**Project Report**

**Title: Train Reservation System Using C++**

**Group Members:**

**Muhammad Azfar Waqas (UW-23-CY-BS-013)**

**Atta Ur Rahman (UW-23-CY-BS-014)**

**Ibrar Ul Hassan Shami (UW-23-CY-BS-018)**

**Table of Contents:**

[1. Introduction: 3](#_Toc157539813)

[1.1 Purpose: 3](#_Toc157539814)

[1.2 Code Structure: 3](#_Toc157539815)

[1.3 Targeted Operating Systems: 4](#_Toc157539816)

[1.4 Target Hardware Platforms: 5](#_Toc157539817)

[2. Overall Description: 5](#_Toc157539818)

[2.1 Project Functions: 5](#_Toc157539819)

[2.2 Maintainability: 6](#_Toc157539820)

[2.3 User Classes and Characteristics: 7](#_Toc157539821)

[2.4 Design and Implementation Constraints: 8](#_Toc157539822)

[3. Non-functional Requirements: 9](#_Toc157539823)

[3.1 Performance Requirements: 9](#_Toc157539824)

[3.2 Security Requirements: 9](#_Toc157539825)

[3.3 Software Quality Attributes: 10](#_Toc157539826)

[4. Other Requirements: 10](#_Toc157539827)

[4.1 User Interface: 10](#_Toc157539828)

[4.2 Software Interface: 10](#_Toc157539829)

[4.3 Hardware Interface: 11](#_Toc157539830)

[4.4 Communication Interface: 11](#_Toc157539831)

[5. Screenshots of the C++ Code: 12](#_Toc157539832)

1. **Introduction:**
   1. **Purpose:**

This document has a detailed explanation of our designed system. It’s built to solve the following Problems:

* Manual Ticketing Hassles
* Overbooking and Seat Allocation
* Limited Access to Information
* Time-Consuming Processes
* Inefficient Record-Keeping Integration
* Resource Optimization
* Enhanced Customer Experience
  1. **Code Structure:**

The basic structure of the code of our project includes:

* **A structure of the following members:**
* Train name
* Train number
* Number of seats
* **Libraries included:**
* iostream
* string
* conio.h
* **Important functions used:**
* Clear screen
* getch ()
* goto start
* cin.ignore
* **User defined functions include:**
* Reservation
* View details
* Cancel
* Print ticket
* Specific train
* Charge
* Login
* Feedback
* **Control Flow Statements:**
* If else
* Switch
* Nested if else
* **Operators include:**
* AND operator
* Equal to operator
* Other Arithmetic operators (+, \*)
  1. **Targeted Operating Systems:**

Train reservation systems are commonly web-based or have components accessible through web interfaces. In such cases, they are designed to be platform-independent and accessible through standard web browsers. As a result, they can run on a variety of operating systems, including:

* **Windows:** Compatibility with popular web browsers like Chrome, Firefox, and Edge.
* **Linux:** Support for browsers like Chrome and Firefox on Linux distributions.
* **macOS:** Compatibility with Safari, Chrome, and Firefox on Apple's macOS.

If the train reservation system includes mobile applications, there might be specific versions designed for:

* **iOS:** A version compatible with Apple devices, running on iPhones and iPads.
* **Android:** An application optimized for Android smartphones and tablets.
  1. **Target Hardware Platforms:**

Train reservation systems, being primarily web-based, are designed to run on standard computing hardware. This includes:

* **Desktops and Laptops:**

The system should be accessible on standard desktop and laptop computers, irrespective of the underlying hardware architecture (x86, x64, etc.).

* **Mobile Devices:**

If there are mobile applications, they are designed for smartphones and tablets, regardless of the specific hardware details.

* **Web Servers:**

Backend components of the system might run on servers, and these servers could be hosted on various hardware platforms. Common server hardware architectures include x86\_64, ARM, and others.

1. **Overall Description:**
   1. **Project Functions:**

Our proposed system will replace the current Train reservation system. The system will have all of the standard features for using online applications in addition to the additional features mentioned below:

* Efficient User Interface
* Realtime Booking and Updates
* Personalized Passenger Profiles
* Enhanced Security Measures
* Feedback
* Improved Mechanism
  1. **Maintainability:**

Maintainability, readability, flexibility, and robustness are crucial aspects of code quality. Well-designed code in a train reservation system should exhibit characteristics that make it easier to understand, modify, and maintain. Here are some aspects of the code that contribute to these qualities:

* **Modular Design:**
* **Contribution:** Breaking down the code into modular components promotes maintainability and readability. Each module can be developed, tested, and maintained independently.
* **Descriptive Variable and Function Names:**
* **Contribution:** Using meaningful and descriptive names for variables, functions, and classes enhances code readability and makes it easier for developers to understand the purpose of each element.
* **Comments and Documentation:**
* **Contribution:** Adequate comments and documentation provide additional context and explanations, aiding developers in understanding the code and its functionalities. This contributes to maintainability and readability.
* **Consistent Coding Style:**
* **Contribution:** Maintaining a consistent coding style across the codebase enhances readability. It ensures that developers can easily understand and follow the code, contributing to a cohesive and harmonious codebase.
* **Code Reusability:**
* **Contribution:** Designing code with a focus on reusability allows developers to leverage existing components in different parts of the system. This enhances flexibility and reduces redundancy.
* **Unit Testing:**
* **Contribution:** Incorporating unit tests ensures that individual components of the code function as intended. This practice contributes to robustness by catching and preventing errors early in the development process.
  1. **User Classes and Characteristics:**

In a train reservation system, various types of users with different roles and responsibilities interact with the code. The expected level of technical expertise varies for each user type based on their involvement in the system. Here are some common types of users and their expected levels of technical expertise:

* **End Users (Passengers):**

Technical Expertise: Low to Moderate

Interaction with code: Minimal to none

* **Customer Support Representatives:**

Technical Expertise: Moderate

Interaction with Code: Limited

* **System Administrators:**

Technical Expertise: High

Interaction with Code: Extensive

* **Developers/Programmers:**

Technical Expertise: High

Interaction with Code: Extensive

* **Management/Decision Makers:**

Technical Expertise: Variable (Low to Moderate)

Interaction with Code: Limited

* **Business Analysts:**

Technical Expertise: Low to Moderate

Interaction with Code: Limited

* **Management/Decision Makers:**

Technical Expertise: Variable (Low to Moderate)

Interaction with Code: Limited

Understanding the diverse user roles and their expected levels of technical expertise is crucial for designing user interfaces, providing adequate training and documentation, and ensuring that the system caters to the needs of all stakeholders as the infrastructure provided should be firm and to the point so no further complications may arise that may delay the project.

* 1. **Design and Implementation Constraints:**

**Any limitations or specific requirements that influenced the code's design are:**

The design of a train reservation system code is influenced by various limitations and specific requirements to ensure that it meets the needs of users, adheres to industry standards, and aligns with organizational goals. Here are some common limitations and specific requirements that might influence the design of a train reservation system code:

* **Regulatory Compliance:**

Compliance with railway regulations, safety standards, and legal requirements can significantly influence the design to ensure the system meets industry-specific guidelines.

* **Performance Requirements:**

The need to handle a large number of concurrent users during peak times, process transactions quickly, and provide a responsive user interface can impact the design for scalability and performance optimization.

* **Data Security and Privacy:**

Strict adherence to data security and privacy regulations, including secure storage of user information, and secure communication protocols.

* **Availability and Reliability:**

Requirements for high availability and reliability to minimize downtime, especially during critical periods such as ticket sales for popular routes.

* **Cross-Platform Compatibility:**

Designing the system to work seamlessly across various operating systems, browsers, and devices to ensure a consistent user experience.

* **User Accessibility:**

Adherence to accessibility standards to make the system usable by individuals with disabilities, including considerations for screen readers and other assistive technologies.

* **Customer Support Features:**

Inclusion of features for customer support, such as ticket cancellation, refund processing, and handling customer inquiries efficiently.

1. **Non-functional Requirements:**
   1. **Performance Requirements:**

The performance requirements include:

* + 1. **Response Time:**
* **Benchmark:** The time taken for the system to respond to user requests.
* **Target:** Response times should be optimized to provide a seamless user experience. For example, searches for train schedules and booking confirmations should occur within a few seconds.
  + 1. **Concurrent User Handling:**
* **Benchmark:** The maximum number of concurrent users the system can handle without performance degradation.
* **Target:** The system should be scalable to accommodate a large number of users during peak times, ensuring that performance remains consistent even under heavy load.
  + 1. **Server Response Time:**
* **Benchmark:** The time taken for the server to process and respond to user requests.
* **Target:** Server response times should be minimized to reduce latency. This includes backend processing times for tasks like retrieving train schedules, checking seat availability, and processing payments.
  + 1. **Memory Usage:**
* **Benchmark:** The amount of memory the system consumes during normal operation.
* **Target:** Efficient memory usage to avoid resource exhaustion. Optimizing memory usage contributes to system stability and performance.
  1. **Security Requirements:**

Every person who interacts with the system whether end user, customer support, System administrator, developer/programmer, Business Analyst, Management and third-party integrators will be given a log in ID and password. is assigned a unique login ID and password to access the system. Before any user can access the system, they shall be required to input a username, and a password. Every password must be of at least 8 characters.

* 1. **Software Quality Attributes:**

A well-designed C++ train reservation system should prioritize reliability, maintainability, and performance. It must offer a user-friendly interface with strong security measures, allowing for easy testing and future extensibility. Additionally, portability, scalability, and a focus on error handling contribute to a high-quality software product.

