

CMPSC 462: Data Structures and Algorithms (Fall 2021)

**Project-1**

Stack, Queue and Deque Applications

Hansi Seitaj

**Dr. Vinayak Elangovan**

Submitted On: 09/22/2021

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| 1 | INTRODUCTION | 3 |
| 2 | DESIGN & IMPLEMENTATION | 3 - 4 |
| 3 | RESULTS / SAMPLE OUTPUTS | 5 - 7 |
| 4 | CONCLUSION | 7 - 8 |
|  |  |  |

1. **INTRODUCTION**

In this first part of the project, I am working with data structures as Stacks, Queues and Deques. Stacks are containers of objects that are pushed (inserted) and popped (removed) according to the last-in first-out (LIFO) rule. Queues are containers of object that are pushed (inserted) and popped (removed) according to the first-in first-out (FIFO) rule. Finally, Deques are known as a double-ended queue, are an order of collection of items very similar to the queue. In addition, you can have elements be pushed (inserted) and popped (removed) according to the front and to a rear end.

In the first part of the project, you will find the implementation of three classes:

1. Stack class
2. Qu (Queue) class
3. Deque class

In the second part of the project (Part-2: Applications) there are displayed 3 different applications by implementing the data structures mention above.

1. Tower of Hanoi
2. Cars lined up at a car wash
3. Using deque for a train functionality
4. **DESIGN & IMPLEMENTATION**

In the first part of the project there is a simple implementation and construct of the three classes mentioned above and there is no specific algorithm to go in details.

In the second part of the project there are three main algorithms used for each part:

1. Tower of Hanoi

Text

Description automatically generated

In a few words, there is a method named ‘ShowStimulation’ that accepts three stack objects and displays to the user the rods content and the disk placement using asterisks. Each disk size is label as: 1 – small disk, 2 – medium disk, 3 – large disk. Due to the 9 spots needed to be displayed (3 per each disk) there is a variable created multiplied by the numbers that represent the disks size. In the end the algorithm accepts the three stacks it can use to play the game and displays the visualization of game and the content of the rods.

1. Cars lined up for a car wash

Text

Description automatically generated

In this case there is a method named ‘carWash’ that accepts a queue object with the cars lined in it. The method display a visual representation of each car going into the car wash process and jumping into the next one until the queue is empty and there is no car in the line.

1. Using deque for train functionality

Text

Description automatically generated

By implementing the class of Deque, I can show deque abstract data structure mechanism. Since deque data structures are known for the speed they can process entries the train, electric train, bus, are some examples that closely show the efficiency of using deque data structure. Moreover, in this case a train object is created and their people entering the train from two ends (rear and front) and sometimes later leaving the two fronts. In this way, there is a conceptualization of the train using deque.

1. **RESULTS / SAMPLE OUTPUTS**

There are a few tests I have implemented to check the application durability and efficiency for each case in part two of the project.

1. Tower of Hanoi

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

In this application there is a low risk taken in consideration because the user only needs to run the program and see the construction of the code commented for each step and see the result. Each step is shown to the user and the content of each rod is also displayed for less confusion and more transparency.

1. Cars lined up for a car wash

Text

Description automatically generated Text

Description automatically generated

This application allows the user to enqueue any type, year, make, model car and the program will show the car washing process for each one at a time (FIFO).

1. Using deque for train functionality

Text

Description automatically generated

In this application, the mechanism is very simple to understand. A certain number of people enter the front and the rear end of the train. In one case, just two of them leave, the program is able to see that the train is not empty and continue the train schedule. In case when there is not a number of people in the train the program displayes that the train is empty and waits for passengers. There is not an actuall big number of people that can be inserted in this case since the train is of a deque form and can accept a large amount of inputs. In other words, it is a small application to show the mechanism of the train ( or a bus which is simmilar in terms of mechanical point of view).

1. **CONCLUSION**

* What have you learnt from this project?

To start, by working on this project I was able to learn the construction of the data structures: Stack, Queue, Deque. Also, creating several applications in regard to these data structures helped me improve my ability to program and up level my understanding of their functionality. Furthermore, now I hold more experience in visualization of the application and got experience in programming close to a user-friendly visualization. Finally, I learnt that these data structures are widely used in the real world of software and there are so many real-life representations that we can use for a better understanding.

* What ways you can expand this project?

I think that the project serves its purpose, and it is very informative and effective. However, if you think in terms of expanding for a wider understanding, there are two ways you can apply that. The first way is that the project can have a wider variety of options to choose for each application in part 2. For instance, instead of just the tower of Hanoi, there can be other mini games that could be presented as an option. Another way to expand the project is to already give to the students 60-80% of the code for each application and let the student work only on 1 to 2 algorithms per application (total 6 algorithms). Furthermore, since you are including the idea and more then half of the code per application, give to the student the test cases they should be able to fulfill in order for the algorithm to be as accurate as they can be. In this way they can present their unique approaches to the applications; present them; discuss the expected results.