C++ Implementation and Documentation Report of Searching Algorithm

Submitted by:
Aaron Christian C. Navarro

Submitted to:
Ms. Justine Louis Neypes

BS Information Technology
NTC_CC105 — Data Structure

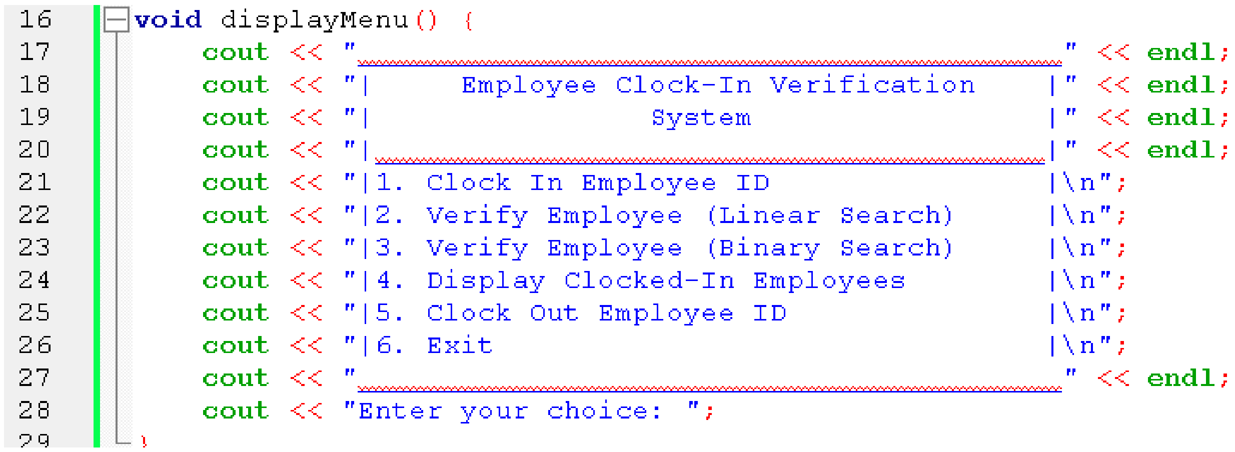
October 30, 2025

Step by step Algorithm

1. Start program.
2. Display title and menu options.
3. Create an empty print queue.
4. Repeat until user chooses Exit:
5. Ask the user for a choice.
    - If 1: Ask for file name and student name → add job to queue.
    - If 2: If the queue is not empty → process and remove first job; else show "No jobs."
    - If 3: Show next job if available.
    - If 4: Display all jobs in queue.
    - If 5: Show total number of jobs.
    - If 6: Display exit message and end program.
6. Else: Show "Invalid choice."
7. End program.

Code Implementation and explanation

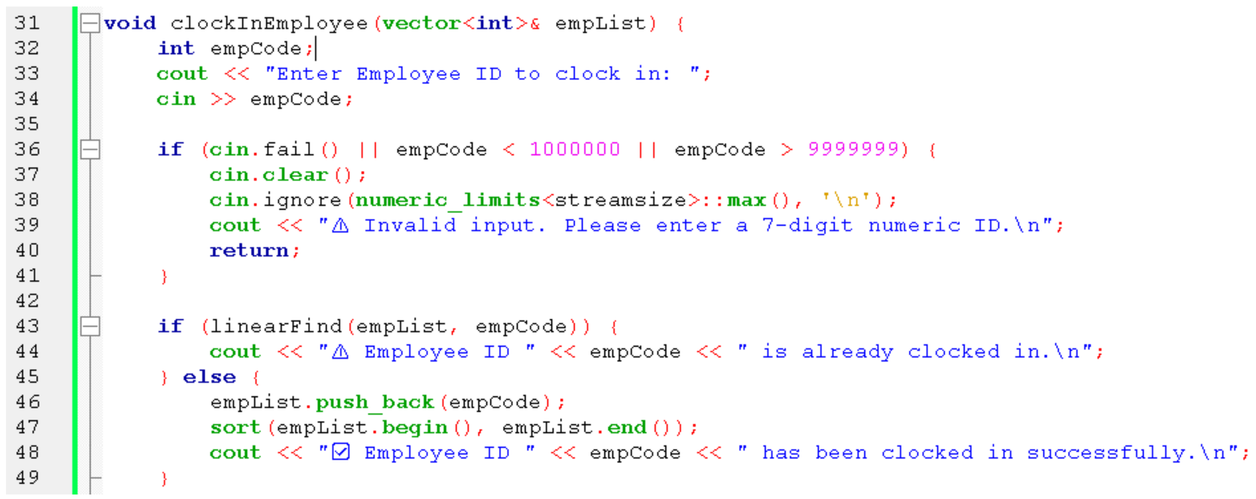
Function: displayMenu()



```
void displayMenu() {
    cout << "_________________________________________" << endl;
    cout << "|      Employee Clock-In Verification    |" << endl;
    cout << "|                System                  |" << endl;
    cout << "|_______________________________________|" << endl;
    cout << "|1. Clock In Employee ID                 |\n";
    cout << "|2. Verify Employee (Linear Search)      |\n";
    cout << "|3. Verify Employee (Binary Search)      |\n";
    cout << "|4. Display Clocked-In Employees         |\n";
    cout << "|5. Clock Out Employee ID                |\n";
    cout << "|6. Exit                                 |\n";
    cout << "_________________________________________" << endl;
    cout << "Enter your choice: ";
}
```

Purpose and Explanation:
In this function, displays the main program menu with all available options for managing employee attendance. It provides a clean and organized design so that the user can easily navigate or select what they need to see. The menu is the main point for accessing all the functions that indicate numbers in the system.

