James Ng Homework 2 Question 3 Group 7

The objective is to test if an array-based stack or a linked list-based stack is faster when performing millions of operations. Since I was using the code from Blackboard, I had to create a stack abstract data type header file for the code to work properly. I just set the values 0 as a base case for the pure virtual functions. After creating the stackADT.h, I did the same with the linked list code from Blackboard, as the array-based stack, and made the linked list abstract data type. Afterwards, I had to make the function that performed operations using the stack from both array based-stack and linked list-based stack. So arrayOps and linkedOps were made to test the timing on both stacks. For 123,456,789 times, both functions perform operations and are timed to compare the speed. In these functions, they perform push, top and pop operations. After they are done performing the operations, the time it took is returned and we get the timing. Every time I have ran this program, the array-based stack was significantly faster, as it ranges from 8,300 to 8,500 milliseconds. Meanwhile, the linked list-based stack was ranging from 28,000 to 30,000 milliseconds. In the main function, there were variables with integer data types created for access to the stacks. I created auto variables to store the values of the timing. Afterwards, I compared the two values to see which is faster and it prints the appropriate response.

James Ng Homework 2 Question 7 Group 7

The objective is to solve a maze using the stack. Referenced from cplusplus.com, I used some ideas and created a structure that would have access to the row and column for the whole code. I made a default constructor, the row and column variables and declared the starting point of the maze from there. Afterwards, I needed a way to show the maze, so the print function was made. I included the maze dimensions and used for loops to print each layer or line of the maze. In the main function, there is the base maze where nothing has happened yet. The maze is ten by eleven to give enough space to demonstrate backtracking and it is categorized as a char so each individual character can be read. The maze is meant to be shown as a before and after format, so the base maze is shown first, then the solved maze. In some way we had to access the stack and keep track of our position, so variables were made to specifically keep track of the whole maze using the stack. I used a while loop that would keep going no matter what, so that we may go through the entire maze. First, an if statement was made to traverse the maze going up, so it checks the row above it so see if it is empty. If it is empty, it will add the position to the stack, add an upside-down exclamation point and subtracts the row from the position. Then it will continue so that it'll go to the next if statement, which will check the right. It checks the column instead. Same goes for checking down and left, afterwards it'll check if there is an 'E' which is the endpoint of the maze. It'll print the maze if it reached the end of the maze and end the program if it is done. Now for the backtracking part, it checks if everywhere else has something filling the space and mark it with its own character 'B'. Then the while loop repeats itself and it will continue traversing the maze until the maze is solved.

James Ng Homework 2 Question 9 Group 7

The objective is to show an airport that has seventy percent of planes going in one direction, twenty percent of planes going in one out of two directions and will always choose whichever has the shorter line and ten percent of planes going one out of three directions and will always choose the shorter line. First, I needed to get the plane percentages, so I ask the user for how many planes they would like, so the number would be stored in a variable called planes. Afterwards, I designated one of the four directions, naming them south, as the direction that seventy percent of planes are going. So, I created a function printSouth so that it would create the runway for south. In the function, I created an integer vector array and used a for loop so that it would only repeat depending on the number of planes. I also made an integer variable that would round the number of planes down using the floor function , so the variable would not become a decimal since we cannot a plane and a half of a plane. It would add the number to the vector and print the runway with the plane number. I had a bit of trouble here as I tried making the integer I in the for loop begin with 1 so that we wouldn't have a plane 0 showing. The program would crash and show an error. I fixed this issue by simply adding 1 into the push_back. Now for the east and west directions. I printed these directions similarly to how south was printed, however I needed to show the runway so I created separate for loops in order to print a horizontal runway so that it would look cleaner. The for loop used the function floor with the number of planes multiplied by twenty percent or 0.2. I first printed the horizontal lines, then the actual planes and finally closing off the runway with horizontal lines once again. I had to print the entire runway, so I created a function called printRunway to print all of the runways onto one screen. So, I used the code from each individual runway and modified it so that the overall image would look neater by adding extra spaces and symbols to indicate what directions it was going.