DATA STRUCTURES AND ALGORITHM

FINAL PROJECT



**Student Information:**

Leonardo Richie

**Class Information:**

Class: L2BC

Lecturer’s name:

Nunung Nurul Qomariah

**Binus University International**

**School of Computer Science**

**2022**

Chess Game Runtime

# Project Description

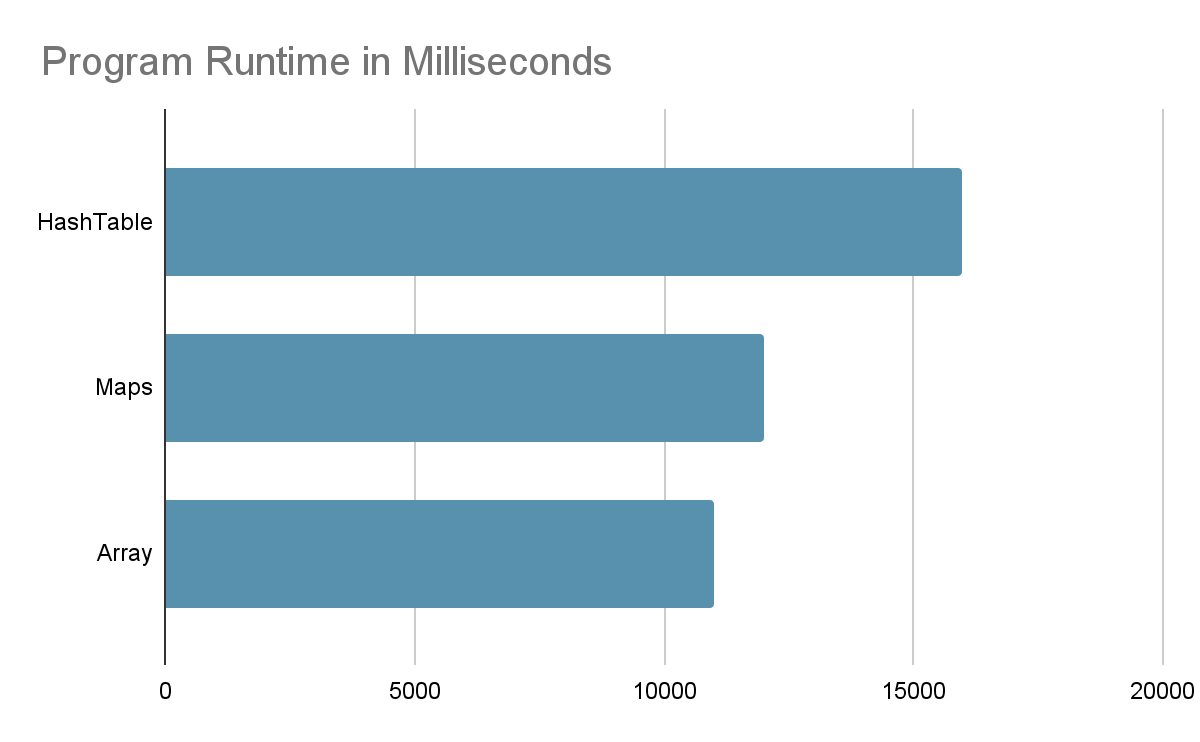
For the data structure final project, we plan on doing multiple data structures applied to our program. In doing that, we want to measure the time length of moving the chess pieces from one place to another using different data types and print it out as an output. And at the end, we plan to compare which are best at completing the task.

Rules and Boundaries for the chess game

* Assume movement set by the user
* Single input from the user
* Usage of “Chrono” class as a time measurement tool
* The position of the piece & color will be stored
* Different distance relocation run each algorithm with different inputs

# Proposed alternative data structure

Array fen kenneth part



|  | Program Runtime in Milliseconds |
| --- | --- |
| Hashtable | 15985 |
| Maps | 11994 |
| Array | 10997 |

As we can see from the graph and the table data, Array is the best way to do the program

# Analysis

For the project, we used multiple data structures to execute the same program. The program is to store the movement of the chess piece and also print out the board. The array is the fastest and the most effective data structure to do this action. Hashmaps are using an array as well so it will not be faster than the array itself. Maps are not the best way because even though they can do instant access by keys, an array can directly be accessed by index.

Fen, array = Kenneth part

The data structure used and their time complexity:

Hashtable O(1) - fastest, O(n) - Worst case

Maps O(log n)

Fen -

Array -

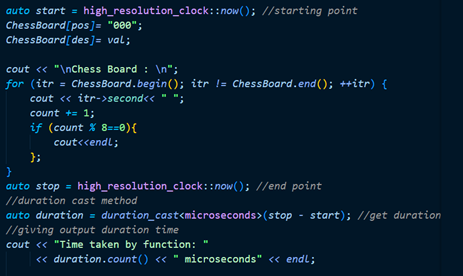
From the time complexity we can see that array is better.

