

A dark blue vertical bar runs down the left side of the slide. A blue arrow-shaped banner points to the right from this bar, containing the text 'CSE4002'. In the bottom-left corner, there are several thin, curved lines in dark blue and light grey that sweep upwards and to the right.

CSE4002

# Fundamentals In Programing

Dilshan Gunasekara  
CL/HDCSE/CMU/120/05

## Assignment Cover Sheet

Qualification		Module Number and Title
HD in Computing and Software Engineering /Network Technology and Cyber Security		CSE 4002 Fundamentals in Programming
Student Name & No.		Assessor
Name: Dilshan Gunasekara Stud No:CL/HDCSE/CMU/120/05		Mrs. Nisansala Athapaththu
Hand over date		Submission Date
Assessment type	Duration/Length of Assessment Type	Weighting of Assessment
Coursework	Software Submission and demonstration	100%

Learner declaration	
<p>I, Dilshan Gunasekara, CL/HDCSE/CMU/120/05 &lt;name of the student and registration number&gt; , certify that the work submitted for this assignment is my own and research sources are fully acknowledged.</p>	

Marks Awarded			
First assessor			
IV marks			
Agreed grade			
Signature of the assessor		Date	

---

**FEEDBACK FORM INTERNATIONAL COLLEGE OF BUSINESS  
& TECHNOLOGY**

**Module** : CSE 4002      **Student**

**:**

**Assessor** : Mrs. Nisansala Athapaththu

**Assignment** : Moon Hotel automation system

**Assessor Feedback:**

**Marks Awarded:**

## Table of Contents

Assignment Cover Sheet .....	1
Acknowledgment .....	6
Introduction to Fundamentals in Programming .....	7
Task 01 .....	7
<b>System Requirements Specification and Logical Diagrams</b> .....	7
<b>System Requirements Specifications</b> .....	7
<b>Non – Functional System Requirements</b> .....	8
Logical Diagrams .....	10
Task 02 .....	13
<b>Implementation of Functional C++ Program</b> .....	13
<b>Modularization</b> .....	16
<b>Data Parsing</b> .....	17
<b>User Input Validation and User-Friendliness</b> .....	20
Task 03 .....	22
<b>Test Document Preparation</b> .....	22
<b>Test Plan</b> .....	24
<b>Test Case</b> .....	25
TC1.....	25
References.....	29



---

#### ORIGINALITY REPORT

---

14%

SIMILARITY INDEX

1%

INTERNET SOURCES

0%

PUBLICATIONS

14%

STUDENT PAPERS

---

#### PRIMARY SOURCES

---

1

Submitted to University of Wales Institute,  
Cardiff

Student Paper

11%

---

2

Submitted to National School of Business  
Management NSBM, Sri Lanka

Student Paper

2%

---

3

Submitted to Curtin University of Technology

Student Paper

<1%

---

4

Submitted to University Of Tasmania

Student Paper

<1%

---

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off

## Acknowledgment

I would like to extend my deepest gratitude to Mrs. Nisansala Athapaththu for her invaluable dedication and commitment to teaching at ICBT Campus. Mrs. Athapaththu's passion for education has been a guiding light, illuminating the path for countless students, including myself.

Her profound knowledge, unwavering support, and inspiring mentorship have not only enriched our academic journey but have also shaped our perspectives and aspirations. Mrs. Athapaththu's ability to instill a love for learning and her tireless efforts in nurturing our growth deserve the highest commendation.

Moreover, I would like to express my heartfelt appreciation to ICBT Campus for providing a platform for such exceptional educators like Mrs. Athapaththu to impart knowledge and wisdom.

Lastly, I extend my sincerest gratitude to my friends and family for their unwavering encouragement and support throughout my educational endeavors. Their belief in me has been a constant source of motivation, driving me to strive for excellence.

Together, Mrs. Nisansala Athapaththu, ICBT Campus, and my cherished friends and family have played instrumental roles in shaping my academic and personal development, for which I am profoundly grateful.

## **Introduction to Fundamentals in Programing**

The foundation for comprehending and developing computer programs is the fundamentals of programming. These fundamental ideas cover a variety of ideas, such as learning the grammar and semantics of a programming language, using control structures to ensure logical progression, and using functions and modularization to simplify difficult jobs. Programmers may easily organize and handle data with the help of essential components like data structures, algorithms, and input-output management. Debugging, testing, and error handling principles guarantee the robustness and dependability of programs. The skill set is further enhanced by optional concepts like version control and object-oriented programming. For those stepping foot in the ever-changing realm of software development, programming fundamentals offer a strong foundation that paves the way for more complex subjects and specialized subfields.

