A picture containing logo

Description automatically generated

Assignment Cover Letter

(Group Assignment)

|  |  |
| --- | --- |
| **Student Names:**  Arish Madataly  Alysha Puti Maulidina  Dafa Ramadhan Syaidina | 2502049706  2502005906  2502022282 |
| **Course Code:** COMP 6048001 | **Course Name:**  Data Structures and Algorithms |
| **Class:** L2AC | **Name of Lecturer:**  Nunung Nurul Qomariyah, S.Kom., M.T.I., Ph.D. |
| **Major:** Computer Science |  |
| **Title of Assignment:**  Attendance Management System |  |
| **Type of Assignment:** Final Project |  |
| **Due Date:**         21 - 06 - 2021 |  |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating**

BINUS International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality**

By signing this assignment, I understand, accept and consent to BINUS International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Arish Madataly

Alysha Puti Maulidina

Dafa Ramadhan Syaidina

Table of Contents

[Introduction 4](#_Toc106747186)

[Problem description 4](#_Toc106747187)

[Objective 4](#_Toc106747188)

[Specifications 5](#_Toc106747189)

[Solution implementation 5](#_Toc106747190)

[Array 5](#_Toc106747191)

[Linked list 6](#_Toc106747192)

[Queue 6](#_Toc106747193)

[Map 7](#_Toc106747194)

[Search function 7](#_Toc106747195)

[Program manual 7](#_Toc106747196)

[Proof of working app 8](#_Toc106747197)

[Conclusion 11](#_Toc106747198)

[Measurement of efficiency 11](#_Toc106747199)

[Conclusion & future improvements 13](#_Toc106747200)

# Introduction

## Problem description

Throughout this semester, we have learned about the foundational concepts of data structures in C++. Upon research on our final project, my team members and I looked to find a system that may allow us to implement and demonstrate different case uses of data structures and deemed that an attendance system would enable us to explore different types of data structures. Given the rise of asynchronous learning as a result of the pandemic, we believed that developing an efficient management of attendance in an academic setting would be most suitable problem for us to examine.

One of the most important things in programming is choosing the right data structure to represent our data, which determines which operations on your data will be efficient and how much storage is required to store a given data of items. When dealing with large datasets, such as one involving school systems where there may be over a hundred of pupils and grade management, O(n) and O(log n) will make a stark difference when, for example, searching function is involved.

## Objective

Given our problem description, our main objective in this final project is to develop an attendance system using various data structures and determine which is the most efficient. The criteria of the evaluation will be based upon:

* Speed (time complexity)
* Memory use

Below are the functionalities explained that are to be implemented:

1. Checking record function is used before an insertion to determine whether a student record already exists or not.
2. Creating, deleting, and modifying/updating a record

# Specifications

The program specifications are as follows:

* Software and library used:
  + Codeblocks IDE
  + Clock() – to measure runtime
  + Standard Template Library (STL) – implement data structure
* Input:
  + Register, delete and update student data
  + Insert grading for student(s)
  + Search for a particular student
* Output:
  + List of students registered
  + Student report card with grades for each course
* Data structure libraries and algorithm that will be experimented:
  + Array
  + Queue
  + Map
  + Linked list

# Solution implementation

## Array

The first data structure that we experimented on was array, as it is one of the most commonly used data structure and relatively easy to implement. In C++, an array is a sequential collection of items (variables) of the same data type.

Chart, calendar

Description automatically generated

The C++ Standard Template Library (STL) offers a dynamic-size array in std::vector – this is the most suitable for when expect the size of our data to vary during runtime. There are a few setbacks in terms of memory use and runtime with std::vector. It may allocate more memory to optimize insertion time, and the size doesn’t shrink automatically by removing elements.

## Linked list

Secondly, we designed a program using linked list, which can be considered as the second most used data structure after array. It is comprised of a chain of nodes where each node contains a value and a pointer to the next node.



Linked list are good for inserting and deleting items in the front or back of a list, but are more disadvantageous when having to delete a record at an arbitrary index in the middle of a list because a node must be found first.

## Queue

Queue can be viewed as a set of elements arranged linearly in a FIFO manner. The operation of inserting an element is referred to as “enqueue” and removing elements from the queue is called “dequeue”. However, for an attendance system, a queue may be less suitable because of inserting courses or students an arbitrary index is complex and will take up a lot of time.

