



BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Group Work)

Student Information:	Surname:	Given Name:	Student ID Number:
----------------------	----------	-------------	--------------------

1.	Wardana	Sandrian	2502016411
2.	Prabowo	Sulthan	2502020075
3.	Ramanda	Rayhan	2501982582

Course Code : COMP6798001

Course Name : Data Structures

Class : L2CC

Lecturer : Ir. Tri Asih Budiono, MIT

Type of Assignments: Term Final Project

Submission Pattern

Due Date : 25 June 2022 **Submission Date** : 25 June 2022

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:

1. Sandrian Wardana
2. Sulthan Prabowo
3. Rayhan Ramanda

Battleship Clone Written in C++

Sulthan, Sandrian, and Reyhan

CONTENTS

[Program Description](#)

[Explanation of Our Code in C++](#)

[Executing Our Code](#)

[Alternative Data Structures](#)

[Video Link](#)

[GitHub Link](#)

[References](#)

1. Program Description

Battleship is a strategy-type guessing game for two players. Each player's fleet of warships are marked on a board. The locations of the fleets are concealed from the other player. Each player takes turns on taking shots to see if they hit their opponent's battleship or not; whoever destroys all of their opponent's ships first wins. This game trains the player's mind to be cunning. By implementing C++ and utilizing some of the ASCII styled art, players can enjoy this authentic Battleship board game.

To give an introduction for our program, we initially came up with the idea of making a tank-shooting game. We went with this idea until we carefully re-assessed the project requirements. In the end, we came up with implementing a Battleship like game in C++. We thought that a Battleship type game would be able to easily be implemented on different data structures.

Unfortunately, we weren't able to implement another data structure into our C++ program but we were able to do a theoretical analysis on it. On the flip side, we have a fully working program implementation of a Battleship game in C++. Our game works very similarly to an actual Battleship game, except that it's a simple implementation of it where you only need to guess the locations of the opponent ship and keep going until you defeat all enemy ships.

Sandrian, Sulthan, and Rayhan worked on this Battleship project. Rizky was in our group at the start until he separated to make his own project in the end.



	A	B	C	D	E	F	G	H	I	J	
1											
2											
3											
4			X								
5						X	X				
6			X					X		X	
7				X						X	
8	X	X						X			
9											
10											

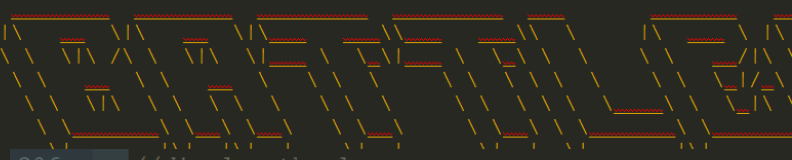
2. Explanation of Our Code in C++

Here, our program's code will be explained. The inner workings of the program will be explained to show how it runs. As a disclaimer, the code snippets will be given as examples but not full code snippets, please check our GitHub for that.

First, there are the simple methods. They're called "simple" methods because they handle the overall presentation of the program in the console. Methods such as `spaceConsole()` and `hashtagConsole()` are purely there to be able to print certain lines of text, in this case, it's to print spaces in the console to create gaps in the text so that they wouldn't be squished all together, and the other is to just simply print a line of hashtags, all taking in an int parameter as the amount of space/hashtags are wanted to be printed.

To continue with simple methods, there are the `displayTitle()`, `winText()`, `loseText()`, `displayLoseText()`, and `displayWinText()`. These methods do exactly as their name suggests [but to clarify, the difference between `loseText()` and `displayLoseText()` (and similar methods) is that the first displays ASCII art, and the second just displays a line of text]. The previously stated simple methods are used in these methods as well to create a tidier presentation. (From Top to Bottom, Pic. 2.1: `spaceConsole()` and `hashtagConsole()`, Pic. 2.2: `displayTitle()`, Pic. 2.3: `displayLoseText()` and `displayWinText()`)

```
13 void spaceConsole(int lineSize){ //simulate clearing the console method
14     for (int i = 0; i < lineSize; i++){
15         cout << " " << endl;
16     }
17 }
18
19
20 void hashtagConsole(int lineSize){ //create hashtag line method
21     for (int i = 0; i < lineSize; i++){
22
28 void displayTitle(){ //displaying the title method
29     spaceConsole(5);
30     hashtagConsole(120);
31     spaceConsole(1);
32     cout << "Battleships: C++ Remake" << endl;
33     cout << R" (
34
35     //display the lose screen
36     //display the win screen
37
38     //display the lose screen
39     //display the win screen
40
41     //display the lose screen
42     //display the win screen
43     void displayLoseText(){
44         spaceConsole(1);
45         cout << "Goddamn it, they got you first. You lost to
46         spaceConsole(1);
47     }
48
49     //display the win screen
50     void displayWinText(){
51         spaceConsole(1);
52         cout << "Congratulations. You win! You successfully
53         spaceConsole(1);
54     }
55 }
```



