Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report

on

"Billing System"

[Code No: COMP 202]

For partial fulfillment of Second Year/first Semester in Computer Engineering

Submitted By:

Arip Sunar [038006-24]

Submitted To:

Er. Sagar Acharya

Department of Computer Science and Engineering

Submission Date: 2026-02-25

Bona fide Certificate

This is to certify that the project entitled

" Billing system "

is a bonafide work carried out by the following students in partial fulfillment of the requirements for the Second Year /First Semester of the Bachelor of Engineering in Computer Engineering, under the guidance of Er. Sagar Acharya, Department of Computer Science and Engineering.

Name of Students :

Arip Sunar

The project work has been carried out with sincerity, dedication, and to the satisfaction of the department, adhering to the academic guidelines.

Project Supervisor

_____

Er. Sagar Acharya

Department of Computer Science and Engineering

Date: 2026-02-25

Acknowledgement

I would like to express my sincere gratitude to my respected teacher, **Sagar Acharya**, for his continuous guidance, encouragement, and valuable support throughout the development of this project.

His clear explanations of Data Structures and Algorithms (DSA) concepts and their practical implementations in C++ greatly helped me understand how theoretical knowledge can be applied in real-world applications. His teaching approach made complex topics like vectors, file handling, searching algorithms, and structured programming easier to grasp and implement effectively in this BillingSystem project.

I am truly thankful for his mentorship, constructive feedback, and constant motivation, which played a significant role in successfully completing this project.

I would also like to extend my sincere appreciation to the Department of Computer Science and Engineering for providing me with the opportunity and platform to apply my theoretical knowledge in a practical and meaningful way. The academic environment, resources, and encouragement provided by the department were instrumental in helping me carry out and complete this project successfully.

I would also like to thank my institution and everyone who directly or indirectly supported me during this project work.

Thank You

Abstract

The **BillingSystem** is a desktop-based billing management application developed using C++ and the raylib library for graphical user interface design. The main objective of this project is to implement core Data Structures and Algorithms (DSA) concepts in a practical, real-world application while providing an interactive and user-friendly billing system.

The system includes a secure user authentication module with login and signup functionality, where user data is stored persistently using file handling techniques. The billing module allows users to add, update, and delete items dynamically, calculate the total bill in real time, detect duplicate items, and generate formatted receipts. All billing items are stored using dynamic arrays implemented through `std::vector`, demonstrating practical use of data structures.

The project follows a modular multi-file architecture to ensure maintainability and scalability. Various DSA concepts such as structures, dynamic arrays, linear search, file I/O operations, string parsing, and enumerations are applied throughout the system. The graphical interface is designed with custom UI components, smooth screen transitions, and a modern themed layout to enhance user experience.

Overall, this project successfully integrates theoretical DSA knowledge with practical implementation, showcasing problem-solving skills, modular programming, and GUI development using C++.

Keywords :c++, Raylib, DSA, Billing Management System, Dynamic Array ,Linear Search, Receipt generation, OOP

Table of Contents

List of Figures

Acronyms/Abbreviations

The list of all abbreviations used in the documentation are as follows :

GUI – Graphical User Interface

DSA – Data Structures and Algorithm

OOP – Object Oriented Programming

PK – Primary key

ER – Entity Relationship

UI– User Interface

I/O – Input/Output

Chapter 1 Introduction

1.1 Background

In today's fast-paced business environment, accurate and efficient billing is essential for both small and large enterprises. Manual billing methods, such as handwritten bills or spreadsheets, are prone to errors, time-consuming, and often lack proper record-keeping, which can lead to mistakes in calculations, item duplication, or missing receipts. To overcome these challenges, computerized billing systems have become widely adopted, automating item entry, total calculation, and receipt generation while providing a structured way to store user and transaction data securely. The BillingSystem project is a desktop application developed in C++ using the Raylib GUI library, designed to demonstrate the practical application of Data Structures and Algorithms (DSA) in a real-world scenario. In its current implementation, the system uses `std::vector` to store users and cart items, with linear search employed for user authentication and duplicate item detection. To further enhance efficiency and performance, advanced DSA concepts can be applied: using a hash table for user authentication allows instant lookup of usernames, replacing linear search and significantly improving login speed; implementing a linked list for cart management enables efficient insertion and deletion of items anywhere in the list without shifting elements, which is especially useful for dynamic billing; and applying sorting algorithms to arrange items by name, quantity, or price before generating receipts ensures a clean and organized display, enhancing usability. By integrating these techniques, the project not only reinforces theoretical knowledge of data structures and algorithms but also provides practical experience in building a modular, efficient, and user-friendly billing system that bridges the gap between academic learning and real-world software development.

1.2 Objectives

The primary objectives of " **Billing System**" are :

- The project develops a desktop billing system with secure login, dynamic cart management, real-time total calculation, and receipt generation, using hash tables for fast user lookup, linked lists for cart management, and sorting algorithms for organized display.
- To implement a secure user authentication system with login and signup functionality.
- To provide dynamic cart management with add, delete, and update operations.
- To calculate real-time totals and generate formatted receipts for billing.

1.3 Motivation and Significance

The motivation behind the BillingSystem project arises from the need to create an accurate, efficient, and user-friendly billing solution for businesses and individuals. Manual billing processes are often error-prone, time-consuming, and difficult to manage, especially when handling large numbers of items or multiple customers. This project aims to automate billing operations, providing secure user authentication, dynamic cart management, and real-time total calculation with receipt generation, all within an interactive GUI built using the Raylib library. The significance of this project lies in its use of advanced DSA concepts to improve performance and usability: hash tables enable fast user lookup and duplicate detection, linked lists allow efficient addition and deletion of cart items without shifting data, and sorting algorithms ensure organized item display for a professional and readable billing interface. Furthermore, file I/O operations provide persistent storage of users and billing data, making the system reliable and practical. Overall, the project not only delivers a functional billing application but also demonstrates the integration of theoretical knowledge with real-world software development, giving hands-on experience in modular programming, algorithmic problem solving, and GUI design,

while laying a foundation for more advanced billing or inventory management systems.