A picture containing window

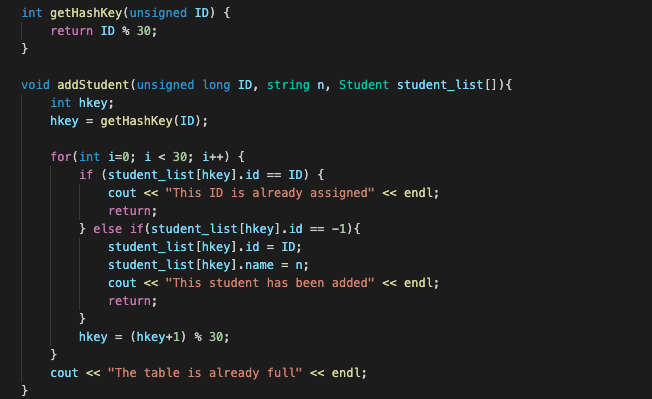
Description automatically generated

## Map

The last data structure that we implemented is map, which as an STL container used to store ordered data so that elements can be looked up in a dictionary-like structure. Each data is assigned to a unique key. What makes a map advantageous compared to the previously mentioned data structure is that data can be traversed bidirectionally in linear time, making it suitable for an application that uses reference or index. Additionally, as the keys stored are all unique which can avoid data redundancy and duplication.

## Search function

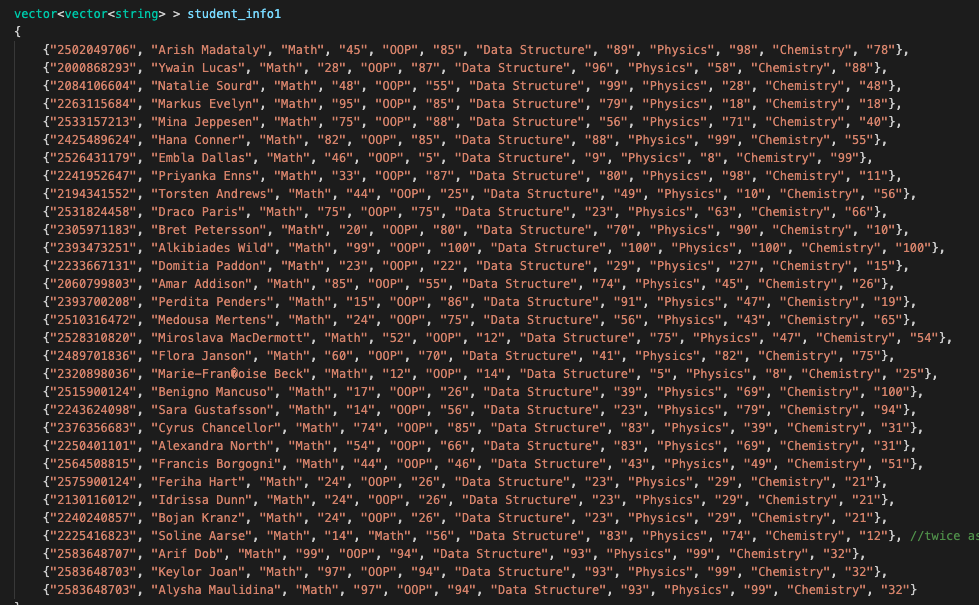
We have decided to use hashing as our main algorithm and a hash table as the container, complemented with linear probing for the collision resolution. Using a hash function is a much faster approach than traversing through an array. With no collision, the hash function will immediately return the index of the value that is searched.



The code snippet above demonstrates the hash key function when adding a student in the array-based system.

## Program manual

The search and modification function of the attendance system lies in the driver file. The file contains a list with 30 student records that we will stimulate our systems on.



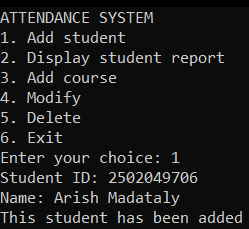
It also contains the clock() function that will measure the elapsed time of each of the systems (this will be further discussed further in the report).