Afterwards, there's the Player class, which handles everything of the game. First, let's explain the private variables.

There are the constant static int variables, they are BoardSize, maximumShips, NOSHIP, ATTACKED, SHIP, and DESTROYED. BoardSize holds the value of the limit of the BoardSize or of the matrix array, maximumShips just holds the value of the limit of ships a player/enemy can have. NOSHIP, ATTACKED, SHIP, and DESTROYED are different values representing the state of the board; in order, represents no ships on the board location with 0, then represents the board location that is attacked but hits no ships with 1, then represents the board location that has a ship with 2, and finally represents the destroyed ships with 3.

The last two variables are bool isCpu and int numOfShips, which determines if something is the cpu or not (aka the player), and holds the value of the number of ships the player or the cpu has while playing the game. (Pic. 2.4: the private variables)

```
82  class Player
83  {
84  private:
85      const static int BoardSize = 10; //Declares the maximum board size
86      const static int maximumShips = 10; //Declares the max amount of ships available
87
88      const static int NOSHIP = 0; //int to show if there is no ship on the board
89      const static int ATTACKED = 1; //int to show if a particular location has been attacked on the board
90      const static int SHIP = 2; //int to show if there is a ship on a certain location on the board
91      const static int DESTROYED = 3; //int to show if the position has a destroyed ship on the board
92
93      int matrixBoard[BoardSize][BoardSize]; //keep track of positions of player's ships
94
95      //need variable to identify type of player
96      bool isCpu;
97
98      //need variable to get number of ships
99      int numOfShips = 0; //player and cpu has this attribute
100 }
```

Next will be explaining the rest of the Player class, but it is best if they're explained through the structure of int main.

The first thing regarding the Player class in int main is Player player(false) and Player cpu(true); these are the constructor methods. To explain, when creating the player and cpu objects, the constructor method takes in a boolean (isCpu) which determines whether the object created is a cpu or not; if it's false, then it's a player, if it's true, then it's a cpu.

In addition, it calls the clearGrid() private method to clear the Battleship board or the matrixBoard, which is a 2D matrix array, so that we would ensure that the board is empty before setting the ships on it. Then, setShips() is called.

setShips() private method basically just sets the ships on the matrix board. It first checks if this method is being called through the player or cpu object.

If it is called through the player object, then it will take input from the user to set the positions of the ship on the board with x and y coordinates. After the input, it will check if the x and y positions the user inputted are already occupied with ship or not, if it's not, it will set the ship and continue to do so until the desired amount of ships are set, if there is a ship at that location, then it will tell the player and ask to re input.

If it is called through the cpu object, then it will do the same thing as what happens if the method is called through the player object, except there is no input and

