A Project Report

On

**“Conversion Calculaor”**

For The Course

**“ICT 1200: Project-I”**

**Supervised by**

Dr. Md. Abir Hossain

Associate Professor

Department of ICT, MBSTU.

**Submitted By**

Abdur Rawf Akand (IT23050)

Md. Tajul Islam (IT 23051)

****

**DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY**

SANTOSH, TANGAIL-1902, Dhaka, Bangladesh

**Contents**

**Page No.**

**Cover Page---------------------------------------------------------------------- 1**

**Declaration---------------------------------------------------------------------------------- 9**

**Acknowledgements---------------------------------------------------------------10**

**Chapter 1: Introduction**------------------------------------------------- **11**

**1.1. Introduction**---------------------------------------------------------------- **11**

**1.2. Motivation of the Project**------------------------------------------------------ **11**

1.2.1. Addressing the Need for Accurate and Efficient Unit Conversion**-- ----12**

1.2.2. Enhancing Practical Utility and Usability**--------------------------**-- ---**12**

1.2.3. Learning and Implementing Programming Concepts**--------------- ---13**

1.2.4. Contributing to User-Centric Software Development**------------------14**

1.2.5. Solving RealWorld Problems with Technology**--------=------------- 14**

1.2.6. Building a Foundation for Future Projects**---------------------------- 14**

**1.3. Project Vision** -------------------------------------------===------------------- **14**

1.3.1. Usability: Prioritizing User Experience**--------------------------------**----- **14**

1.3.2. Precision and Accuracy: Delivering Reliable Results**---------------------- 15**

1.3.3. Flexibility: Supporting Diverse Unit Systems and Conversion Needs-----**- 16**

1.3.4. Scalability: Building a Framework for Future Expansion**------------------ 16**

1.3.5. Educational Value: A Tool for Learning and Growth**---------------------- 17**

1.3.6. Reliability and Offline Access: A Dependable Resource**------------------- 18**

**1.4 Main Objective of the Project------------------------------------------18**

1.4.1. Versatile Unit Conversion Across Multiple Categories**---------------- 18**

1.4.2. Accuracy and Precision in Conversions**--------------------------**------ **19**

1.4.3. Efficient and User-Friendly Interface**---------------------------------- 19**

1.4.4. Educational Value and Learning Outcomes**--------------------------- 20**

1.4.5. Scalability and Future Expansion**--------------------------------------21**

1.4.6. Offline Access and Lightweight Design**--------------------------------21**

**1.5 Conclusion----------------------------------------------------------------- 21**

**Chapter - 2: Related Project Overview ---------------- 23**

**2.1. Introduction**--------------------------------**------------------------------------- 23**

**2.2 Related Project--------------------------------------------------------------- 24**

2.2.1. Online Unit Converters**------------------------------------------------- 24**

2.2.2. Mobile Conversion Apps**----------------------------------------------- 24**

2.2.3. Pre-installed System Converters**---------------------------------------- 25**

2.2.4. Specialized Scientific and Engineering Conversion Tools**------------- 25**

**2.3 Problem of Existing Project------------------------------------------------ 26**

2.3.1. Dependence on Internet Connectivity**---------------------------------- 26**

2.3.2. Lack of History and Data Persistence**---------------------------------- 26**

2.3.3. Limited Categories and Units**--------------------------------------------- 27**

2.3.4. Advertisements and Privacy Concerns**------------------------------------ 27**

2.3.5. Lack of Customization and Flexibility**------------------------------------ 27**

2.3.6. Complexity in Specialized Tools**------------------------------------------ 27**

2.3.7. Inconsistent User Interface**----------------------------------------------- 28**

2.3.8. Inability to Clear or Manage History**-------------------------------------- 28**

2.3.9. Cost and Licensing**-------------------------------------------------------- 28**

**2.4 Proposed Project Overview------------------------------------------------ 29**

2.4.1. Offline Functionality**------------------------------------------------- 29**

2.4.2. Wide Range of Conversion Categories**------------------------------- 29**

2.4.3. History Feature**------------------------------------------------------- 30**

2.4.4. Data Persistence**------------------------------------------------------ 30**

2.4.5. User-Friendly Interface**----------------------------------------------- 30**

2.4.6. Customizable Conversion Options**----------------------------------- 30**

2.4.7. Efficiency and Accuracy**--------------------------------------------- 31**

**2.5 Conclusion------------------------------------------------------------------ 31**

**Chapter 3:Project Design------------------------------------------ 32**

**3.1. Introduction** ------------------------------------------------------------------------- **32**

**3.2 Hardware and Software Requirement------------------------------------- 33**

3.2.1 Hardware Requirements**------------------------------------------------ 33**

3.2.2 Software Requirements**------------------------------------------------- 33**

**3.3 Proposed System Design---------------------------------------------------- 35**

3.3.1. Main Menu**------------------------------------------------------------- 35**

3.3.2. Conversion Functions**-------------------------------------------------- 36**

3.3.3. History Management**--------------------------------------------------- 37**

3.3.4. Clear History**----------------------------------------------------------- 38**

3.3.5. File Handling for Data Persistence**------------------------------------- 38**

**3.4 Conclusion------------------------------------------------------------------- 39**

**Chapter - 4: Implementation------------------------------ 40**

**4.1. Introduction** -------------------------------------------------------------------------- 40

**4.2. lengthConversion----------------------------------------------------------- 40**

4.2.1. Function Declaration**---------------------------------------------------- 40**

4.2.2. Variable Declarations**--------------------------------------------------- 41**

4.2.3. Menu Display**----------------------------------------------------------- 41**

4.2.4. User Input**--------------------------------------------------------------- 42**

4.2.5. Switch Statement**-------------------------------------------------------- 42**

4.2.6. Case Statements for Each Conversion**---------------------------------- 42**

4.2.7. File Closure**--------------------------------------------------------------45**

**4.3 WeightConversion------------------------------------------------------------45**

4.3.1. Function Declaration**--------------------------------------------------- 45**

4.3.2. Variable Declarations**-------------------------------------------------- 45**

4.3.3. Menu Display**---------------------------------------------------------- 48**

4.3.4. User Input**-------------------------------------------------------------- 48**

4.3.5. Switch Statement**------------------------------------------------------ 48**

4.3.6. Case Statements for Each Conversion**--------------------------------- 48**

4.3.7. File Closure**----------------------------------------------------------- 49**

**4.4. TemperatureConversion-------------------------------------------------- 49**

4.4.1. Function Declaration**-------------------------------------------------- 49**

4.4.2. Variable Declarations**------------------------------------------------- 49**

4.4.3. Menu Display**---------------------------------------------------------- 51**

4.4.4. User Input**-------------------------------------------------------------- 51**

4.4.5. Switch Statement**------------------------------------------------------ 51**

4.4.6. Case Statements for Each Conversion**--------------------------------- 51**

4.4.7. File Closure**------------------------------------------------------------ 52**

**4.5 VolumeConversion---------------------------------------------------------- 52**

4.5.1. Function Declaration**------------------------------------------ --------- 52**

4.5.2. Variable Declarations: **-------------------------------------------------- 52**

4.5.3. Menu Display**----------------------------------------------------------- 53**

4.5.4. User Input**-------------------------------------------------------------- 53**

4.5.5. Switch Statement**------------------------------------------------------- 53**

4.5.6. Case Statements for Each Conversion**--------------------------------- 53**

4.5.7. File Closure**------------------------------------------------------------ 56**

**4.6 Currency Conversion---------------------------------------------------- 56**

4.6.1. Function Declaration--------------------------------------------------- **56**

4.6.2. Variable Declarations-------------------------------------------------- **56**

4.6.3. Declares three variables----------------------------------------------- **56**

4.6.4. File Opening------------------------------------------------------------- **57**

4.6.5. Currency Conversion--------------------------------------------------- **57**

**4.7 History------------------------------------------------------------------------ 58**

4.7.1. Function Declaration**---------------------------------------------------- 58**

4.7.2. Variable Declaration and Initial Display**-------------------------------- 58**

4.7.3. Option 1: Search By Date**----------------------------------------------- 58**

4.7.4. Opening and Reading the File**------------------------------------------- 59**

4.7.5. Searching for the Date**--------------------------------------------------- 59**

4.7.6. No Matching History Found**-------------------------------------------- 60**

4.7.7. Option 2: Show All History**--------------------------------------------- 60**

**4.8. Clear History---------------------------------------------------------------- 61**

4.8.1. Function Declaration**---------------------------------------------------- 61**

4.8.2. File Opening in Write Mode**-------------------------------------------- 61**

**4.9. Code Execution and Results**-------------------------------------------------------- **61**

**Chapter - 5: Limitation and Future Scope------------------ 67**

**5.1 Introduction------------------------------------------------------------------ 67**

**5.2 Limitations of the Current Project--------------------------**----**----------- 67**

5.2.1. Limited Conversion Types**---------------------------------------------- 67**

5.2.2.User Interface Constraints**----------------------------------------------- 67**

**5.3 Future Scope of the Project-------------------------------------------------- 69**

5.3.1. Enhanced Conversion Categories: **------------------------------------- 69**

5.3.2. Multilingual **------------------------------------------------------------ 69**

**5.4 Conclusion------------------------------------------------------------------- 69**

**Chapter 6: Conclusion----------------------------------------------- 71**

**6.1 Summary of the Project----------------------------------------------------- 71**

**6.2 Lessons Learned------------------------------------------------------------- 71**

**6.3 Final Remarks on the Conversion Calculator----------------------------- 72**

**6.4 Conclusion------------------------------------------------------------------- 73**

**References ---------------------------------------------------------------------------- 74**

**Declaration**

This is to certify that the work presented in this project is carried out by the candidate under the supervision of **Dr. Md. Abir Hossain** in the department of Information and Communication Technology, MBSTU, Tangail, Bangladesh. It is also declared that neither of this project has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Signature of Supervisor

**Dr. Md. Abir Hossain**

                                                                                  Associate Professor

Dept. of ICT, MBSTU

**Acknowledgements**

I am immensely grateful to the Almighty Allah for the successful completion of my dissertation. I wish to begin by extending my heartfelt gratitude to my supervisor, Dr. Md. Abir Hossain, Associate Professor in the Department of Information and Communication Technology at MBSTU. His unwavering support paved the way for the continuation of my ICT 1200 Project-I.

I would also like to express my deep appreciation for my supervisor's invaluable guidance, insightful suggestions, constant encouragement, and unwavering support throughout the entirety of the project. However, words seem insufficient to truly acknowledge the tremendous effort put forth by our esteemed teacher.

As always, our gratitude towards our family knows no bounds. The unwavering love and support from our parents continue to be the cornerstone of our lives.

