# Bus Simulation

Ali Helmi

# TECHNICAL ASSESSMENT REPORT

REVISION – Draft

November 17, 202

# Table of Contents

# 1. Initial Stage

## *1.1. First Impression*

When I first read the description of the exercise, I had lots of ambiguities on what quantities to define and what others to measure. Then I realized that I could make all the parameters I was thinking of hard-coding into actual variables.

### 1.1.1. Questions

- What is the maximum capacity for each bus (how many persons it can carry at a time)?
- What is the route of the bus after its final stop? Does it trace its path back the stops? or does it circle around going from the last destination to the beginning first stop?
- What are the destinations of the passengers?

### 1.1.2. Solutions

My solution for the first question was making the bus capacity as an input, since we can't swipe through the values because we are swiping through the maximum and minimum number of queue of passengers at each stop. For us to able to simulate those variables we needed to fix the bus capacity, and rather than hard coding the number I chose to make it determined by the user. As for the second question, I just made the code versatile to be changed at any point in time. I had the program assume that the route for the buses is to circle around after the last stop and go back to the first stop directly.

As for the last question, I assumed they were random, which raised more questions such as how far and how many stops can it be? Are the passengers allowed to pick the same spot? That means that the passengers would be dropped off at the pickup stop, but at the same time it means they would take an entire trip around the route or two and so on. So, in order to prevent all that I just made the randomizer picks a bus stop from the current trip and avoid the current stop.

I usually break up every task to smaller ones and make functions out of those smaller ones in order to create building blocks I could utilize and shape to where the project takes me. I also tend to feel inclined to automate as many tasks as possible in order to avoid tedious work.

# 2. Coding Stage

## *2.1. Train of thoughts*

Initially I only made the buses as elements of hash maps with simple types, however, I later realized that there was much more information of which I needed to keep track. I decided to go with classes instead and creating a class that contains all the important information while also flexible enough to be manipulated in necessary ways. So, I created all the required functions inside the class and kept adding as needed. Such approach allowed me to have multiple unique instances of such class and each containing information about its respective bus.

As for the bus schedule, I had two ideas which essentially boiled down to strict and non-strict schedules. Examples of which are represented in a real-life example of subways/metro and public transportation buses. Subways do not wait for passengers to board; thus, they have strict schedule. While on the other hand, buses do not leave until every passenger waiting in the bus-stop has boarded. This difference plays a huge factor in determining the arrival time of every transportation entity used as means of transportation. If a bus got delayed in one stop it could affect the entire network leading to a snowballing effect. However, in case of empty stops when no passengers are waiting, subways are forced to wait regardless while buses can use that time to catch up to their original timeline.

Concluding from the fact that this assessment used buses instead of simulation means that the idea to keep the schedule flexible. I could also take advantage of this by relaying the real-time whereabout of each bus to its adjacent buses in order to have more coherent network.

### 2.1.1. Simulation Approach

When I first started coding this assessment, I had two ideas for simulation, one that utilizes threads and the other utilizes a global timer. The idea behind the global timer approach is having one variable that keeps track of time the entire simulation, and after every predetermined periods certain actions happen and only when their conditions are met in a timely manner. This approach required all functions to be redesigned into executing one action at a time rather than a holistic action on all the quantities. Another takeaway I got from the description was the unique specification of five buses and ten bus stops which means double the amount of buses, and that meant for the starting point of each bus is to have a vacancy between its adjacent buses to be left empty for initial movement.

Plotting outputs took the shape of a graph with axis of time (per minute) vs. Stops and five lines drawn on this graph each representing a bus location at certain point in time. I also had an idea of running ten simulations and averaging the result in one single simulation run to get a better and more accurate results. This idea can be easily accomplished by adjusting the timing rations while maintain the original parameters.

### 2.1.2. Other Possible Approaches

The idea behind the threaded system is that every bus is represented in a thread with their own time being measured locally. This approach requires all functions to have loops and execute all actions at once, for example: when boarding's function is executed it boards all passengers with a single call. The delay being inside every iteration of that function allows the bus to wait appropriately and realizing a more realistic environment. This approach also provides the feature of threads "buses" talking to each other and getting information on each other's whereabouts, and waiting accordingly.

