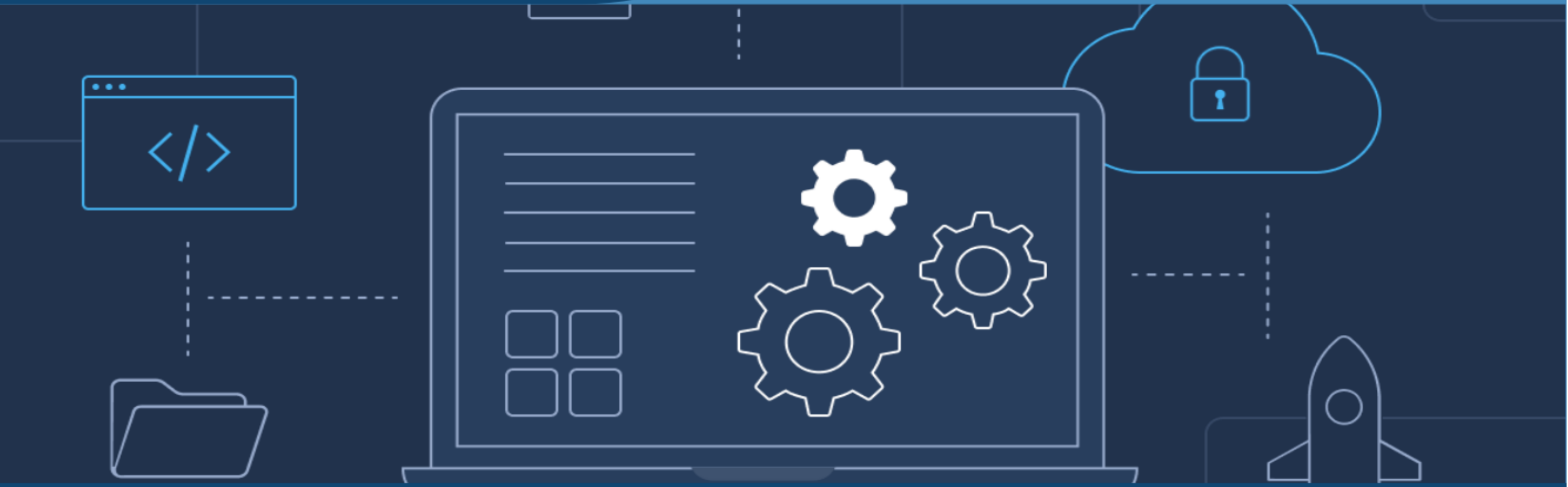
SOFTWARE ENGINEERING MANAGEMENT AND DEVELOPMENT

Middlesex University Mauritius

BUS TRACKING AND ARRIVAL TIME PREDICTION SYSTEM



STUDENT NAME:	STUDENT NUMBER:	ROLE:
IMAAN HAYFAA KUREEMBOKUS	M00952402	SCRUM MASTER
RAZEEN MODAYKHAN	M00952403	SECRETARY
CHIMARAOKE MBATA	M00909998	DEVELOPER 1
LISA GRACE EASTON	M00934573	TESTER
EDWIN ABDON SHAYO	M00906834	DEVELOPER 2

TUTOR: MR ADITYA SANTOKHEE

DATE: 21ST APRIL 2024

Table of Contents

Contents

Table of Figures

Introduction:

This report presents an overview of the Bus tracking system, a platform designed in C++ with the use of algorithms and data structures to streamline the monitoring of and management of buses from a vast CSV file of buses. With the CSV file containing over one thousand buses, the system allows the user to use features such as monitoring different buses available and displaying their departure and arrival times, their locations and add and delete buses from the CSV file.

The Bus tracking system works on user inputs which directly works with the CSV file. Updates to the CSV file from functionalities such as: add buses and delete buses directly changes the CSV file to save the inputs from the user.

Project Description:

The different functionalities associated with the software are:

1. Add Bus: When this option is entered, the user is prompted to enter different details about the bus, like the 'Bus Number', 'Bus Driver's Name', 'Bus Arrival Time', 'Bus Departure', starting point and finishing point.
   Upon entering all the details, the data is written onto the CSV file.

2. Delete Bus: When entering this option, the user will be prompted to input the bus number that they wish to delete the record of. Upon finding the record from the hash table, the data is accordingly deleted from the CSV file.

3. Search Bus: This option allows the user to enter the 'Bus Number' assigned to every bus which is a unique identifier. The user is then able to view all the details associated with that bus.