**Chapter 1**

**Introduction**

#### 1.1 Introduction:

In the realm of daily life, as well as scientific and technical fields, unit conversions play an integral role. From converting lengths when measuring distances, to adjusting temperatures between scales, and even calculating weights during shopping, the need for accurate and efficient conversions is ever-present. Manually converting units can be time-consuming and prone to errors. To simplify this process, a **Conversion Calculator** becomes an essential tool.

The **Conversion Calculator** is designed to convert units across four main categories: **length, weight, temperature, currency and volume**. Users can select the type of conversion they wish to perform from a simple menu interface, and after providing the necessary input, the program will compute and display the result. This not only saves time but also ensures accurate conversions without the need for manual calculation. Furthermore, the program includes a feature that records all conversions performed into a history file, allowing users to review previous results at any time. Users can also clear the history if they no longer need the saved conversions.

#### 1.2. Motivation of the Project:

The idea behind the **Conversion Calculator** project stems from the universal need for accuracy and efficiency when converting between different units of measurement. In both professional and everyday settings, the ability to quickly and reliably perform unit conversions is essential. Whether in engineering, science, mathematics, commerce, or even household tasks, there are constant demands to convert lengths, weights, temperatures, and volumes. This project was motivated by a desire to provide a solution that simplifies these tasks through automation, reducing the risk of human error and enhancing productivity.

**1.2.1. Addressing the Need for Accurate and Efficient Unit Conversion**

In today’s fast-paced world, time is often of the essence. Manual unit conversions, though possible, are time-consuming and susceptible to mistakes, especially when dealing with complex conversions or less familiar units. Whether it is converting from one system (e.g., metric) to another (e.g., imperial) or handling various scientific measures, accurate conversion is critical. The **Conversion Calculator** seeks to solve this problem by providing a reliable tool that performs these conversions instantly and accurately.

The primary motivation was to eliminate the need for memorizing conversion formulas or performing repetitive calculations. By automating the process, the **Conversion Calculator** reduces the cognitive load on users and ensures that they can get the correct results without effort. This is particularly important for users who need to perform conversions regularly or in scenarios where a small mistake in conversion could lead to significant issues, such as in scientific research, engineering, or even financial calculations involving units of measure.

**1.2.2. Enhancing Practical Utility and Usability**

One of the key motivators for this project was to create a tool that is not only accurate but also practical and easy to use. Many people struggle with complex conversion tools or online calculators that may require an internet connection or present a steep learning curve. In contrast, this project was motivated by a desire to design a solution that can be easily navigated by anyone, regardless of their technical background.

The user-friendly interface allows users to select the category of conversion they need—whether it's length, weight, temperature, or volume—then enter their values and receive results instantly. This focus on simplicity and ease of use ensures that the **Conversion Calculator** can serve a broad audience, from students to professionals, or even individuals managing household tasks. The addition of features like saving the history of conversions further improves usability, providing users with a means of tracking their previous calculations without needing to re-enter data repeatedly.

**1.2.3. Learning and Implementing Programming Concepts**

From a technical perspective, the motivation behind the project also includes personal development in programming skills, specifically in C language. As a first-year project, the **Conversion Calculator** provides a practical opportunity to learn and implement several key programming concepts, such as:

* **Modular programming**: Breaking down the problem into different functions like lengthConversion(), weightConversion(), temperatureConversion(), and volumeConversion(), each of which handles specific tasks.
* **File handling**: By incorporating the history feature, the project offers hands-on experience with file input/output operations, a critical skill in many programming tasks.
* **Control flow and user input validation**: The program teaches effective use of control structures like loops, conditionals, and switch statements, as well as handling user input dynamically.

The motivation to build a real-world application that is not only functional but also well-structured encouraged the exploration of good coding practices. By implementing this project in C, students learn how to manage memory efficiently, write optimized code, and develop software with practical utility.

**1.2.4. Contributing to User-Centric Software Development**

The motivation behind the project is also grounded in a desire to develop user-centric software. The **Conversion Calculator** is designed with the user experience in mind. The goal is to create a program that addresses real-world needs while also being lightweight and efficient. The decision to save conversion histories and provide options for clearing them reflects a commitment to putting the user first. Users can track their conversion activities over time, making it useful for tasks requiring repetitive or complex conversions.

**1.2.5. Solving Real-World Problems with Technology**

This project is motivated by the broader goal of using technology to solve real-world problems. Unit conversion may seem like a simple task, but the reality is that it plays a vital role in many industries. For instance, engineers need precise conversions for measurements in construction or mechanical design. Scientists often need to convert between temperature scales in research, while nutritionists and cooks frequently convert between different units of weight and volume. The **Conversion Calculator** simplifies these tasks for a wide range of users.

**1.2.6. Building a Foundation for Future Projects**

Finally, this project serves as a foundation for future development. The motivation behind the **Conversion Calculator** is not only to complete a functional program but also to build skills and a framework that can be expanded upon in the future. As the first-year project, it provides a starting point for more complex applications, such as adding additional conversion types (like as: currency, time, speed) in later stages of study.

#### 1.3. Project Vision:

The vision for the **Conversion Calculator** project is to develop an intelligent, user-friendly, and versatile tool that seamlessly assists users in converting various units of measurement with maximum accuracy and efficiency. The project aspires to be more than just a basic conversion utility; it aims to evolve into a robust and essential application that caters to both everyday users and professionals in diverse fields, such as education, engineering, science, commerce, and beyond. The key components of this vision include usability, precision, flexibility, scalability, and educational value.

**1.3.1. Usability: Prioritizing User Experience**

One of the primary goals of the **Conversion Calculator** is to provide a seamless and straightforward experience for users. The vision is to create a tool that anyone—whether a student, professional, or individual with minimal technical skills—can use without difficulty. This usability is achieved through a clean, intuitive interface that guides the user step-by-step, requiring minimal effort to select conversion categories, input values, and view results.

The vision also includes creating a program that offers quick results, helping users save time and reduce the burden of manual conversions. By offering features like history tracking, where previous conversions are saved for reference, and clear history functions, users can conveniently access or remove their data without having to redo past work. These design elements are focused on simplifying the conversion process while making the experience hassle-free and efficient.

#### 1.3.2. Precision and Accuracy: Delivering Reliable Results

A crucial aspect of the **Conversion Calculator’s** vision is the delivery of precise and accurate results. The program ensures that conversions are mathematically sound and reliable across a range of units, such as length, weight, temperature, and volume. This accuracy is particularly important in professional settings where even small errors in conversion could lead to significant issues, such as engineering miscalculations or scientific research inaccuracies.

The vision sees this project becoming a tool trusted for critical tasks that require the utmost precision, with the ability to handle a wide variety of conversion scenarios. For example, scientists can rely on the tool for exact temperature or volume conversions, while architects can use it for accurate length and area measurements during construction planning.

**1.3.3. Flexibility: Supporting Diverse Unit Systems and Conversion Needs**

A key part of the project’s vision is the flexibility to adapt to different user needs. The **Conversion Calculator** currently supports a wide range of units from different systems, such as metric and imperial, to ensure that users from diverse fields or regions can find the conversions they need. Whether it’s converting kilometers to miles, Celsius to Fahrenheit, or liters to gallons, the vision is to provide a comprehensive set of conversions to cover all essential categories.

In a future iteration of the project, the vision also includes the possibility of enabling users to choose between different unit conversion standards. For example, some fields, like the food and beverage industry, may use specific conversion standards that vary slightly from scientific or technical fields. Providing customizable or pre-set standards for conversion would further enhance the calculator’s applicability.

**1.3.4. Scalability: Building a Framework for Future Expansion**

Another key aspect of the project’s vision is its scalability. The **Conversion Calculator** is designed as a foundational project, with the potential to grow and evolve as new features and functions are added. This scalability ensures that the project can move beyond its current capabilities to incorporate advanced programming techniques, additional conversion categories, and potentially even external integration.

The vision includes a future in which the calculator can scale to handle more complex tasks, such as:

* **Mobile and Web Versions:** As part of its scalability, the vision is to adapt the calculator for mobile devices and web applications. This would enable users to access the tool on smartphones, tablets, or web browsers without needing to download or install a program.

By making the calculator easily extendable, future developers can also contribute to or customize the project, ensuring that it remains relevant and functional in a rapidly evolving technological landscape.

**1.3.5. Educational Value: A Tool for Learning and Growth**

The **Conversion Calculator** is envisioned as a tool that not only serves a functional purpose but also provides educational value. As part of a first-year programming project, it introduces fundamental concepts in coding, such as loops, conditionals, file handling, and modular programming. The project encourages hands-on learning, enabling students to understand how theoretical programming concepts translate into real-world applications.

Beyond its role in education for developers, the calculator can also serve as an educational resource for users. In future versions, the tool could provide explanations for each conversion formula used, allowing users to better understand the mathematics behind the calculations. This feature would be especially useful for students studying science, mathematics, or engineering, where understanding the rationale behind conversions is essential.

In a broader sense, the project vision includes transforming this simple conversion calculator into an interactive learning tool. It could potentially feature quizzes, tips, or exercises that help users test their understanding of unit conversions, making it both a practical and educational tool.

**1.3.6. Reliability and Offline Access: A Dependable Resource**

The **Conversion Calculator** is envisioned as a reliable tool that users can depend on at any time, even without internet access. Unlike many online calculators, this project is built as a standalone application that functions offline, making it particularly useful in environments where internet connectivity is not available. The vision is to ensure the program remains a dependable resource for users in any context, whether in a remote location, during travel, or in areas with limited online access.

By emphasizing reliability, the project aspires to build user trust and ensure that the calculator becomes an essential tool for daily use. This vision includes improving the underlying code to handle larger or more complex data inputs without crashing, ensuring smooth and uninterrupted performance.

**1.4 Main Objective of the Project:**

The primary objective of the **Conversion Calculator** project is to develop a comprehensive, accurate, and user-friendly tool that enables seamless conversions across a wide range of units. Designed with versatility and ease of use in mind, the calculator aims to assist users in converting measurements related to **length, weight, volume, temperature, and number systems** quickly and efficiently. The project's ultimate goal is to provide users with a tool that is both educational and functional, helping them avoid the complexities of manual conversions and reduce errors associated with calculating measurements manually.

**1.4.1. Versatile Unit Conversion Across Multiple Categories**

One of the core objectives of the project is to create a tool that offers versatility by supporting a wide variety of conversion categories. Users need a reliable tool to convert units from different measurement systems, including both metric and imperial units. The conversion categories include:

* **Length Conversions:** Convert between units like meters, kilometers, feet, inches, miles, etc.
* **Weight Conversions:** Handle conversions between kilograms, grams, pounds, ounces, etc.
* **Volume Conversions:** Enable conversions between liters, milliliters, gallons, cubic meters, and more.
* **Temperature Conversions:** Provide accurate conversions between Celsius, Fahrenheit, and Kelvin scales.

