come and observe the animals. The zoo will house tigers, penguins, turtles, and other animals. A zoo is a business that is responsible for its animals, so a zoo is able to purchase animals, feed the animals, and collect profits on each animal. Random events can happen every day, such as an animal gets sick and dies, an animal gives birth, a boom in zoo attendance, or nothing at all. Zoo is able to manage its own bank account, withdraw from the account for purchases, and add to the account for generated revenue.
2. Key objects
   1. Menu class
   2. MyRandom class
3. Key structures
   1. Exhibit
      1. Keeps track of count of a type of animal
      2. Keeps track of exhibit capacity (exhibit size) of a type of animal
4. Key enums
   1. AnimalType (DEPRECATED – MOVED TO ANIMAL CONSTANTS CLASS)
      1. Tiger
      2. Penguin
      3. Turtle
      4. Custom
   2. FeedType (extra credit)
      1. Cheap
      2. Generic
      3. Premium
   3. EventType
      1. Nothing
      2. Boom
      3. Birth
      4. Sickness
   4. ExitReason
      1. Quit
      2. Bankrupt
5. Key constants (static constants)
   1. Int Start exhibit size
      1. Starting size of all exhibits = 10
   2. Double Starting bank account
      1. Starting amount in bank account = 100,000
   3. Int feed multiplier size
      1. Size of feed multiplier lookup array
   4. Double cheap multiplier
      1. Multiplier value for cheap feed = 0.5
   5. Double generic multiplier
      1. Multiplier value for generic feed = 1.0
   6. Double premium multiplier
      1. Multiplier value for premium feed = 2.0
   7. String event\_file
      1. Name of out file to write/read random events
      2. “event\_file.txt”
   8. Int animal exhibit count (DEPRECATED – MOVED TO ANIMAL CONSTANTS)
      1. The number of different exhibits for animals
      2. Used as the size for animal\_exhibits array
      3. = 4
   9. Int starting\_value (DEPRECATED – MOVED TO ANIMAL CONSTANTS)
      1. Constant value zero to be used to multiple places
6. Private data members
   1. Menu menu;
   2. Double bank account
   3. Animal \*\*\*animal\_exhibits
      1. Pointer to array of pointers to array of pointers to animals
         1. Tiger \*[] – array of pointers to tigers
         2. Penguin \*[] – array of pointers to penguins
         3. Turtle \*[] – array of pointers to turtles
         4. CustomAnimal \*[] – array of pointers to custom animals
   4. Double daily bonus profit (tiger bonus)
      1. Holds daily bonus profit for tiger bonus
      2. Resets to zero at start of each day
   5. Double food cost multiplier (Extra credit)
      1. Multiplier dependent on user selection of daily feed type
      2. If feed type is:
         1. Cheap – multiplier = 0.5
         2. Generic – multiplier = 1.0
         3. Premium – multiplier = 2
   6. FeedType enum todays feed type (extra credit)
      1. Enum to represent feed type selected for today
      2. Changes after user chooses feed type daily
   7. Bool exit status
      1. Boolean denoting if user wants to quit the game
      2. Starts at exit status = false
   8. Exhibit \*exhibit\_count (pointer to struct array)
      1. Pointer to array of exhibit structures that keep track of count of each animal and the size of their exhibit (capacity)
   9. Double \*feed\_multiplier\_lookup
      1. Pointer to array of doubles that represent the different feed type multipliers
7. Public member functions
   1. Constructor/default constructor
      1. Seed random number generator
      2. Instantiate menu object (probably not necessary)
      3. Initialize bank account to constant starting bank
      4. Initialize exit status to false
      5. Initialize daily profit bonus to “starting value” = 0
      6. Initialize today’s feed type to generic
      7. Initialize food cost multiplier to generic (will auto convert to 1.0)
      8. Initialize feed multiplier lookup table array
      9. Set animal\_exhibits to point to new Animal \*\* array of size [animal exhibit count = 4]
         1. Loop through array (animal exhibit count)
            1. At each index, create a new Animal \* array of size [start exhibit size = 10]
            2. Initialize all new pointers to nullptr