## **Task 01**

### **System Requirements Specification and Logical Diagrams**

#### **System Requirements Specifications**

##### **Room Management**

###### **1. View Reserved and Available Rooms**

Permit the receptionists to verify the current status of every room, differentiating between ones that are reserved and those that are not.

###### **2. Search Rooms by Unique IDs**

Enable receptionists to efficiently manage rooms by giving them the ability to instantly discover certain rooms by providing their unique identifiers.

##### **Order Management**

###### **1. View Bookings**

Give front desk staff access to an extensive list of all upcoming reservations that includes pertinent information such as guest names, room categories, and reservation dates.



## 2. Prepare Bills

Provide a tool that lets front desk staff create guest invoices based on reservations, guaranteeing billing accuracy and streamlining the check-out procedure.

### **User Authentication**

#### 1. Receptionist Log In

Provide a secure login process wherein receptionists must enter their own usernames and passwords to access the system. This will guarantee that only authorized staff members are able to handle reservations for rooms and billing.

### **Bill Display**

#### 1. Display The Bill Amount

Include an easy-to-read display of the entire bill amount for every reservation so that receptionists and visitors can both comprehend and validate the financial transactions.

### **Non – Functional System Requirements**

Non-functional requirements are features of a system that specify its limitations, attributes, and features instead of particular actions. Performance, usability, security, and other global system features are frequently covered by these criteria. These are Requirements for Moon Hotel Management System.

#### 1. **Compatibility**

For any web-based interfaces, the system should work with the majority of operating systems and web browsers that receptionists frequently use.

Payment gateways and other external systems should integrate seamlessly.

#### 2. **Compliance**

Industry standards and any data protection laws should be complied with by the system.

To guarantee quality and dependability, it should follow best practices for code and design.

### **3. Performance**

Under typical working settings, the system needs to react to user input in a span of two seconds.

At least fifty receptionists should be able to access the system simultaneously without experiencing noticeably worse performance.

### **4. Security**

Mechanisms for user permission and authentication should adhere to industry standards for security.

Encrypting customer data is recommended for both transmission and storage, including personal information and booking details.