# Code Explanation

**Chrono**



By using Chrono, we can count how long it takes for the program to be executed. We aim to find which data structure is the fastest and most effective for chess movements.



First state the starting point (auto start = high\_resolution\_clock::now()) to show where the timer starts and the end point (auto stop =high\_resolution\_clock::now()) to show where the timer ends. We put the timer between the program that changes the position of the chess piece and also print the board out. Auto duration = duration= cast<microseconds> (stop-start) is to get the time from the stop and start in microseconds. Next is printing out the microseconds by cout<< duration.count().

Fen: Kenneth part

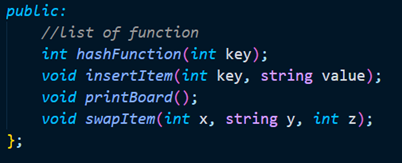
Array: Kenneth part

Hashtable

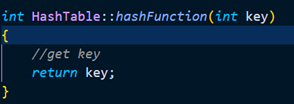
**Creating HashTable Class**



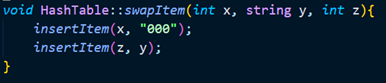
Static const int hash buckets = 65 to declare the hashmap size which is 65. Next is to create a linear data structure, a list that consists of int and string for the hash bucket.



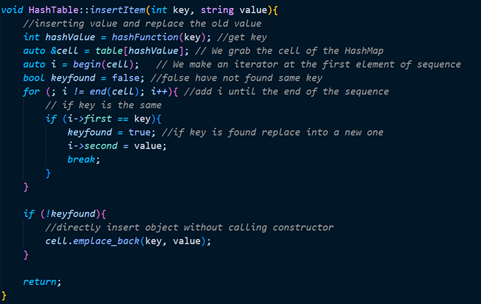
There are 4 function available in the class, first is the hashfunction to get the keys, insertItem to insert value to the hashtables, printBoard to print out the 8x8 board and SwapItem to swap the pieces of the chess.



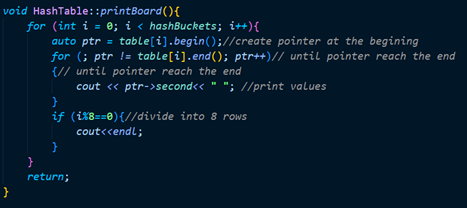
The hashTable function directly returns the key, because we do not need any probing.



The swapItem function is to swap the position of both chesspiece. It uses InsertItem function 2 times, first to change the old position to 000 value and the new position with the previous value. This function take x,y,z as the input. X as the first position, y as the value that wants to be moved, z is the new position destination.



The insertItem function takes the key and value input for the new position. First is to get the key by using Hashfunction(key) and put it inside the hashvalue variable. &cell = table[hashvalue] is to grab the cell of the hashmap. I is an iterator that started at the first element of the sequence. Keyfound boolean is to determine if the value if found or not. Next by using the loop the iterator moves from the beginning to the end, and finding the key. If the key is found, keyfound is true and the value will be input. i->first points to the keys and i->second points to the value. If the key is not found, the key and value will be input.