The project’s objective is to ensure that these conversions are comprehensive and easy to perform, meeting a wide array of user needs in both professional and everyday contexts. The goal is to make the tool flexible enough to be useful for everyone, from students needing quick homework solutions to professionals in industries where precise measurements are crucial.

**1.4.2. Accuracy and Precision in Conversions**

Another fundamental objective is to guarantee the highest level of accuracy and precision in every conversion performed by the tool. When users rely on the calculator to handle critical tasks, such as engineering measurements or scientific data, even small miscalculations can lead to significant consequences. As such, the project places a strong emphasis on ensuring that every conversion is mathematically accurate and adheres to standardized conversion formulas.

To achieve this, the program is designed to handle various decimal places, ensuring that even conversions requiring high precision (such as those used in research or high-stakes industries) are performed flawlessly. The objective is to build trust in the calculator’s reliability, making it a dependable tool for users needing exact results.

**1.4.3. Efficient and User-Friendly Interface**

The project’s objective also focuses on creating a tool that is easy to use, even for individuals who may not have a technical background. The interface design is central to this, providing a smooth and straightforward experience for users. The aim is to minimize the steps required to input data, select conversion types, and retrieve results, thereby reducing the time spent on manual calculations.

This means that the program will offer a command-line interface (CLI) where users can easily navigate between different conversion types, input their data, and receive results instantly. Another key objective is to offer clear instructions, error messages, and feedback to ensure that users are always informed about the next steps or any mistakes made in their inputs.

Looking forward, an additional objective is to create a program that can be expanded to include graphical user interfaces (GUIs) or mobile apps, providing even more user-friendly interaction.

**1.4.4. Educational Value and Learning Outcomes**

An important objective of the **Conversion Calculator** is to provide a learning experience, not only for users but also for the developers themselves. As a first-year project, the calculator introduces the developers to core programming concepts such as:

* Input/output handling
* Loops and conditionals
* Data validation
* Modularity and code structuring
* Basic mathematical operations

By completing this project, developers gain practical experience in building software applications from the ground up, learning how to apply their theoretical knowledge in real-world scenarios. Additionally, the calculator itself can serve as an educational tool for users, helping them understand how various units are related and converted. In future versions, the program could include explanations of the conversion formulas, making it even more educational for users interested in learning the math behind the conversions.

**1.4.5. Scalability and Future Expansion**

Another key objective is to build the **Conversion Calculator** with scalability in mind. While the current version focuses on essential unit conversions, the tool is designed to be expandable. The objective is to create a flexible code structure that can easily incorporate new features, unit categories, or even external data sources in the future.

For example, future iterations of the project may include more specialized units (such as currency conversions or scientific constants), add advanced error-checking algorithms, or integrate with web APIs to offer real-time data (e.g., for currency conversions). The objective is to ensure that the tool remains adaptable, with the capacity to grow and evolve as new needs arise.

**1.4.6. Offline Access and Lightweight Design**

Given that not all users will have access to the internet at all times, an important objective is to ensure that the **Conversion Calculator** works offline. This ensures that users in remote areas or situations where internet access is unreliable can still perform conversions without needing to be online. The tool is designed to be lightweight and fast, making it easily accessible on any basic computing platform, without requiring significant processing power or memory.

This objective ensures that the tool remains reliable in any context, making it a practical and dependable resource for users who need it for various purposes.

* 1. **Conclusion:**

This **Conversion Calculator** project is designed to be a practical tool for performing unit conversions efficiently and accurately. Its modular structure, ease of use, and added functionality like historical tracking makes it a highly useful application. By handling conversions in critical areas such as length, weight, temperature, and volume, this project demonstrates how a simple, user-centered program can make a real difference in everyday life as well as in more technical fields.

In summary, the **Conversion Calculator** project offers a practical solution to a common problem: the need for quick and accurate unit conversions. By covering multiple categories of conversions, including length, weight, temperature, and volume, the calculator provides users with a versatile tool that can be used in a wide range of scenarios. The project not only enhances productivity by automating conversion tasks but also demonstrates the effectiveness of C programming in creating functional and efficient applications.

**Chapter 2**

**Related Project Overview**

**2.1 Introduction:**

A conversion calculator is a tool used to convert various units from one form to another. In everyday life, there are frequent instances where one needs to convert measurements like length, weight, temperature, volume, and numbers between different unit systems. For example, converting kilometers to meters, Celsius to Fahrenheit, or grams to kilograms is a common task in scientific, industrial, and even casual settings. The need for accurate conversions is crucial in many fields such as engineering, construction, cooking, and academics. A reliable conversion calculator helps users avoid errors that may arise from manual calculations, ensuring precision and efficiency.

In this project, the focus is on developing a comprehensive conversion calculator written in C programming language. The calculator includes functionality for converting units of length, weight, temperature, and volume, as well as offering a history tracking feature. The project provides a user-friendly command-line interface where users can easily navigate through different conversion options. Additionally, the history feature allows users to track their past conversions, making it easier to refer back to previous results.

The project is structured to be modular, meaning different types of conversions are separated into their respective functions for ease of development and maintenance. The calculator also includes the ability to clear the history when needed, ensuring the user can maintain a clean record of current calculations. By integrating file handling, the project enables persistent storage of conversions, ensuring that data is not lost between sessions.

The goal of this conversion calculator project is to create a tool that is both practical and educational for new programmers. It demonstrates key programming concepts such as functions, file handling, control structures, and user interaction, making it a valuable learning experience while solving real-world problems.

**2.2 Related Project:**

Conversion calculators have been developed in various formats, ranging from simple online tools to complex software applications. These calculators are commonly used to convert measurements such as length, weight, temperature, volume, and other units between different systems (e.g., metric to imperial). Some of the most common related projects in the field of unit conversions are:

#### 2.2.1. ****Online Unit Converters****

Many websites offer online unit conversion tools where users can enter a value and choose from predefined units for conversion. Popular websites such as **UnitConverters.net** and **ConvertWorld.com** provide an easy-to-use interface for converting between various units like meters to miles, Celsius to Fahrenheit, or liters to gallons. These tools often include categories for a wide range of conversions, from basic measurements to specialized scientific and engineering conversions.

However, these online converters have limitations. Users need internet access to utilize these tools, and some websites can be cluttered with ads, making them less efficient. Additionally, these online tools may not offer persistent data storage, meaning once a session ends, the history of conversions is lost.

#### 2.2.2. ****Mobile Conversion Apps****

Apps like **Unit Converter** by Smart Tools or **Converter+** for smartphones provide a portable solution for users to perform conversions on the go. These apps typically cover a wide range of categories including scientific conversions, currency exchange rates, and unit conversions. Mobile apps also allow users to access these tools offline, which is an advantage over web-based solutions.

However, while mobile apps are convenient, they often come with limitations such as requiring frequent updates, which may affect their usability over time. Some apps may also collect user data or display intrusive ads, which could compromise user experience.

#### 2.2.3. ****Pre-installed System Converters****

Most modern operating systems, like Windows or macOS, come with pre-installed converters as part of the calculator application. These system converters allow users to convert common units like temperature, length, weight, and volume directly on their computers without the need to search online or install additional software.

While system-based converters are convenient, they may not be customizable or expandable to include specific units that some users need. Moreover, these built-in tools lack advanced features such as historical data tracking, multiple conversions at once, or the ability to clear history selectively.

#### 2.2.4. ****Specialized Scientific and Engineering Conversion Tools****

In technical fields like physics, chemistry, and engineering, specialized software tools like **Mathematica**, **MATLAB**, and **WolframAlpha** are used to perform complex conversions and calculations. These tools not only convert units but also handle equations, simulations, and advanced mathematical modeling. They are particularly useful for professionals and researchers who require high levels of precision and integration into larger calculations.

However, these specialized tools tend to be expensive and may have a steep learning curve, making them inaccessible to casual users or beginners. Additionally, they are often overkill for simple conversions, which can be achieved with lighter tools.

**2.3 Problem of Existing Project:**

Although various conversion tools exist in different formats such as online websites, mobile applications, and built-in system software, they often present several limitations and challenges that make them less than ideal for certain users. These problems can range from usability issues to technical and functional drawbacks. Here are some of the key problems observed in existing conversion calculators:

#### 2.3.1. ****Dependence on Internet Connectivity****

Many popular unit conversion tools, especially online-based calculators, require constant internet access. Websites such as **UnitConverters.net** or **ConvertWorld.com** cannot function offline, which is a major limitation for users who need to perform conversions in remote locations or where internet access is unreliable. This restriction affects accessibility and convenience, especially in situations where quick conversions are required.

#### 2.3.2. ****Lack of History and Data Persistence****

Most of the existing tools, especially online converters and simple system-based calculators, do not retain any history of conversions performed by the user. Once a session ends or a browser window is closed, all previous conversions are lost. For users who need to reference past calculations, this can be a significant inconvenience. Without the ability to review or retrieve previous conversions, users might have to re-enter data, leading to wasted time and potential errors in re-calculating.

#### 2.3.3. ****Limited Categories and Units****

While many tools offer basic conversions such as length, weight, and temperature, they often lack comprehensive support for specialized units or categories. For example, scientific or engineering conversions like pressure, force, energy, or even currency and data storage may not be included in basic calculators. Additionally, the inability to add or customize new conversion types makes these tools rigid and less useful for specific or advanced use cases.

#### 2.3.4. ****Advertisements and Privacy Concerns****

Many free mobile apps and online conversion tools are often cluttered with intrusive advertisements that disrupt the user experience. These ads not only slow down the interface but can also lead to distractions or accidental clicks. Additionally, some apps collect user data for targeted advertising, raising privacy concerns for users who are looking for a simple, ad-free conversion solution. These issues are particularly prevalent in free versions of apps and websites that rely on advertising revenue.

#### 2.3.5. ****Lack of Customization and Flexibility****

Pre-installed system converters, like those found in operating systems, are often limited in terms of flexibility. Users are typically unable to customize these calculators to their specific needs. For example, if a user frequently needs to convert units not included in the default options, there is no way to add those units to the system-based tools. Moreover, these converters lack advanced features such as saving multiple conversion results at once, batch processing of data, or managing specific settings for preferred units.