instead, it creates a random number with the limit of the maximumShips variable. (Pic. 2.5: creating objects in int main, Pic. 2.6: setShips())

```
323 int main() {
324     srand(time(0)); //initializing the randomizer
325     displayTitle();
326     //set up two Players: (player and cpu) and set up each
327     Player player(false); //creating player and cpu object
328     Player cpu(true);
329
330     //start turn based game loop
331 }

115 void setShips() {
116     if (isCpu == false) { //checking if it's setting the ship for the play
117         cout << "You need to set 10 ships!" << endl;
118         spaceConsole(1);
119         //setting the player's ship coordinate
120         while (numOfShips < maximumShips) { //checking the amount of ship
121             int setXCoord = takeInputBoard("To set ship coordinate, please enter X: ");
122             int setYCoord = takeInputBoard("Please enter Y: ");
123
124             if (matrixBoard[setXCoord][setYCoord] != SHIP) { //if matrix is not occupied
125                 numOfShips++; //add on
126                 matrixBoard[setXCoord][setYCoord] = SHIP; //set the ship
127                 spaceConsole(30);
128                 displayBoard(); //display the board
129                 cout << "You have to set " << maximumShips - numOfShips << " more ships." << endl;
130             }
131             else { //if matrix is occupied with a ship already
132                 cout << "A ship is already placed there, please try again." << endl;
133             }
134         }
135     }
136
137     else { //checking if it's setting the ship for the cpu
138         //have a randomizer create the cpu's set locations
139         while (numOfShips < maximumShips) { //checking the amount of ship
140             int xPosition = rand() % maximumShips;
141             int yPosition = rand() % maximumShips;
142
143             if (matrixBoard[xPosition][yPosition] != SHIP) {
144                 numOfShips++;
145                 matrixBoard[xPosition][yPosition] = SHIP;
146             }
147         }
148     }
149 }
```

One notable method to mention now is the takeInputBoard(). This int public method handles taking the input of the user and does validation checks. It takes in a string parameter to set the text that needs to be printed before taking the input.

It then takes a string input. Afterwards, it starts the validation in a try catch statement that's looped inside a do while loop; that string input is converted into an int variable, doing this will be able to throw an exception if the user accidentally inputs anything other than an int.

Next, if it passes through the int conversion, it will be checked if the input is greater than the BoardSize; since this method is used to take in x and y coordinate inputs, we do not want the player to input anything greater than the board size. If the statement is true, it will throw an exception and ask the user to re-input.

To end the do while loop of the try catch statement, it runs the do while loop on a boolean which will switch to true when the try catch statement does not return any exceptions. Finally, the takeInputBoard will return an int which is taken as the final input of the user. (Pic. 2.7: takeInputBoard())

```

241 //method to take user input + exception handling.
242 int takeInputBoard(string prompt) {
243 //string prompt helps to display the prompt that is needed
244 //setup int value to return
245 int input;
246 string rawInput;
247 cout << prompt << endl;
248 cin >> rawInput;
249 bool convertSuccess = false;
250
251 //do while loop to check if input can be converted to int
252 do {
253 try {
254 input = stoi(rawInput); //convert string to int
255
256 if (input > BoardSize){ //then check if the input is greater than board size
257 throw 99; //throw exception
258 }
259 convertSuccess = true; //change bool to true at the end of the try block
260 }
261 catch (...) { //if it catches any error
262 cout << "Invalid input, please try a number within the board size\n";
263 cin >> rawInput; //ask for input again
264 }
265
266 } while (!convertSuccess); //do while bool is false
267
268 return input - 1;
269
270 }
271
272

```

Next in the int main, a do while loop is used to run the main battleships game. First, the displayBoard() method is called based on the Player class object of player or cpu, this method just does what the name suggests. Finally then calls the method doTurn() which basically handles the turn based attacking feature of the game.

doTurn() takes a parameter which is the Player object, which is connected to a pointer to call that specific address of the object. The reason that it's like this is to be able to have the method actually affect the object's numOfShips variable so that when it reaches 0, it can stop the do while loop and end the game. What doTurn() does is that it mainly determines if the player or the cpu is attacking; if the player is attacking, it takes input, runs through the validation method, and then calls the main attack() method. If the cpu is attacking, it does the same thing as the player except a random number generator is used to determine the positions of attack.

What the attack() method does is it checks the location of the board based on the user input of x and y, and then either changes the variable of the position on the board to ATTACKED or DESTROYED or prints a statement that you've attacked at the same spot, already destroyed the ship at that spot, or if the input is invalid. Then it either returns true and it signals to stop the attack loop in the do while loop of doTurn() or if it returns false, it will keep looping until an attack is successfully returned true.