## *2.2. Difficulties*

The main difficulty I would say is available time since my time is already occupied with my college senior-level classes. Or perhaps I might have overcomplicated this problem, which is the more probable cause.

# 3. Requirements

## *3.1. System Definition*

For the system to work, multiple standard libraries are required, and thus were included in the script. The definition of my system is represented in hierarchy of organized files each has its own distinguishable purpose. The first file being the libraires header file which houses all the “include” calls, in addition to my own generated namespace library which consists of helper functions. The second header file includes all the initialization and setting parameters at starting point assuring consistent results every time the simulation is launched. The third header file has all the function’s definitions and declarations of the functions used in simulation.

## *3.2. Characteristics*

As aforementioned, this project consists of a number of source files and header files as demonstrated in the Github repository. Each file’s content is explained as follows:

### 3.2.1. libraries.h

This header file contains all libraries which are required and included in the project. The libraires are included in this file so it can utilize header guard feature and have all of them organized in one place to be used throughout the project. The other benefit of this header is that it also contains my own customizable library and namespace, tailored for this project specifically, to be used throughout the project as well. The namespace I created provides many different helper functions, mainly the print functions which are used for debugging.

Multiple definitions of the same function name allows me to create functions for different types and purposes, and can be called with single function name for all that functionalities to be used. The recursive function is one of the variations adds another capability which is allowing the function to accept a variable number of arguments that are not pre-determined.

### 3.2.2. initial_quantities.h

This header file is fairly simple, and its sole purpose is to lay the groundwork for the project and act as a foundation initiating quantities to be used and manipulated during the simulation. Quantities such as structures and classes that need to be created beforehand in order to be used.

### 3.2.3. bus_simulation_functions.h

The most important header file of all as it consist of all the functions that make up the backbone of this project. As explained, I prefer to break up each step to this project into smaller tasks and those smaller tasks are represented in functions included in this header file. Every function has a specific unique purpose while also allows flexible use. Initially functions used to fulfill tasks holistically in one execution, but then were retailored to only fulfill one action at a time and leave the iteration for when the function is called.

### 3.2.4. bus_simulation.cpp

The plan was for this source file to be the main source file and present options board for the user to choose from and input certain quantities. The user would be asked to choose between choices of simulations (threaded or global timer), and then enter few necessary quantities since there are too many unknowns to base a simulation and output results. However, time frame was an issue that I could not overcome, so I kept the structure of the other source code files which were intended to host types of simulations in case this project is completed in the future. As of this time on bus_simulation.cpp is used, and it is used as both the main file and the host of simulation. The simulation designed in this file is way simpler as intended and merely used as demonstration of the idea. The file does not ask the user for anything except for bus size since it was the most important quantity to tie all other variables together.

# 4. Common Practices and Conventions

## *4.1. Approaches*

As already established, my common practices when approaching projects or problems are represented in two main categories which are building blocks and automation. Building blocks approach is essentially constructing many functions each with specific purpose to be used freely to build said project and reshape as the project progresses resembling Lego structures metaphorically. Automation approach is intuitively explained as it sounds which fucoses on eliminating tedious repetitions. Automation is shown in many incidents throughout my code as well as building blocks.

## *4.2. Naming*

As far as naming is concerned, I usually prefer to stay consistent throughout all files that make a certain project. I am also flexible if the project is collaborative as long as all collaborators stay consistent as well. My personal naming system is simple and consists of two protocols: lowercase_letters and UppercaseLetters. As shown in the examples those are the two styles used everywhere in naming everything from variables to files. Functions usually get the upper case technique and variables get the lower case, there are few special rules as well such as including an extra underscore before the name if the variable is meant to be private.

## *4.3. Header Files*

As implied from the previous sections, before I start working in projects I use Visual Studio's feature that is filtering system and I create filter to organize my files. I create many files each with its own respective usage, and using this method had always allowed me to navigate through the aspects of my work easier and fined what I am looking for since all files are organized. This method also minimizes redundancy and redefinitions.