#### 2.3.6. ****Complexity in Specialized Tools****

While some advanced software like **MATLAB** or **WolframAlpha** offers powerful and precise unit conversion functionalities, they are often over-complicated for casual users who only need to perform basic conversions. These tools are designed for professionals in scientific and engineering fields and may require a high level of technical expertise to operate. Furthermore, they are typically resource-intensive, expensive, and have a steep learning curve, making them inaccessible to beginners or non-technical users.

#### 2.3.7. ****Inconsistent User Interface****

The user interfaces of many conversion tools vary greatly in terms of ease of use. Online converters often suffer from cluttered layouts with excessive text or ads, making it difficult for users to quickly navigate to the desired conversions. Mobile apps may also have inconsistent designs, with buttons or options that are too small, not intuitive, or hidden behind menus. This lack of a streamlined, user-friendly interface hampers the overall experience and reduces efficiency, especially for users looking for quick conversions.

#### 2.3.8. ****Inability to Clear or Manage History****

Even in cases where some tools provide a history feature, there are limitations in how users can interact with that history. For example, users may not have the ability to clear specific entries or reset their entire history in many systems. This creates clutter, especially when users perform many conversions over time. Without the ability to clear or selectively delete past records, the history feature loses its usefulness and becomes a hindrance.

#### 2.3.9. ****Cost and Licensing****

Advanced tools like **MATLAB** and **WolframAlpha** require expensive licenses, which are out of reach for many casual users or students. Even mobile apps may have premium versions that lock important features behind a paywall. This cost factor limits the availability of fully-featured conversion tools for users who might be looking for a simple, free alternative that performs basic but essential conversions efficiently.

**2.4 Proposed Project Overview:**

The proposed project, **Conversion Calculator**, is designed to address the limitations of existing conversion tools by offering a simple, flexible, and offline solution that caters to a wide range of users. This tool is intended to serve as a comprehensive, user-friendly utility for performing various types of conversions in categories such as **length, weight, volume, temperature**, and **number systems**. The goal is to provide a robust calculator that can perform essential conversions quickly and efficiently, while offering advanced features like **data persistence, history tracking, and offline functionality**.

#### 2.4.1. ****Offline Functionality****

One of the key features of the proposed calculator is that it does not require an internet connection. Users can perform all conversions offline, making it accessible in any environment, including remote areas or places with limited connectivity. This ensures that the tool is always available when needed, regardless of the user’s location.

#### 2.4.2. ****Wide Range of Conversion Categories****

Unlike many existing tools that focus on just one or two categories, the proposed project covers multiple conversion categories, including:

* **Length Conversion:** Meters, kilometers, feet, inches, miles, and centimeters.
* **Weight Conversion:** Grams, kilograms, pounds, ounces, and stones.
* **Volume Conversion:** Liters, gallons, milliliters, and cubic meters.
* **Temperature Conversion:** Celsius, Fahrenheit, and Kelvin.

This broad spectrum of options makes the calculator versatile and useful for various fields, such as education, daily use, engineering, and scientific purposes.

#### 2.4.3. ****History Feature****

One major shortcoming of many existing tools is the lack of a history feature. In the proposed calculator, each conversion is logged and saved in a text file, allowing users to review their previous calculations. This feature is especially beneficial for users who perform multiple conversions or need to reference past results for comparison or documentation purposes.

* The history can be viewed in a user-friendly format, displaying the conversions performed during each session.
* A **clear history** function is also included, allowing users to delete past records if necessary, ensuring the tool stays uncluttered.

#### 2.4.4. ****Data Persistence****

All conversion data is logged in a file, making it possible for users to store their work over time. This data can be retrieved even after the program has been closed and reopened, ensuring that users do not lose important calculations. The persistence of data is critical for professionals or students who require continuous access to their past conversions over extended periods.

#### 2.4.5. ****User-Friendly Interface****

The calculator is designed with simplicity and ease of use in mind. Users are presented with a clear and intuitive **menu-based interface** where they can easily select the type of conversion they wish to perform. Each category (length, weight, temperature, and volume) has sub-options to ensure that the desired unit conversions are readily accessible. The interface guides users step-by-step through the input and output processes, ensuring that even non-technical users can navigate it comfortably.

#### 2.4.6. ****Customizable Conversion Options****

Although the initial version of the calculator includes predefined unit conversions, future iterations of the project could allow for customizable options where users can add specific units or conversion formulas according to their needs. This flexibility ensures that the tool can evolve based on user requirements and feedback, making it adaptable to various industries or academic fields.

#### 2.4.7. ****Efficiency and Accuracy****

The proposed calculator emphasizes computational accuracy, ensuring that all conversions are performed with precision. Each formula used for conversion is derived from well-established mathematical principles. Whether converting between metric and imperial systems or between temperature scales, users can trust that the results are accurate and reliable.

**2.5 Conclusion:**

In conclusion, the **Conversion Calculator** project is designed to offer a versatile, efficient, and user-friendly tool for performing a wide variety of conversions. By addressing the shortcomings of existing calculators, such as limited functionality, lack of offline access, and absence of history tracking, this project provides a robust solution that can be used in multiple domains, including education, engineering, and daily life.

With its offline functionality, comprehensive range of conversion categories, data persistence, and history tracking, the proposed calculator ensures convenience and reliability for users who require quick and accurate conversions. The intuitive interface and clear documentation further enhance the user experience, making it accessible to individuals with varying levels of technical expertise.

By offering a flexible, portable, and resource-efficient tool, the Conversion Calculator not only fills the gaps left by existing tools but also serves as a foundation for further enhancements and features in future versions. As such, this project represents a significant step forward in providing a comprehensive and reliable solution for all types of conversion needs.

**Chapter 3**

**Project Design**

**3.1. Introduction:**

The design and implementation phase of the **Conversion Calculator** project plays a critical role in ensuring the system operates smoothly and fulfills the intended functionality. This chapter outlines the foundational structure of the project, covering the design methodologies, system architecture, and the actual process of converting the design into an executable program.

The **Conversion Calculator** is implemented using the C programming language, chosen for its simplicity, efficiency, and low-level control over system resources. The design phase involves understanding the requirements, creating a detailed system design, and then translating that design into functional code. This stage ensures that all major components—such as length, weight, temperature, volume, and number conversions—work seamlessly together.

The chapter also elaborates on the necessary hardware and software requirements, as well as the tools and technologies used to build and execute the project. In addition, the system's modular design and architecture are discussed, which highlight how various conversion functionalities are segregated into separate modules to maintain clarity and ease of maintenance. The chapter ends by detailing the header files and supporting functions that enable the system to handle multiple types of conversions and maintain the history of the operations performed.

The goal of this chapter is to provide a comprehensive overview of the project's design, laying out the step-by-step process from initial planning to final implementation, with an emphasis on how each component contributes to the overall functionality of the system. This approach ensures that the program is scalable, efficient, and easy to maintain, while also catering to the primary needs of the user.

**3.2 Hardware and Software Requirement:**

The **Conversion Calculator** project relies on both hardware and software components to function effectively. This section outlines the minimum hardware specifications required to run the application and the software tools used to design, develop, and test the system.

**3.2.1 Hardware Requirements**

To successfully run the **Conversion Calculator**, a basic computing environment is required. The hardware specifications are modest, ensuring the project can be executed on a wide range of systems. Below are the minimum hardware requirements:

* **Processor:** Intel Pentium 4 or higher
* **RAM:** 512 MB (minimum), 1 GB or more (recommended for better performance)
* **Storage:** 50 MB of free space for installation and data storage
* **Display:** Standard VGA (1024x768 resolution or higher)
* **Input Devices:** Keyboard and mouse for user interaction
* **Operating System:** Compatible with Windows, macOS, or Linux operating systems that support C program compilation.

These hardware specifications ensure that the program runs efficiently without requiring significant system resources. As a lightweight application, it can function on most modern computing devices, including desktops and laptops.

**3.2.2 Software Requirements**

The development and execution of the **Conversion Calculator** involve the use of several software tools and libraries, mainly centered around C programming. The following software components are required:

* **Operating System:**
  + The program is platform-independent and can be compiled and run on any system that supports C programming, including:
    - Microsoft Windows (XP or later)
    - macOS
    - Linux distributions such as Ubuntu or Fedora
* **Integrated Development Environment (IDE):**
  + An IDE is required for writing, compiling, and debugging the C code. Some recommended IDEs include:
    - **Code::Blocks:** A lightweight, open-source IDE for C/C++ programming.
    - **Dev-C++:** A free, portable IDE that supports C/C++ languages.
    - **Visual Studio Code:** A modern code editor that can be configured for C programming with extensions.
* **C Compiler:**
  + A C compiler is necessary to translate the written code into machine-readable instructions. Some common options are:
    - **GCC (GNU Compiler Collection):** Available on Windows, macOS, and Linux, GCC is widely used for C program compilation.
    - **Clang:** An alternative to GCC, available on macOS and Linux.
    - **MinGW:** A minimalist GNU for Windows that provides a GCC compiler for Windows.
* **Libraries and Header Files:**
  + Standard C libraries and header files are required to implement various functionalities such as file handling, time operations, and mathematical calculations. These include:
    - <stdio.h>: Standard input/output operations.
    - <math.h>: Mathematical functions used in conversions.
    - <time.h>: Time functions to record conversion history with timestamps.
    - <stdlib.h>: Used for memory management and utility functions.
* **Text Editor:**
  + A basic text editor is required to write and edit C source code files. Some popular choices include:
    - Notepad++ (Windows)
    - Sublime Text (Cross-platform)
    - Vim or Nano (Linux)
* **File Management Tools:**
  + Since the project involves storing and retrieving conversion history in text files, a file management system or command-line tool is necessary to handle file operations.

**3.3 Proposed System Design:**

The proposed system is a console-based application built in C to perform a variety of unit conversions, including length, weight, temperature, and volume. This system also includes functionality to record conversion history, allowing users to review or clear previous conversions. To achieve modularity, ease of use, and maintainability, the system is designed with a structured, menu-driven approach and employs multiple functions to handle specific tasks.

Below is a detailed breakdown of the design and implementation of each component in the system.

**3.3.1.** **Main Menu**

The main menu serves as the entry point for users to interact with the application. It provides the user with seven choices, each corresponding to a different action within the program:

* **1. Length Conversion**
* **2. Weight Conversion**
* **3. Temperature Conversion**
* **4. Volume Conversion**
* **5. History**
* **6. Clear History**
* **7. Exit**

Each of these menu options is mapped to a function that performs a specific task. For instance, selecting "1" initiates the length conversion function, which then provides further options for specific length conversions. The main menu is implemented within a while loop, which allows the user to repeatedly access the menu until they decide to exit the program by selecting option 7.