Loop through specific animal exhibit array

Set each pointer to nullptr

* + 1. Set exhibit\_count to point to new Exhibit array of size [animal exhibit count = 4]
       1. Loop through array (animal exhibit count)
          1. Set exhibit capacity to [start exhibit size = 10]
          2. Set animal count to [starting value = 0]
  1. Destructor
     1. Delete animal\_exhibits
        1. Loop through array (animal exhibit count)
           1. Loop through (exhibit count[animal type (which is index i)].count)
           2. Delete animal \*
           3. Delete array [] at this index (animal \*\*)
        2. Delete array [] animal\_exhibits (animal \*\*\*)
     2. Delete exhibit\_count array
        1. Delete [] exhibit\_count (array of exhibit structures)
     3. Delete feed multiplier lookup array
        1. Delete [] feed\_multiplier\_lookup
  2. Start
     1. Description
        1. Public driver function to run game. Game entry point.
     2. Print welcome message
     3. \*\*Print note that all 3 extra credits are included in this program\*\*
     4. Print bank account
     5. Prompt user to buy 3 animals
        1. 1 or 2 of each
     6. Create/add animal to exhibit
        1. Animals are 1 day old
        2. Loop through number of user input for each animal type
           1. Add animal

This creates animal

Adds to exhibit

And subtracts from bank

* + 1. Call runTycoon to start simulation

1. Private member functions
   1. runTycoon
      1. description:
         1. driver function to run the simulation for “each day”. Contains a while loop to run all the tasks necessary in a day of the Zoo Tycoon business.
      2. Local variable
         1. Int day counter = 0
      3. While the user has not chosen to exit, and the business is not bankrupt (i.e. exit status != true)
         1. Increment day counter + 1
         2. Print day number: day counter
         3. Run beginning-of-day
         4. Run mid-day
         5. Run end-of-day
   2. Beginning-of-day
      1. Description:
         1. The tasks at the start of each day of the Zoo Tycoon
      2. Reset daily bonus profit to zero
      3. All animals increase age by 1
      4. Print bank account
      5. Ask user for today’s feed type (extra credit)
         1. Change food cost multiplier
         2. Change todays feed type data member
      6. Pay feeding cost of all animals
      7. Print all animals fed message
      8. Print bank account
   3. Midday
      1. Description:
         1. The tasks and events that happen during the middle portion of the day at the Zoo
      2. Random event happens
      3. Write status message to file, then read from file and print to screen (extra credit)
      4. Calculate profit for all animals
         1. Add to bank
         2. Add daily bonus profit to bank (tiger bonus)
         3. Print todays profits have been calculated message
      5. Print bank account
      6. Ask user if they want to buy a new animal (3 days old)
         1. Tiger
         2. Penguin
         3. Turtle
         4. New animal (extra credit)
            1. Prompt for traits of new animal
         5. Add animal (overloaded option for custom animal)
            1. This makes the animal at 3 days old
            2. Adds to exhibit
            3. Subtracts from bank
   4. End-of-day
      1. Description:
         1. The tasks at the end of the day at the Zoo Tycoon.
      2. Print exhibit count
      3. Print bank account
      4. Check if bankrupt
         1. If yes, print message “game over”
            1. Abstract this into a function
            2. Optional
            3. Number of days opened?
            4. Tell them number of animals
            5. Ending balance
            6. All the animals were released and went on a rampage in the city
            7. Reason: you went bankrupt
         2. End the game (set exit status to true)
      5. Ask user if they want to keep playing or end the game
         1. If end the game, print message “game over”
            1. Abstract this into a function
            2. Optional
            3. Number of days opened?
            4. Tell them number of animals
            5. Ending balance
            6. All the animals were released an went on a rampage in the city
            7. Reason: you gave up
         2. End the game (set exit status to true)
   5. Random event
      1. Description:
         1. Random events happen on a daily basis. There could be a boom in zoo attendance, an animal gives birth, an animal gets sick and dies, or nothing happens.
      2. Draw (randomly selects)
      3. If draw == 1
         1. Nothing happens
      4. If draw == 2
         1. Boom in zoo attendance
      5. If draw == 3
         1. Baby is born (animal gives birth)
      6. If draw > 3
         1. Animal gets sick
   6. Draw (const)
      1. Description:
         1. Randomly select the random event (depends on food quality)
      2. If feed type is:
         1. Generic or premium: randomly select number between 1-4 (normal odds)
         2. Cheap: randomly select number between 1-5 (higher chance for sickness)
      3. Returns integer of random event selection
   7. Nothing happens (const)
      1. Write/read status message that nothing happened today
         1. Call write\_read function
            1. Arguments:

EventType = Nothing

* 1. Boom in zoo attendance
     1. Local variable:
        1. Int todays\_bonus
     2. Calculate random bonus between 250-500 for EACH tiger
        1. Randomly select number between 250-500
           1. Set number to todays\_bonus
        2. Multiply todays\_bonus by total number of tigers
     3. Set that number to daily bonus profit (tiger bonus)
     4. Write/read status message that a boom in zoo attendance happened
        1. Call write\_read function
           1. Arguments:

EventType = Boom

Animal \* = nullptr

Int todays\_bonus

* 1. Baby is born
     1. Create temporary local AnimalType enum variable “selected\_type”
     2. Create temporary local animal \*pointer “selected\_animal”
     3. Create temporary local Vector<AnimalType> animals\_checked
     4. Pick animal to have baby
        1. Passing in animals\_checked
        2. Passing in selected\_animal pointer
        3. Passing in selected\_type
     5. If no animals are eligible at all (pick animal to have baby returns false)
        1. Run random event again
     6. If an animal was selected to have a baby (pick animal to have baby returns true)
        1. Run give birth
           1. Pass in selected\_type
           2. Pass in selected\_animal (by pointer)
  2. Pick animal to have baby
     1. Description:
        1. Randomly selects animal to give birth, while keeping track of which animal exhibits are not able to give birth
        2. Recursively calls itself until it has checked all animals for eligibility
        3. returns Boolean true if an eligible animal was selected
        4. returns Boolean false if no animal was able to be selected
     2. Arguments:
        1. Vector<AnimalType> &animals\_checked
        2. Animal \* & selected\_animal (pass in a pointer by reference)
        3. AnimalType &selected\_type
     3. Local variable:
        1. Boolean animal\_chosen = false
     4. If size of animals\_checked is equal to animal\_exhibit\_count
        1. Return false
     5. Run check animals count
        1. Pass in:
           1. Animal\_chosen
           2. Selected\_type
           3. Animals\_checked
     6. If not animal\_chosen:
        1. We must not have any animals..
        2. Return false
     7. Loop through “selected\_type” exhibit to check for age
        1. Call pointer -> isAdult
        2. If found one that is old enough, copy animal pointer address to temporary animal pointer reference passed in (“selected\_animal”)
        3. Return true
     8. If no animal of type is old enough, pick another animal type
        1. Recursively call “Pick animal to have baby”
           1. Passing in animals\_checked vector by reference
           2. Passing in \*selected\_animal animal pointer by reference
           3. Passing in selected\_type by reference
  3. Give birth
     1. Description:
        1. An animal of specified type gives birth according to how many babies their species produces at a time.
     2. Arguments:
        1. AnimalType type
        2. Animal \*animal
     3. In for loop of animal’s number of babies (animal->getNumBabies)
        1. If animal type is custom
           1. Add animal (overloaded) – get traits via animal->get[trait]

Type

Age = should be 0

Name

Cost

Num babies

Feeding cost

Payoff

* + - 1. Else
         1. Add animal (normal)

Type

Age = should be 0

* + 1. Write/read status message that an animal gave birth
       1. Call write\_read function
          1. Arguments