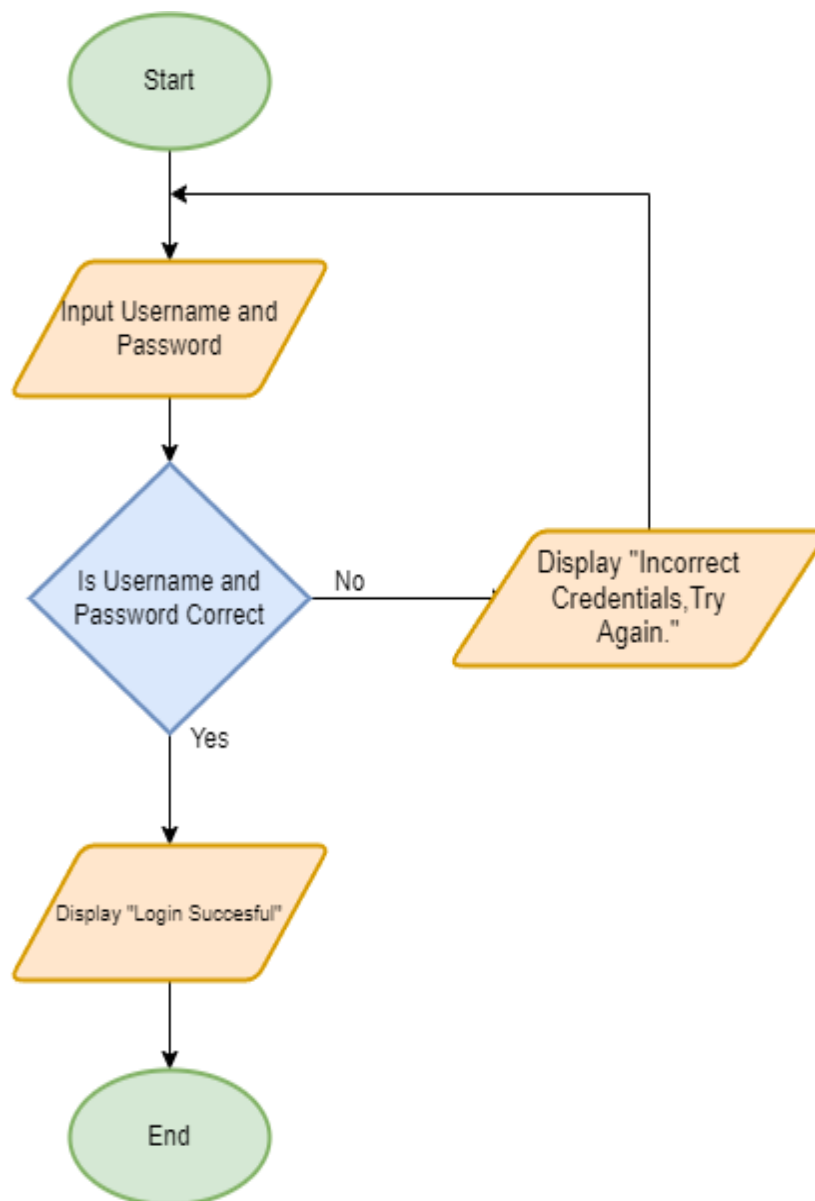
### **5. Usability**

In order to ensure that receptionists can complete routine jobs with little training, the user interface should be simple to understand.

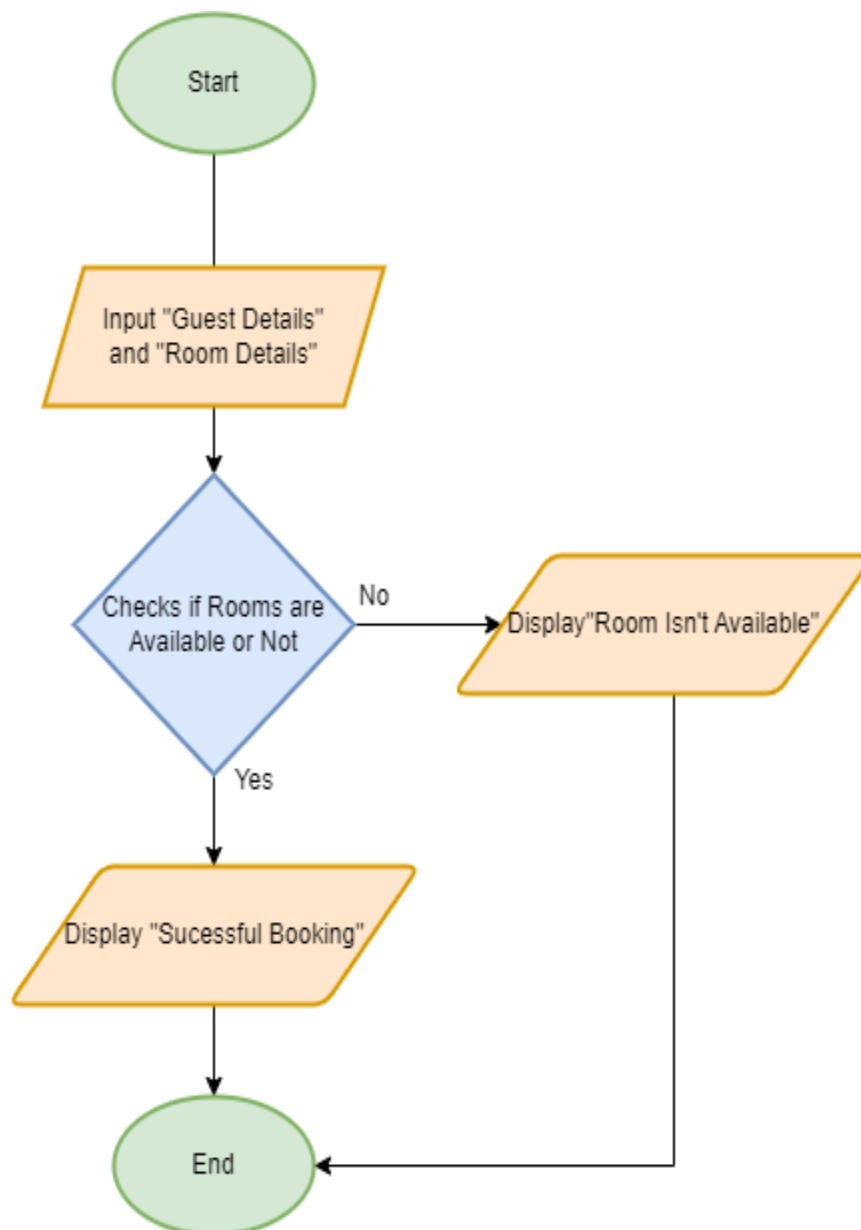
The system must adhere to accessibility guidelines and be usable by people with impairments.