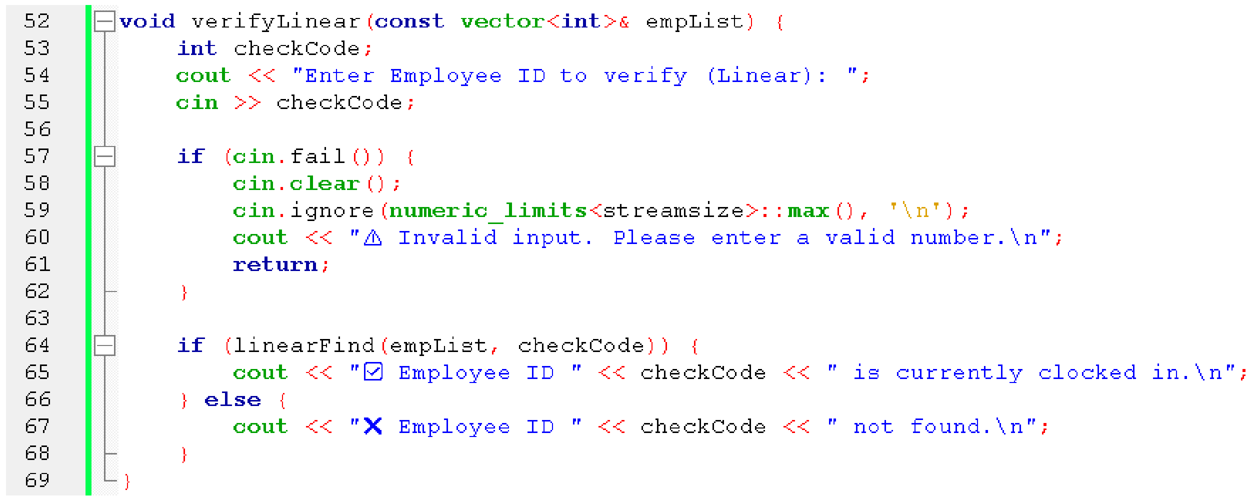
Function: clockInEmployee()



```cpp
void clockInEmployee(vector<int>& empList) {
    int empCode;
    cout << "Enter Employee ID to clock in: ";
    cin >> empCode;

    if (cin.fail() || empCode < 1000000 || empCode > 9999999) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a 7-digit numeric ID.\n";
        return;
    }

    if (linearFind(empList, empCode)) {
        cout << "⚠ Employee ID " << empCode << " is already clocked in.\n";
    } else {
        empList.push_back(empCode);
        sort(empList.begin(), empList.end());
        cout << "☑ Employee ID " << empCode << " has been clocked in successfully.\n";
    }
```

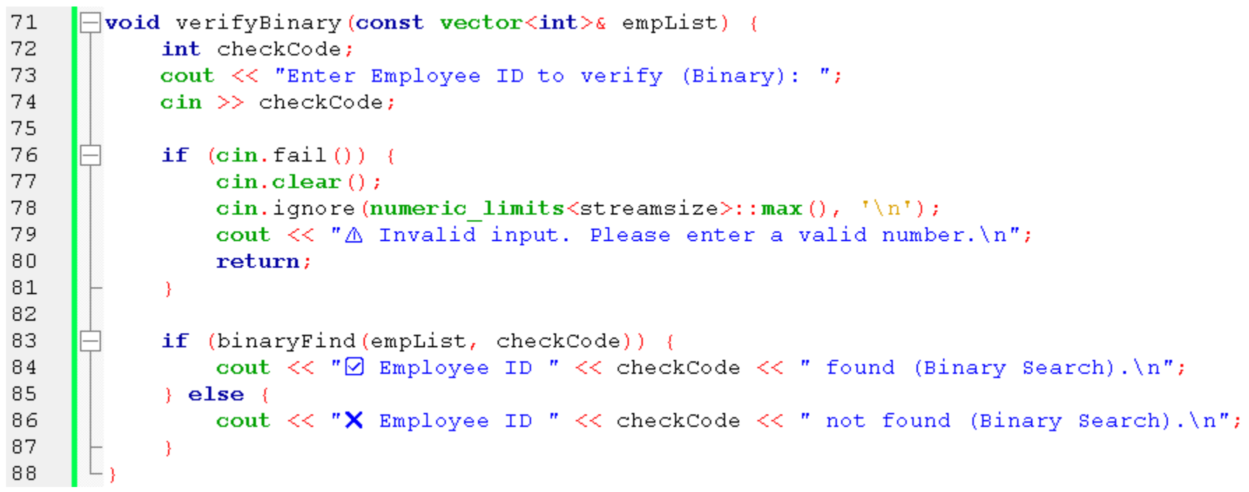
Purpose and Explanation:
So for the clocking employee() allows the user to clock in an employee by entering their ID number. It validates the input to ensure it is a proper 7 digit number and checks if the ID already exists in the system itself. This process helps it to prevent duplicates and ensures accurate employee records.