1. **Other Requirements:**
   1. **User Interface:**

* **Console application:**

The simplest approach would be a text-based interface where users interact through commands and menus. This could be developed purely in C++.

* **Graphical User Interface (GUI):**

For a more user-friendly experience, you could create a GUI using external libraries like Qt or widgets. These libraries provide widgets and functionalities to build graphical interfaces like windows, buttons, and text boxes.

* **Web application:**

You could develop a web application using C++ with frameworks like Wt or CppCMS. This would allow users to access the reservation system through a web browser, providing greater flexibility and platform independence.

* 1. **Software Interface:**
* **Internal APIs:**

Different modules within the C++ code could interact through internally defined APIs (Application Programming Interfaces). This allows for modular development and separation of concerns within the program.

* **Database interface:**

If the system uses a database to store data like train schedules, availability, and reservations, you would need an interface to interact with it. C++ provides libraries like libpq for PostgreSQL or ODBC for various database systems.

* **External APIs:**

The system might interact with external APIs for functionalities like email sending, or SMS notifications. These APIs would provide specific communication protocols and functions to integrate with the C++ code.

* 1. **Hardware Interface:**
* **Direct hardware interaction:**

Certain projects might involve direct interaction with hardware like ticket printers or display screens. This would require specific libraries or drivers depending on the hardware used.

* **Network interface:**

The system would likely use a network connection to communicate with databases, external APIs, or other external systems. This interaction would happen through standard network protocols like TCP/IP.

* 1. **Communication Interface:**
* **Network protocols:**

As mentioned above, the system would utilize various network protocols like TCP/IP, HTTP, or HTTPS for communication with different entities.