## Logical Diagrams

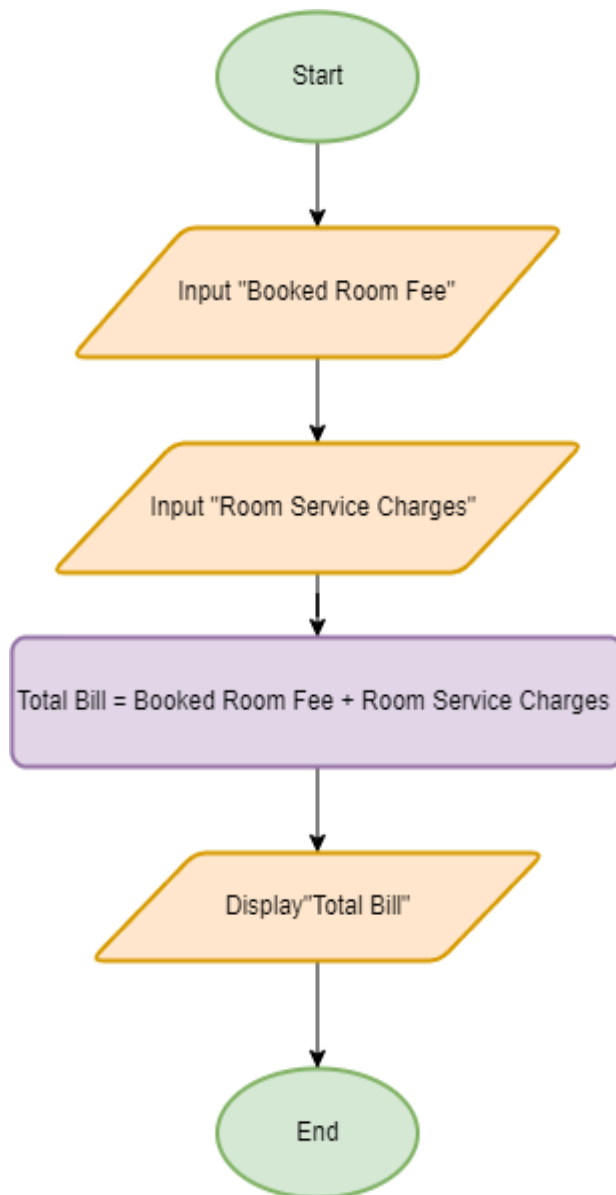
### Diagram For User Authentication



## Diagram For Room Management



**Diagram For Order Management and Bill Display.**



## Task 02

### Implementation of Functional C++ Program

#### 1. Sequence Structure

```
int main() {  
    // Sequence structure  
    const int numRooms = 5;  
    Room rooms[numRooms] = { {1, false}, {2, false}, {3, false}, {4, false}, {5, false}  
  
    int choice;  
    do {  
        choice = displayMenu();  
  
        // Processing user choice based on the menu  
        switch (choice) {  
            // Cases for different menu options  
        }  
  
    } while (choice != 5);  
  
    return 0;  
}
```

In programming, sequence structure is the basic idea of carrying out statements in a predefined order and moving linearly from one instruction to the next within a program. The basic flow of control in a program is represented by this structure, where statements are executed in the order they are written in the code, sequentially. It follows a sequential process in which there is no branching or need for decision-making; instead, one action leads to the next. Sequence structure makes a task or process intuitive and simple to follow by essentially reflecting the steps' natural

evolution. It provides a simple and unambiguous logic for structuring program flow, acting as the cornerstone upon which more intricate program structures are constructed. The flow of control within a sequence structure is determined by the order in which the statements are performed, guaranteeing that each statement is carried out in the correct order to produce the intended result. All things considered, sequence organization is essential for structuring program logic and enabling the methodical and orderly completion of tasks.