Chapter 2 Related Works

Several researchers, developers, and students have created billing systems and point-of-sale applications that meet common business needs. Many existing systems emphasize basic billing functions, like user login, product entry, bill calculation, and receipt printing. For example, traditional console-based billing applications have often been developed using C or C++ with simple text input and output. Items are stored in arrays, and billing operations are carried out one after the other. These systems typically use basic data structures. They are effective for small inventories but lack modern user interfaces and efficient data management.

Other implementations have expanded billing features to include product databases, inventory tracking, and report generation. These are usually built with languages like Java or Python and frameworks such as Swing or Tkinter for GUI development. Such systems often store data in relational databases using SQL. This improves scalability and persistence, but their architectural complexity increases with database integration. Mobile billing apps developed on platforms like Android emphasize portability and integration with cloud storage.

In terms of algorithm improvements, some academic and open-source projects have looked into using hash tables for quick product or user lookup, linked lists for dynamic sales lists, and sorting algorithms to organize itemized bills. However, few student projects incorporate these advanced data structure ideas into a unified desktop

GUI application using C++ and Raylib, as this project does. Most educational billing systems remain console-based or depend on higher-level GUI frameworks without focusing deeply on algorithm optimization.

Compared to existing works, the BillingSystem project uniquely combines a custom graphical interface, persistent file storage, and practical use of fundamental and advanced data structure techniques. It employs hash tables for user authentication, linked lists for cart management, and sorting algorithms for clear display, all in a modular, flexible way. This improves both system performance and educational value, making it more efficient than basic implementations and more algorithmically sophisticated than typical GUI applications designed for academic use.

Chapter 3 Design and Implementation

3.1 Implementation

The BillingSystem is implemented as a desktop application using C++ and the Raylib library for creating the graphical user interface. The GUI consists of three main screens: Login, Signup, and Billing, with smooth screen transitions managed through program state control. Custom UI components such as buttons, input boxes, and password fields are developed using Raylib's drawing and input-handling functions to provide an interactive and user-friendly experience.

The system functions include secure user authentication, dynamic addition and deletion of cart items, real-time total calculation, and receipt generation. Data structures such as vectors, hash tables, linked lists, and sorting algorithms are used to manage users and billing items efficiently.

For data storage, the program uses file-based persistence instead of a traditional database. User credentials and billing information are stored and retrieved using file I/O operations (e.g., users.dat and receipt.txt), which act as a simple database to maintain data consistency across sessions.

3.1.1 Implementation Planning

The implementation of the BillingSystem project was carefully planned to demonstrate the practical application of Data Structures and Algorithms (DSA) in solving a real-world billing problem. As this project was assigned as a mini academic task, the primary focus was on selecting appropriate data structures that ensure efficiency, scalability, and logical organization of data.

During the planning phase, the system was divided into major functional modules: user authentication, cart management, billing calculation, and receipt generation. For user authentication, efficient searching was considered essential. A hash table structure was planned to enable faster username lookup and duplicate detection, reducing time complexity compared to simple linear search. For managing billing items dynamically, data structures such as vectors and linked lists were chosen to allow flexible insertion and deletion of items, similar to how real shopping carts function.

To maintain organized and professional output, sorting algorithms were included in the design to arrange cart items by name, quantity, or price before generating receipts. This ensures better readability and structured presentation of billing information. Additionally, file I/O operations were integrated to simulate a simple database system, allowing persistent storage of user credentials and billing records.

3.1.2 Requirement Analysis

The system requirements were identified to ensure proper functionality and performance.

Functional requirements : The system must provide secure user authentication with login and signup functionality. It should allow users to add, update, and delete billing items dynamically, calculate the total amount in real time, and generate a formatted receipt. The system must store and retrieve user credentials and billing data using file I/O operations to ensure data persistence. Additionally, it should support efficient searching, sorting, and duplicate detection using appropriate data structures such as hash tables, linked lists, or vectors.

Non-functional requirements : The system should be user-friendly, with a clear and responsive graphical interface built using Raylib. It must ensure efficient performance, especially in user lookup and cart operations, by applying suitable DSA concepts. The application should be reliable, maintaining data consistency during file operations. It should also be modular and maintainable, allowing future improvements or scalability without major restructuring.

3.1.3 System Design

The proposed BillingSystem was designed using a modular and layered architectural approach to ensure maintainability, scalability, reliability, and a clear separation of functional responsibilities. The implementation was carried out as a standalone desktop application using C++ with the Raylib GUI library, rather than relying on any external frameworks or modifying third-party systems. This approach preserves the integrity of the core program, allows easy future upgrades, and ensures that each module can operate independently while interacting seamlessly with other components. The system follows a layered model in which the presentation layer handles all graphical user interface operations, capturing user interactions such as login credentials, cart item inputs, and billing commands, while the application logic layer processes these inputs to perform authentication, cart updates, total calculation, duplicate detection, and receipt generation. The data storage layer manages persistent data using file I/O operations, storing user credentials in users.dat and billing transactions in receipt.txt.