It will keep looping the do while loop in int main until the attacks have reduced the numOfShips down to 0, in which it will stop the loop. (From Top to Bottom, Pic. 2.8: doTurn(), Pic. 2.9: attack(), Pic. 2.9: The do while loop of int main)


```

273 //method to determine the attack turn of player or cpu
274 void doTurn(Player *enemy) { //enemy pointer
275     if (isCpu == false) { //check for player
276         spaceConsole(1);
277         cout << "It's time for you to attack." << endl;
278         spaceConsole(1);
279
280         int xInput;
281         int yInput;
282
283         do {
284             //take input of player of the coordinates they want
285             xInput = takeInputBoard("Enter the x-coordinate");
286             yInput = takeInputBoard("Enter the y-coordinate");
287             //call attack function taking the coordinates as arguments
288             //method to attack the target coordinates
171 bool attack(Player *enemy, int newX, int newY) { //The "*" and "Input));
172     //check the coordinate:
173     //if coordinate is occupied with ship, destroy it
174     if (enemy->matrixBoard[newX][newY] == SHIP) {
175         enemy->matrixBoard[newX][newY] = DESTROYED;
176         enemy->numOfShips--; //decrease by 1 of either the player or the cpu
177         return true;
178     }
179     //else if coordinate is not occupied with a ship, mark it as attacked
180     else if (enemy->matrixBoard[newX][newY] == NOSHIP) {
181         enemy->matrixBoard[newX][newY] = ATTACKED;
182         return true;
183     }
184     //else if coordinate is already destroyed, tell user that it has been destroyed
185     else if (enemy->matrixBoard[newX][newY] == DESTROYED) {
186         if (isCpu == false) cout << "This ship has been destroyed";
187         return false;
188     }
189     //else if coordinate is already attacked, tell user that it has been attacked
190     else if (enemy->matrixBoard[newX][newY] == ATTACKED) {
191         if (isCpu == false) cout << "You've already attacked this coordinate";
192         return false;
193     }
194     //start turn based/game loop
195     do {
196         spaceConsole(30);
197         //first display the board; cpu board on top and player board on bottom
198         cout << "The enemy has " << cpu.getNumOfShips() << " ship(s) left." << endl;
199         cpu.displayBoard();
200         spaceConsole(1);
201         player.displayBoard();
202         spaceConsole(1);
203         cout << "You have " << player.getNumOfShips() << " ship(s) left." << endl;
204         //ask player to attack
205         player.doTurn(&cpu);
206         //let cpu do its thing
207         cpu.doTurn(&player);
208     } while(player.getNumOfShips() > 0 && cpu.getNumOfShips() > 0);
209 }

```

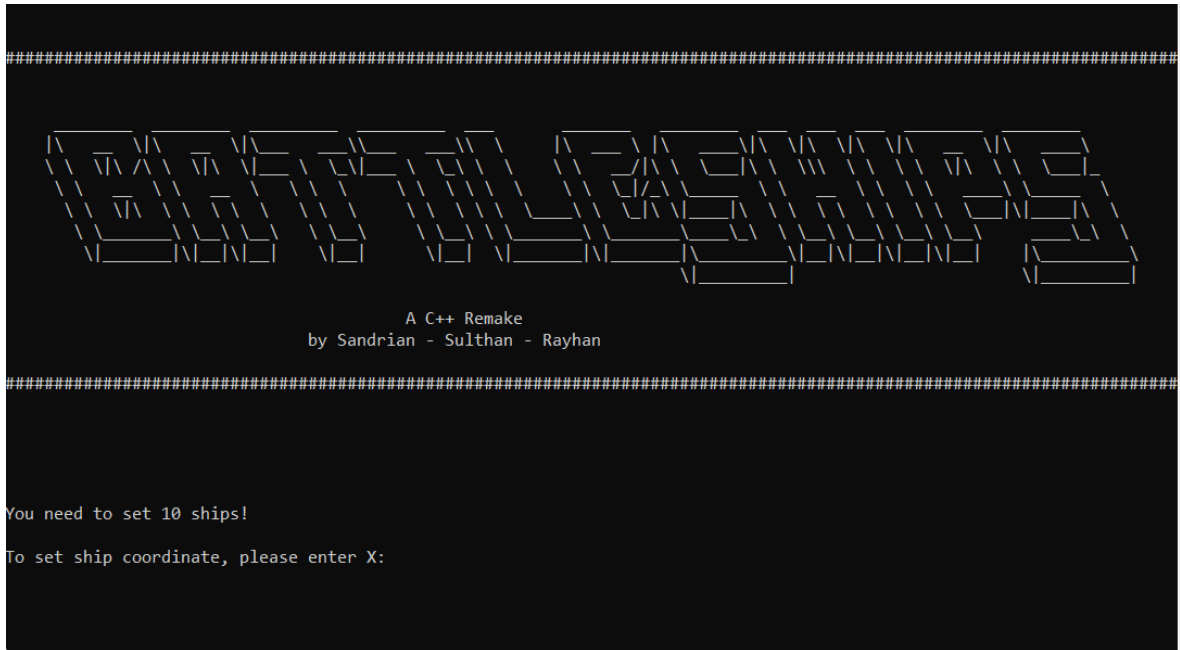
Depending on if the player's or cpu's numOfShips variable is 0, it will display a win screen or a lose screen. The way that either screens present is that it calls the ASCII art method, displays the results of player and cpu board, and then displays the win/lose sentence text with the corresponding method. With the help of the simple methods, it