## 2. Selection Structure (Switch-Case)

```
int choice;
do {
    choice = displayMenu();

    switch (choice) {
    case 1:
        viewRooms(rooms, numRooms);
        break;
    case 2:
        bookRoom(rooms, numRooms);
        break;
    case 3:
        viewBookings(rooms, numRooms);
        break;
    case 4:
        prepareBill();
        break;
    case 5:
        std::cout << "Exiting the program. Thank you!\n";
        break;
    default:
        std::cout << "Invalid choice. Please try again.\n";
    }
} while (choice != 5);

return 0;
```

A selection structure enables a program to make decisions based on predetermined conditions. It is commonly done using if-else statements or switch-case constructions. Essentially, it allows the software to select amongst different actions based on the truth value of a certain condition.

Here are some key points about selection structure:

**Decision Making:** By analyzing one or more circumstances and executing several sets of statements depending on how these conditions turn out, selection structures help programs make decisions easier to understand and implement.

**Conditional Branching:** Depending on the truth value of the conditions stated in the selection structure, the program branches into various execution routes.

**Branching Constructs:** If-else statements or switch-case constructs are used in the majority of programming languages to design selection structures. Based on the assessment of conditions, these components enable the program to select among several possibilities or actions.

**Multiple Alternatives:** The selection structure allows for the execution of numerous code blocks by the program depending on the satisfaction of various requirements.

**Controlled Execution:** Selection structures make programs more flexible and adaptive by regulating the flow of execution according to conditions. This enables programs to react quickly and effectively to a variety of circumstances.

Selection structure is fundamental for implementing decision-making logic in programs, allowing them to adapt their behavior based on specific conditions or criteria.

### 3. Repetition-Structure(do-while-loop)

```
do {  
    // Display menu and get user choice  
    choice = displayMenu();  
    // Process user choice  
    switch (choice) {  
        // Cases for different menu options  
    }  
} while (choice != 5); // Repeat until the user chooses to exit
```

A repetition structure, sometimes referred to as a loop, enables a computer to run a series of instructions again up until a predetermined condition is satisfied. One such repetition structure is the do-while loop, which runs its code block once at least and then repeatedly as long as a given condition holds true.

Here are some key points about the do-while loop:



**Initial Execution:** The do-while loop, in contrast to other loop structures, runs its code block at least once before verifying the loop condition. This guarantees that the code block is run at least once, irrespective of the loop condition's starting state.

**Conditional Execution:** The do-while loop assesses the loop condition following the initial execution. The loop iterates and runs the code block once again if the condition is true. Until the condition is no longer true, this process is repeated.

**Exit Condition:** When the loop condition evaluates to false, the loop comes to an end. As a result, the number of times the loop runs can vary based on the starting condition and subsequent assessments.

**Use Cases:** You can use the do-while loop to make sure a block of code runs at least once, regardless of the condition, and then repeat the execution of the piece of code based on a condition.

The do-while loop comes in handy when you have to carry out an operation that needs to be completed in at least one iteration, such as processing a list of potentially empty items or asking the user for input until they provide valid input. It offers a strong and adaptable mechanism for integrating repeated operations into programs.

## Modularization

The code exhibits modularization by breaking up the functionality into several modules or functions, each in charge of carrying out particular duties:

### Room Management Module

```
// Function to view available rooms
void viewRooms(const Room rooms[], int numRooms) {
    // Function body
}

// Function to book a room
void bookRoom(Room rooms[], int numRooms) {
    // Function body
}
```

Functions like `viewRooms()` and `bookRoom()` are responsible for handling tasks related to managing rooms.

## Order Management Module

```
// Function to view booked rooms
void viewBookings(const Room rooms[], int numRooms) {
    // Function body
}
```

The viewBookings() function is responsible for managing bookings.

## Bill Display Module

```
// Function to prepare a bill
void prepareBill() {
    // Function body
}
```

The prepareBill() function prepares and displays the bill.

## Data Parsing

Function arguments are an efficient way to handle data flow across modules, enabling communication and data sharing:

### Passing Room Details to viewRooms():

```
void viewRooms(const Room rooms[], int numRooms) {
    // Access room details from the array 'rooms[]' and display them
}
```

To access and show room information, the viewRooms() function takes as inputs an array of room details (rooms[]) and the number of rooms (numRooms).

### Passing Room Details to bookRoom():

```
void bookRoom(Room rooms[], int numRooms) {  
    // Modify room details in the array 'rooms[]' to indicate reservation  
}
```

Similarly, the bookRoom() function also receives an array of room details (rooms[]) and the number of rooms (numRooms) as arguments, allowing it to update the reservation status of rooms.

Overall, the code given is more organized, readable, and maintainable thanks to the modularization and efficient data passing that guarantee each module wraps a particular functionality and communicates with other modules as needed.