#### 3.3.2. ****Conversion Functions****

Each conversion type (length, weight, temperature, and volume) is implemented as a separate function in the program. These conversion functions perform specific types of conversions depending on the user’s selection. Below is a breakdown of each conversion function:

* **Length Conversion (lengthConversion)**:  
  This function provides users with eight length-related conversions. For example, "Meters to Kilometers" or "Miles to Meters." Once the user selects a conversion option, the function prompts for the required input value and calculates the result. The results are printed to the console and recorded in a text file (File12.txt) for historical reference.
* **Weight Conversion (weightConversion)**:  
  Similar to the length conversion function, the weight conversion function offers eight conversions, including "Grams to Kilograms" and "Pounds to Ounces." The function performs the selected conversion and outputs the result both on the console and in the history file.
* **Temperature Conversion (temperatureConversion)**:  
  The temperature conversion function allows users to switch between Celsius, Fahrenheit, and Kelvin units. It includes six conversion options, such as "Celsius to Fahrenheit" and "Kelvin to Celsius." The results are shown on the console and stored in File12.txt.
* **Volume Conversion (volumeConversion)**:  
  This function provides six options for converting between liters, gallons, milliliters, and cubic meters. For example, users can convert "Liters to Gallons" or "Cubic Meters to Liters." Like the other functions, this conversion’s results are displayed on the console and appended to the history file.

Each conversion function uses printf to display conversion instructions and results to the user. These functions rely on conditional statements to navigate between options and are built with basic mathematical operations for unit conversions. Furthermore, all results are recorded in File12.txt to maintain a persistent history of conversions.

#### 3.3.3. ****History Management****

The history management feature in the system allows users to review all previous conversions or filter conversions by date. This feature is implemented within the history() function, and it works as follows:

* **File-Based Storage**:  
  The system stores each conversion result in a text file named File12.txt. Each session is marked with an opening date and time, which is written to the file upon program initialization. This file-based approach provides a persistent history of conversions across different sessions, making it easy for users to track their conversion history.
* **History Viewing Options**:  
  The history() function offers two options:
  + **View by Date**: The user can enter a specific date (in the format MMM DD, such as "Oct 24") to filter the history records for that particular day. The program scans through the file for entries that match the specified date, displaying all conversions made on that day.
  + **View All**: If the user wants to view the entire history, the program reads through the entire file and displays each entry on the console.
* **Efficient Data Retrieval**:  
  The history function uses fgets to read the file line by line, which ensures efficient retrieval and filtering of data. When a specific date is searched, the function scans for the date within each record to show only the relevant entries.

#### 3.3.4. ****Clear History****

To allow users to manage their conversion history, the system includes a function (clearHistory) that erases all data in File12.txt. Selecting the "Clear History" option opens the file in write mode, which automatically deletes its contents. This feature provides users with privacy and a fresh start when needed.

#### 3.3.5. ****File Handling for Data Persistence****

The system uses file handling to log and manage each session’s activities. The main file used for this purpose is File12.txt. Each time the program starts, it records the session’s start time and date in this file, followed by any conversion results throughout the session. The file-handling mechanism includes the following details:

* **Writing to File**:  
  The program opens File12.txt in append mode ("a"), allowing it to add new entries without overwriting the existing ones. Each conversion function appends its results to this file, ensuring that all conversion history is retained across sessions.
* **Reading from File**:  
  The program reads the file in both the history function (to display records) and the clear history function (to check if the file is non-empty before clearing). This file structure ensures easy readability, as each session is organized chronologically with date and time stamps.
* **Session Marking**:  
  The ctime() function is used to log each session's start time and date in a human-readable format. This information helps users understand when conversions were made, especially when using the history search function.

**3.4 Conclusion:**

This conversion calculator system provides users with a reliable and efficient way to perform various unit conversions, view historical conversion records, and clear history when needed. Designed as a console-based application using the C programming language, the system is structured for ease of use with a simple menu-driven interface that makes navigation intuitive. The modular approach with separate functions for each conversion type promotes code clarity and future maintainability. Additionally, the use of file-based storage ensures that the user's conversion history is preserved across sessions.

Through the integration of multiple conversion functions, error handling, and file-based persistence, the application meets essential requirements for a unit conversion tool. This system demonstrates how fundamental programming techniques in C—such as file handling, conditional statements, and modular design—can be applied to create a user-friendly and versatile utility program. Future improvements could include expanding the range of conversions, introducing a graphical interface, and integrating databases for enhanced data management.

**Chapter 4**

**Implementation**

* 1. **Introduction:**

This chapter provides a comprehensive overview of the implementation process of the "Conversion Calculator" program. The program was designed to provide users with a convenient tool for converting between various units in categories such as length, weight, temperature, and volume. Each conversion module within the calculator is implemented to facilitate simple and efficient calculations for the user. Additionally, to enhance user experience and interaction, the program includes functionalities for recording, viewing, and clearing the conversion history, which are stored in a text file for easy access.

The program is structured using modular functions that handle distinct types of conversions, a history feature for session tracking, and an option to clear previous data. The implementation begins with a main menu interface that allows the user to navigate through the various conversion categories and options. Each conversion module is implemented as a separate function for clarity and reusability, with user inputs processed and results displayed, both on-screen and recorded in a text file. The history feature provides a means for users to view past conversions by date or display the entire session history.

**4.2. lengthConversion:**

**4.2.1. Function Declaration**

void lengthConversion() {

This line declares a function named lenghtConversion that takes no arguments and returns no value. It is designed to perform various conversions between units of length.

**4.2.2. Variable Declarations:**

float meters, kilometers, feet, inches, miles, centimeters;

int choice;

FILE \*file;

meters, kilometers, feet, inches, miles, centimeters are floating-point variables that will store the input values and results for different unit conversions.

choic is an integer variable used to store the user's selection from a menu.

file is a file pointer of type FILE used to manage the file in which conversion results will be saved.

File Opening

file = fopen("File12.txt", "a");

 fope opens a file named "File12.txt" in append mode ("a"), which means data written to this file will be added at the end without overwriting existing content.

 If File12.txt" doesn’t exist, it will be created.

Menu Display

**4.2.3. Menu Display**

printf("\n.... Length Conversion ....\n");

printf("1. Meters to Kilometers\n");

printf("2. Kilometers to Meters\n");

printf("3. Feet to Inches\n");

printf("4. Inches to Feet\n");

printf("5. Meters to Miles\n");

printf("6. Miles to Meters\n");

printf("7. Centimeters to Meters\n");

printf("8. Meters to Centimeters\n");

printf("Select a option: ");

This block displays the available conversion options to the user in a simple menu format. Each option corresponds to a different unit conversion.

**4.2.4.** **User Input**

scanf("%d", &choice);

This line captures the user's input and stores it in the choice variable.

**4.2.5. Switch Statement**

switch (choice) {

The switch statement evaluates choice and performs the appropriate conversion based on the selected option.

**4.2.6. Case Statements for Each Conversion**

For each case, the function prompts the user for an input value, performs the conversion, displays the result, and writes it to the file. Let’s go through each case:

**Case 1: Meters to Kilometers**

case 1:

printf("Enter meters: ");

scanf("%f", &meters);

printf("%.2f meters = %.2f kilometers\n", meters, meters / 1000);

fprintf(file,"%.2f meters = %.2f kilometers\n", meters, meters / 1000);

break;

* Prompts the user to enter a value in meters.
* Converts meters to kilometers by dividing by 1000.
* Displays and writes the result to the file, formatted to two decimal places.

**Case 2: Kilometers to Meters**

case 2:

printf("Enter kilometers: ");

scanf("%f", &kilometers);

printf("%.2f kilometers = %.2f meters\n", kilometers, kilometers \* 1000);

fprintf(file,"%.2f kilometers = %.2f meters\n", kilometers, kilometers \* 1000);

break;

* Prompts the user to enter kilometers.
* Converts kilometers to meters by multiplying by 1000.
* Displays and saves the result.

**Case 3: Feet to Inches**

case 3:

printf("Enter feet: ");

scanf("%f", &feet);

printf("%.2f feet = %.2f inches\n", feet, feet \* 12);

fprintf(file,"%.2f feet = %.2f inches\n", feet, feet \* 12);

break;

* Prompts for a value in feet.
* Converts feet to inches by multiplying by 12.
* Displays and writes the result.

**Case 4: Inches to Feet**

case 4:

printf("Enter inches: ");

scanf("%f", &inches);

printf("%.2f inches = %.2f feet\n", inches, inches / 12);

fprintf(file,"%.2f inches = %.2f feet\n", inches, inches / 12);

break;

* Prompts for inches.
* Converts inches to feet by dividing by 12.
* Outputs and saves the result.

**Case 5: Meters to Miles**

case 5:

printf("Enter meters: ");

scanf("%f", &meters);

printf("%.2f meters = %.2f miles\n", meters, meters / 1609.34);

fprintf(file,"%.2f meters = %.2f miles\n", meters, meters / 1609.34);

break;

* Asks for meters.
* Converts meters to miles by dividing by 1609.34.
* Shows and writes the result.

**Case 6: Miles to Meters**

case 6:

printf("Enter miles: ");

scanf("%f", &miles);

printf("%.2f miles = %.2f meters\n", miles, miles \* 1609.34);

fprintf(file,"%.2f miles = %.2f meters\n", miles, miles \* 1609.34);

break;

* Requests miles input.
* Converts miles to meters by multiplying by 1609.34.
* Displays and logs the result.

**Case 7: Centimeters to Meters**

case 7:

printf("Enter centimeters: ");

scanf("%f", &centimeters);

printf("%.2f centimeters = %.2f meters\n", centimeters, centimeters / 100);

fprintf(file,"%.2f centimeters = %.2f meters\n", centimeters, centimeters / 100);

break;

* Prompts for centimeters.
* Converts centimeters to meters by dividing by 100.
* Displays and records the result.

**Case 8: Meters to Centimeters**

case 8:

printf("Enter meters: ");

scanf("%f", &meters);

printf("%.2f meters = %.2f centimeters\n", meters, meters \* 100);

fprintf(file,"%.2f meters = %.2f centimeters\n", meters, meters \* 100);

break;

* Prompts for meters.
* Converts meters to centimeters by multiplying by 100.
* Shows and saves the result.

**4.2.7. File Closure**

fclose(file);

}

* Closes the file after all operations are done, ensuring data is saved and resources are released.

**4.3 WeightConversion:**

**4.3.1. Function Declaration**

void weightConversion() {

This line declares the weightConversion function, which takes no arguments and has a void return type, meaning it does not return any value. This function handles conversions between various weight units.