4. View Bus Availability: The user can choose this option to display all buses available from the CSV file. The function uses the hash table data structure to store each bus information accordingly from the CSV file then outputs the data onto the terminal. The user is also able to filter the buses based on their Bus Numbers, Arrival and Departure time.

Report Layout:

This report offers an in-depth overview of the Bus tracking system, encompassing its project description, functionalities, design, algorithms used and testing implementations. These will provide insights into the methodologies used in this project's development process and performance evaluation.

Design:

The diagram below represents the simplified UML class diagram that shows the structure of the ‘Bus’ class and its relationship with its member functions. The ‘bus.h’ header file contains constants, class definitions for ‘Bus’ and the functions that handle the program.
The ‘Bus’ class represents a bus object with attributes like bus number, departure time, driver name, current station, from location, to location, and ETA.

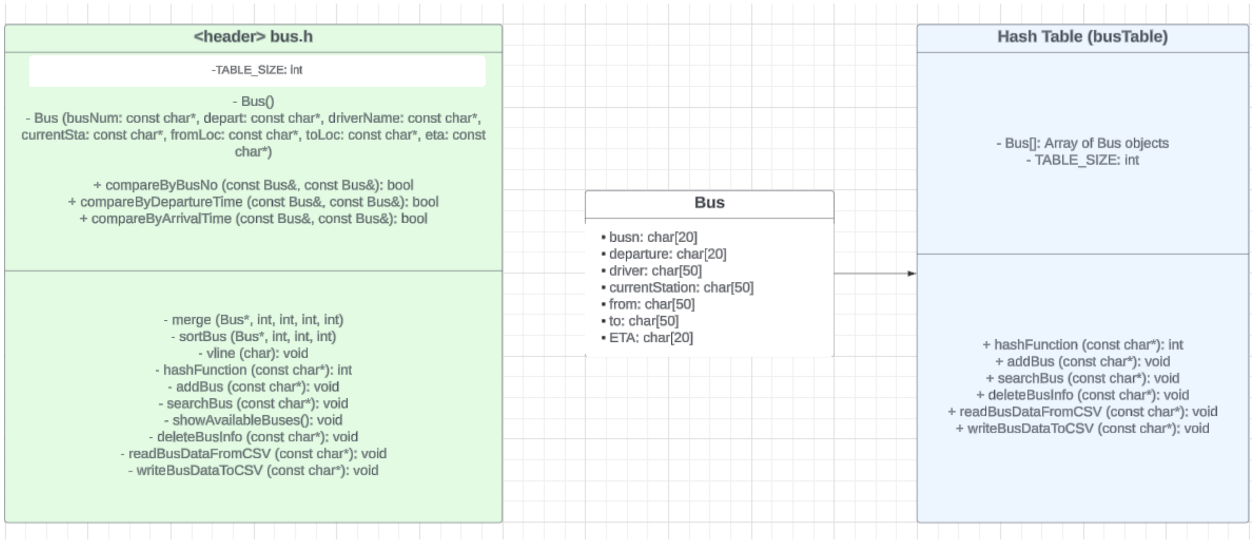

<header> bus.h
-TABLE_SIZE: int
- Bus()
- Bus (busNum: const char*, depart: const char*, driverName: const char*, currentSta: const char*, fromLoc: const char*, toLoc: const char*, eta: const char*)
+ compareByBusNo (const Bus&, const Bus&): bool
+ compareByDepartureTime (const Bus&, const Bus&): bool
+ compareByArrivalTime (const Bus&, const Bus&): bool
- merge (Bus*, int, int, int, int)
- sortBus (Bus*, int, int, int)
- vline (char): void
- hashFunction (const char*): int
- addBus (const char*): void
- searchBus (const char*): void
- showAvailableBuses(): void
- deleteBusInfo (const char*): void
- readBusDataFromCSV (const char*): void
- writeBusDataToCSV (const char*): void
Bus
• busn: char[20]
• departure: char[20]
• driver: char[50]
• currentStation: char[50]
• from: char[50]
• to: char[50]
• ETA: char[20]
Hash Table (busTable)
- Bus[]: Array of Bus objects
- TABLE_SIZE: int
+ hashFunction (const char*): int
+ addBus (const char*): void
+ searchBus (const char*): void
+ deleteBusInfo (const char*): void
+ readBusDataFromCSV (const char*): void
+ writeBusDataToCSV (const char*): void

Figure 1- UML Class Diagram (generated on Lucid Chart)

Pseudocode:

The pseudocode below outlines the implementation of the Bus Tracking system. The different operations associated with the system are adding, deleting, searching information and sorting buses based on different criteria. The system utilizes a hash table for efficient data storage and retrieval.