Function: verifyLinear()



```cpp
void verifyLinear(const vector<int>& empList) {
    int checkCode;
    cout << "Enter Employee ID to verify (Linear): ";
    cin >> checkCode;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a valid number.\n";
        return;
    }

    if (linearFind(empList, checkCode)) {
        cout << "☑ Employee ID " << checkCode << " is currently clocked in.\n";
    } else {
        cout << "❌ Employee ID " << checkCode << " not found.\n";
    }
}
```

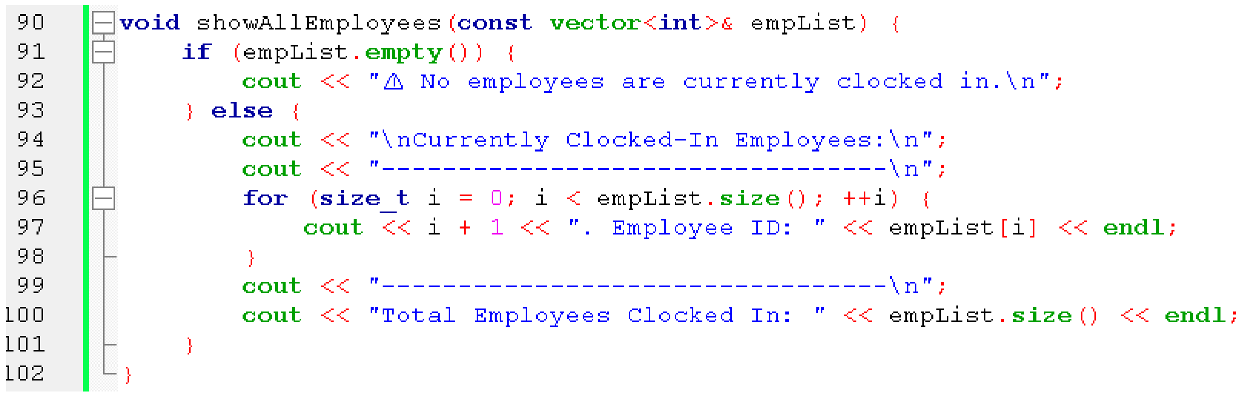
Purpose and Explanation:
This function uses a linear search to verify an employee ID exists in the list. And also it checks each record one by one until it matches or reaches the end. This method is easy to implement and understand, making it for small sets of data. But the downside is that it can be slower when the number increases to a larger number like 1 to 100
Function: verifyBinary()



```cpp
void verifyBinary(const vector<int>& empList) {
    int checkCode;
    cout << "Enter Employee ID to verify (Binary): ";
    cin >> checkCode;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a valid number.\n";
        return;
    }

    if (binaryFind(empList, checkCode)) {
        cout << "☑ Employee ID " << checkCode << " found (Binary Search).\n";
    } else {
        cout << "❌ Employee ID " << checkCode << " not found (Binary Search).\n";
    }
}
```

Purpose and Explanation:
In this function it performs a binary search to locate an employee ID in the list. It separates it into a search range in half, making it more effective for sorting data. This method is much faster than the linear search for larger lists like 1 to 100. It helps also to demonstrate how searching algorithms improve to perform based on the data.