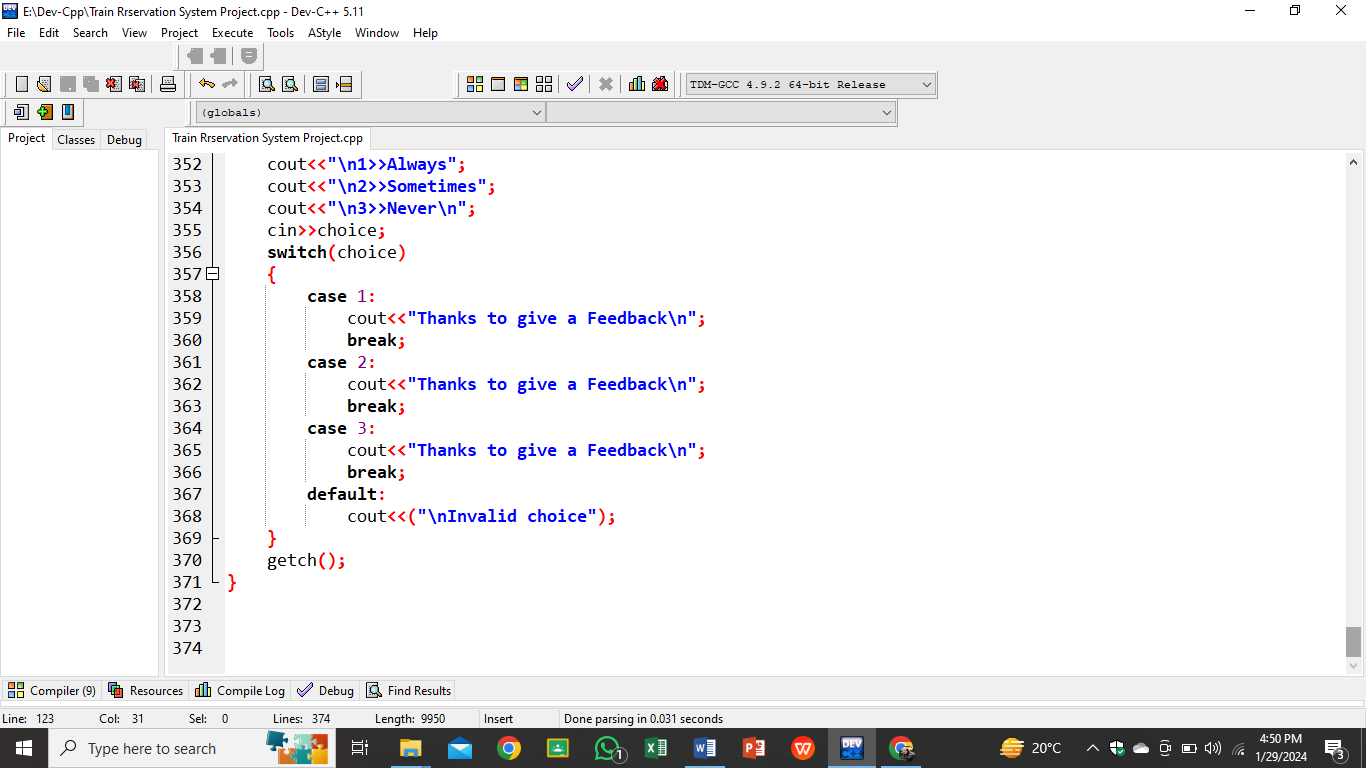
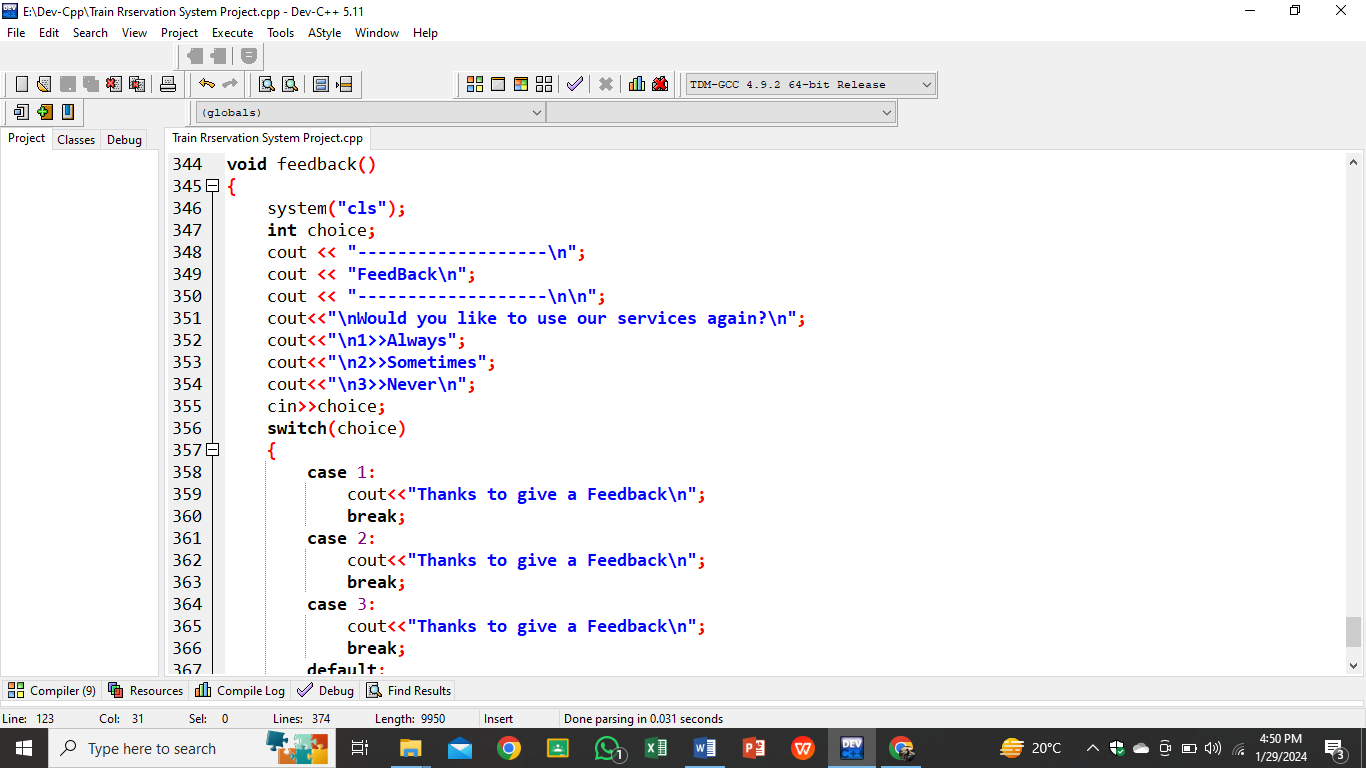
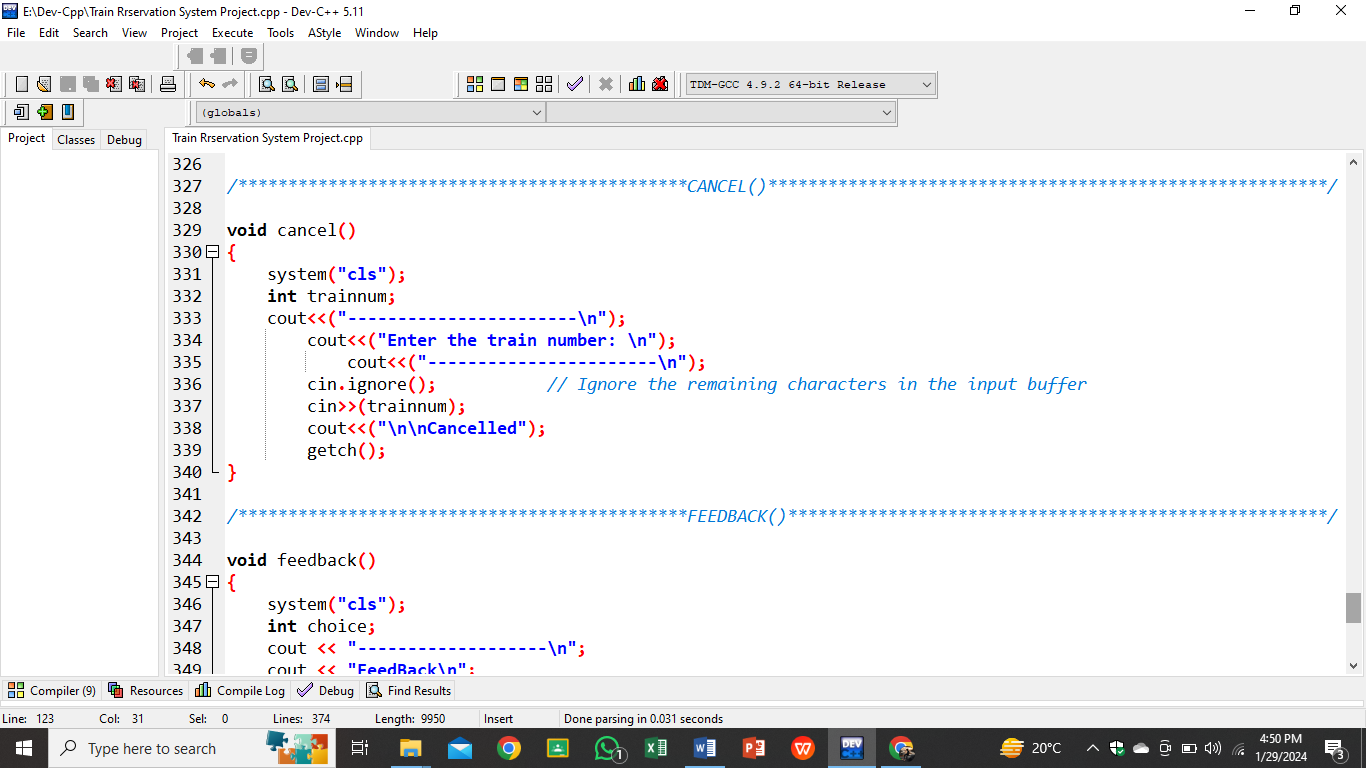
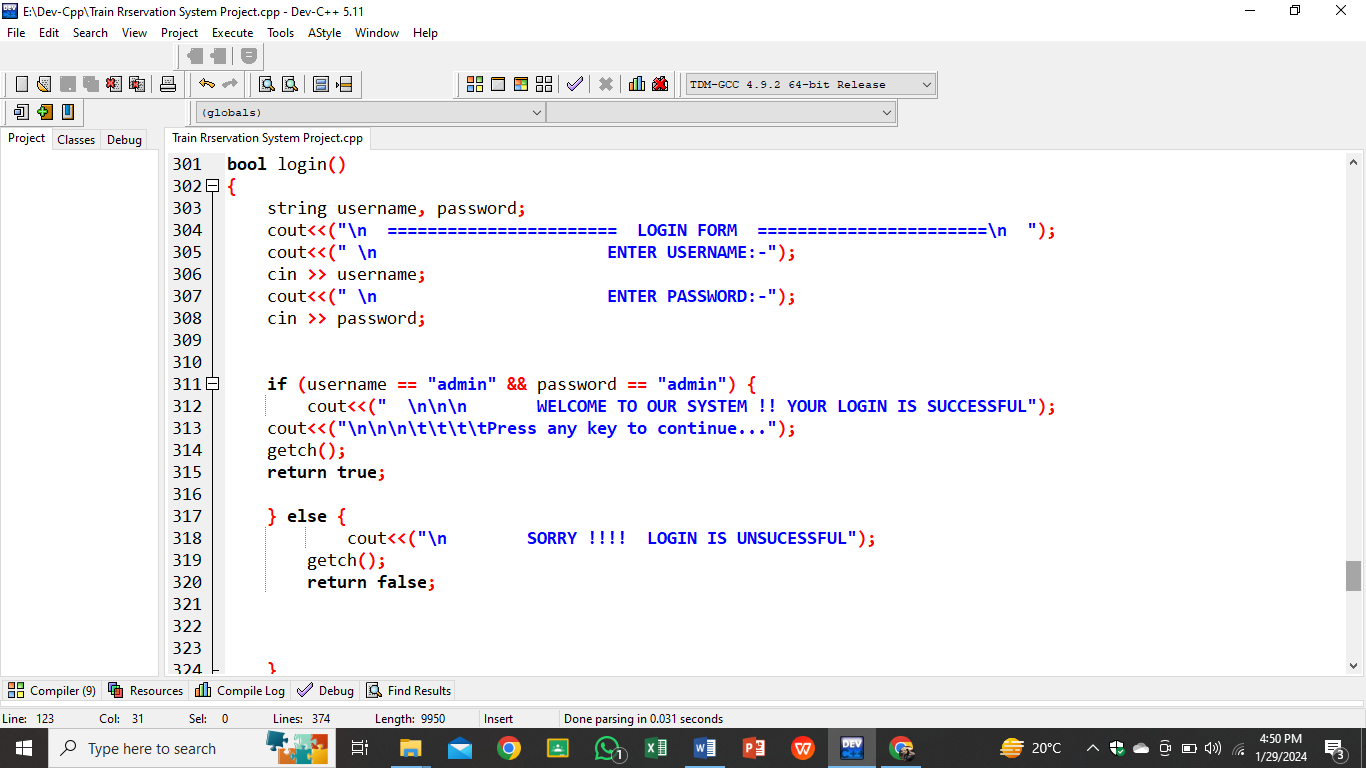
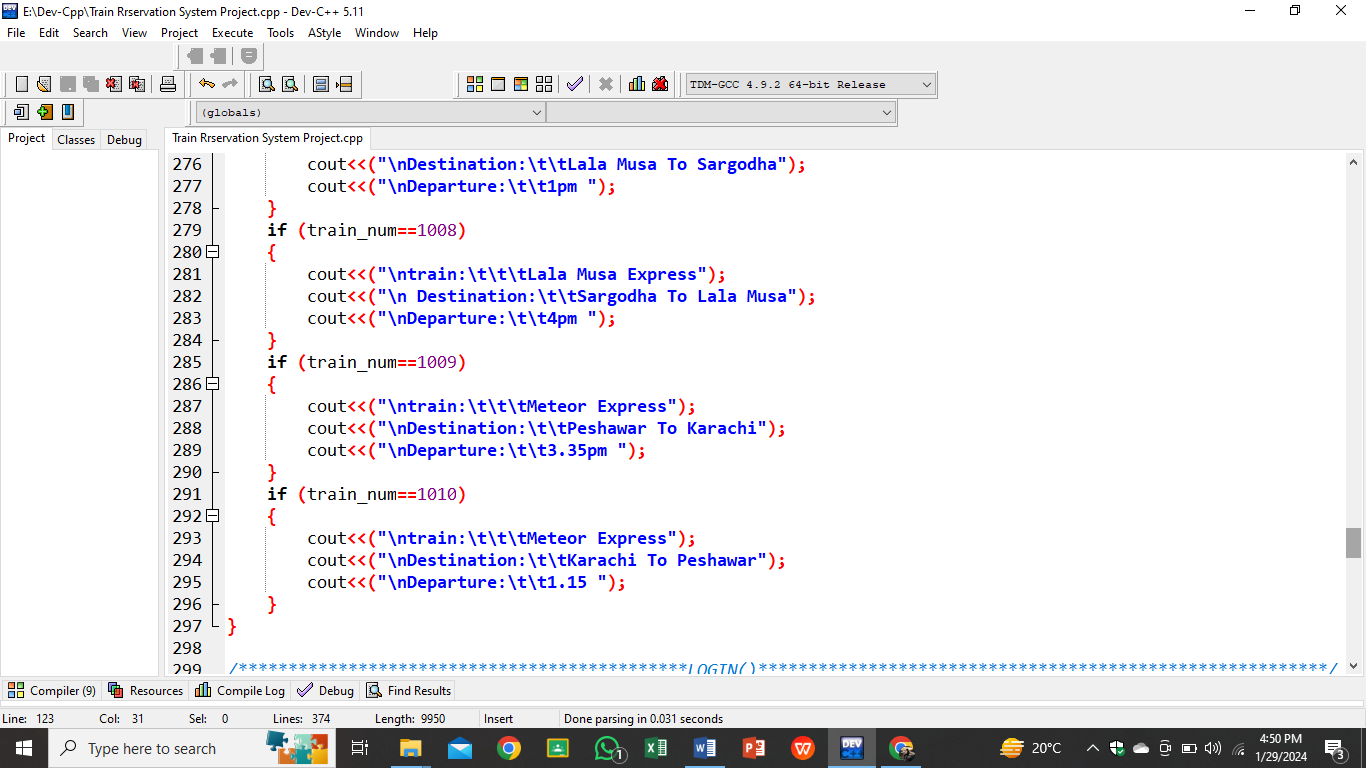