## Data Handling Techniques

### Array Usage

The rooms[] array is appropriately used to store room details. Each element of the array represents a room, and the array serves as a container for organizing and managing room-related data.

```
const int numRooms = 5; // Number of rooms in Moon Hotel  
Room rooms[numRooms] = { {1, false}, {2, false}, {3, false}, {4, false}, {5, false} }
```

The rooms[] array is defined in this code sample to store room information, including the room ID and reservation status. Every entry in the array represents a single hotel room and is an instance of the Room struct. Initial reservation statuses and distinct room IDs are included in the array's initialization.

## Struct Usage

A struct named Room is defined to represent individual rooms, encapsulating attributes such as room ID and reservation status:

```
// Structure to represent a room
struct Room {
    int roomId;
    bool isReserved;
};
```

This code sample establishes the blueprint for a room using the Room struct, which has two attributes: roomId, an integer that serves as the room's unique identification, and isReserved, a boolean that indicates whether the room is now available or reserved (false). A single hotel room is represented by each instance of the Room struct, which contains all of the room's characteristics in a single data structure.

Effectively managing room details, the rooms[] array offers a hierarchical container for arranging and retrieving room-related data.

The Room struct allows room data to be represented coherently and logically by encapsulating attributes specific to each rooms. The efficiency, readability, and maintainability of the code are improved by this organized method of processing data.

## User Input Validation and User-Friendliness

### Validation

Although the code relies on simple validation, like validating menu choices, more sophisticated validation methods can be used to improve data integrity and guard against mistakes. As an illustration:

**Checking for invalid input types:** To avoid type mismatches and associated runtime issues, make sure user inputs are of the expected type (integer, for menu options, for example).

**Validating input ranges:** To avoid out-of-range errors, confirm that user inputs are within permitted ranges. For example, when a user books a room, make sure the room ID they enter is genuine and falls within the range of room IDs that are accessible.

**Managing edge circumstances:** Take into account probable mistake scenarios and edge cases, such as situations in which a user enters non-numeric characters or tries to make a reservation for a room that already exists.

### User-Friendliness

In order to improve the user experience, user-friendliness must be improved by offering more precise instructions, educational messaging, and simple interactions. The following ideas will help to improve user-friendliness:

**More precise instructions:** Give users precise, easy-to-follow directions or prompts to help them navigate the features of the software. To help users understand the purpose of each option, for instance, provide brief descriptions of each one while presenting the menu.

**Informative error messages:** When users enter incorrect data or make mistakes, give them informative feedback as an alternative to generic error messages. Provide information on the type of mistake and how users can fix it. For example, when a user books a hotel, input an invalid ID; show the user a notice stating that the ID is invalid and ask them to submit a legitimate ID.

**Feedback on input validation:** Tell users right away if their input is incorrect and ask them to submit the information again. Give users precise instructions on the acceptable format or range of acceptable inputs to help them supply accurate data.

Through the use of strong input validation techniques, as well as clear instructions and helpful feedback, the application can be made more user-friendly overall and reduce the possibility of errors or misunderstandings. This strategy makes it easier for users to engage with the application and makes the UI more intuitive and user-friendly.

## **Navigation, Accuracy, Creativity, Completeness**

### **Navigation**

Users can navigate the software with ease thanks to its menu-driven structure. Users don't require complicated commands or inputs to choose the necessary functionality because it is presented as a clear menu with numbered possibilities. This method makes it easier for users to interact with the application and guarantees that they can easily browse through its various features.

### **Accuracy**

Based on user input, the application accurately completes necessary operations such as showing available rooms, reserving rooms, viewing bookings, and creating bills. Every function performs its intended role with accuracy, guaranteeing that users obtain accurate and trustworthy information about room availability, bookings, and invoicing.

### **Creativity**

Although the program's implementation is simple, there is opportunity for improvement in terms of user interaction, error management, and feature additions. Creative error messages and prompts, for instance, can make the user experience more interesting. Furthermore, adding capabilities like room searches or dynamic room management could improve the program's usefulness and user experience.

### **Completeness**

By including essential features like booking, bill preparation, and room administration, the system satisfies the needs. Still, there is room for improvement to make the system more robust and complete. For example, adding capabilities like data permanence, error reporting, or user authentication could make the system more complete and offer a more all-encompassing way to manage hotel operations.