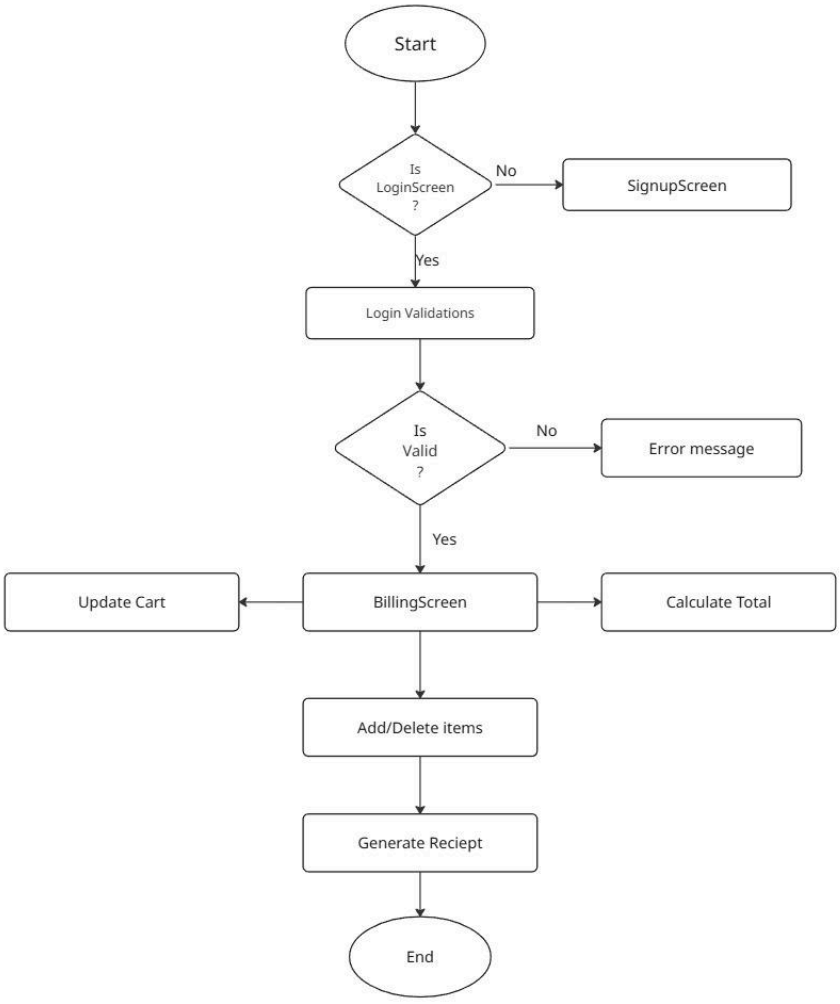

Fig 3.1 : System Flowchart

Structurally, the system consists of Login, Signup, and Billing modules, supported by helper and UI components, that interact through well-defined interfaces. This modular and layered design promotes loose coupling and high cohesion, allowing independent development, testing, and maintenance of individual components. The backend logic implements user authentication using linear search in vectors, with potential optimization through hash tables for faster lookup, and manages dynamic cart operations through vectors and linked lists. Sorting algorithms are applied to organize cart items and receipts, ensuring a professional and readable output.

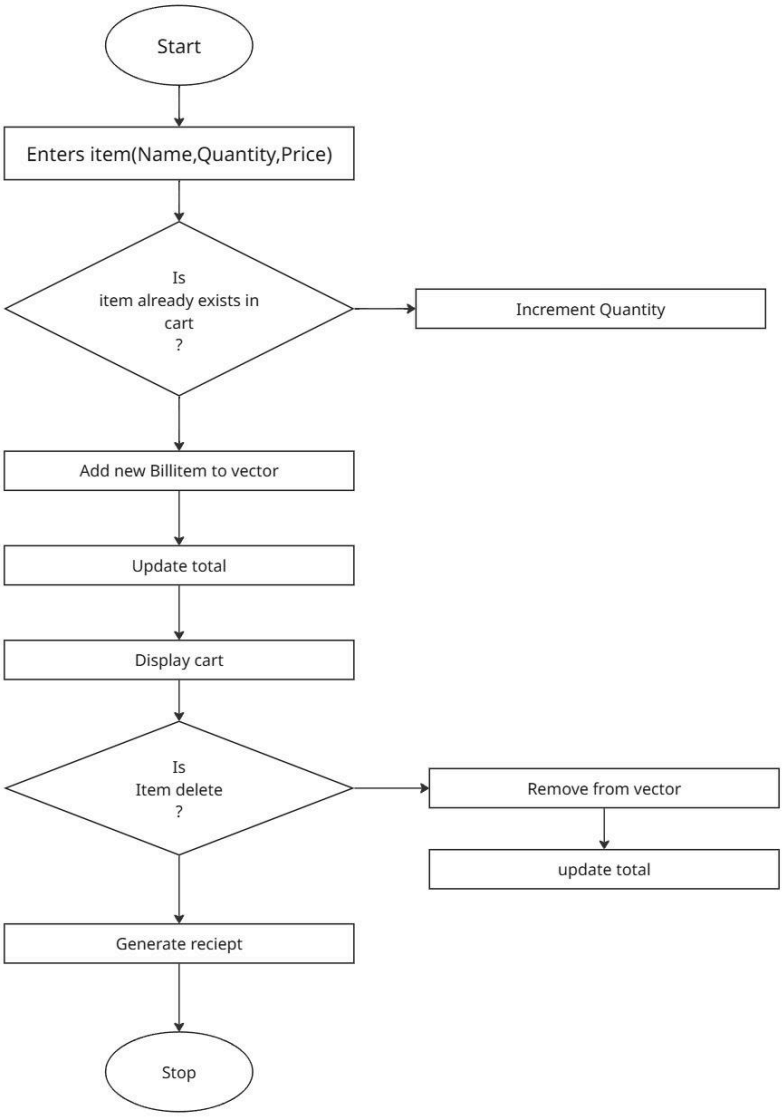

Fig 3.1 : Billing Module Flowchart

The frontend GUI integrates interactive elements such as custom buttons, input boxes, password fields, and visually appealing layouts. The screens are designed with a dark cyberpunk theme featuring glowing effects and smooth transitions between Login, Signup, and Billing screens. These interface enhancements improve usability, accessibility, and overall user experience, while clearly separating functional operations from visual presentation.

The **billing logic module** manages real-time cart updates, calculates totals dynamically, detects duplicate items to update quantities, and allows deletion of individual items. Receipt generation is handled through formatted file output, ensuring persistent storage of transactions. Flowcharts and use case diagrams were used during the design phase to model the user interaction flow and system operations, while the ER diagram represents the relationships between users, billing items, and receipts. These visual models provide a structured understanding of system behavior, data relationships, and process logic.

