Ahmad El-Bobou

162

Fall 2018

Lab 3 Design and Reflection

To create the program, I'm planning on handling everything in the main function through the game class. The game class will store all the necessary variables for the game in member variables. The game class will have 5 member functions aside from the default constructor: 1. A menu to see if the player wants to play, 2. A function to get the required variable information, 3. A function that takes dynamically allocated arrays and fills them with the rolls for each player, 4. a game loop that iterates through the rounds and displays each round and final score, and 5. A destructor that frees the memory. My plan for instantiating objects according to user input is to instantiate them in the scope of the game class function that gets the rolls with an if - else if statement. This will ensure I don't needlessly create 4 objects to account for all possible user inputs. In order to store the rolls from the short lived die objects, I will input those into dynamically allocated arrays that the game loop will access.

During implementation, I ended up choosing to dynamically allocate the objects. I did this in order to avoid creating four objects when only 2 were needed. In order to reuse the pointer, I copied the information from the rolls to an array and then deleted pointer and set it to nullptr. This happens in the getRolls() function in the Game class. I ended up using the input validation files from project 1 but changed the functions to be able to reused in more programs. I create a function that checks for an int above a minimum and another function that checks for an int above a minimum and below a maximum.

While creating this program, I had a bit of difficulty deciding how I wanted to use die objects in the Game class. I ended up using them as I explained in the above paragraph. I'm still

not quite sure this was the best way to do it. I wanted to save on memory space by not creating objects that wouldn't be used. I'd love to see a comment from the TA about what you would have done if you have the time. Otherwise this lab was a fun exercise.

Test Cases	Driver Function	Input	Expected Outcome	Observed Outcome
Ensure regular die class produces random numbers		Created a for loop that iterated 100 times. Executed this code 5 times.	Random list of 100 numbers between 1 and # of sides selected each time code is executed.	Random list of 100 numbers between 1 and # of sides selected each time code is executed.
Ensure loaded die class produces random numbers		Created a for loop that iterated 100 times. Executed this code 5 times.	Random list of 100 numbers between 1 and # of sides selected each time code is executed.	Random list of 100 numbers between 1 and # of sides selected each time code is executed.
Ensure loaded die rolls higher on average		Created a for loop that rolled the regular die and loaded die 100 times and took the average of the rolls.	Loaded die will almost always outroll the regular die.	Loaded die ALWAYS outrolled the regular. (There is a possibility this might not happen, but it is VERY unlikely.)
Ensure loaded and regular dice don't roll higher than number of sides.		Created dice of both type with sides of 5, 100, 500.	No roll will be over 5, 100, or 500.	No roll was over 5, 100, or 500 for their respective die.
Input validate beginMenu		1. Y 2. y 3. N 4. n 5. 1 2 t a n 6. l R 0.0 2 e y	1. Game runs. 2. Game runs. 3. Game quits. 4. Game quits. 5. Prompt to reenter 4 times then quits. 6. Prompts to reenter 5 times then runs.	1. Game runs. 2. Game runs. 3. Game quits. 4. Game quits. 5. Prompt to reenter 4 times then quits. 6. Prompts to reenter 5 times then runs.

Test - isIntAboveX(x) function		X = 0 1. -1 1 2. 5 3. g 0 3 4. 4.4 f 2 5. 4d 40	1. Prompts to reenter then accepts 2. Accepts 3. Prompts to reenter 2 times then accepts. 4. Prompts to reenter 2 times then accepts 5. Prompts to reenter times then accepts	1. Prompts to reenter then accepts 2. Accepts 3. Prompts to reenter 2 times then accepts. 4. Prompts to reenter 2 times then accepts 5. Prompts to reenter times then accepts
Ensure the game loop iterates the rolls for the number of rounds		1. rounds = 1 2. rounds = 50 3. rounds = 200 4. rounds = 10000	1. iterates 1 time 2. iterates 50 times 3. iterates 200 times 4. iterates 10000 times.	1. iterates 1 time 2. iterates 50 times 3. iterates 200 times 4. iterates 10000 times.
Ensure destructor runs	Temporarily include if statement that checks if the pointers are nullptrs in the destructor and couts a msg	n/a	Ptrs are nullptrs as reported by msg	ptrs are nullptrs as reported by msg
Test combo of loaded, unloaded, and varying sides 1		Loaded and Unloaded - different sides 1. y 2. 20 3. 1 4. 0 5. 5	Results 1. Game runs 2. 20 rounds are played 3. P1 is loaded 4. P2 is regular 5. P1 has 5 sides	Results 1. Game runs 2. 20 rounds are played 3. P1 is loaded 4. P2 is regular 5. P1 has 5 sides

		6. 10	6. P2 has 10 sides	6. P2 has 10 sides
Test combo of loaded, unloaded, and varying sides 2		Loaded and Unloaded - different sides 1. y 2. 10 3. 0 4. 1 5. 6 6. 3	Results 1. Game runs 2. 10 rounds are played 3. P1 is regular 4. P2 is loaded 5. P1 has 6 sides 6. P2 has 3 sides	Results 1. Game runs 2. 10 rounds are played 3. P1 is regular 4. P2 is loaded 5. P1 has 6 sides 6. P2 has 3 sides
Test both loaded and varying sides		Both Loaded - different sides 1. y 2. 10 3. 1 4. 1 5. 6 6. 3	Results 1. Game runs 2. 10 rounds are played 3. P1 is loaded 4. P2 is loaded 5. P1 has 6 sides 6. P2 has 3 sides	Results 1. Game runs 2. 10 rounds are played 3. P1 is loaded 4. P2 is loaded 5. P1 has 6 sides 6. P2 has 3 sides
Test both regular and varying sides		Both Loaded - different sides 1. y 2. 10 3. 0 4. 0 5. 6 6. 3	Results 1. Game runs 2. 10 rounds are played 3. P1 is regular 4. P2 is regular 5. P1 has 6 sides 6. P2 has 3 sides	Results 1. Game runs 2. 10 rounds are played 3. P1 is regular 4. P2 is regular 5. P1 has 6 sides 6. P2 has 3 sides
Test both loaded and same sides		Both Loaded - same sides 1. y 2. 10 3. 1	Results 1. Game runs 2. 10 rounds are played 3. P1 is loaded	Results 1. Game runs 2. 10 rounds are played 3. P1 is loaded

		4. 1 5. 6 6. 6	4. P2 is loaded 5. P1 has 6 sides 6. P2 has 6 sides	4. P2 is loaded 5. P1 has 6 sides 6. P2 has 6 sides
Test both regular and same sides		Both Loaded - same sides 1. y 2. 10 3. 0 4. 0 5. 6 6. 6	Results 1. Game runs 2. 10 rounds are played 3. P1 is regular 4. P2 is regular 5. P1 has 6 sides 6. P2 has 6 sides	Results 1. Game runs 2. 10 rounds are played 3. P1 is regular 4. P2 is regular 5. P1 has 6 sides 6. P2 has 6 sides