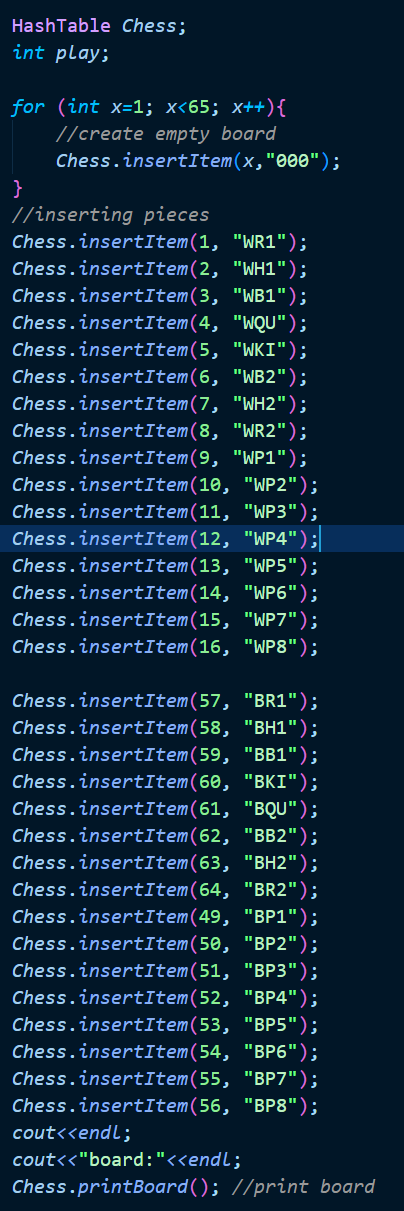
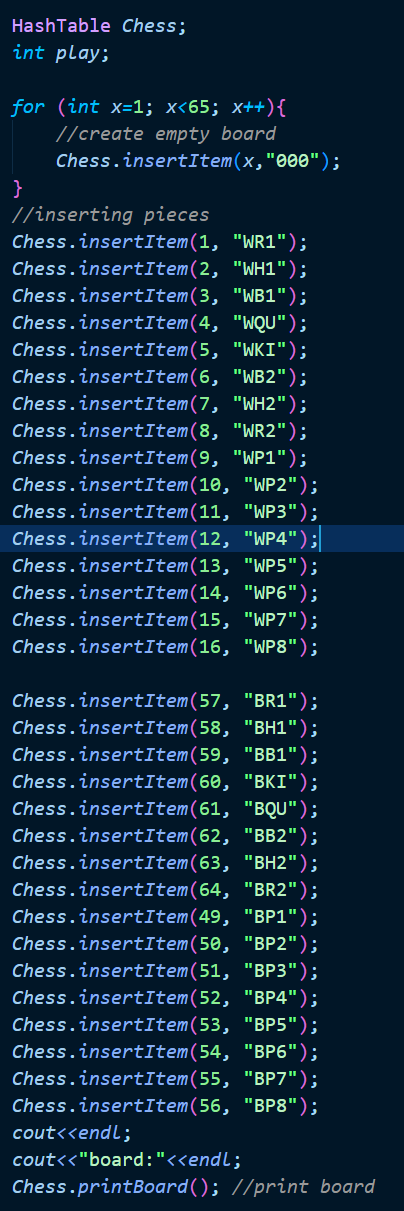


The printBoard function is to printout the board of 8x8.

By using the nested loop, first loop is to get through the hash bucket, second loop is for the pointer to get to the second which is the value. Ptr will start at the beginning and end at the end. ptr->second is to get access to the value since ptr->first is to get the key. The if statement i%8 is to show that every 8 value printed the next value will go to the next line below it so that the 8x8 board is formed.

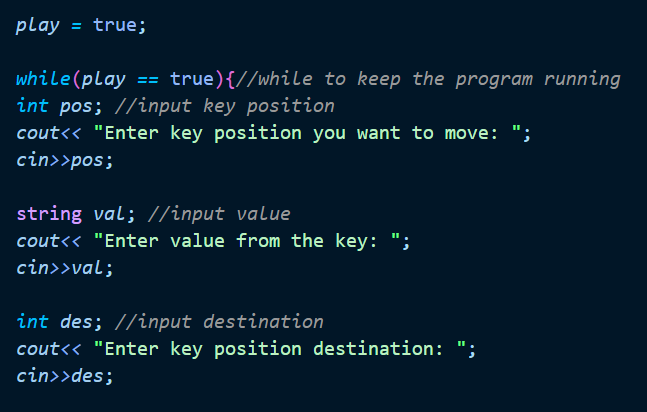
**Main file**

Creating the chess Board

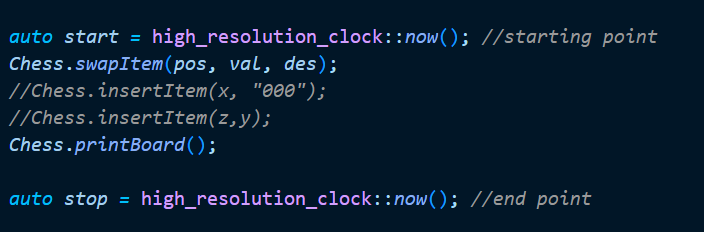


For the main file, first create the hashtable object called chess. The for loop is to create all of the 8x8 board as “000” which means empty. By using the InsertItem function the chess piece is inserted. Next is Chess. printBoard() function to printout all of the board content.

**Getting inputs**



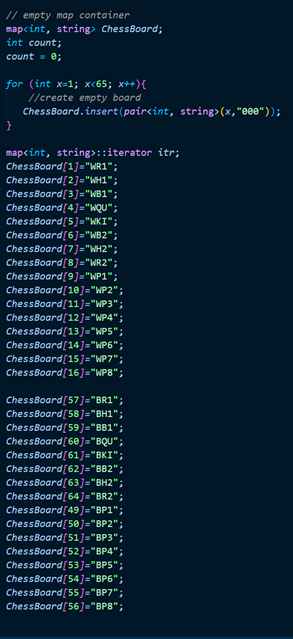
To get the input from the user, cin is used to get the previous position, destination, and also the value. By using the while loop, the program can keep on going and ask the user to input again.