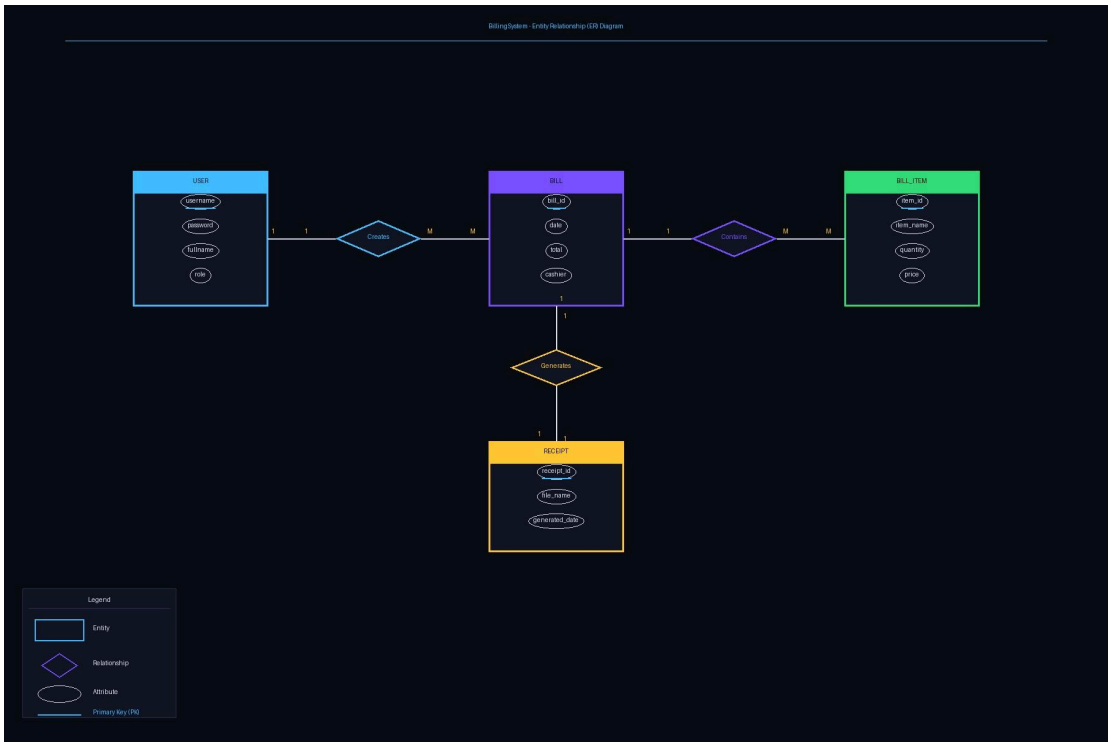

Fig 3.2 : ER diagram

Overall, the system design integrates efficient data handling, modular programming, and practical application of DSA concepts, bridging the gap between theoretical knowledge and real-world software development. By combining a layered architecture, dynamic data structures, persistent storage, and an interactive GUI, the BillingSystem provides a robust, scalable, and user-friendly solution for desktop-based billing management.

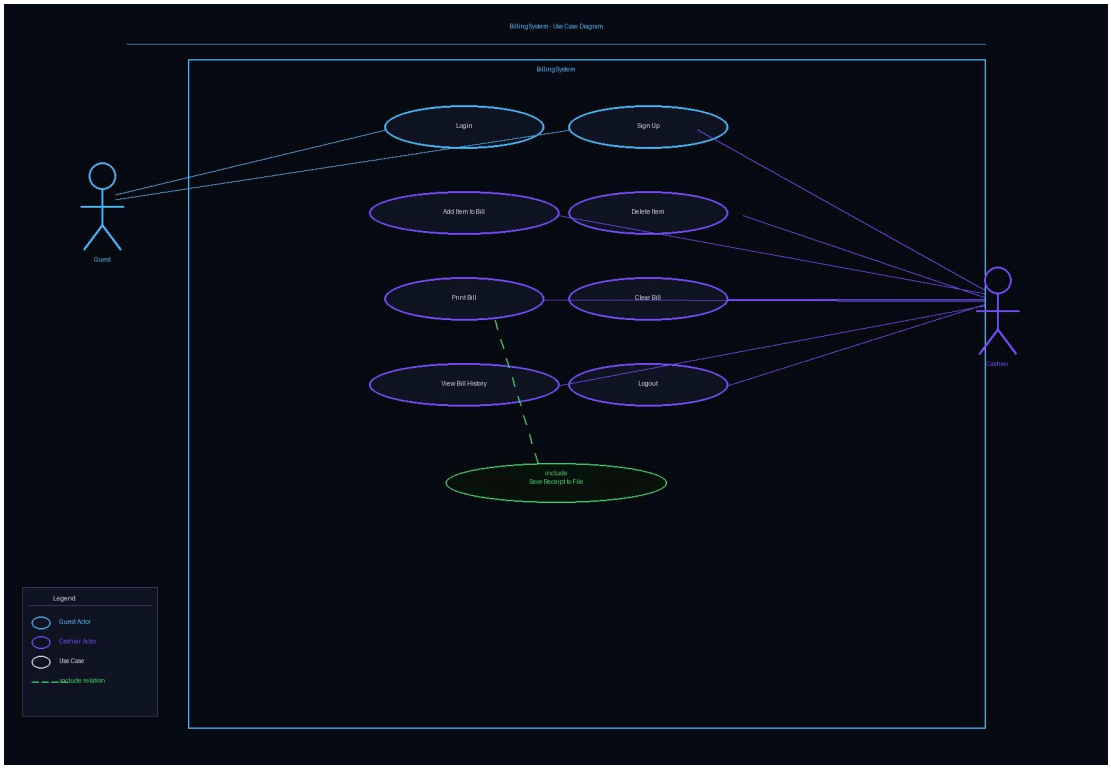

Fig 3.3 : Use Case Diagram

3.1.4 Development

The BillingSystem is a desktop application built with C++ and the Raylib graphics library, designed to emulate a real-world billing management environment. The application includes three primary screens: Login, Signup, and Billing. The Login

and Signup screens provide a secure user authentication system, enabling existing users to sign in and new users to register, with all user information stored persistently in a local file. The Billing screen allows users to add items with their name, quantity, and price to a dynamic bill, view all items in a scrollable table, calculate the grand total in real time, and generate a formatted receipt saved as a text file.