The individual files – Map.cpp, Array.cpp, Linked\_list.cpp, Queue.cpp – are where all of the functions including insertion and deletion can be performed on, and where the user input happens.

## Proof of working app

Array

Adding student record

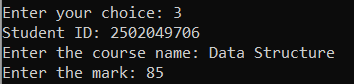


Displaying student record

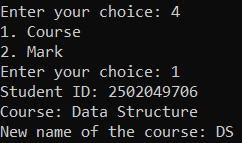
Graphical user interface

Description automatically generated

Adding course and mark for a student



Updating name of course



Updating score for a particular course

Text

Description automatically generated

A picture containing text

Description automatically generated

Deleting a course

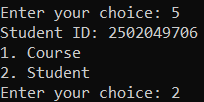
Text

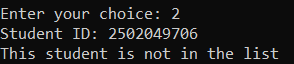
Description automatically generated

Graphical user interface

Description automatically generated

Deleting a student record





Linked list

Adding student, marks, and displaying report for all students

Text

Description automatically generated

Map

# Conclusion

## Measurement of efficiency

At the end, we have managed to successfully implement the measurement of efficiency in the map and array versions of our attendance system using the clock() library. It returns the approximate CPU time consumed by the program, and is said to be the most stable f. With this, we measured the time taken to insert a course for an existing student and display their report i.e. for 10, 20 and 30 students.



We tested the different systems 10 times to gain an average run time, as shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| For 10 | Time (seconds) | | | | | | | | | | Avg |
| No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Map | 0.07000 | 0.09000 | 0.10400 | 0.074000 | 0.10100 | 0.10300 | 0.09200 | 0.10100 | 0.10000 | 0.10400 | 0.0938 |
| Array | 0.07300 | 0.08400 | 0.08500 | 0.08300 | 0.08300 | 0.08400 | 0.08700 | 0.08300 | 0.08800 | 0.08600 | 0.0836 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| For 20 | Time (seconds) | | | | | | | | | | Avg |
| No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Map | 0.16000 | 0.1590000 | 0.17300 | 0.074000 | 0.191000 | 0.17100 | 0.17400 | 0.16600 | 0.17700 | 0.17600 | 0.1621 |
| Array | 0.16900 | 0.16800 | 0.16600 | 0.16600 | 0.16500 | 0.16300 | 0.16700 | 0.15700 | 0.15800 | 0.15500 | 0.1634 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| For 30 | Time (seconds) | | | | | | | | | | Avg |
| No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Map | 0.28000 | 0.24600 | 0.25700 | 0.25700 | 0.24500 | 0.24500 | 0.26000 | 0.26200 | 0.25500 | 0.27100 | 0.2333 |
| Array | 0.23200 | 0.23300 | 0.23200 | 0.23400 | 0.23400 | 0.23100 | 0.23400 | 0.23600 | 0.23700 | 0.23700 | 0.2340 |

From the findings above, we can infer that the system with the map data structure yields the fastest and most optimal time of execution, on average, for all tests.

In theory, when considering far larger data sets (such as an entire school system) than the one used in our tests, it can be said that implementing a map dataset is more efficient because we know that it is constant and has no risk of collision when executing a search function because it allocates a memory only when a student is added. Although for smaller data sets, array may be appropriate as searching using a hash key is faster when no collisions are involved. But as data sets dynamically grow, the array searching algorithm approaches the worst-case scenario when the tables have to be sequentially traversed from the start which means it is the same as simple array and does not bear any functions of a hash table.

As the trend suggests, if there exist larger data sets, a map would obviously be more suitable of use because the time for it to perform functions such as modify or search would much be more lower than array.

## Conclusion & future improvements

The result of our tests proved our initial theory in that the map data structure and linked list is much more efficient in implementation of such attendance system. Furthermore, using a small pool of data, the array system performs much better compared to that of its map counterpart.

There is no doubt that this project leaves room for many potential developments. In further explorations, we hope to experiment with different types of collision resolution for our search function. Furthermore, we have experimented with AVL trees for the hash table but made the program slower as the AVL tree had to be run for every data inserted – however, for bigger data it can potentially solve the worst-case scenario that may occur in the hash table in the array-based attendance system. We also plan to fix our queue-based system In the future, perhaps using double ended queue.