For hashMaps to switch the position, the program just need to use swapItem() for pos,val,des as the input from the user. The swap Item will directly move the item to the new position and change the old position to “000”.

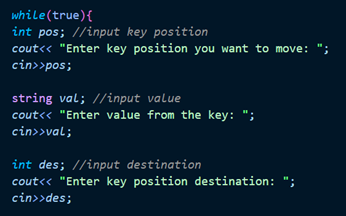
Maps

**Creating ChessBoard**



Creating a chess board using maps. First declare the map <int, string> ChessBoard; to create the map object. Declare the count variable to print out the chessboard, printing endl every 8 value to create 8x8 board. map<int, string>::iterator itr; is to declare an iterator. The iterator is use to search value through the map. Next is inserting all the value. By using ChessBoard[int] = value, the chess pieces are inserted (black and white pieces).

**Getting inputs**



By using the cin, the user will input the position of the first value, the value, and the destination of the new position.

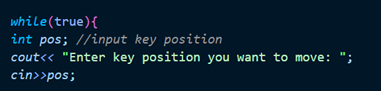
**Changing the positions and printing out the board.**

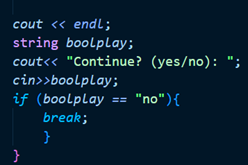


To change the position of the board, by using ChessBoard[index]=value 2 times, first change the old position value to 000 then change the new position index to value

Using the for loop and the iterator at the beginning of the map until the end. itr->second, pointing to the value (itr->first pointing to the keys). count+= 1 to show the number of value printed. There is and if function at the last to print endl every 8 value so that the 8x8 board is formed.

**While loop**





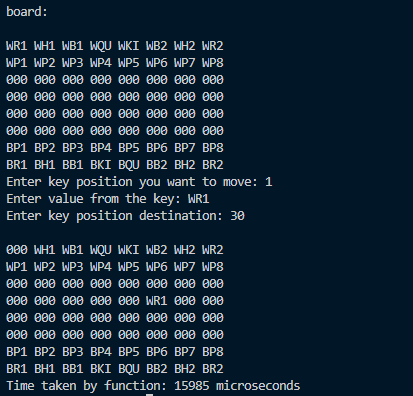
For the program to always run, while(true){} is used. As long as the while loop runs the program will run continuously. At the end of the program, it will ask the user to input yes or no, if they want to continue the program they type yes, if they want to stop, they can type no. the if statement is to break the loop once the user type no.

# Program Manual

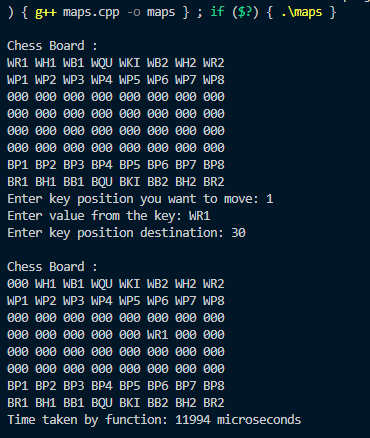
The program for every data structure is similar which is to print out the board and get the user input from one position to another position which is used to move the chess piece.

The program will directly switch the position so that the chess piece can move

**Hashtable**

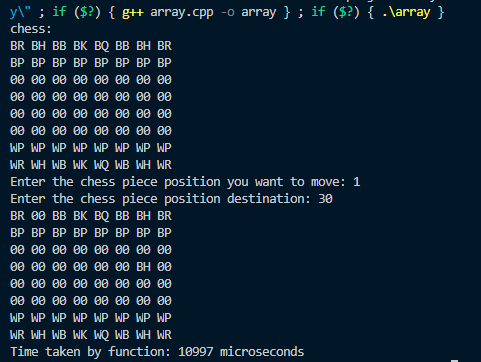


**Maps**



Fen : kenneth part

**Array**



# Result

As the result of the project, the array is the best way to execute the program because for the chess program each chess piece is stored in 1 position starting from 1 to 64. The empty chess board will be “000” value. The array is faster because we do not need any keys, we can directly put the chess piece by indexing from 1 to 64. As for hashmaps and maps, they both need keys to be accessed and it will cost more time because keys are not necessary for chess to work. Even though all of the data structure has instant access, the array is better.

From the program executed, by using Chrono, the time taken for each program is shown at the end of the program. Hashtable took 15985 microseconds, HashMaps took 11994 microseconds and maps took 10997 for the program to be executed. It is proven that the array has a faster execution. It is caused by the indexing of the array, hashmaps and maps need keys to access the value, so it took more time to create the keys. The array can just directly index and does not use the key to access value.