### 4.3.1. Helper Functions

One of those files always holds my helper functions for that particular project, and almost every time I must have printing helper functions. I use printing in debugging all the time, and I place them before and after the lines I am suspecting to be the cause of the problem. For example I place one print before and after a loop call and see if the program managed to enter said loop.

### 4.3.2. Namespaces

I sometimes create my own namespace, but mostly I use C++'s standard library (std). However, I try to avoid having "using namespace std;" in my code. Although using namespace std makes using the library easier but carries risks with the convenience which makes said convenience not that appealing. The main reasons I avoid using namespace std are to avoid confusions between other namespaces and to avoid conflicts with other similar definitions. As projects scale up, the use of namespaces becomes more and more crowded, and I think keeping each namespace attached to their respective declarations is a lot safer and neater. More so when there are multiple collaborators one could create a class, structure, or a function with a similar to one of those of a particular library or namespace and that could lead to all sorts of bugs and errors. One more issue that occurred to me personally at some point is including "using namespace std;" in multiple files and that broke down the whole thing which lead me to start avoiding it all together.

# 5. Results and Conclusion

As of this stage, the program works properly and as intended. It also shows the potential to out appropriate results. All files, functions, and libraries were built and compiled successfully. I tested every single function using my debugging method I mentioned earlier, and I even tested various combinations between those functions which all succeeded. I validated the entire system and the simulation functionality. The "building blocks" functions proved to work in conjunction with each other flawlessly, and even time complexity is taken into account when building every part of this project. However, as aforementioned a proper host needs to be placed and make use of those finished products. Essentially, the same idea I used for demonstrating the simulation approach where functions are used and called in an organized coherent manner. As of now the bus simulation is only capable of moving the bus once at the beginning of the simulation. My diagnosis is that somewhere in the simulation area the readiness status for the bus to move is not changed correctly which is an incredibly easy fix. The program is already capable of outputting the results properly to a csv file where it can be plotted. The data is structured to output each bus's bus stop location and the time stamp at that point in time. Every bus movement is registered with location and time in respect to the global timer.

The distances between buses are not kept uniform because of the randomness of number of passengers at every bus stop causing buses to be delayed more than others and that effect ends up snowballing affecting adjacent buses. There are solutions for this including but limited to having a strict schedule as I explained above like we see in metros/subways.

Since there are other unknowns that prevent from sweeping through the number of people at each queue, and depending on the approach I used of randomizing passengers at every bus stop, I conclude that maximum number of people is limited by the bus size and the minimum number is as low as zero.

The periodic table is generated by the program itself and every entry is a new row. Row entry consists of each bus position and timestamp when that entry was recorded.

All pre-defined parameters are set in the project as indicated in the manual, and unknown quantities were set as variables which caused the program to be adjusted accordingly.

## *5.1. Suggestions*

The groundwork is set for this project, and what is left is to make use of the functions and place them in proper places to realize the final artistic puzzle. I laid the foundation for two types of simulations; however, they are not the sole option to go for. My functions are flexible to be used in any sort of situation. The output is grouped in a vector of vector of pair of points which then is written into csv file once the simulation is done.

# Appendix A: Examples (Other Possible Approaches)

```
int BusLocationFinder(std::unordered_map<char, int> bus_locations, char target_bus) {
	std::unordered_map<char, int>::const_iterator stop_finder = bus_locations.find(target_bus);
	if (stop_finder == bus_locations.end()) {
		return 0;
	}
	return stop_finder->second;
}

char OccupiedBy (std::unordered_map<char, int> bus_locations, int target_stop) {
	std::unordered_map<char, int>::iterator bus_finder = bus_locations.begin();

	while (bus_finder != bus_locations.end()) {

		if ((bus_finder->second == target_stop)) {
			return bus_finder->first;
		}
		bus_finder++;
	}
	return 'Z';
}
void PassengerDestination(std::unordered_map<int, std::vector<int>>& passenger_destination, int passengers_limit, std::vector<int> stops, int current_stop) {
	int waiting_passengers = PassengerRandomizer(passengers_limit);
	Print("Waiting: ", waiting_passengers);
	std::vector<int>::iterator stops_iterator = stops.begin();
	int stops_index = 0;
	while (stops_iterator != stops.end()) {

		if (*stops_iterator == current_stop) {
```