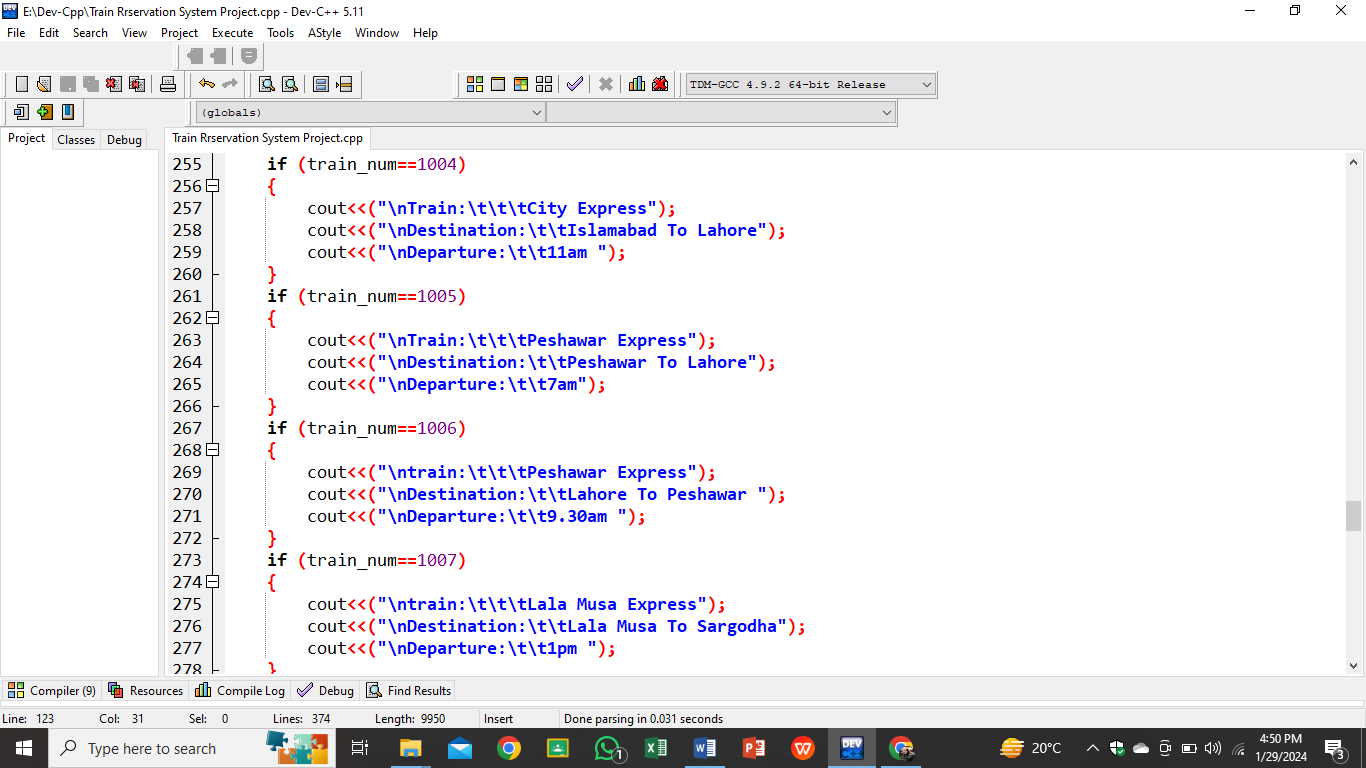
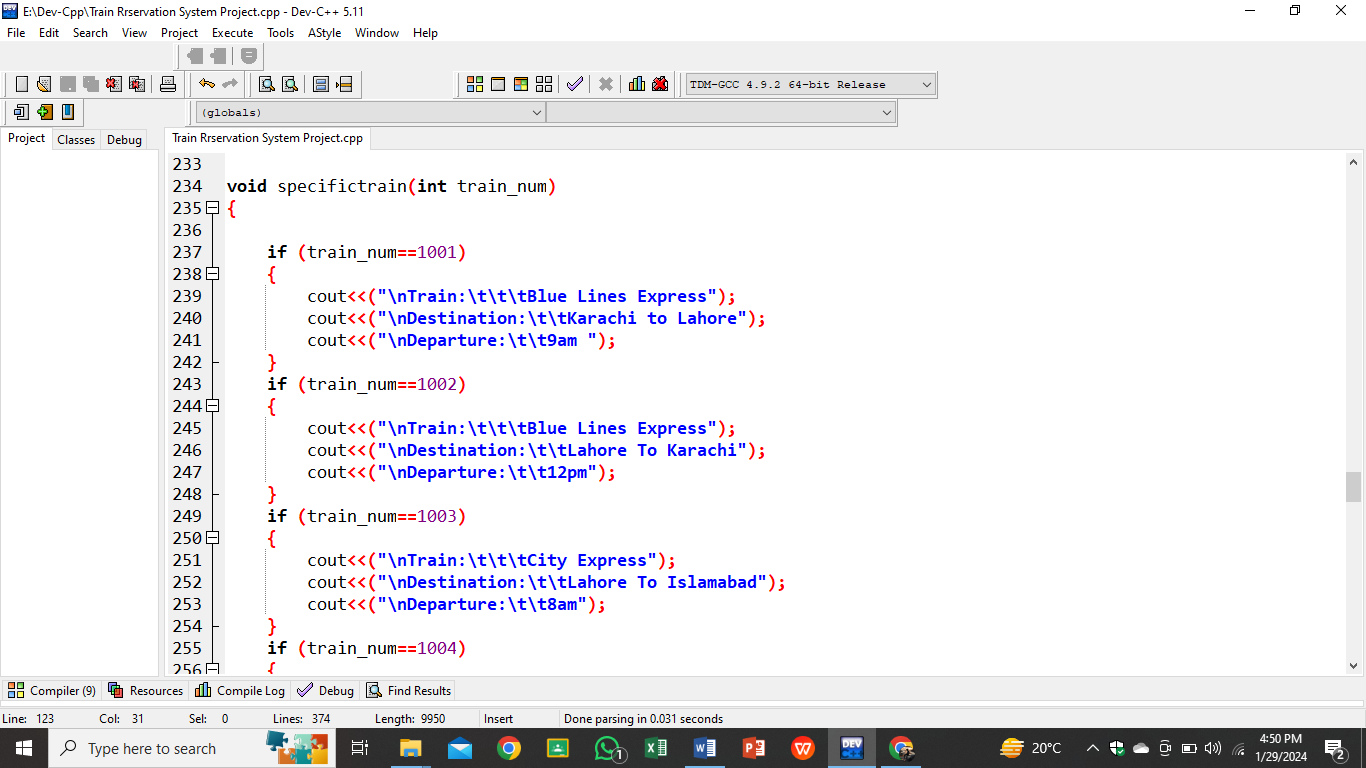
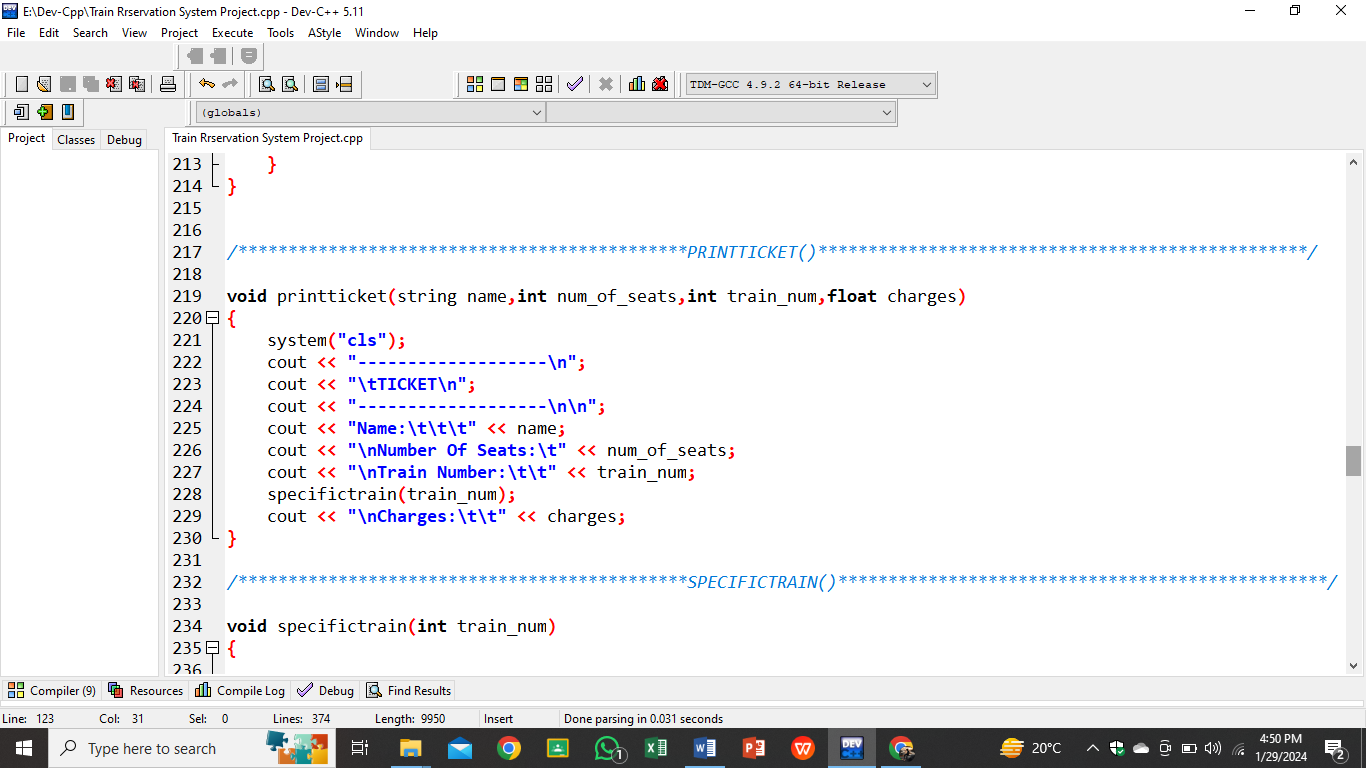
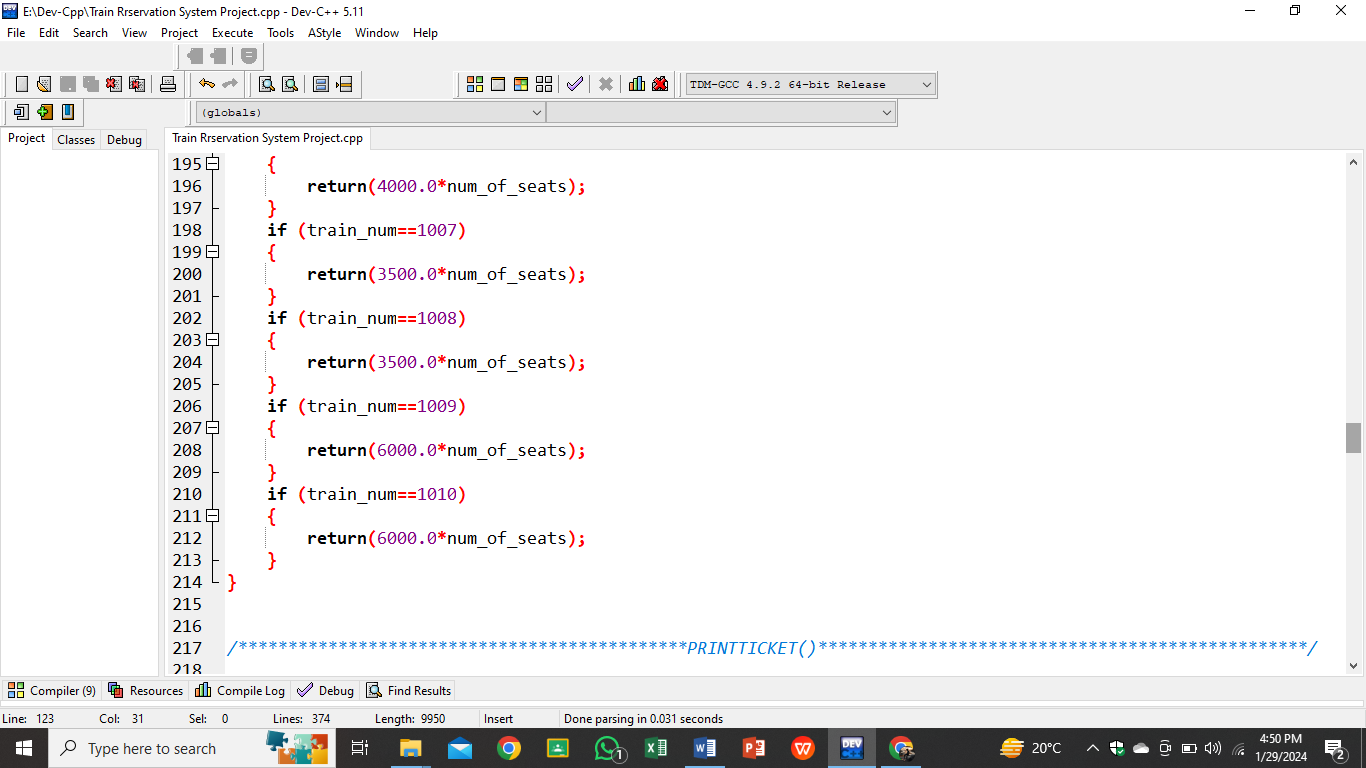