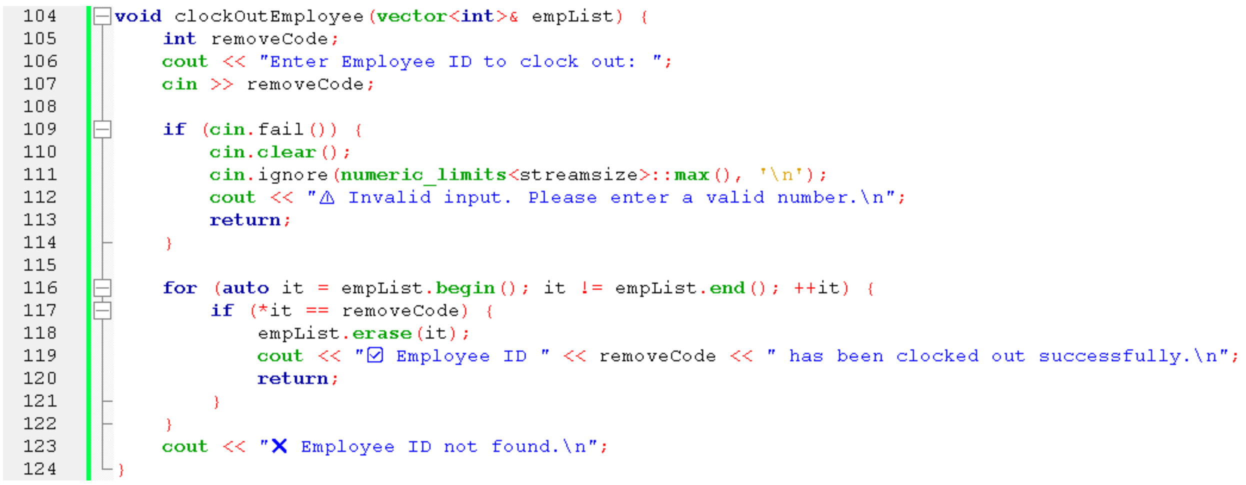
Function: showAllEmployees()



```cpp
void showAllEmployees(const vector<int>& empList) {
    if (empList.empty()) {
        cout << "⚠ No employees are currently clocked in.\n";
    } else {
        cout << "\nCurrently Clocked-In Employees:\n";
        cout << "----------------------------------\n";
        for (size_t i = 0; i < empList.size(); ++i) {
            cout << i + 1 << ". Employee ID: " << empList[i] << endl;
        }
        cout << "----------------------------------\n";
        cout << "Total Employees Clocked In: " << empList.size() << endl;
    }
}
```

Purpose and Explanation:
In this part, it displays all employeeIDs currently clocked in. this list is in order with numbering ans shows the tidal count of the employee. This makes it easy for the user to review who is present and late during that time. It is especially useful for monitoring attendance and ensuring that records are accurate and updated at the same time

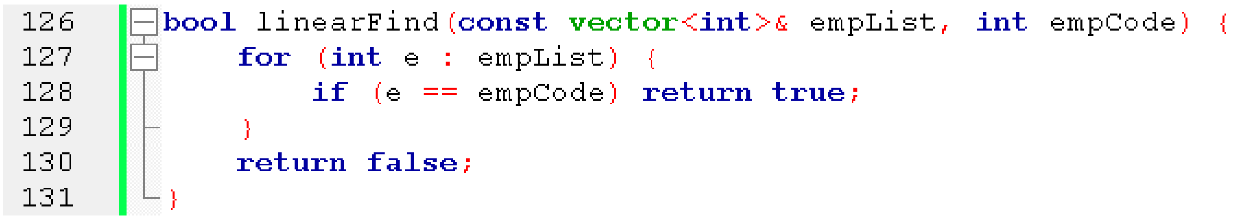
Function: clockOutEmployee()



```cpp
void clockOutEmployee(vector<int>& empList) {
    int removeCode;
    cout << "Enter Employee ID to clock out: ";
    cin >> removeCode;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a valid number.\n";
        return;
    }

    for (auto it = empList.begin(); it != empList.end(); ++it) {
        if (*it == removeCode) {
            empList.erase(it);
            cout << "☑ Employee ID " << removeCode << " has been clocked out successfully.\n";
            return;
        }
    }
    cout << "✗ Employee ID not found.\n";
}
```

Purpose and Explanation:
From the word clockout employee, it means the function is by removing an employee’s ID from the list when they clock out. It searches using the vector and erases the ID once it finds a specific employee’s ID. this ensures that the employee is no longer marked as present, because you remove it. And also preventing dupes as well id miss type Employee’s ID.

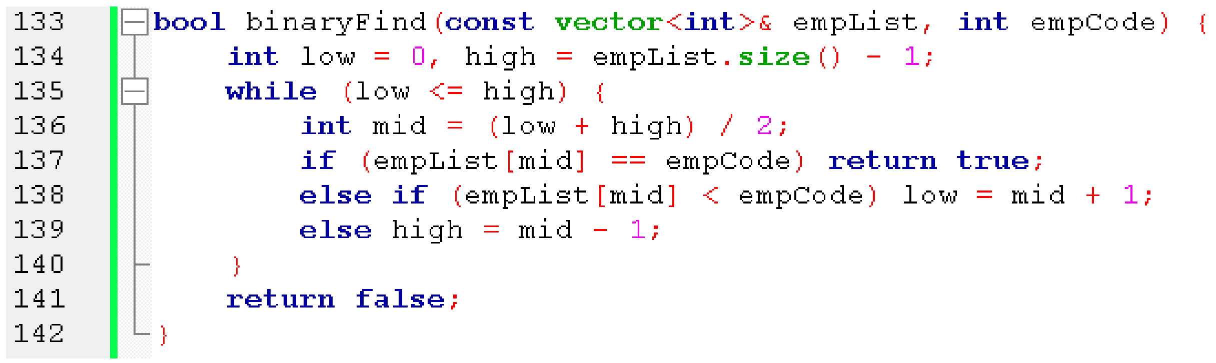
Function: linearFind()



```cpp
bool linearFind(const vector<int>& empList, int empCode) {
    for (int e : empList) {
        if (e == empCode) return true;
    }
    return false;
}
```

Purpose and Explanation:
LinearFind performs a linear search to check a specific employee ID exist in the list it goes each element in the vector one by one and compares it to the input of the user. If it match i will return true if not it will continuing until it finds the specific position.