The project applies several key Data Structures and Algorithms (DSA) concepts, such as dynamic arrays using vectors for managing users and bill items, structs for organizing data, linear search for validating users and detecting duplicate items, file input/output for persistent storage, and enums for handling screen navigation. The graphical interface follows a dark cyberpunk theme with neon cyan and purple highlights, incorporates the Poppins font via Raylib's font system, and displays a custom logo on the login screen. All UI elements—including buttons, text input boxes, and password fields—are custom-built from scratch without relying on any third-party UI frameworks, providing a fully interactive and visually cohesive experience.

The major development phases were:

- Planning and Requirement Analysis
- Design Phase
- Setting Up the Development Environment
- Implementation
- Testing and Debugging
- Final Build and Deployment
- Challenges Faced

3.1.5 Testing and Debugging

Once the implementation was completed, the application underwent thorough testing. Each screen was tested individually to verify that all input fields accepted text correctly, buttons responded properly to mouse clicks, error messages appeared for invalid entries, and navigation between screens functioned smoothly. The billing functionality was checked by adding multiple items, deleting items, scrolling through long lists, and generating receipts. User authentication was tested by attempting logins with incorrect credentials, creating new accounts, and ensuring that duplicate usernames were not allowed. During this testing phase, several issues were identified and resolved, including cursor positioning within input fields, scroll limits for the bill table, and font rendering problems.

3.1.6 Project Release and Enhancement

The BillingSystem was implemented in a structured and modular manner to ensure maintainability, scalability, and clarity. The project was organized into multiple files, starting with globals.h and globals.cpp, which defined shared data structures, color constants, global variables, and font helper functions, allowing consistent access across all modules. The ui.cpp file was then developed to include reusable interface components such as Button, InputBox, PasswordBox, GlowLine, DrawBackground, and DrawLogo, all custom-built using Raylib drawing functions for use across the Login, Signup, and Billing screens. The helpers.cpp file handled data operations including loading and saving users, checking for duplicate usernames, validating login credentials, and calculating cart totals using file I/O with users.dat. The screens were implemented sequentially: the Login screen featured username and password fields, sign-in button, error messages, and a top logo; the Signup screen included full name, username, password, and confirm password fields with validation and navigation back to login; and the Billing screen comprised an add-item panel, a scrollable bill table, and a summary bar showing the grand total with print and clear

options. Finally, main.cpp initialized the window, loaded fonts and the logo, restored saved data, and ran the main loop, continuously rendering the active screen based on the current state, ensuring smooth operation and cohesive user interaction.

3.2 System Requirement Specifications

The system was developed and tested with specific software and hardware requirements to ensure proper functionality, performance, and compatibility with the Billing System . These requirements are outlined as follows.

3.2.1 Software Specifications

The software components necessary for developing and operating the system include:

- **Operating System :** Windows 10 or later
- **Compiler:** GCC 14.0 or later (w64devkit)
- **Graphics Library** : Raylib 5.0
- **Integrated Development Environment (IDE):** Visual Studio Code
- **Font :** Poppins (Regular, Bold, SemiBold)

These software components provide the foundation for implementing backend logic, frontend interfaces, database management, and seamless integration with the Moodle LMS.

3.2.2 Hardware Specification

The hardware requirements for running the system effectively are as follows:

- **Processor:** Intel Core i3 or equivalent (minimum); Intel Core i5 or higher is recommended for optimal performance
- **Memory (RAM):** 2 GB minimum; 4 GB or more is recommended

- **Storage:** 50 MB free disk space
- **Display:** 1100 x 700 pixels resolution
- **Input Devices:** keyboard and mouse

These specifications ensure that the system operates efficiently, supports real-time monitoring and tracking features, and provides a smooth user experience for both students and instructors.

Chapter 4 Discussion on the achievements

4.1 Project Overview

The BillingSystem is a fully operational desktop application developed using C++ and the Raylib graphics library. This project was created as part of the Data Structures and Algorithms practical coursework, aiming to apply theoretical DSA concepts in a practical, real-world scenario. The system emulates a basic billing environment similar to those used in small shops, retail outlets, or businesses that require transaction management and bill generation. It offers a complete workflow, beginning with user registration and authentication, followed by the ability to add items, calculate totals, generate printable receipts, and review the entire billing history, providing a realistic and functional billing management experience.

Login and Signup System

The Login Screen is the first interface displayed when the application starts. It features a visually styled card panel with the application logo at the top, a username input field, a password field with masked characters, a Sign In button, and a link to navigate to the Signup Screen. When a user enters their credentials and clicks Sign In or presses Enter, the system checks the username using a hash table implemented with an unordered_map, providing an average lookup time of O(1). If the credentials are valid, the user is directed to the Billing Screen; otherwise, an error message appears in red below the Sign In button. The login screen also supports Tab key navigation between input fields to improve usability.

The Signup Screen allows new users to register an account. It includes four input fields for full name, username, password, and confirm password. Upon clicking the Create Account button, the system validates the inputs by ensuring no field is empty,

verifying that the password and confirm password match, and checking the hash table to ensure the username is not already taken. If all checks pass, the new user is added to both the users vector and the hash table, and the account information is saved to the users.dat file for persistent storage. A success message is displayed in green, and the user can return to the Login Screen using the Back to Login button within the card panel.

Billing Screen

The Billing Screen serves as the main workspace of the application and becomes accessible only after a successful login. It is organized into multiple sections for efficient user interaction. The top bar displays the application name, the currently logged-in user's name, a History button to view previous bills, and a Logout button. The left panel contains the Add Item form, where users can input an item's name, quantity, and price, and then click the Add to Bill button to add it to the cart, which is stored as a vector. Before adding, the system performs a linear search through the cart to check for existing items; if an item already exists, its quantity is incremented instead of creating a duplicate entry.