helps the presentation of the console. (Pic. 2.10: The if else statement for checking winner and printing the results)

```
348 //check winner + print results
349 if (cpu.getNumOfShips() == 0) { //if player wins
350     spaceConsole(50);
351     hashtagConsole(120);
352     winText();
353     hashtagConsole(120);
354     //spaceConsole(1);
355     cout << "CPU Board:" << endl;
356     cpu.displayBoard();
357     spaceConsole(1);
358     player.displayBoard();
359     cout << "Your Board:" << endl;
360     displayWinText();
361 }
362 else if (player.getNumOfShips() == 0) { //if cpu wins
363     spaceConsole(50);
364     hashtagConsole(120);
365     loseText();
366     hashtagConsole(120);
367     //spaceConsole(1);
368     cout << "CPU Board:" << endl;
369     cpu.displayBoard();
370     spaceConsole(1);
371     player.displayBoard();
372     cout << "Your Board:" << endl;
373     displayLoseText();
374 }
375
```

We've added extensive commenting on the source code itself, so feel free to check it out yourself.

3. Executing Our Code



Pic. 3.1, As you see here, we implemented basic 3D ASCII art and prompted the user to set 10 ships on a 10x10 sized board.

On display, the game board symbols of “V” to represent the ship on the board, “*” represents an attacked location that doesn’t contain a ship, “X” represents a ship that is destroyed, and “-” just represents an untouched position on the board. The opponent’s board won’t show the ship positions (obviously).

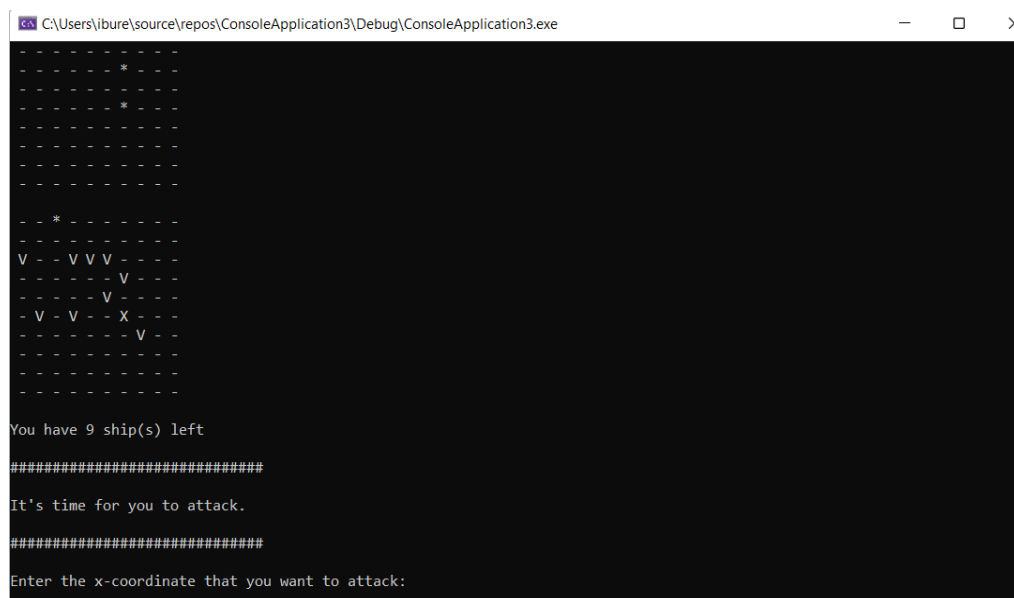


Fig. 3.2, One of the human player ship is destroyed marked with “X”, and an attacked position is marked with “*”.