Overall, the application shows adequate navigation, correctness, and completeness in fulfilling the objectives; however, there is room for improvement in terms of functionality and user

experience. The program can become more adaptable, user-friendly, and comprehensive in meeting the needs of the hotel management system by adding creative aspects and extra functions.

### **Task 03**

#### **Test Document Preparation**

A methodical way of describing the different situations, circumstances, and activities that must be tested in a software program is called test case documentation. It offers a methodical framework for confirming the software's dependability, performance, and functionality in many scenarios. Typically, test case documentation consists of the following elements:

**Test Case ID:** A special number that is linked to every test case for tracking and simple reference.

**Test Case Name:** A succinct name or heading that expresses the goal or intention of the test case.

**Description:** A thorough explanation of the test case that includes information about the particular scenario, state, or capability being examined.

**Test Data:** The parameters or input data needed to carry out the test scenario. This covers any requirements or preconditions needed in order for the test to be carried out properly.

**Expected Result:** The anticipated behavior or result of the software upon execution of the test case. This acts as a reference point for assessing the real test results that were attained.

**Actual Result:** The software's actual performance as observed during testing. To ascertain if the test case has succeeded or failed, this is compared to the anticipated outcome.

**Conclusion:** A synopsis or evaluation of the test case's execution that includes any problems, observations, or suggestions for improvement.

All things considered, test case documentation is essential to the software development lifecycle since it guarantees the software product's quality, performance, and dependability. It assists in reducing risks, verifying requirements, and providing end customers with a solid and trustworthy solution.

### **Areas to Improve**

#### **1. Improved User Interface (UI):**

Improve the user interface's usability and aesthetic appeal. To enhance the overall user experience, think about implementing simple layouts and contemporary design concepts.

To make it more intuitive and user-friendly, enhance the menu options and navigation. Use clear labeling and visual clues to help users navigate the program with ease.

## **2. Security and Authentication:**

Enhance security protocols to protect user data and login credentials. To improve security, use strong authentication techniques like multi-factor authentication.

Apply role-based access control (RBAC) to user roles to limit access to sensitive functions. This guarantees that some system functions can only be carried out by authorized users.

## **3. Handling Errors and Validation:**

Put in place thorough error handling procedures to give users informative error messages when there are system or input issues. Make sure error messages are understandable, instructive, and provide guidance on how to fix the problem for users.

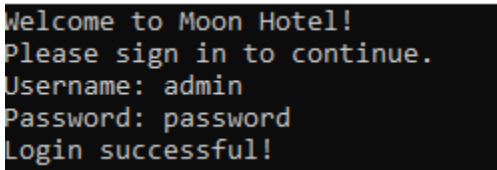
Thoroughly verify user inputs to guard against inaccurate data entry and guarantee data integrity. In order to reduce the possibility of processing incorrect data, apply input validation tests for each and every user input.



## Test Plan

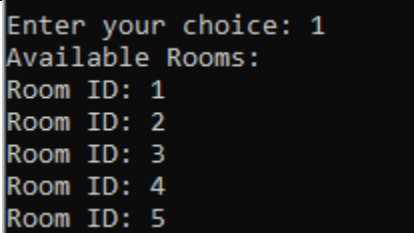
Test Case ID	Test Case Name	Description	Expected Result
TC1	User Login	The User will be Logging on from here	Directed to the main menu. If successful
TC2	Display the available Hotel Rooms	The user can view the available Hotel Rooms	Can view the available Hotel Rooms when successful.
TC3	Calculate and print the bill	Calculate the bill and printing it.	Along with the Room name, ID, and unit price, the bill's final amount is also shown.
TC4	User Log out	User will be Logging out from here	prompt verifying successful logout; display login page.
TC5	Help	There is a help menu where you may get instructions on how to use the system.	Instructions on how to operate the system will be shown.
TC6	Exit	Exits the program	Exits the program

## Test Case

<b>Test Case ID</b>	TC1
<b>Test Case Name</b>	User Login
<b>Operation</b>	Confirming and validating the provided username and password
<b>Test Data</b>	Username: admin Password: password
<b>Expected Results</b>	Directed to the main menu. If successful
<b>Preview</b>	 <pre>Welcome to Moon Hotel! Please sign in to continue. Username: admin Password: password Login successful!</pre>
<b>Conclusion</b>	Login Function works properly