EventType = Birth

Animal \* = animal

* 1. Animal gets sick
     1. Description
        1. An animal in the zoo has gotten sick. There’s a slim chance they might make it.
     2. If today’s feed type == premium
        1. 50/50 chance of getting sick or redraw
        2. Select random number between 1 and 2
        3. If 1
           1. No animals got sick
           2. Call random event again
        4. Else if 2
           1. Animal dies
     3. Else
        1. Animal dies
  2. Animal dies
     1. Description:
        1. Randomly select an animal that will get “sick” and die. Remove one animal of that type from the exhibit.
     2. Create temporary local Vector<AnimalType> animals\_checked
     3. Create temporary local variable – AnimalType selected\_type
     4. Create temporary local boolean animal\_chosen = false
     5. Run check animals count
        1. Pass in:
           1. Animals\_checked
           2. Selected\_type
           3. Animal\_chosen
     6. If animal chosen
        1. Remove animal of that type (animal died)
           1. Call remove animal passing in “selected\_type”
     7. Else If not animal\_chosen
        1. We must not have any animals left..
        2. Run random event again
  3. Check Animals count
     1. Description:
        1. Randomly loops through animal exhibits to find eligible animals to either give birth or get sick and die. Keeps track of which exhibits have been checked, and if an eligible animal has been found.
     2. Arguments
        1. Vector<AnimalType> &animals\_checked
           1. Vector of animal types by reference to keep track of which animal exhibits have already been checked, as to prevent infinite loops due to random selection of animals
        2. AnimalType &selected\_type
           1. animal type by reference denoting which animal type was randomly selected
        3. Boolean &animal\_chosen
           1. boolean by reference denoting if an eligible animal has been chosen
     3. Do:
        1. Pick random animal and set to “selected\_type”
        2. Check if selected\_type exhibit has any of that type of animal
           1. If exhibit\_count[selected\_type].count > 0

Set animal\_chosen = true

* + - 1. Check if selected\_type has already been checked
         1. If (Std::find(animals\_checked.begin(), animals\_checked.end(), selected\_type) == animals\_checked.end())
         2. If true – animal is not in animals\_checked vector yet

Add selected\_type to animals\_checked

Animals\_checked.push\_back(selected\_type)

* + - 1. While: !animal\_chosen (animal\_chosen is false) AND size of animals\_checked != animal\_exhibit\_count
  1. Pick random animal (const)
     1. Description:
        1. Randomly picks an animal from animal\_exhibit
     2. Pick random number between 0 – animal\_exhibit\_count-1
     3. Return random number static\_casted into AnimalType
  2. Remove animal
     1. Description:
        1. Randomly selects an animal from a specific exhibit passed as an argument. Deletes randomly selected animal in the exhibit. Shifts the rest of the animals in the animal’s exhibit array down to fill in the space of the deleted animal. Deletes the pointer of the last animal in the exhibit, because it is now duplicated. Decreases exhibit count of that animal by 1.
     2. Arguments:
        1. AnimalType selected\_type
     3. Local variable
        1. Animal \*temp\_animal – pointer to animal to be deleted
     4. Randomly select an animal
        1. Pick a random index between 0 and exhibit count – 1
     5. Point temp\_animal to animal at that index
     6. Shift all animals from right side of index down except last one
        1. Loop from index of deleted animal to count – 1
     7. Delete animal pointer at the end of the array because it is now a duplicate
        1. Delete animal pointer at index exhibit count – 1
           1. Set to nullptr
     8. Decrease exhibit count of animal type by 1
     9. Write/read animal died from sickness message
        1. Call write\_read function
           1. Arguments