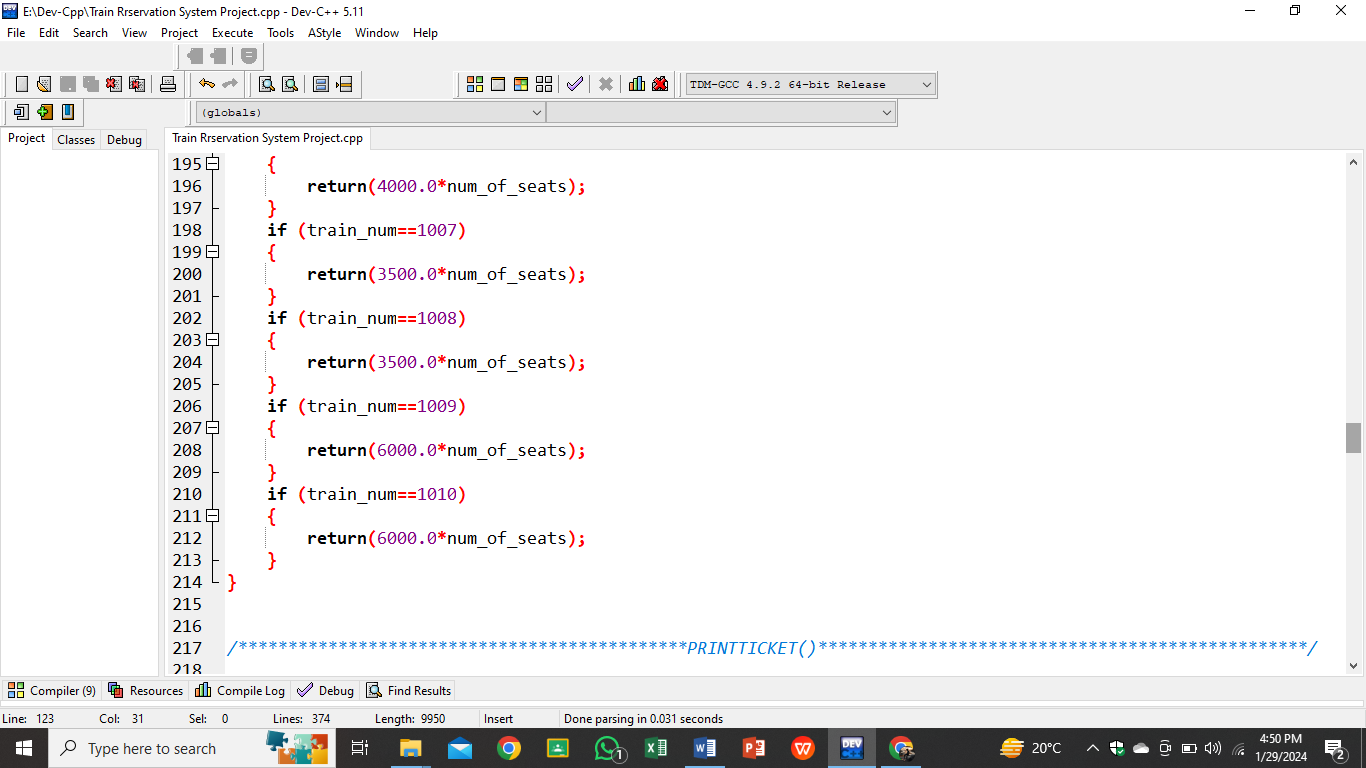
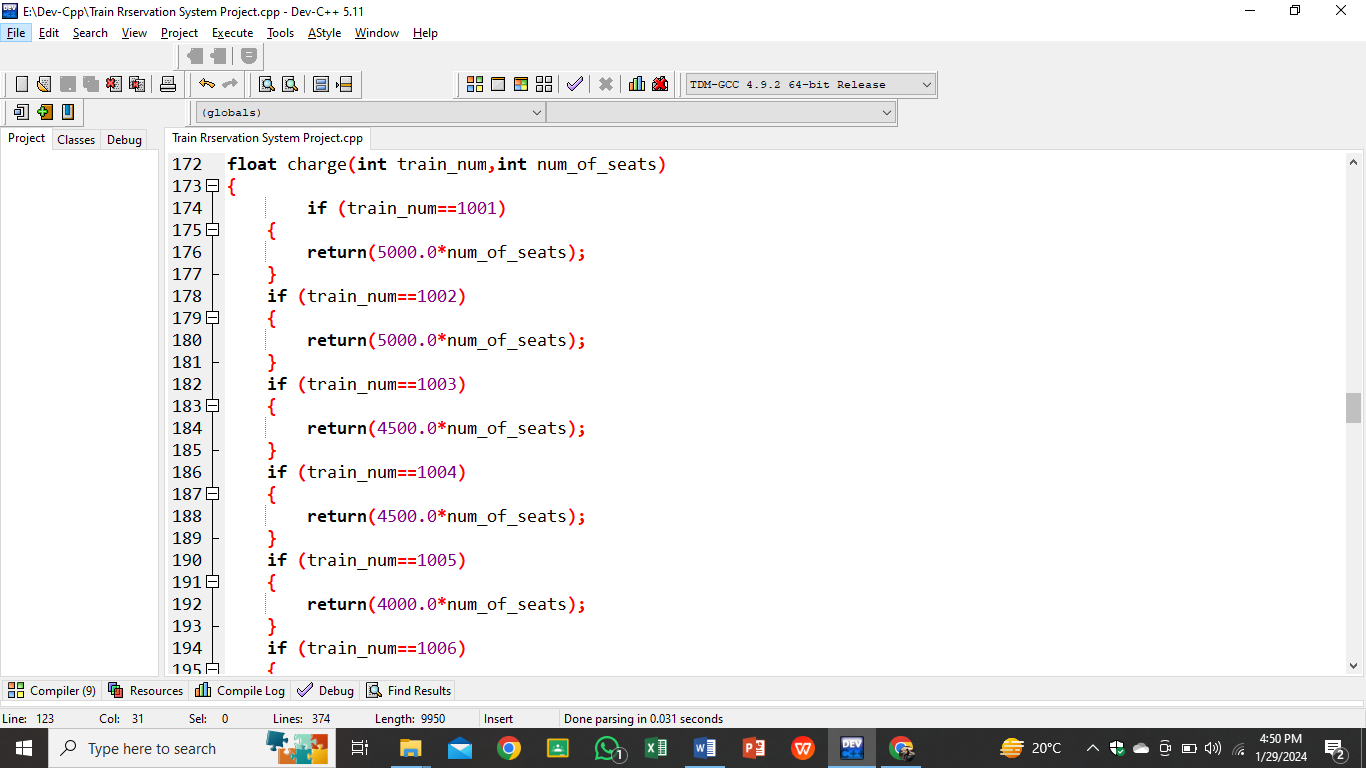
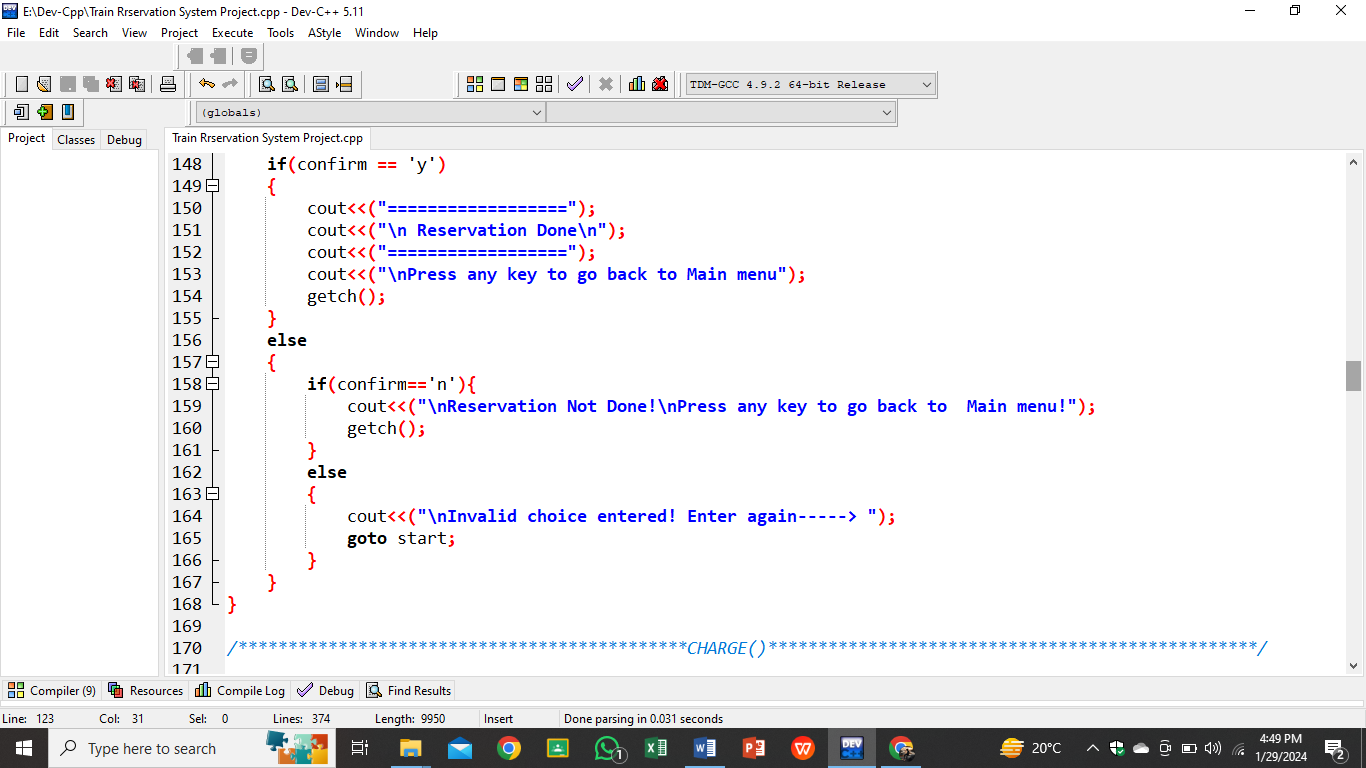
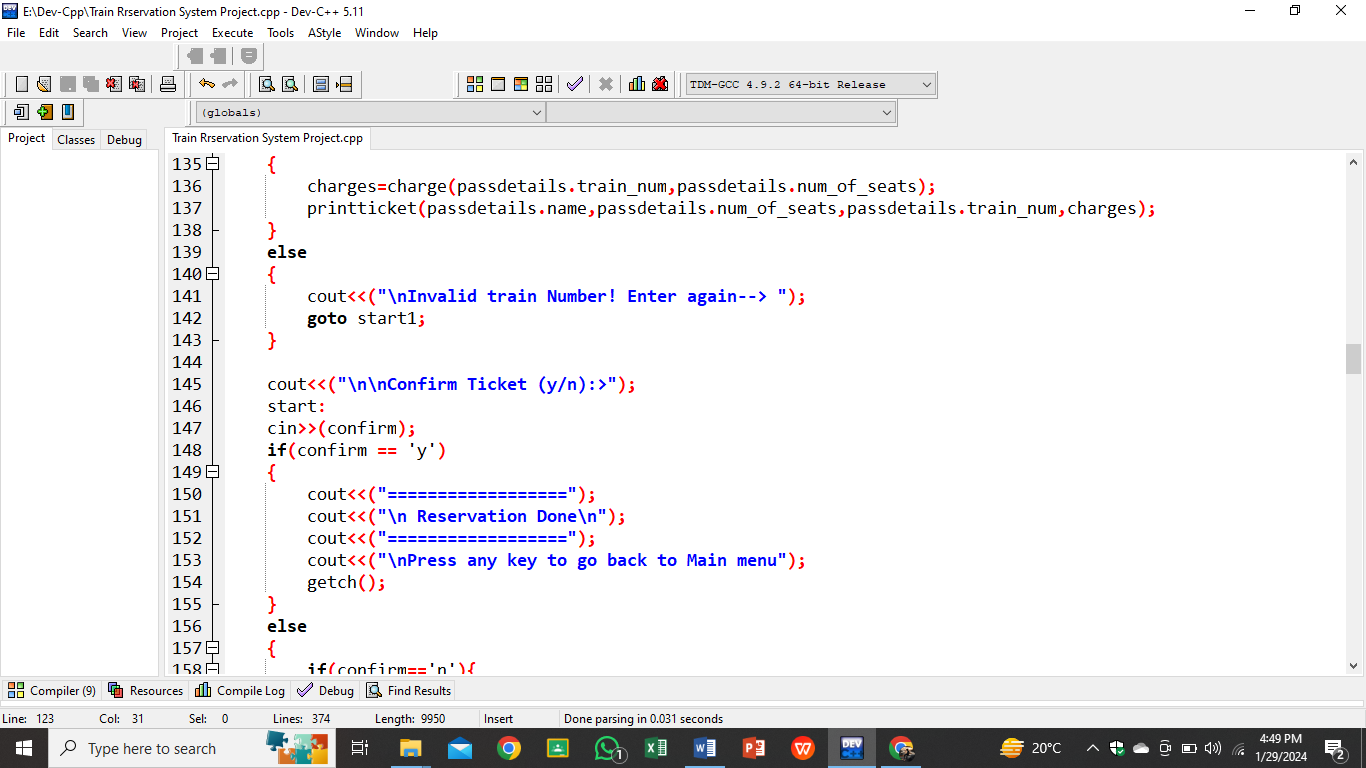
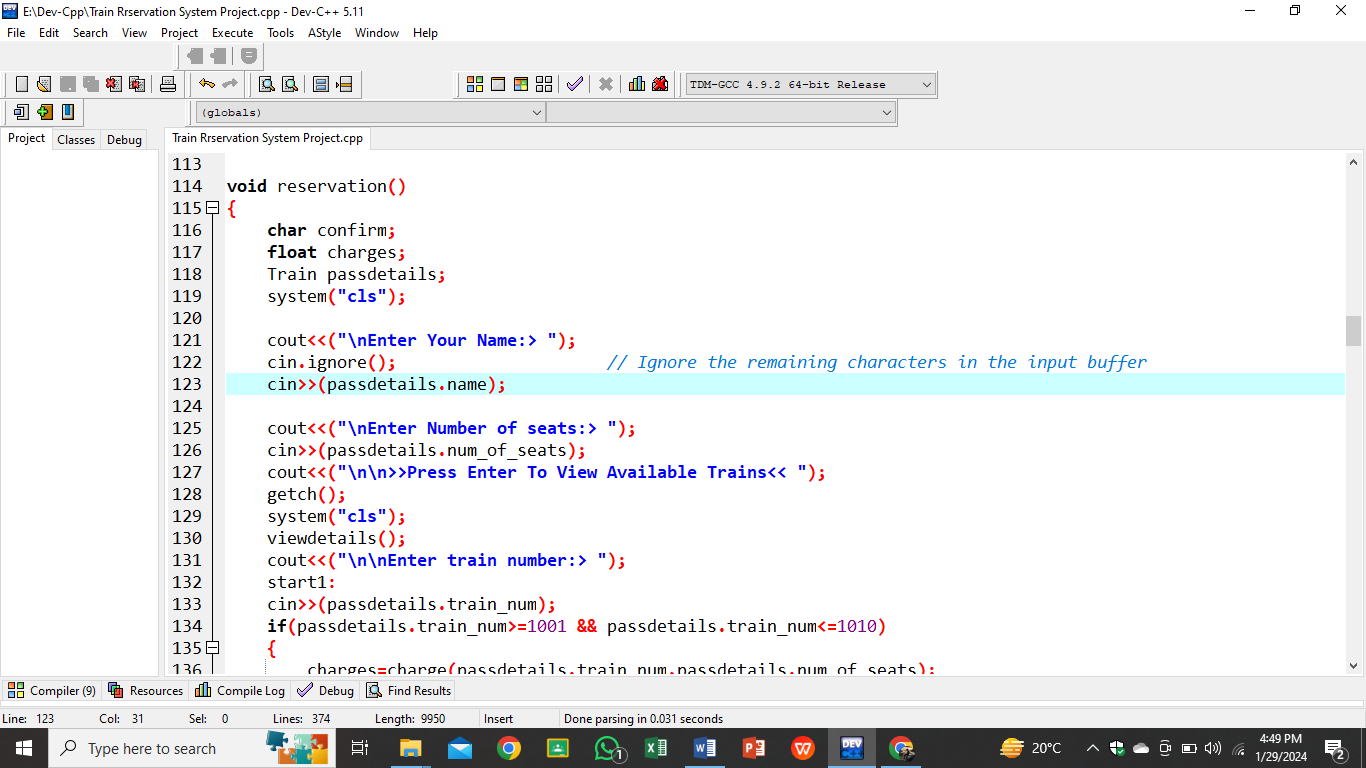