**4.3.2. Variable Declarations**

FILE \*file;

file = fopen("File12.txt", "a");

float grams, kilograms, pounds, ounces, stones;

int choice;

* file is a pointer of type FILE used for file operations, allowing the function to write conversion results to a file.
* fopen("File12.txt", "a") opens "File12.txt" in append mode ("a"), creating the file if it doesn’t exist. Results will be added at the end of this file.
* The float variables (grams, kilograms, pounds, ounces, and stones) store the input values and converted results for different weight units.
* choice is an integer that stores the user’s selection for the type of conversion.

**4.3.3. Menu Display**

printf("\n.... Weight Conversion .....\n");

printf("1. Grams to Kilograms\n");

printf("2. Kilograms to Grams\n");

printf("3. Pounds to Ounces\n");

printf("4. Ounces to Pounds\n");

printf("5. Kilograms to Pounds\n");

printf("6. Pounds to Kilograms\n");

printf("7. Stones to Pounds\n");

printf("8. Pounds to Stones\n");

printf("Select a option: ");

This block displays a menu with options for different weight conversions.

**4.3.4. User Input**

scanf("%d", &choice);

The scanf function reads the user's input and stores it in choice.

**4.3.5. Switch Statement**

switch (choice) {

The switch statement uses the value of choice to determine which conversion to perform.

**4.3.6. Case Statements for Each Conversion**

Each case prompts the user for input, performs the conversion, displays the result, and writes it to the file. Let’s look at each case.

**Case 1: Grams to Kilograms**

case 1:

printf("Enter grams: ");

scanf("%f", &grams);

printf("%.2f grams = %.2f kilograms\n", grams, grams / 1000);

fprintf(file, "%.2f grams = %.2f kilograms\n", grams, grams / 1000);

break;

* Prompts the user to enter a value in grams.
* Converts grams to kilograms by dividing by 1000.
* Prints the result on the screen and writes it to the file.

**Case 2: Kilograms to Grams**

case 2:

printf("Enter kilograms: ");

scanf("%f", &kilograms);

printf("%.2f kilograms = %.2f grams\n", kilograms, kilograms \* 1000);

fprintf(file, "%.2f kilograms = %.2f grams\n", kilograms, kilograms \* 1000);

break;

* Prompts for a value in kilograms.
* Converts kilograms to grams by multiplying by 1000.
* Displays and records the result.

**Case 3: Pounds to Ounces**

case 3:

printf("Enter pounds: ");

scanf("%f", &pounds);

printf("%.2f pounds = %.2f ounces\n", pounds, pounds \* 16);

fprintf(file, "%.2f pounds = %.2f ounces\n", pounds, pounds \* 16);

break;

* Asks the user to enter pounds.
* Converts pounds to ounces by multiplying by 16.
* Shows and writes the result.

**Case 4: Ounces to Pounds**

case 4:

printf("Enter ounces: ");

scanf("%f", &ounces);

printf("%.2f ounces = %.2f pounds\n", ounces, ounces / 16);

fprintf(file, "%.2f ounces = %.2f pounds\n", ounces, ounces / 16);

break;

* Prompts for ounces input.
* Converts ounces to pounds by dividing by 16.
* Displays and saves the result.

**Case 5: Kilograms to Pounds**

case 5:

printf("Enter kilograms: ");

scanf("%f", &kilograms);

printf("%.2f kilograms = %.2f pounds\n", kilograms, kilograms \* 2.20462);

fprintf(file, "%.2f kilograms = %.2f pounds\n", kilograms, kilograms \* 2.20462);

break;

* Prompts the user to enter kilograms.
* Converts kilograms to pounds by multiplying by 2.20462.
* Displays and writes the result.

**Case 6: Pounds to Kilograms**

case 6:

printf("Enter pounds: ");

scanf("%f", &pounds);

printf("%.2f pounds = %.2f kilograms\n", pounds, pounds / 2.20462);

fprintf(file, "%.2f pounds = %.2f kilograms\n", pounds, pounds / 2.20462);

break;

* Asks the user for pounds.
* Converts pounds to kilograms by dividing by 2.20462.
* Shows and saves the result.

**Case 7: Stones to Pounds**

case 7:

printf("Enter stones: ");

scanf("%f", &stones);

printf("%.2f stones = %.2f pounds\n", stones, stones \* 14);

fprintf(file, "%.2f stones = %.2f pounds\n", stones, stones \* 14);

break;

* Prompts the user to enter stones.
* Converts stones to pounds by multiplying by 14.
* Displays and logs the result.

**Case 8: Pounds to Stones**

case 8:

printf("Enter pounds: ");

scanf("%f", &pounds);

printf("%.2f pounds = %.2f stones\n", pounds, pounds / 14);

fprintf(file, "%.2f pounds = %.2f stones\n", pounds, pounds / 14);

break;

* Asks for pounds input.
* Converts pounds to stones by dividing by 14.
* Outputs and saves the result.

**4.3.7. File Closure**

fclose(file);

}

This line closes the file after the conversion operation, ensuring data is saved, and resources are released.

This function provides a simple user interface for converting between different weight units, with results saved in a file for future reference.

**4.4. TemperatureConversion:**

**4.4.1. Function Declaration**

void temperatureConversion() {

The function temperatureConversion is declared with no parameters and a void return type, meaning it does not return any value.

**4.4.2. Variable Declarations**

float celsius, fahrenheit, kelvin;

int choice;

FILE \*file;

file = fopen("File12.txt", "a");

* celsius, fahrenheit, and kelvin are float variables to store user input and conversion results for temperature in Celsius, Fahrenheit, and Kelvin respectively.
* choice is an integer to store the user’s selection for the conversion type.
* file is a pointer of type FILE, which is used to open "File12.txt" in append mode ("a"), so conversion results can be saved to the file.

**4.4.3. Menu Display**

printf("\n.... Temperature Conversion ....\n");

printf("1. Celsius to Fahrenheit\n");

printf("2. Fahrenheit to Celsius\n");

printf("3. Celsius to Kelvin\n");

printf("4. Kelvin to Celsius\n");

printf("5. Fahrenheit to Kelvin\n");

printf("6. Kelvin to Fahrenheit\n");

printf("Select a option: ");

This section displays a menu with temperature conversion options.

**4.4.4. User Input**

scanf("%d", &choice);

This line captures the user’s selection and stores it in the choice variable.

**4.4.5. Switch Statement**

switch (choice) {

The switch statement checks the value of choice and performs the corresponding conversion based on the selected option.

**4.4.6. Case Statements for each Conversion**

Each case statement performs one specific conversion, displaying the result on the screen and writing it to the file.

**Case 1: Celsius to Fahrenheit**

case 1:

printf("Enter Celsius: ");

scanf("%f", &celsius);

printf("%.2f Celsius = %.2f Fahrenheit\n", celsius, (celsius \* 9 / 5) + 32);

fprintf(file, "%.2f Celsius = %.2f Fahrenheit\n", celsius, (celsius \* 9 / 5) + 32);

break;

* Prompts the user to enter a value in Celsius.
* Converts Celsius to Fahrenheit using the formula Fahrenheit = (Celsius \* 9 / 5) + 32.
* Displays and writes the result to the file.

**Case 2: Fahrenheit to Celsius**

case 2:

printf("Enter Fahrenheit: ");

scanf("%f", &fahrenheit);

printf("%.2f Fahrenheit = %.2f Celsius\n", fahrenheit, (fahrenheit - 32) \* 5 / 9);

fprintf(file, "%.2f Fahrenheit = %.2f Celsius\n", fahrenheit, (fahrenheit - 32) \* 5 / 9);

break;

* Prompts the user to enter a Fahrenheit value.
* Converts Fahrenheit to Celsius using the formula Celsius = (Fahrenheit - 32) \* 5 / 9.
* Shows and logs the result.

**Case 3: Celsius to Kelvin**

case 3:

printf("Enter Celsius: ");

scanf("%f", &celsius);

printf("%.2f Celsius = %.2f Kelvin\n", celsius, celsius + 273.15);

fprintf(file, "%.2f Celsius = %.2f Kelvin\n", celsius, celsius + 273.15);

break;

* Prompts for Celsius input.
* Converts Celsius to Kelvin by adding 273.15.
* Displays and writes the result.

**Case 4: Kelvin to Celsius**

case 4:

printf("Enter Kelvin: ");

scanf("%f", &kelvin);

printf("%.2f Kelvin = %.2f Celsius\n", kelvin, kelvin - 273.15);

fprintf(file, "%.2f Kelvin = %.2f Celsius\n", kelvin, kelvin - 273.15);

break;

* Asks the user to input a value in Kelvin.
* Converts Kelvin to Celsius by subtracting 273.15.
* Shows and logs the result.

**Case 5: Fahrenheit to Kelvin**

case 5:

printf("Enter Fahrenheit: ");

scanf("%f", &fahrenheit);

printf("%.2f Fahrenheit = %.2f Kelvin\n", fahrenheit, (fahrenheit - 32) \* 5 / 9 + 273.15);

fprintf(file, "%.2f Fahrenheit = %.2f Kelvin\n", fahrenheit, (fahrenheit - 32) \* 5 / 9 + 273.15);

break;

* Prompts for Fahrenheit input.
* Converts Fahrenheit to Kelvin using the formula Kelvin = (Fahrenheit - 32) \* 5 / 9 + 273.15.
* Outputs and saves the result.

**Case 6: Kelvin to Fahrenheit**

case 6:

printf("Enter Kelvin: ");

scanf("%f", &kelvin);

printf("%.2f Kelvin = %.2f Fahrenheit\n", kelvin, (kelvin - 273.15) \* 9 / 5 + 32);

fprintf(file, "%.2f Kelvin = %.2f Fahrenheit\n", kelvin, (kelvin - 273.15) \* 9 / 5 + 32);

break;

* Asks the user for a Kelvin value.
* Converts Kelvin to Fahrenheit using Fahrenheit = (Kelvin - 273.15) \* 9 / 5 + 32.
* Shows and logs the result.

**4.4.7. File Close**

fclose(file);

}

This line closes the file after the conversion process is complete, ensuring data integrity and freeing system resources.

Overall, this function provides an interactive way to convert between temperature units, saving results to a file for future reference.

**4.5 VolumeConversion:**

**4.5.1. Function Declaration**

void volumeConversion() {

This function, volumeConversion, has no parameters and a void return type, indicating it does not return any values.

**4.5.2. Variable Declarations**

float liters, gallons, milliliters, cubicMeters;

int choice;

FILE \*file;

file = fopen("File12.txt", "a");

* liters, gallons, milliliters, and cubicMeters are float variables used to store values for volume in different units.
* choice is an integer to store the user’s menu selection.
* file is a FILE pointer used to open "File12.txt" in append mode ("a"), which allows saving results to the file without overwriting its previous content.