TC1

TC2

<b>Test Case ID</b>	TC2
<b>Test Case Name</b>	Display the available Hotel Rooms.
<b>Operation</b>	Displaying the available Hotel Rooms.
<b>Test Data</b>	Option 1
<b>Expected Results</b>	Can view the available bakery items when successful.
<b>Preview</b>	 <pre>Enter your choice: 1 Available Rooms: Room ID: 1 Room ID: 2 Room ID: 3 Room ID: 4 Room ID: 5</pre>
<b>Conclusion</b>	Display available Hotel Rooms function works properly

### TC3

<b>Test Case ID</b>	TC3
<b>Test Case Name</b>	Calculate and print the bill
<b>Operation</b>	Calculating the overall cost of the order.
<b>Test Data</b>	Option 2 Item ID: 1 Item ID: 2 Item ID: 0 (to close the bill and to get the final total)
<b>Expected Results</b>	Along with the item's name, unit price, the bill's final amount is also shown.
<b>Preview</b>	<pre>Enter your choice: 2 Enter the Room ID to book: 1 Room booked successfully! ===== Moon Hotel Menu ===== 1. View Rooms 2. Book Room 3. View Bookings 4. Prepare Bill 5. Help 6. Logout 7. Exit ===== Enter your choice: 2 Enter the Room ID to book: 2 Room booked successfully! ===== Moon Hotel Menu ===== 1. View Rooms 2. Book Room 3. View Bookings 4. Prepare Bill 5. Help 6. Logout 7. Exit ===== Enter your choice: 4 Total Bill Amount: \$200.00 Bill saved to 'bill.txt' file.</pre>
<b>Conclusion</b>	Successfully obtaining the final total amount of the bill

## TC4

<b>Test Case ID</b>	TC4
<b>Test Case Name</b>	User log out
<b>Operation</b>	User logging out of the system
<b>Test Data</b>	Option 6
<b>Expected Results</b>	Prompt verifying successful logout; display login page.
<b>Preview</b>	<pre>===== Moon Hotel Menu ===== 1. View Rooms 2. Book Room 3. View Bookings 4. Prepare Bill 5. Help 6. Logout 7. Exit ===== Enter your choice: 6 Logged out successfully. Welcome to Moon Hotel! Please sign in to continue. Username:</pre>
<b>Conclusion</b>	Logging out of the system successfully occurred

## TC5

<b>Test Case ID</b>	TC5
<b>Test Case Name</b>	Help
<b>Operation</b>	Displaying alternatives for help so that you can utilise the system properly
<b>Test Data</b>	Option 5
<b>Expected Results</b>	Instructions on how to operate the system will be shown.
<b>Preview</b>	<pre>Enter your choice: 5 This is a Moon Hotel Management System. You can view available rooms, book a room, view bookings, prepare bill, and more. If you need further assistance, please contact hotel staff. Moon Hotel Menu</pre>
<b>Conclusion</b>	Help function works properly

## TC6

<b>Test Case ID</b>	TC6
<b>Test Case Name</b>	Exit
<b>Operation</b>	Exits the program successfully
<b>Test Data</b>	Option 7
<b>Expected Results</b>	Program is concluded saving the data
<b>Preview</b>	<pre>Enter your choice: 7 Exiting the program. Thank you!  C:\Users\Sandarv\Downloads\Assignment 121\HotelMangementSystem\x64\Debug\HotelMangementSystem.exe (process 15832) exited with code 0. To automatically close the console when debugging stops, enable Tools-&gt;Options-&gt;Debugging-&gt;Automatically close the console when debugging stops. Press any key to close this window . . .</pre>

<b>Conclusion</b>	Exit function works properly
-------------------	------------------------------

## References

*Educative answers - trusted answers to developer questions* (no date) *Educative*. Available at: <https://www.educative.io/answers/what-are-the-basic-fundamental-concepts-of-programming> (Accessed: 13 March 2024).

GfG (2023) *Introduction to C++ programming language*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/introduction-to-c-programming-language/> (Accessed: 13 March 2024).

*Programming fundamentals: Introduction to programming - programming - beginner* (no date) *Skillsoft*. Available at: <https://www.skillsoft.com/course/programming-fundamentals-introduction-to-programming-1e866b9b-f229-42d5-8839-acff84da523e> (Accessed: 13 March 2024).

S, R.A. (2023) *Er diagrams in DBMS: Entity relationship diagram model: Simplilearn*, *Simplilearn.com*. Available at: <https://www.simplilearn.com/tutorials/sql-tutorial/er-diagram-in-dbms> (Accessed: 13 March 2024).