The right section features a scrollable bill table that lists all items along with their name, quantity, unit price, row total, and a delete button for each entry. The bottom summary bar displays the grand total, calculated in real time by iterating through the cart vector, and includes two action buttons. The Print Bill button saves the current bill as a new node at the head of an in-memory linked list and writes a formatted receipt to a text file, while the Clear Bill button removes all items from the cart, resetting the workspace for a new transaction.

History Screen

The History Screen serves as a fourth interface that allows users to view all previously printed bills by traversing the linked list from head to tail. Each bill is displayed as an individual card showing details such as the bill number, date, cashier name, a table listing all items with their quantities and prices, and the grand total. The screen also visually represents the linked list structure, indicating whether each node points to a subsequent node or is the tail node with a NULL pointer, effectively demonstrating the practical use of linked lists. Additionally, the History Screen supports mouse wheel scrolling, enabling users to browse through a large number of saved bills efficiently.

4.2 Database Implementations

Linked List

A custom singly linked list was implemented to manage the bill history. Each node, defined as a BillNode struct, stores the complete details of a single printed bill along with a next pointer pointing to the following node in the list. New bills are always inserted at the head of the list, ensuring that the most recently printed bill appears first. The History Screen traverses the linked list from head to tail using a pointer and displays each node as a bill card. This implementation effectively demonstrates how a linked list can be applied in a real-world scenario to maintain an ordered history of records, even when the total number of records is not known in advance.

Hash Table

An unordered_map from the C++ Standard Library was utilized as a hash table to store and efficiently look up user credentials. In this implementation, the username

serves as the key, while the User struct acts as the value. Using a hash table provides an average time complexity of O(1) for insertion, deletion, and lookup operations, which is significantly faster than the O(n) complexity of a linear search through a vector. This greatly improves the performance of login validation and duplicate username checks, especially as the number of registered users increases. Each time the application starts, the hash table is rebuilt by loading all user data from the file and inserting each entry into the unordered_map, ensuring that the table is always up to date with persistent storage.

Linear Search

Linear search was implemented in two places. The first is in the Add Item logic on the Billing Screen which iterates through the cart vector one by one to find an item with the same name before adding a new entry. The second was previously used for login but has now been replaced by the Hash Table for improved performance.

Vector(Dynamic Array)

The std::vector data structure from the C++ Standard Library was used in two places in the application. The first is a vector of User structs that stores all registered users loaded from the users.dat file. The second is a vector of BillItem structs that acts as the shopping cart and dynamically grows as items are added and shrinks as items are deleted. Vectors were chosen because they provide dynamic resizing, constant time access by index, and efficient iteration which are all required for the billing operations in this application.

Achievements

The most significant achievement of this project is the successful development of a fully functional desktop billing application from scratch using C++ and the Raylib graphics library. Every component of the graphical user interface including buttons, input fields, password boxes, scroll functionality, and screen transitions were built entirely without any third party UI framework. The application features a professional dark themed interface with the Poppins font, custom logo display, neon accent colors, hover effects, and blinking cursors in input fields demonstrating strong attention to detail and design quality that goes beyond what is typically expected in a practical coursework project.

A major achievement is the successful implementation of multiple Data Structures and Algorithms concepts in a single real world application. The Vector was used for dynamic cart and user management, the Hash Table using unordered_map was used for fast O(1) user authentication, the Linked List was used to maintain a complete bill history that can be traversed and viewed on a dedicated History Screen, the Linear Search was used for duplicate item detection, and File I/O was used for persistent data storage of user accounts and receipts. Integrating all of these DSA concepts together into one cohesive and working application is the core academic achievement of this project.

Finally the project was successfully organized into a clean eleven file modular structure following good software engineering principles such as separation of concerns and code reusability. The project was also version controlled using Git and published to a public GitHub repository demonstrating awareness of professional development workflows. Most importantly the BillingSystem is not just an academic exercise but a genuinely useful application that a small shop owner or cashier could

realistically use to create bills, manage items, print receipts, and review transaction history making it a meaningful and practical contribution beyond the classroom.

Chapter 5 Conclusion and Recommendation

The BillingSystem project successfully demonstrates how multiple Data Structures and Algorithms concepts can be applied together in a single real world desktop application. The Vector provides dynamic item management, the Hash Table provides fast user authentication, the Linked List maintains a persistent bill history, and the File I/O ensures data survives application restarts. Together these concepts form the backbone of a practical and fully functional billing application that meets all the defined objectives of the practical coursework and provides a solid foundation for further development and enhancement.

5.1 Limitations

Despite its successful implementation, the BillingSystem has several limitations that should be acknowledged. User credentials are stored in a plain text file without any encryption making it unsuitable for high security environments. The bill history stored in the Linked List exists only in memory during the current session and is completely lost when the application is closed as it is not saved to a file. The application window size is fixed at 1100 by 700 pixels and does not support resizing or fullscreen mode. The receipt is saved as a plain text file and does not support direct printing to a physical printer. The system supports only one active user session at a time with no administrator panel for managing users and there is no option to edit an item once it has been added to the bill. These limitations represent areas where the application can be improved and expanded in future development to make it a more complete and production ready billing solution.

5.2 Future Enhancement

There are several enhancements that can be made to the BillingSystem in the future to make it a more complete and production ready application. The most important enhancement would be to integrate a proper database system such as SQLite to replace the current plain text file storage allowing the application to handle a much larger number of users and transactions securely and efficiently. Password encryption using hashing algorithms such as SHA-256 should also be implemented to protect user credentials from unauthorized access. The bill history stored in the Linked List should be made persistent by saving it to a file so that previous bills are not lost when the application is closed and reopened. Additionally the ability to edit item name, quantity, and price directly in the bill table and direct printing support to a physical printer would make the application significantly more practical in a real shop environment.