EventType = Sickness

Animal \* = temp\_animal

* + 1. Delete animal object pointed to by temp\_animal
  1. Add animal
     1. Description
        1. Adds a new animal to the exhibit by creating a new animal, adds them to the exhibit, subtracts their cost from the bank, then prints out a message saying a new animal has been added
     2. Arguments:
        1. AnimalType type
        2. Int age
           1. Default = 0
     3. Local variable
        1. Animal \*new\_animal
     4. Call make animal
        1. Pass in Type and age
        2. Returns a pointer to new\_animal
     5. Call add to exhibit
        1. Pass in type and pointer to new\_animal
     6. Call subtract from bank
        1. Passing animal cost
           1. New\_animal->getCost()
     7. Call print animal added message
        1. Pass in pointer to new\_animal
  2. Add animal (overloaded for custom animal)
     1. Description
        1. Overloaded version of add animal specifically for creating new custom animals. Adds a new animal to the exhibit by creating a new animal, adds them to the exhibit, subtracts their cost from the bank, then prints out a message saying a new animal has been added
     2. Arguments
        1. AnimalType type
        2. Int Age
        3. Const string &name
        4. Int cost
        5. Int number of babies
        6. Int feeding cost
        7. Int payoff
     3. Local variable
        1. Animal \*new\_animal
     4. Call make animal (overloaded)
        1. Pass in type, age, name, cost, number of babies, feeding cost, and payoff
        2. Returns a pointer to new\_animal
     5. Call add to exhibit
        1. Pass in type and pointer to new\_animal
     6. Call subtract from bank
        1. Passing animal cost
           1. New\_animal.getCost()
     7. Call print animal added message
        1. Pass in pointer to new\_animal
  3. Make animal
     1. Description
        1. Creates a new animal of specified type at specified age
     2. Arguments
        1. AnimalType type
        2. Int age
     3. Local variables
        1. Animal \*new\_animal
     4. Create new animal of specified type and age
     5. Returns pointer to new\_animal
  4. Make Animal (overloaded)
     1. Description
        1. Overloaded version of make animal specifically for creating a new custom animal. Creates a new animal of specified type at specified age
     2. Arguments
        1. AnimalType type
        2. Int age
        3. Const string &name
        4. Int cost
        5. Int number of babies
        6. Int feeding cost
        7. Int payoff
     3. Local variables
        1. Animal \*new\_animal
     4. Create new custom animal at specified age
     5. Returns pointer to new\_animal
  5. Add to exhibit
     1. Description
        1. Adds new animal pointer into their exhibit array
     2. Arguments
        1. AnimalType type
        2. Const Animal \* &animal\_ptr
     3. Check if exhibit is at capacity
        1. If exhibit count == exhibit capacity
        2. Call increase exhibit
           1. Pass in AnimalType
     4. Add pointer to animal type exhibit at index (exhibit count)
        1. Animal\_exhibits[type][exhibit\_count[type].count] = animalptr
     5. Increase exhibit count by 1
        1. Exhibit\_count[type].count += 1
  6. Increase exhibit
     1. Description
        1. Increases exhibit size by a factor of start\_exhibit\_size. I.e all exhibits will be increased by a size of 10 at a time.
     2. Arguments
        1. AnimalType type
     3. Local variable
        1. Int new\_array\_size
        2. Animal \*\*new\_array
     4. Set new\_array\_size to current exhibit capacity + start\_exhibit\_size (10)
        1. Exhibit\_count[type].capacity + start\_exhibit\_size
     5. Create a new animal pointer array of size new\_array\_size
        1. Set new\_array to new Animal\*[new\_array\_size]
     6. Copy pointers of old array into new\_array
        1. Loop for exhibit\_count[type].capacity
           1. Copies pointer over to new\_array
           2. Set new\_array[index] = old\_array[index]
     7. Delete old exhibit array
        1. Delete [] animal\_exhibits[type]
     8. Set new array to exhibit array pointer
        1. Animal\_exhibits[type] = new\_array
     9. Point new\_array to nullptr
        1. New\_array = nullptr
     10. Set new exhibit capacity
         1. Exhibit\_count[type].capacity = new\_array\_size
  7. Increase age of all animals
     1. Description
        1. Increases age of all animals in the zoo
     2. Loop for animal exhibit count
        1. Loop for exhibit\_count[index (type)].count
           1. Animal\_exhibit[index (type)][index (animal count)] -> incrementAge()
  8. Feed all animals
     1. Description
        1. Feeds all the animals in the zoo. Subtracts their feeding cost from the bank account
     2. Loop for animal exhibit count
        1. Loop for exhibit\_count[index (type)].count
           1. Call subtract from bank

Pass in argument:

Animal\_exhibit[index (type)][index (animal count)] -> get feeding cost()