Below are the explanations for the core functions of the program:

1. hashFunction(busNumber):
   This function calculates the hash index (int) for a given bus number (string) using a hashing algorithm.

2. addBus(busNumber, departure, driver, currentStation, from, to, ETA):
   This function is responsible for adding a new bus to the system by calculating its hash index and storing it in the hash table.

3. searchBus(busNumber):
   Searches for a bus in the hash table based on its bus number and returns the bus object if found, otherwise returns a message indicating that the bus is not found.

4. deleteBus(busNumber):
   Deletes a bus from hash table based on its bus number by setting the corresponding hash table entry to empty.

5. sortBuses(sortBy):
   Collects all the buses from the hash table, sorts them based on the specified criteria (BusNo, DepartureTime, ArrivalTime), and returns a sorted list.

Data Structure:

The reason as to why a Hash Table has been chosen for this program is due to the fast access to the elements (bus records). The data can easily be manipulated using its member functions and the CSV file is easily updated based on the changes made by the user.

Using a Hash function, the program calculates the index in the hash table where a bus object should be stored or retrieved. This direct access based on indexes caters for efficient data retrieval.

Moreover, Hash tables have a higher adaptability in search operations when the key is known. Since hash tables offer a constant-time complexity O (1) for search operations, regardless of the number of elements stored, the time taken to find an element is nearly constant, making searches very efficient.

The sorting mechanism operates on an array of elements extracted from the hash table whereby the information in the original hash table remains unchanged. The array is sorted using the ‘sortBus’ function which implements the Merge Sort algorithm.

[Jayashree Domala, 29th May 2022]

Algorithm:

1. Add Function:
   The function begins by prompting the user to input details about the new bus, such as departure time, driver’s name, current station, destination, and ETA. Once the user inputs the details, a ‘Bus’ object is created which represents a new bus record. The ‘hashfunction’ is called to determine the index at which the record should be placed in the hash table. The ‘writeBusDataToCSV’ function writes all bus records from the hash table to the CSV file.

2. Delete Function:
   After entering the ‘bus number’ as input, the ‘hashFunction’ calculates the hash index for the bus record in the hash table. If a bus record exists at the calculated index, it is effectively deleted from the hash table. Moreover, the ‘writeBusDataToCSV’ ensures that the deletion is also reflected on the CSV file.

3. Merge Sort Algorithm:
   The Merge Sort algorithm is one of the most efficient sorting algorithms. It is based on the divide-and-conquer strategy. Merge Sort continuously cuts down a list into multiple sub lists until each has only one item, then merges those sub lists into a sorted list.
   [Vaibhav Khandelwal, 25th October 2023]

   In this case, the Merge Sort algorithm is applied on the 'buses' array which represents a collection of 'Bus' objects that are sorted based on different criteria during the sorting progress. The 'Bus' objects are passed to the 'sortBus' function which applies the merge sort algorithm on the array.

   The different filters available the sorting process involves:
   1. Sorting by Bus Number (unique identifier- Ascending Order)
   2. Sorting by Departure time (Ascending Order)
   3. Sorting by Arrival time (Ascending Order)

   - Time Complexity:
     By using the divide and conquer approach, the array is divided into two halves recursively until each subarray contains only one element. Then the two sorted arrays are merged back together. The divide step has a negligible time complexity while the merge step has a time complexity of O(n) per level of recursion and since the recursion depth is $\log_2 n$, the merge step's total complexity is O (n log n).

   - Justification:
     The Merge sort algorithm has a time complexity of O (n log n) making it efficient for sorting large datasets. Compared to Quick Sort algorithm, the Merge Sort algorithm uses less memory making it feasible in memory-constrained environments.

4. Linear Search Algorithm:
   A linear search algorithm is the simplest approach employed to search for an element in a data set. It examines each element until it finds a match, starting from the beginning of the data set until the end. The search is finished and terminated once the target element is located.
   [Ravikiran A S, 27th July 2023]

   In the context of the Bus Tracking system, the Linear Search algorithm iterates through each element ('Bus' objects) in the hash table of buses until it finds a target element or reaches the end of the collection.

- Time Complexity:
  The time complexity of linear search is O(n), where n is the number of elements in the collection being searched. In the case of the program, the ‘searchBus’ function uses a linear search algorithm to find and display details of a bus based on the ‘bus number’ entered by the user with a time complexity of O(n) where n is the number of buses in the hash table.

- Justification:
  Linear search is the most convenient algorithm as it involves sequentially checking each bus record in the hash table until the desired element is found. This simplicity makes it easy to understand and operate through the code.