**4.5.3.Menu Display**

printf("\n.... Volume Conversion ....\n");

printf("1. Liters to Gallons\n");

printf("2. Gallons to Liters\n");

printf("3. Milliliters to Liters\n");

printf("4. Liters to Milliliters\n");

printf("5. Cubic Meters to Liters\n");

printf("6. Liters to Cubic Meters\n");

printf("Select a option: ");

This section displays a menu with options for converting between different volume units, prompting the user to make a selection.

**4.5.4. User's Input**

scanf("%d", &choice);

This reads the user's menu choice and stores it in the choice variable.

**4.5.5. Switch Statement**

switch (choice) {

The switch statement checks the value of choice and performs the corresponding conversion based on the selected option.

**4.5.6. Case Statements for Each Conversion**

Each case performs a specific conversion, displays the result, and writes it to the file.

**Case 1: Liters to Gallons**

case 1:

printf("Enter liters: ");

scanf("%f", &liters);

printf("%.2f liters = %.2f gallons\n", liters, liters / 3.78541);

fprintf(file, "%.2f liters = %.2f gallons\n", liters, liters / 3.78541);

break;

* Prompts the user to enter a volume in liters.
* Converts liters to gallons using gallons = liters / 3.78541.
* Displays and writes the result to the file.

**Case 2: Gallons to Liters**

case 2:

printf("Enter gallons: ");

scanf("%f", &gallons);

printf("%.2f gallons = %.2f liters\n", gallons, gallons \* 3.78541);

fprintf(file, "%.2f gallons = %.2f liters\n", gallons, gallons \* 3.78541);

break;

* Prompts the user to enter a volume in gallons.
* Converts gallons to liters using liters = gallons \* 3.78541.
* Shows and logs the result.

**Case 3: Milliliters to Liters**

case 3:

printf("Enter milliliters: ");

scanf("%f", &milliliters);

printf("%.2f milliliters = %.2f liters\n", milliliters, milliliters / 1000);

fprintf(file, "%.2f milliliters = %.2f liters\n", milliliters, milliliters / 1000);

break;

* Prompts for a volume in milliliters.
* Converts milliliters to liters using liters = milliliters / 1000.
* Displays and logs the result.

**Case 4: Liters to Milliliters**

case 4:

printf("Enter liters: ");

scanf("%f", &liters);

printf("%.2f liters = %.2f milliliters\n", liters, liters \* 1000);

fprintf(file, "%.2f liters = %.2f milliliters\n", liters, liters \* 1000);

break;

* Prompts the user to enter a value in liters.
* Converts liters to milliliters using milliliters = liters \* 1000.
* Displays and writes the result to the file.

**Case 5: Cubic Meters to Liters**

case 5:

printf("Enter cubic meters: ");

scanf("%f", &cubicMeters);

printf("%.2f cubic meters = %.2f liters\n", cubicMeters, cubicMeters \* 1000);

fprintf(file, "%.2f cubic meters = %.2f liters\n", cubicMeters, cubicMeters \* 1000);

break;

* Prompts the user for a volume in cubic meters.
* Converts cubic meters to liters using liters = cubicMeters \* 1000.
* Shows and writes the result to the file.

**Case 6: Liters to Cubic Meters**

case 6:

printf("Enter liters: ");

scanf("%f", &liters);

printf("%.2f liters = %.2f cubic meters\n", liters, liters / 1000);

fprintf(file, "%.2f liters = %.2f cubic meters\n", liters, liters / 1000);

break;

* Prompts the user for a volume in liters.
* Converts liters to cubic meters using cubicMeters = liters / 1000.
* Displays and logs the result.

**4.5.8. File Closure**

fclose(file);

}

After the conversion process, the file is closed to ensure data integrity and free up system resources.

This function provides an interactive way to perform volume conversions and saves the results to a file for later reference.

**4.6. CurrencyConversion():**

The currencyConversion() function is designed to convert a specified foreign currency amount to Bangladeshi Taka (BDT) based on predefined exchange rates. It also logs the conversion results to a file named File12.txt.

* + 1. **Function Declaration:**

void currencyConversion() {

Declares a function currencyConversion with no parameters and a return type of void, meaning it doesn’t return any value.

* + 1. **Variable Declarations:**

float foreignAmount, result;

int choice;

FILE \*file;

* + 1. **Declares three variables:**

foreignAmount: to store the input amount in the foreign currency.

result: to store the calculated amount in BDT.

choice: to store the user’s currency selection.

file: a pointer of type FILE to handle file operations.

* + 1. **File Opening:**

file = fopen("File12.txt", "a");

Opens the file File12.txt in append mode ("a"), allowing data to be added to the end of the file. If the file doesn’t exist, it will be created.

* + 1. **Currency Conversion :**

printf("\n.... Currency Conversion ....\n");

Prints a title for the currency conversion menu.

printf("1. USD (US Dollar) to BDT\n");

printf("2. INR (Indian Rupee) to BDT\n");

printf("3. PKR (Pakistani Rupee) to BDT\n");

printf("4. NPR (Nepalese Rupee) to BDT\n");

printf("5. LKR (Sri Lankan Rupee) to BDT\n");

printf("6. MVR (Maldivian Rufiyaa) to BDT\n");

printf("7. BTN (Bhutanese Ngultrum) to BDT\n");

printf("8. AFN (Afghan Afghani) to BDT\n");

printf("9. JPY (Japanese Yen) to BDT\n");

printf("10. KRW (South Korean Won) to BDT\n");

Displays options (1 to 10) for different currencies that the user can convert to BDT.

printf("Select a currency option: ");

scanf("%d", &choice);

Prompts the user to select a currency by entering a number between 1 and 10, and stores the input in choice.

printf("Enter the amount: ");

scanf("%f", &foreignAmount);

Prompts the user to enter the amount in the selected foreign currency and stores it in foreignAmount.

switch (choice) {

Begins a switch statement that checks the value of choice to determine which currency conversion case to execute.

conversion logic, which is explained here for case 1 (USD to BDT) and is similar for other cases.

case 1:

result = foreignAmount \* 109.5; // Approximate conversion rate for USD to BDT

printf("%.2f USD = %.2f BDT\n", foreignAmount, result);

fprintf(file, "%.2f USD = %.2f BDT\n", foreignAmount, result);

break;

**result = foreignAmount \* 109.5;:** Calculates the equivalent amount in BDT by multiplying foreignAmount by the conversion rate for USD to BDT.

**printf("%.2f USD = %.2f BDT\n", foreignAmount, result);**: Displays the conversion result to the user.

**fprintf(file, "%.2f USD = %.2f BDT\n", foreignAmount, result);**: Writes the conversion result to File12.txt.

**break;** Ends the current case to prevent fall-through to the next case.

**4.7 History:**

**4.7.1. Function Declaration**

void history() {

The function history has no parameters and returns nothing (void). It displays and manages history records from a file.

**4.7.2. Variable Declaration and Initial Display**

int tt;

printf("\n.... History ....\n");

printf("1. Search By Date.\n");

printf("2. Show All.\n");

printf("Select a option: ");

scanf("%d", &tt);

* tt is an integer variable to store the user's choice.
* The function displays options for the user to either search by a specific date or to show all history records.
* scanf captures the user’s choice, storing it in tt.

**4.7.3. Option 1: Search By Date**

if (tt == 1) {

FILE \*file;

char line[256], dateInput[8];

int found = 0;

printf("Enter the date (Like as, Oct 24): ");

scanf(" %[^\n]", dateInput);

* If the user chooses option 1, a FILE pointer file is created to open and read the file.
* line is a character array for reading lines from the file.
* dateInput stores the date entered by the user (formatted as "Oct 24").
* found is an integer flag to check if any entries matching the date are found.

**4.7.4. Opening and Reading the File**

file = fopen("File12.txt", "r");

if (file == NULL) {

printf("Error opening file!\n");

return;

}

printf("History for %s:\n", dateInput);

* The file "File12.txt" is opened in read mode ("r").
* If the file cannot be opened, an error message is displayed, and the function exits.
* A message displays the search criteria using the input date.

**4.7.5. Searching for the Date**

while (fgets(line, sizeof(line), file)) {

if (strstr(line, "Opening Date and Time:") != NULL) {

fgets(line, sizeof(line), file);

if (strstr(line, dateInput) != NULL) {

found = 1;

printf("%s", line);

while (fgets(line, sizeof(line), file) && strstr(line, "Opening Date and Time:") == NULL) {

printf("%s", line);

}

printf("\n");

}

}

}

* This loop reads each line of the file to search for entries with the specified date.
* strstr(line, "Opening Date and Time:") != NULL checks for a line indicating a new entry with a timestamp.
* If the line contains "Opening Date and Time:", the next line is read to look for dateInput in the timestamp.
* When the date is found:
  + found is set to 1, indicating that a matching entry exists.
  + The matching date line and subsequent lines are printed until the next "Opening Date and Time:" line, which signals a new entry.

**4.7.6. No Matching History Found**

if (!found) {

printf("No history found for the date %s.\n", dateInput);

}

fclose(file);

* If no matching entries are found (found is still 0), a message informs the user.
* The file is closed after reading.

**4.7.7. Option 2: Show All History**

else if (tt == 2) {

FILE \*file;

file = fopen("File12.txt", "r");

while (!feof(file)) {

char ch = fgetc(file);

printf("%c", ch);

}

fclose(file);

}

* If the user chooses 2, the entire content of "File12.txt" is displayed.
* A while loop reads each character (fgetc(file)) until the end of the file (feof(file)).
* Each character is printed, and the file is closed afterward.

**4.8. Clear History:**

The clearHistory function is a straightforward way to delete all records stored in the File12.txt file. Here's a line-by-line explanation:

**4.8.1. Function Declaration**

void clearHistory() {

The function clearHistory takes no parameters and has a void return type, meaning it performs an action without returning any value.

**4.8.2. File Opening in Write Mode**

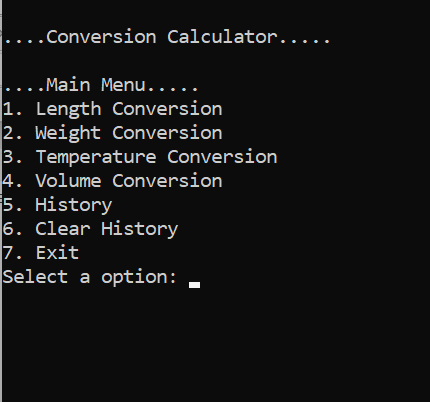
FILE \*file;

file = fopen("File12.txt", "w");

* A FILE pointer file is declared to handle file operations.
* fopen("File12.txt", "w") opens File12.txt in **write mode** ("w").
  + Opening a file in write mode ("w") **erases all existing content** in the file. This is how the history is "cleared."