On the technical and functional side the application window could be made resizable with fullscreen support for different screen sizes and resolutions. An administrator panel could be added to allow the admin to manage registered users including creating, deleting, and resetting passwords along with support for multiple simultaneous user sessions for larger businesses with more than one cashier. A sales report feature could be added that analyzes the bill history linked list and generates daily, weekly, and monthly sales summaries to help business owners track their revenue. Finally the application could be ported to Android or iOS as a mobile billing application making it accessible on smartphones and tablets which are more commonly used in small businesses today.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). MIT Press.
2. Lippman, S. B., Lajoie, J., & Moo, B. E. (2012). *C++ primer* (5th ed.). Addison-Wesley Professional.
3. Raysan5. (2024). *Raylib: A simple and easy-to-use library to enjoy videogames programming*. GitHub. https://github.com/raysan5/raylibRaylib. (2024). *Raylib cheatsheet*. https://www.raylib.com/cheatsheet/cheatsheet.html
4. Stroustrup, B. (2013). *The C++ programming language* (4th ed.). Addison-Wesley Professional.

APPENDIX

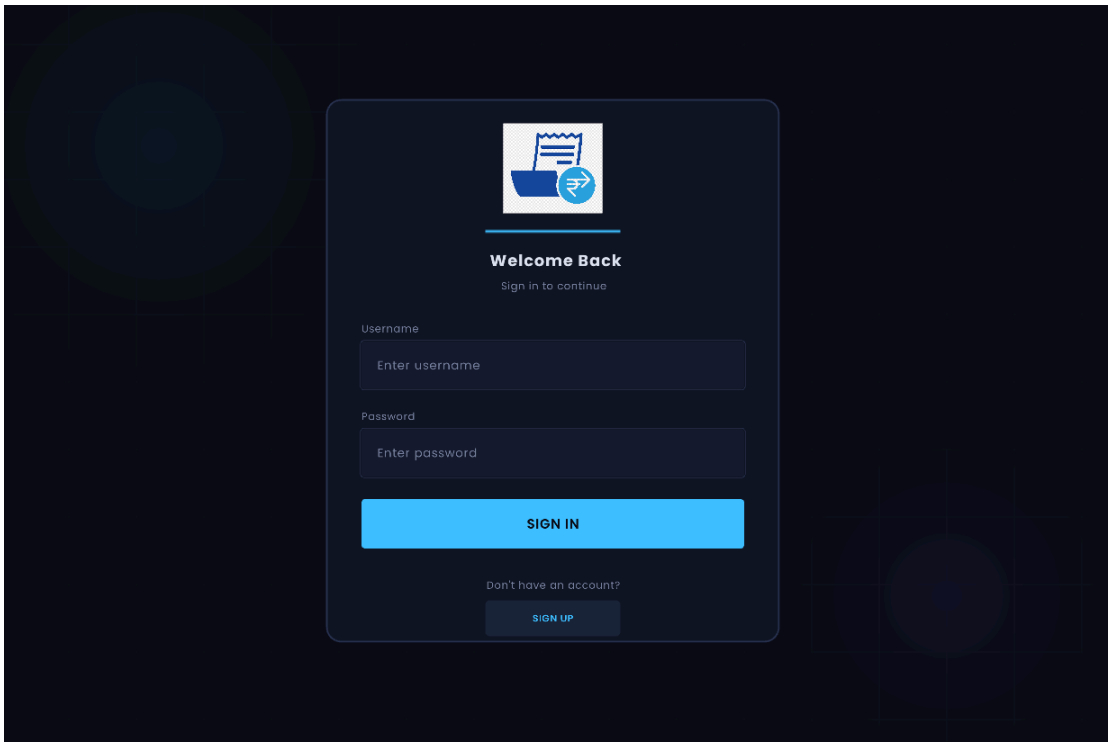

Fig A.1 Login Interface

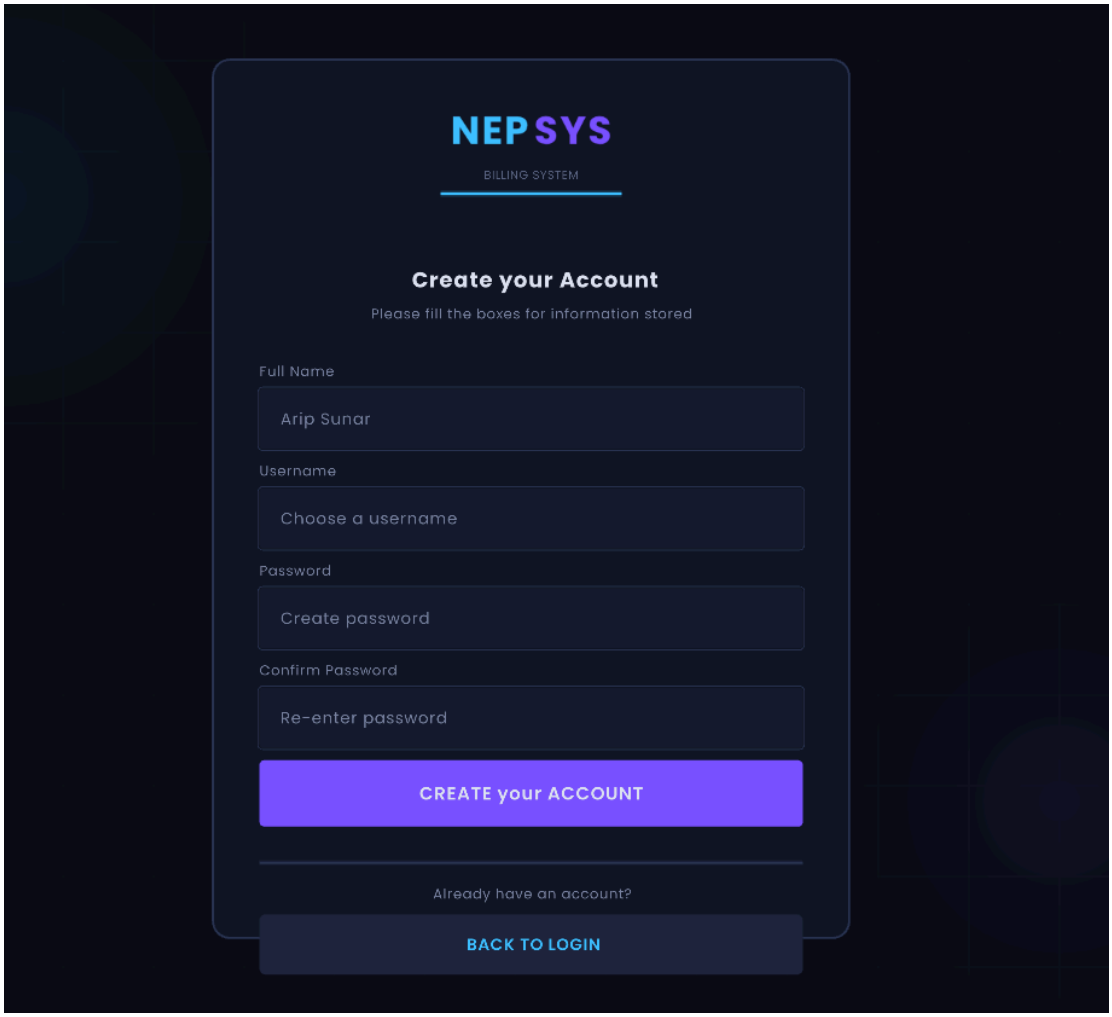

Fig A.2 Signup Interface

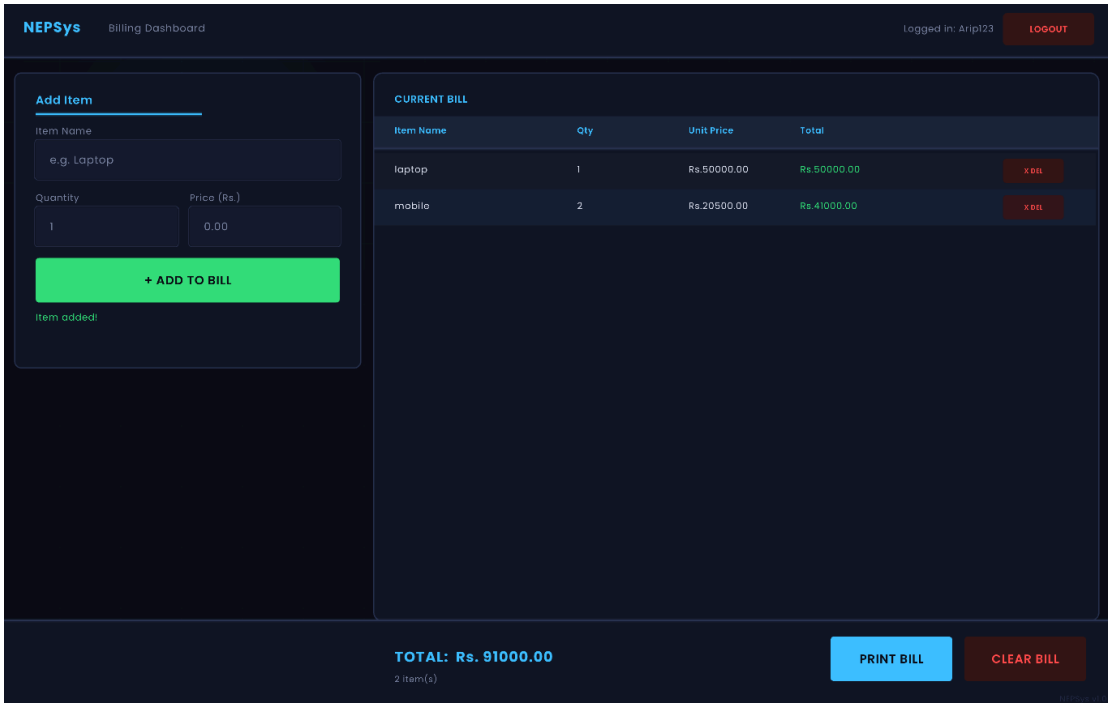

Fig A.3 Billing Dashboard

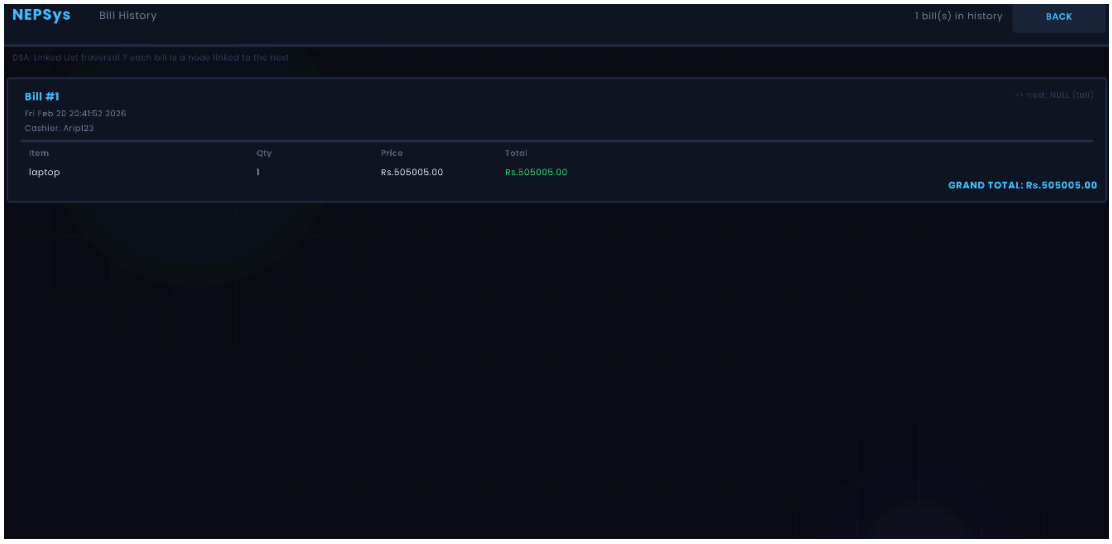

Fig A.4 History Dashboard