Testing:

Statement of Testing Approach:

The Catch2 framework is used for unit testing on various functions of the bus tracking system. The Catch2 provides a structured and expressive way to write unit test and ensure correctness in various scenarios and functionalities.

Table of Test Cases:

Test Case	Testing Approach	Status (Success/Failure)
	Function	
sortBus	The function populates the table with test data for buses with predefined values. The 3 different scenarios are then tested using ‘performSort’ to ensure that the ‘sortBus’ correctly sorts buses.	Success
showAvailableBuses	This test case verifies that the ‘showAvailableBuses’ function behaves correctly in both scenarios, when there are available buses and none to display.	Success
merge	The ‘merge’ functions test the merging ability of pre-sorted arrays based on a specific criterion. If the function works as expected, all REQUIRE statements will pass.	Success

compareByBusNo	The function is tested in scenarios where one bus number is less than the other and vice versa.	Success
compareByDepartureTime	This function creates two bus objects 'bus1' and 'bus2' with their specific attributes. Scenarios are created where one bus departs earlier than the other and vice versa.	Success
compareByArrivalTime	This function creates two bus objects 'bus1' and 'bus2' with their specific attributes. Scenarios are created where one bus arrives earlier than the other and vice versa.	Success
vline	The function is expected to behave as expected by producing a line of 100 '*' characters and validating both content and length of the output.	Success
hashFunction	This test case checks for consistency, ensuring that the same input consistently produces the same hash value.	Success
addBus	A hash index is used as a bus number to ensure that the bus table is not empty, therefore indicating that a bus has been added.	Success
searchBus	A bus with the number '1234' and other specific details are assigned to a bus and added to the bus table. REQUIRE statements are used to find the bus number '1234'. To test the other end of the spectrum, the function is used to capture the output of searching for a bus with a non-existing number '9999' and the message 'Bus not found' is displayed.	Success
deleteBusInfo	A bus with number '1234' and its details is added to the bus table using the 'test:addBus'. The existence of the bus is first checked before attempting deletion.	Success
readBusDataFromCSV	This test case ensures that the functions 'writeBusDataToCSV' and 'readBusDataFromCSV' correctly handle writing bus data to a CSV file and reading data from a CSV file back into the bus table, maintaining data integrity throughout the process.	Success
writeBusDataToCSV		Success
	Input Validation and Error Handling	
validateTimeFormat	This section tests the function with invalid time formats where the colon is missing or	Success

	in an incorrect position, incorrect lengths for time formats and out of range values.	

Conclusion:

In conclusion, the development and testing of the bus tracking system aims at creating a robust and efficient solution for managing bus information.
Through the implementation of key functionalities such as adding, searching, deleting buses, sorting algorithms, and file input/output operations prove to be optimal and further consolidated with the integration of test cases using the Catch2 framework.

Summary:

The Bus Tracking system is designed to store and manage bus information using a hash table data structure. It allows users to add new buses, delete, search and sort buses. The system provides a menu-driven interface for user-interaction through the terminal and leverages hashing for efficient storage and retrieval of bus data.

Limitations and Critical Reflection:

1. Hash Table size: The fixed size of the hash table (‘TABLE_SIZE = 10000’) limits the scalability of the system. As the number of buses increases, collisions in the hash table may occur more frequently, affecting performance.

2. Merge sort array may not be the optimal sorting algorithm since the data from the hash table has to be transferred into an array for sorting process. In this case, the heap sort algorithm would work best as it performs well for sorting large arrays and requires no additional memory space beyond the original array making it more memory efficient.

Improvements:

1. Search Functionality: Advanced search options may be implemented involving data filtering to provide more flexibility in finding specific buses.
2. Database Integration: A database system can be put in place for persistent storage of bus data. This enhances data management, scalability, and reliability of the system, allowing for efficient retrieval of and manipulation of bus information.
3. User authentication: User authentication and authorization mechanisms to ensure that only authorized users can access and modify bus data.

References:

1. Jayashree Domala (29th May 2022) [Available at: https://levelup.gitconnected.com/sorting-and-hash-tables-python-12c12dd9c8fb]
   [Accessed on: 17th April 2024]

2. Ravikiran A S (27th July 2023) – SimpliLearn [Available at: https://www.simplilearn.com/authors/ravikiran-a-s]
   [Accessed on: 19th April 2024]

3. Vaibhav Khandelwal (25th October 2023) – SimpliLearn [Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/merge-sort-algorithm#what_is_a_merge_sort_algorithm]
   [Accessed on: 19th April 2024]