* 1. Calculate profits for all animals
     1. Description
        1. Loops through exhibit and adds all animal’s payoffs to the bank, including the tiger bonus
     2. Local variable
        1. Int total profit = 0
     3. Loop for animal exhibit count
        1. Loop for exhibit\_count[index (type)].count
           1. Add to total profit

Total profit += Animal\_exhibit[index (type)][index (animal count)] -> get payoff()

* + 1. Add tiger bonus
       1. Add to total profit
          1. Total profit += tiger bonus
    2. Call add to bank
       1. Pass in argument:
          1. Total profit
    3. Print today’s profits have been calculated message
       1. Pass in argument
          1. Total profit
  1. Print animal added message (const)
     1. Description
        1. Prints out a message that a new animal was added to the exhibit
     2. Arguments:
        1. Const Animal \*animal
     3. Print out message saying an animal was added to the exhibit
        1. Use menu.formatprompt
           1. “A new [animal->getName] was added to the exhibit. It is only [animal->getAge] days old”
  2. Print animals age increased (const)
     1. Description:
        1. Prints out message that all animals age increased by 1 day.
     2. Use menu.formatprompt
        1. “All animals are one day older”
  3. Print all animals fed message (const)
     1. Description:
        1. Prints out message that all animals have been fed
     2. Use menu.formatprompt
        1. “All animals have been fed!”
  4. Print today’s profits have been calculated message (const)
     1. Description:
        1. Prints out message that today’s profits have been calculated
     2. Arguments:
        1. Int profits
     3. Use menu.formatprompt
        1. “Your zoo made a killing today.. you earned $[profits] in total profit..”
  5. Buy new animal
     1. Description
        1. Prompt the user if they want to buy a new animal at the end of the day. Then create the new animal if they choose
     2. Local constant
        1. int adult age = 3
     3. Ask the user, “Do you want to buy an adult animal?”
     4. Give them yes/no option (menu.confirm)
     5. If no, return and move on with the program
     6. If yes,
        1. Print list of animal options (menu.chooseOne)
           1. Tiger
           2. Penguin
           3. Turtle
           4. New animal (Extra credit)
        2. If they choose tiger, penguin, or turtle
           1. Call add animal and pass in

Animal type

Int age = 3 (adult age local constant)

* + - 1. If they choose new animal
         1. Call buy custom animal
  1. Buy custom animal (extra credit)
     1. Description
        1. If the user has chosen to make a new animal. Prompt them for the animal’s traits then create the new custom animal.
     2. Local constant
        1. int adult age = 3
     3. prompt strings
        1. what is the name of your new animal?
        2. What is the cost of your new animal?
        3. What is the number of babies your new animal produces?
        4. What is the feeding cost of your new animal?
        5. What is the payoff of your new animal?
     4. Prompt user for inputs to create a new animal (menu.promptForInteger)
        1. String animal name (custom prompt)
        2. Int cost – range between 100 – 10000
        3. Int number of babies – range between 1 – 10
        4. Int feeding cost – range between 10-100
        5. Int payoff – range between 100 – 2000
     5. Call add animal (overloaded) and pass in
        1. Type = Custom
        2. Age = adult age (3)
        3. Name
        4. Cost
        5. Number of babies
        6. Feeding cost
        7. Payoff
  2. Prompt for feed type (extra credit)
     1. Description:
        1. Prompts user for the kind of feed type they want to use today
     2. Prompt message to user
        1. “What type of feed do you want to use today”
     3. Give them list of options (menu.chooseOne)
        1. Cheap
        2. Generic
        3. Premium
     4. Set todays feed type to user selection
        1. Todays feed type = Static\_cast<FeedType> (user\_selection – 1)
     5. Set todays food cost multiplier
        1. Call change food cost multiplier
           1. Pass in todays feed type
  3. Change food cost multiplier (extra credit)
     1. Description:
        1. Changes the food cost multiplier used to calculate total feeding cost for the day
     2. Arguments
        1. FeedType type
     3. If feed type = cheap
        1. Change multiplier to 0.5
     4. If feed type = generic
        1. Change multiplier to 1
     5. If feed type = premium
        1. Change multiplier to 2
  4. Add to bank
     1. Description:
        1. Adds amount passed as argument to the zoo’s bank account
     2. Arguments:
        1. Int amount
     3. Add amount to bank account
  5. Subtract from bank
     1. Description:
        1. Subtracts amount passed as argument to the zoo’s bank account
     2. Arguments:
        1. Int amount
     3. Subtracts amount from bank account
  6. Print bank account (const)
     1. Description:
        1. Prints bank account amount to user
     2. Print “bank account: “ bank account
  7. Is bankrupt (const)
     1. Description:
        1. Determines if the zoo is bankrupt or not. Returns a boolean true if bankrupt, false if not bankrupt yet.
     2. Return boolean
        1. bank account is less than 1
  8. check if bankrupt
     1. description:
        1. Checks if zoo is bankrupt, if so, this function ends the game.
     2. Call is bankrupt
        1. To check if zoo is out of money
     3. If yes
        1. Call game over
           1. Arguments:

day counter

ExitReason Bankrupt

* 1. Prompt to keep playing
     1. Description:
        1. Ask user if they want to continue playing or quit the game
     2. Prompt user will message
        1. “Do you want to keep playing?”
     3. Use menu.confirm (yes/no)
        1. If yes
           1. let the game continue
        2. If no
           1. Call game over

Arguments:

Day counter

ExitReason Quit

* 1. Game over
     1. Description:
        1. Tells user the game is over, their stats at the end of the game, what happened to the animals, and the reason why the game is over
     2. Arguments:
        1. Int day counter
        2. ExitReason reason
     3. Print message “game over” and stats
        1. Number of days opened (day counter)
        2. Tell them number of animals
           1. Print exhibit count
        3. Ending balance
           1. Print bank account
        4. “All the animals were released and went on a rampage in the city”
        5. Reason:
           1. If reason == Quit:

Print “Because you gave up”

* + - * 1. If reason == Bankrupt:

Print “Because you went bankrupt”

* + 1. End the game (set exit status to true)
  1. Print exhibit count (const)
     1. Description:
        1. Prints the current status of each exhibit, the count of the animals and the exhibit capacity
     2. Print
        1. “Tiger exhibit:”
           1. “Capacity: “ capacity
           2. “Count: “ count
        2. “Penguin exhibit:”
           1. “Capacity: “ capacity
           2. “Count: “ count
        3. “Turtle exhibit:”
           1. “Capacity: “ capacity
           2. “Count: “ count
        4. “New animals exhibit:”
           1. “Capacity: “ capacity
           2. “Count: “ count
  2. Write\_read (const) (extra credit)
     1. Description
        1. Formats an event message string
        2. Writes event to file
        3. Reads event from file and prints to screen
     2. Arguments:
        1. EventType event
        2. Animal \* animal
           1. Default = nullptr
        3. Int todays\_bonus
           1. Default = 0
     3. Use a switch to check event type to create an output message
        1. Event = Nothing
           1. “Today.. nothing happened at the zoo..”
        2. Event = Boom
           1. “Today.. a boom in zoo attendance occurred.. you earned $ [todays\_bonus] per Tiger for a total of $ [daily bonus profit] extra profit.. people really love tigers.. :v ”
        3. Event = Birth
           1. “Today.. a(n) [animal->getName] had [animal->getNumBabies] babies..”
        4. Event = Sickness
           1. “Today.. a(n) [animal->getName] died from illness at the age of [animal->getAge].. the zoo mourns its loss..“
     4. Call write message
        1. Pass in string message
     5. Call read message
  3. Write\_message (const) (extra credit)
     1. Description:
        1. Writes a message of a random event to an output file
     2. Arguments:
        1. Const string & message
     3. Open output file
     4. Write string to file
     5. Close output file
  4. Read\_message (const) (extra credit)
     1. Description:
        1. Reads a message from random event file and prints it to screen
     2. Open random event file for reading
     3. Read into a string variable
     4. Print string output using menu.formatPrompt
     5. Close file