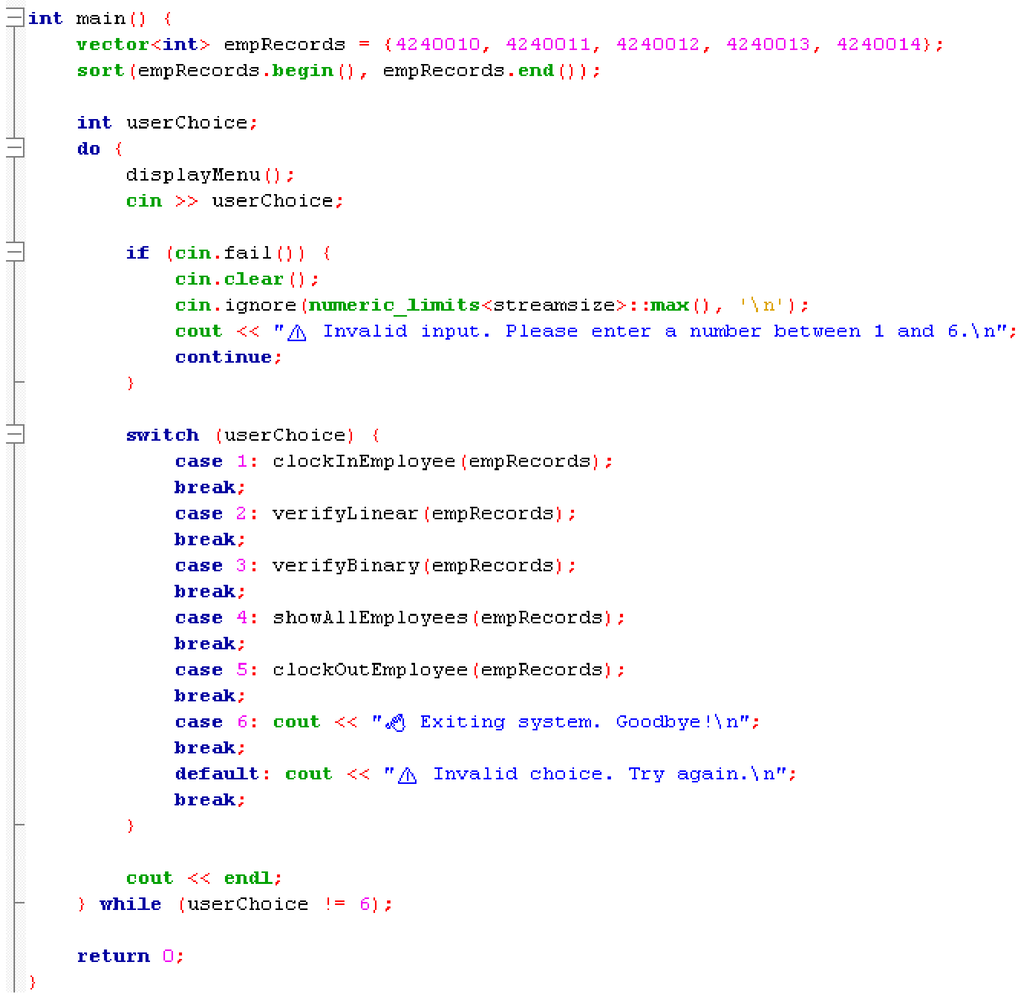
Function: binaryFind()



```cpp
bool binaryFind(const vector<int>& empList, int empCode) {
    int low = 0, high = empList.size() - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (empList[mid] == empCode) return true;
        else if (empList[mid] < empCode) low = mid + 1;
        else high = mid - 1;
    }
    return false;
}
```

Purpose and Explanation:
Same as the linear search function but it works much faster than the linear search function because the list is sorted and repeatedly divided into a range in half until it locates the employee's ID. This makes it ideal for checking the employee;s much faster and efficiently.