When targeting the same or destroyed tile, it would throw an exception and return to x coordinate again.

```
Enter the x-coordinate that you want to attack:
4
Enter the y-coordinate that you want to attack:
7
You've already attacked that location.
Enter the x-coordinate that you want to attack:
7
Enter the y-coordinate that you want to attack:
8
This ship has been destroyed.
Enter the x-coordinate that you want to attack:
8
Enter the y-coordinate that you want to attack:
9
```

Pic. 3.3, Targeting the same attacked position or destroyed ship results in re-prompting the user to enter the correct input.

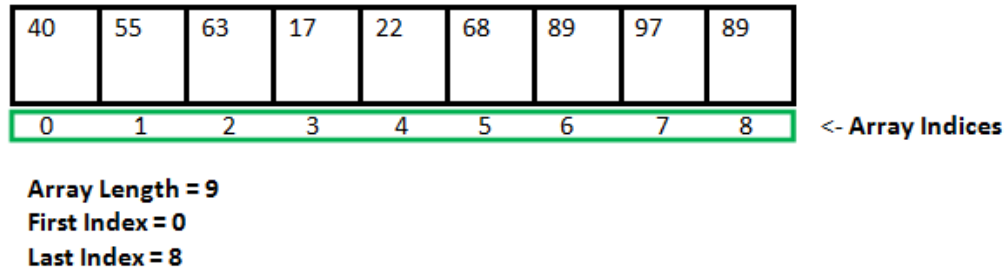
When either one of all CPU's or Player's ships are destroyed, the game will trigger Lose/Win ASCII art.

[illegible]

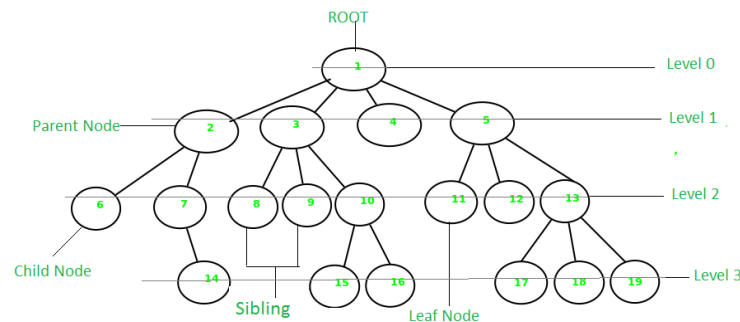
Pic. 3.4, In this case, the human player won and the 3D ASCII art of winning appeared.

4. Alternative Data Structures

The current data structure that the code is using is known as an array. An array is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array . It has advantages such as retrieving or sorting data efficiently, and is able to locate data at an index position . However, the disadvantage of this is that it can only store a fixed size of elements in the array (Pawar, 2022).



An alternative to this data structure is by using trees. A tree is a popular data structure that is non-linear in nature. Unlike other data structures like array, stack, queue, and linked list which are linear in nature, a tree represents a hierarchical structure. While access or searching is slower compared to arrays, inserting or deleting is faster with binary trees. Unlike arrays, trees don't have an upper limit on the number of nodes (Geeks for Geeks, 2022).



5. Video Link

Link contains the demo of the program.

<https://www.youtube.com/watch?v=ckdbP8b2RNY>

6. GitHub Link

The github link that contains this report, video link, and the entire source code of the project.

<https://github.com/The-Riz5-Iz6-Wiz4/DataStructuresFinalProject>

7. References

- <https://www.geeksforgeeks.org/binary-tree-set-1-introduction/>
- <https://www.geeksforgeeks.org/arrays-in-c-cpp/>
- <https://www.geeksforgeeks.org/>
- <https://stackoverflow.com/>
- <https://www.w3schools.com/>
- <https://www.programiz.com/>
- <https://cplusplus.com/>
- https://youtu.be/0cZ5aXcU_oA