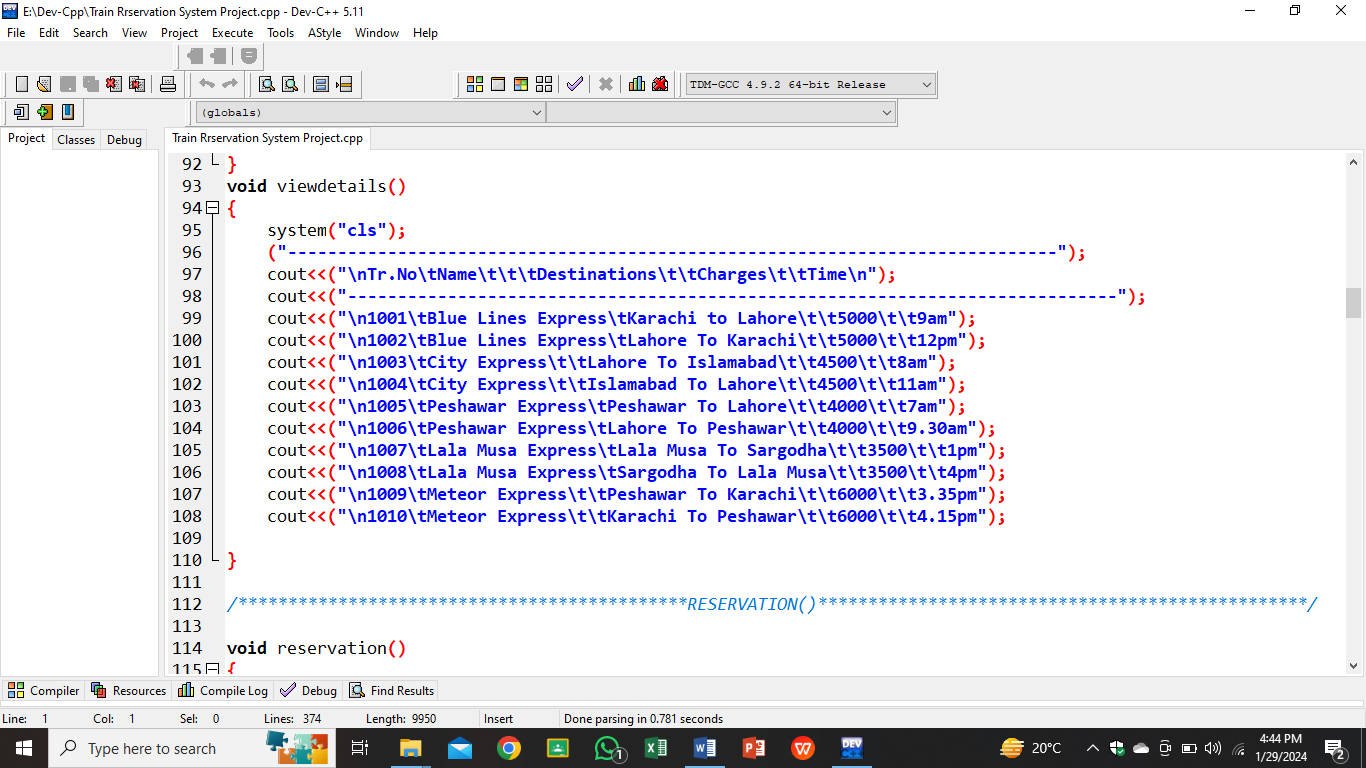
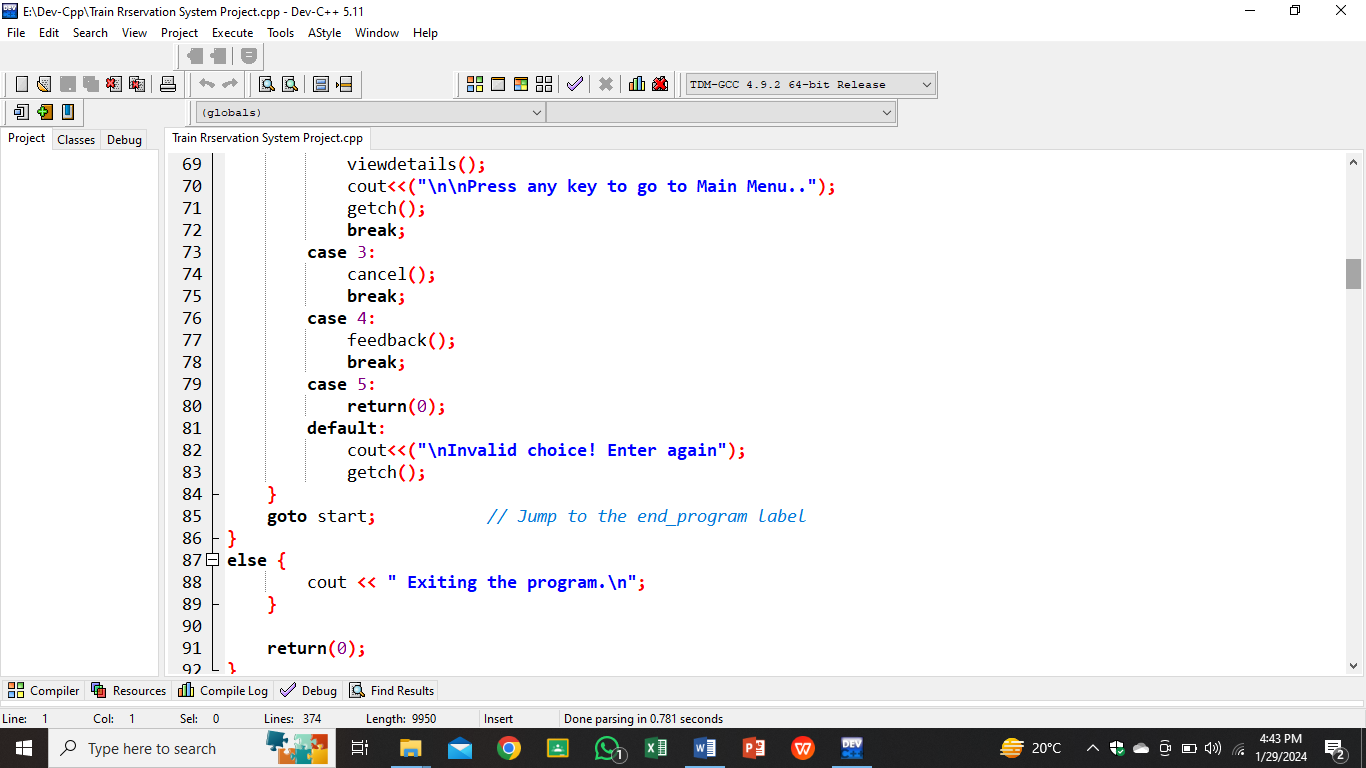
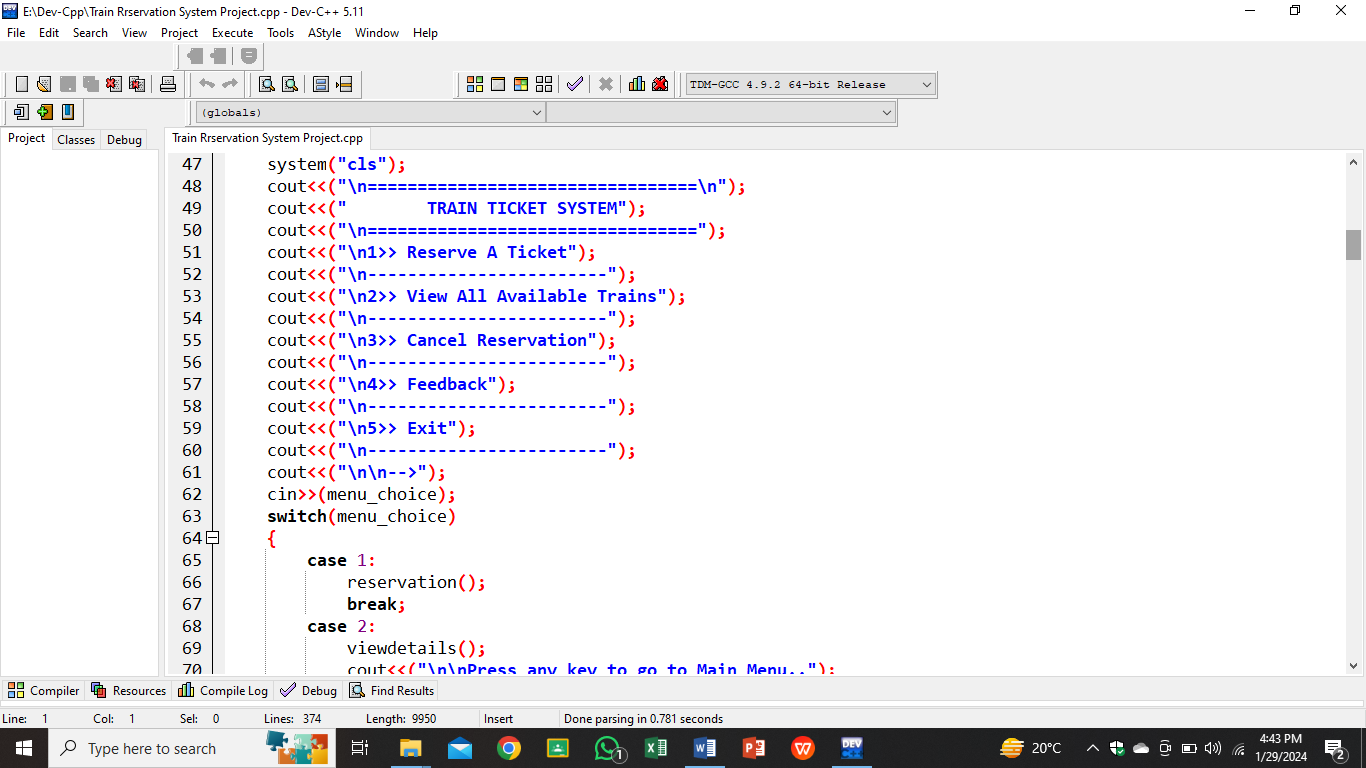
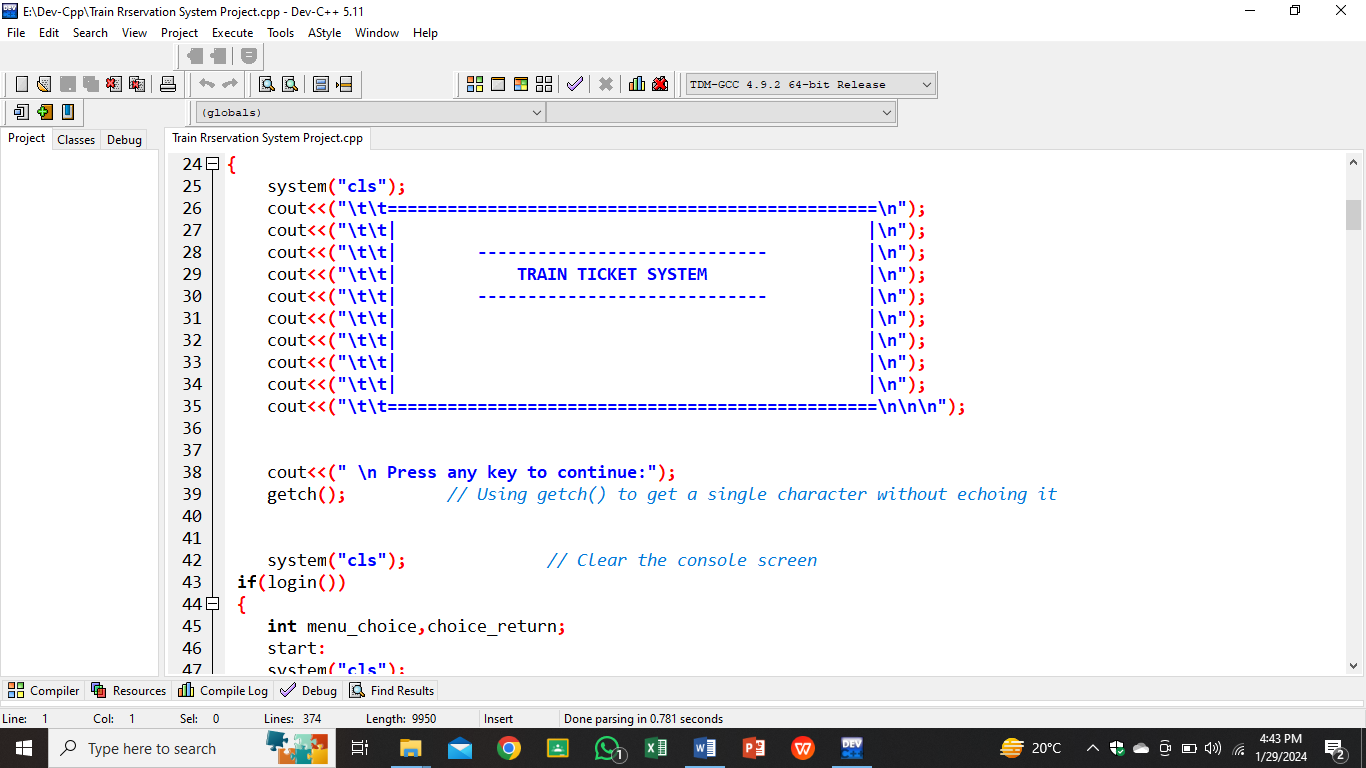
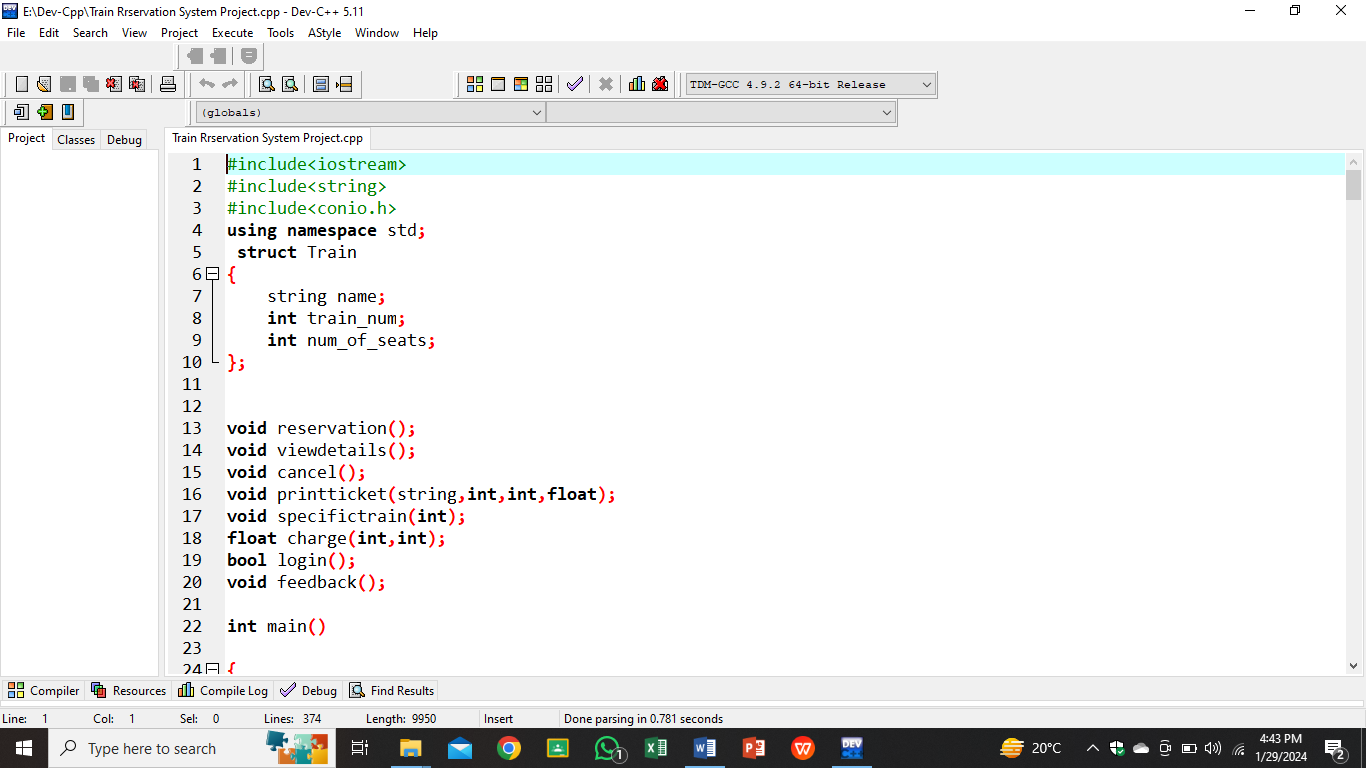
* **Data formats:**

Data exchange between different parts of the system or external entities would happen through specific data formats like JSON, XML, or plain text.

* **Messaging systems:**

Certain applications might employ messaging systems like message queues or pub/sub systems for asynchronous communication between components.

1. **Screenshots of the C++ Code:**

****