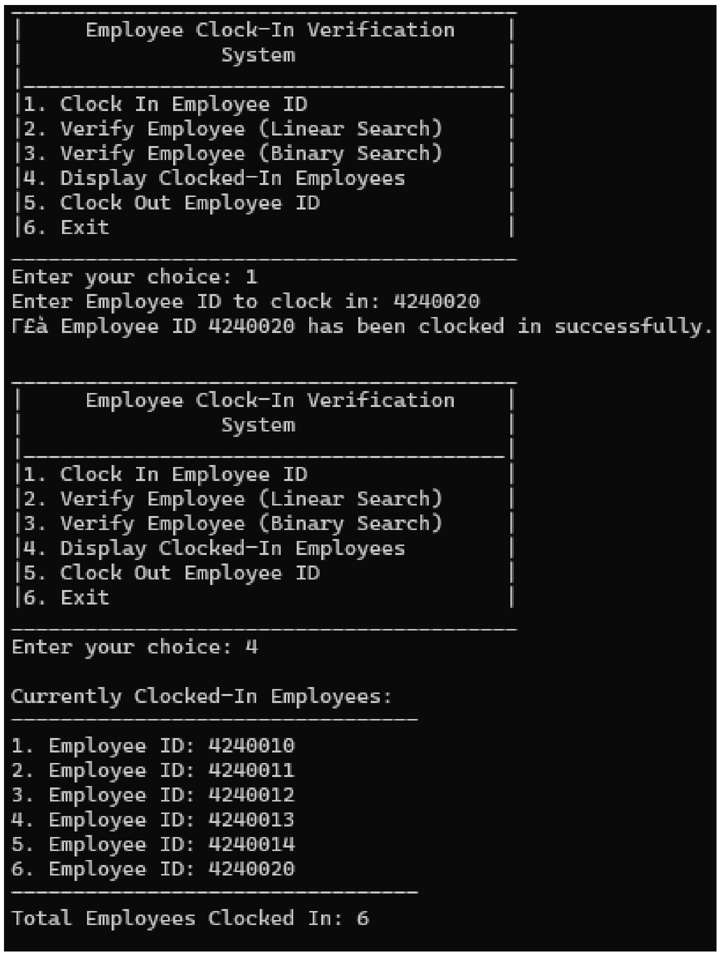
Main function:



```cpp
int main() {
    vector<int> empRecords = {4240010, 4240011, 4240012, 4240013, 4240014};
    sort(empRecords.begin(), empRecords.end());

    int userChoice;
    do {
        displayMenu();
        cin >> userChoice;

        if (cin.fail()) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "⚠ Invalid input. Please enter a number between 1 and 6.\n";
            continue;
        }

        switch (userChoice) {
            case 1: clockInEmployee(empRecords);
            break;
            case 2: verifyLinear(empRecords);
            break;
            case 3: verifyBinary(empRecords);
            break;
            case 4: showAllEmployees(empRecords);
            break;
            case 5: clockOutEmployee(empRecords);
            break;
            case 6: cout << "👋 Exiting system. Goodbye!\n";
            break;
            default: cout << "⚠ Invalid choice. Try again.\n";
            break;
        }

        cout << endl;
    } while (userChoice != 6);

    return 0;
}
```

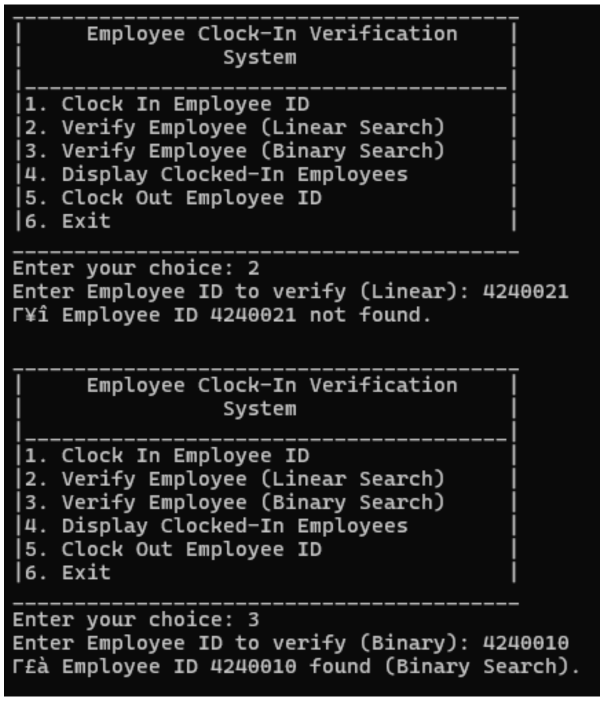
Purpose and Explanation:
In the Main function part this serves as the control center while on the while the displayMenu() is the design version of this function. As you can see there is the vector that automatically displays the given employees to have an already made interaction with the user. And if statement as well if the user wrongly type a letter instead of number in will loop only.

Result:



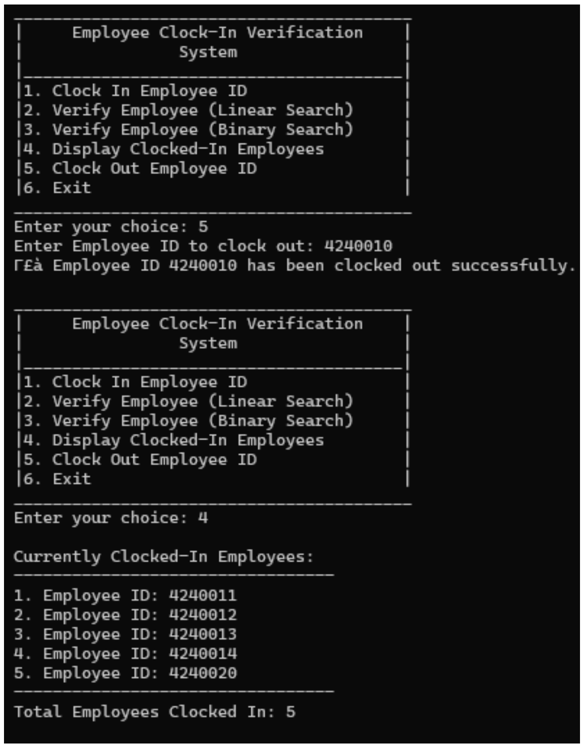
```
 _________________________________________
|     Employee Clock-In Verification     |
|                System                  |
|________________________________________|
|1. Clock In Employee ID                 |
|2. Verify Employee (Linear Search)      |
|3. Verify Employee (Binary Search)      |
|4. Display Clocked-In Employees         |
|5. Clock Out Employee ID                |
|6. Exit                                 |
 _________________________________________
Enter your choice: 1
Enter Employee ID to clock in: 4240020
Γ£à Employee ID 4240020 has been clocked in successfully.

 _________________________________________
|     Employee Clock-In Verification     |
|                System                  |
|________________________________________|
|1. Clock In Employee ID                 |
|2. Verify Employee (Linear Search)      |
|3. Verify Employee (Binary Search)      |
|4. Display Clocked-In Employees         |
|5. Clock Out Employee ID                |
|6. Exit                                 |
 _________________________________________
Enter your choice: 4

Currently Clocked-In Employees:
_________________________________
1. Employee ID: 4240010
2. Employee ID: 4240011
3. Employee ID: 4240012
4. Employee ID: 4240013
5. Employee ID: 4240014
6. Employee ID: 4240020
_________________________________
Total Employees Clocked In: 6
```

On the first part of the output the user select the option1 and enter a employee's ID , Then the system confirms that the employee has been clocked in successfully and adds to the list.
And then the user input 4 to display the list that proves that the new employee's ID is printed on the display function.



```
-----------------------------------------
|     Employee Clock-In Verification    |
|               System                  |
|_______________________________________|
|1. Clock In Employee ID                |
|2. Verify Employee (Linear Search)     |
|3. Verify Employee (Binary Search)     |
|4. Display Clocked-In Employees        |
|5. Clock Out Employee ID               |
|6. Exit                                |
-----------------------------------------
Enter your choice: 2
Enter Employee ID to verify (Linear): 4240021
Γ¥î Employee ID 4240021 not found.

-----------------------------------------
|     Employee Clock-In Verification    |
|               System                  |
|_______________________________________|
|1. Clock In Employee ID                |
|2. Verify Employee (Linear Search)     |
|3. Verify Employee (Binary Search)     |
|4. Display Clocked-In Employees        |
|5. Clock Out Employee ID               |
|6. Exit                                |
-----------------------------------------
Enter your choice: 3
Enter Employee ID to verify (Binary): 4240010
Γ£à Employee ID 4240010 found (Binary Search).
```

Next, the user tests the verification options. The Linear Search correctly identifies that employee 4240021 does not exist, and the Binary Search also reports that 424004010 is not found. When verifying ID 4240010, the system confirms that it is found in the records



```
-----------------------------------------
|     Employee Clock-In Verification    |
|               System                  |
|_______________________________________|
|1. Clock In Employee ID                |
|2. Verify Employee (Linear Search)     |
|3. Verify Employee (Binary Search)     |
|4. Display Clocked-In Employees        |
|5. Clock Out Employee ID               |
|6. Exit                                |
-----------------------------------------
Enter your choice: 5
Enter Employee ID to clock out: 4240010
Γ£à Employee ID 4240010 has been clocked out successfully.

-----------------------------------------
|     Employee Clock-In Verification    |
|               System                  |
|_______________________________________|
|1. Clock In Employee ID                |
|2. Verify Employee (Linear Search)     |
|3. Verify Employee (Binary Search)     |
|4. Display Clocked-In Employees        |
|5. Clock Out Employee ID               |
|6. Exit                                |
-----------------------------------------
Enter your choice: 4

Currently Clocked-In Employees:
---------------------------------
1. Employee ID: 4240011
2. Employee ID: 4240012
3. Employee ID: 4240013
4. Employee ID: 4240014
5. Employee ID: 4240020
---------------------------------
Total Employees Clocked In: 5
```

Afterward, the user clocks out Employee ID 4240010, and when the updated list is displayed, that ID is no longer seen on the display function ,confirming that it is successfully removed. Finally, the user exits the system, and a farewell message is displayed.

Reflection

Through this activity, I got hands-on experience with how Linear Search and Binary Search actually work, and it was pretty cool seeing them in action in a real program. I also learned how to put together functions, loops, and conditions to build a menu system that people can actually use and navigate through easily. Working on the Employee Clock-In Verification System really opened my mind on how the stuff we code can solve actual problems everyday like reviewing in a test for upcoming midterms

Getting the program to run without crashing, especially when someone types in something weird or invalid, took some trial and error. And dealing with the list updates after employees clocked in or out? That had its moments. Binary search also threw me for a loop at first—pun intended. I had to run it a few times before it clicked why it's so much faster than linear search. But honestly, working through all these challenges made me a better problem-solver, and now I feel way more comfortable writing C++ programs that are actually organized and functional.

Code:

```
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
using namespace std;

bool linearFind(const vector<int>& empList, int empCode);
bool binaryFind(const vector<int>& empList, int empCode);
void displayMenu();
void clockInEmployee(vector<int>& empList);
void verifyLinear(const vector<int>& empList);
void verifyBinary(const vector<int>& empList);
void showAllEmployees(const vector<int>& empList);
void clockOutEmployee(vector<int>& empList);

void displayMenu() {
    cout << "_______________________________________________" << endl;
    cout << "|      Employee Clock-In Verification      |" << endl;
    cout << "|                  System                   |" << endl;
    cout << "|____________________________________________|" << endl;
    cout << "|1. Clock In Employee ID                    |\n";
    cout << "|2. Verify Employee (Linear Search)      |\n";
    cout << "|3. Verify Employee (Binary Search)      |\n";
    cout << "|4. Display Clocked-In Employees         |\n";
    cout << "|5. Clock Out Employee ID                  |\n";
    cout << "|6. Exit                                         |\n";
    cout << "_______________________________________________" << endl;
    cout << "Enter your choice: ";
}

void clockInEmployee(vector<int>& empList) {
    int empCode;
    cout << "Enter Employee ID to clock in: ";
    cin >> empCode;

    if (cin.fail() || empCode < 1000000 || empCode > 9999999) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a 7-digit numeric ID.\n";
        return;
    }

    if (linearFind(empList, empCode)) {
        cout << "⚠ Employee ID " << empCode << " is already clocked
in.\n";
    } else {
        empList.push_back(empCode);
        sort(empList.begin(), empList.end());
        cout << "✅ Employee ID " << empCode << " has been clocked in
successfully.\n";
```

```
    }
}

void verifyLinear(const vector<int>& empList) {
    int checkCode;
    cout << "Enter Employee ID to verify (Linear): ";
    cin >> checkCode;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a valid number.\n";
        return;
    }

    if (linearFind(empList, checkCode)) {
        cout << "✅ Employee ID " << checkCode << " is currently clocked
in.\n";
    } else {
        cout << "❌ Employee ID " << checkCode << " not found.\n";
    }
}

void verifyBinary(const vector<int>& empList) {
    int checkCode;
    cout << "Enter Employee ID to verify (Binary): ";
    cin >> checkCode;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a valid number.\n";
        return;
    }

    if (binaryFind(empList, checkCode)) {
        cout << "✅ Employee ID " << checkCode << " found (Binary
Search).\n";
```

```
    } else {
        cout << "❌ Employee ID " << checkCode << " not found (Binary
Search).\n";
    }
}

void showAllEmployees(const vector<int>& empList) {
    if (empList.empty()) {
        cout << "⚠ No employees are currently clocked in.\n";
    } else {
        cout << "\nCurrently Clocked-In Employees:\n";
        cout << "--------------------------------\n";
        for (size_t i = 0; i < empList.size(); ++i) {
            cout << i + 1 << ". Employee ID: " << empList[i] << endl;
        }
        cout << "--------------------------------\n";
        cout << "Total Employees Clocked In: " << empList.size() << endl;
    }
}

    void clockOutEmployee(vector<int>& empList) {
    int removeCode;
    cout << "Enter Employee ID to clock out: ";
    cin >> removeCode;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "⚠ Invalid input. Please enter a valid number.\n";
        return;
    }

    for (auto it = empList.begin(); it != empList.end(); ++it) {
        if (*it == removeCode) {
            empList.erase(it);
            cout << "✅ Employee ID " << removeCode << " has been clocked
out successfully.\n";
            return;
        }
    }
    cout << "❌ Employee ID not found.\n";
}

bool linearFind(const vector<int>& empList, int empCode) {
    for (int e : empList) {
        if (e == empCode) return true;
    }
    return false;
}

bool binaryFind(const vector<int>& empList, int empCode) {
    int low = 0, high = empList.size() - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (empList[mid] == empCode) return true;
        else if (empList[mid] < empCode) low = mid + 1;
        else high = mid - 1;
    }
    return false;
}

int main() {
    vector<int> empRecords = {4240010, 4240011, 4240012, 4240013,
4240014};
    sort(empRecords.begin(), empRecords.end());

    int userChoice;
    do {
        displayMenu();
        cin >> userChoice;

        if (cin.fail()) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "⚠ Invalid input. Please enter a number between 1 and
6.\n";
```

```
            continue;
        }

        switch (userChoice) {
            case 1: clockInEmployee(empRecords);
            break;
            case 2: verifyLinear(empRecords);
            break;
            case 3: verifyBinary(empRecords);
            break;
            case 4: showAllEmployees(empRecords);
            break;
            case 5: clockOutEmployee(empRecords);
            break;
            case 6: cout << "👋 Exiting system. Goodbye!\n";
            break;
            default: cout << "⚠ Invalid choice. Try again.\n";
            break;
        }

        cout << endl;
    } while (userChoice != 6);

    return 0;
}
```