**4.9. Code Execution and Results**

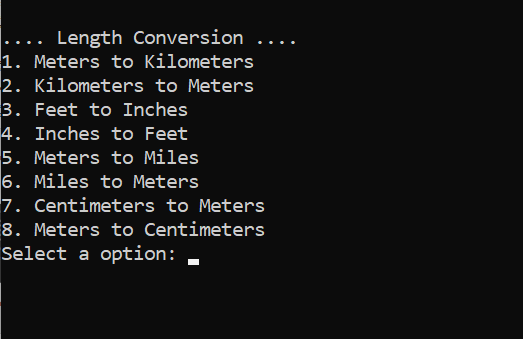
First of all, if we run our code we will find a interface like below:



User can select any Option.

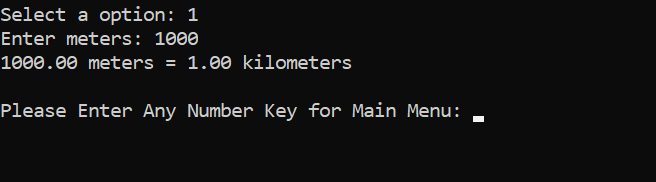
Let user want to convert length and he select 1

Then he find a Menu like this.



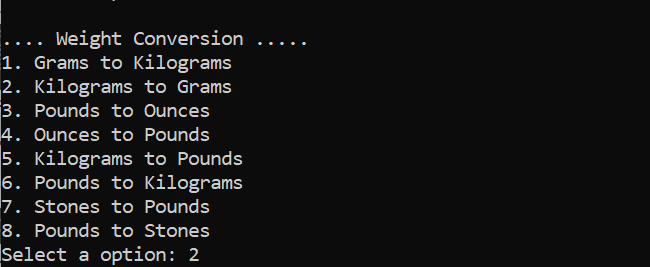
Let user want to convert Meters to Kilometers.Then he select 1.

Then he can gives input and he can show converted output.



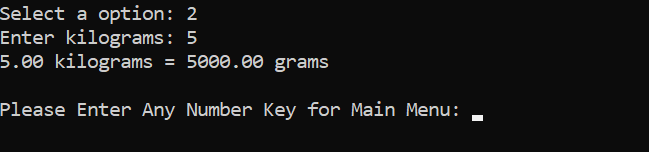
Then he have to press any number for going Main menu.

Now let user want to use weight conversion. Then he select 2 and he find menu like this .

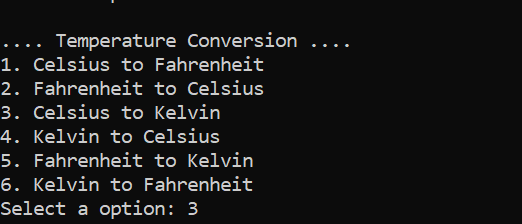


Let he want to convert Kilograms to Grams. Then he will select 2.

Then he will give input as kilograms and find output in grams.

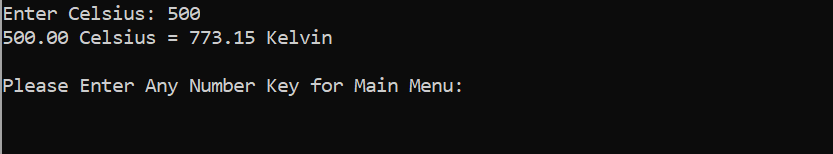


If user select option 3 he will find Temperature Conversion. He can select any kinds of temperature conversion.



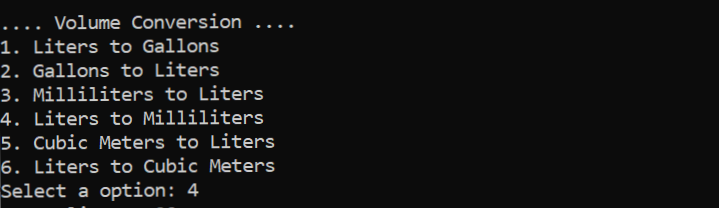
Let user select Celsius to kelvin . And select option 3.

Then he will find option to give his input in celcius and he will find his output in kelvin.



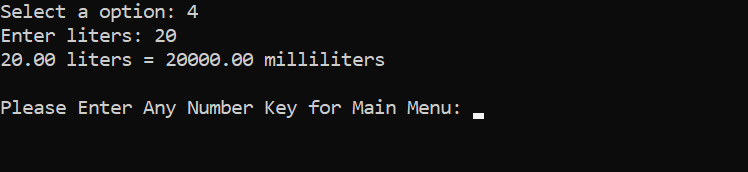
Then he user enter any number for going main menu.

If user select option 4 ,then he will go on volume conversion.



Let user select option 4 that convert Liters to Mililiters.

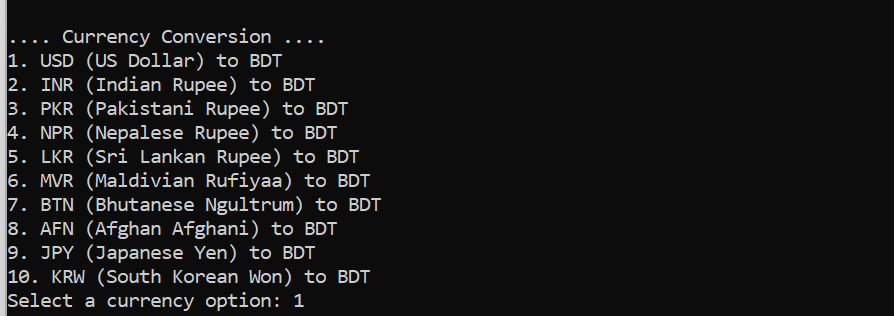
Then he will find option to give his inpute as liters and he will find output in mililiters.



Then user press any number to go main menu.

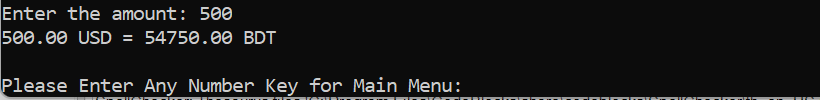
If user want to convert Currency he need to select option 5.

Then he will find a interface like this



Let he want convert USD to BDT, and select option 1,

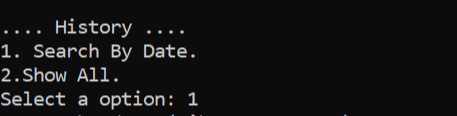
Then he can give his amount,



User can select 6 for history. Then will find two types of history:

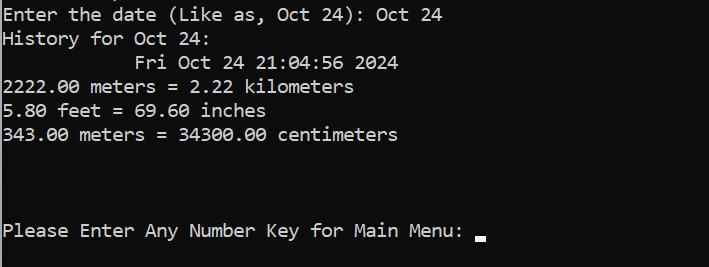
1. Search by date
2. Show all.

In search by date, user input a specific date and he will find history for the date. And in show all option he will able to all history

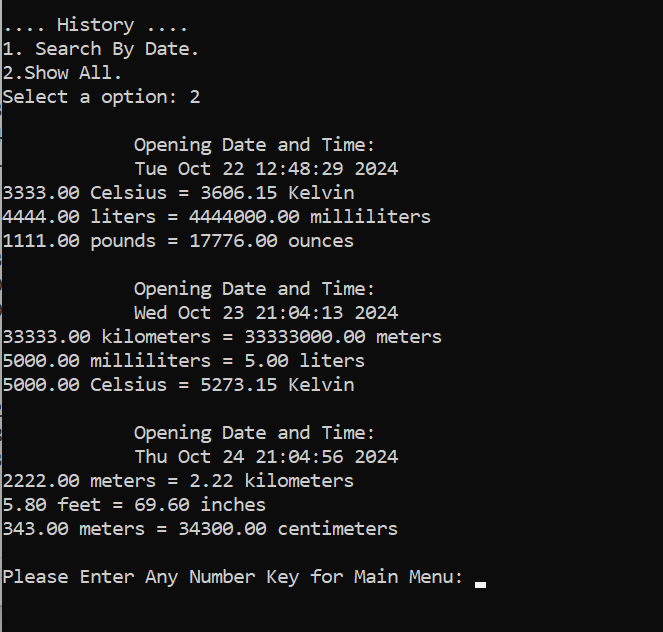


Let user select option 1.

Here user is asked for a search , after input a date by user ,it will print history for this specific date with day and time .



By chose 2, user will find history of all date with day and time.



If the user select option 7, then all the history will delete . And print a message like: history is cleared.

And if user want to exiting the calculator, he should select 8.

**Chapter 5**

**Limitation and Future Scope**

**5.1 Introduction:**

In software development, it is crucial to critically assess both the limitations of a project and the potential for future improvements. This chapter will delve into the specific limitations of the Conversion Calculator project, highlighting areas that need enhancement or re-evaluation. Additionally, we will explore various avenues for future development, ensuring that the calculator evolves to meet user needs and technological advancements. By understanding the current constraints and envisioning future possibilities, we aim to create a more robust and versatile tool for unit conversions.

**5.2 Limitations of the Current Project:**

While the Conversion Calculator serves its primary function effectively, several limitations hinder its overall performance and user experience:

**5.2.1. Limited Conversion Types**:

The calculator currently supports a predefined set of conversion categories, including length, weight, temperature, and volume. However, many users may require additional conversion types, such as speed (e.g., miles per hour to kilometers per hour), area (e.g., square feet to square meters), and currency conversions (e.g., USD to EUR). The absence of these conversion types limits the tool's utility, particularly for users needing more comprehensive conversion capabilities.

* + 1. **User Interface Constraints**:

The calculator operates in a text-based console environment, which may be intimidating or confusing for some users, especially those accustomed to graphical user interfaces (GUIs). A more visually engaging interface could enhance user interaction and satisfaction, making the calculator more approachable for a broader audience. The current design does not leverage modern UI/UX principles, which could improve the overall user experience.

1. **Error Handling**:

The existing implementation lacks robust error handling mechanisms. When users input invalid data or unexpected values, the program does not adequately address these scenarios, which may lead to crashes or incorrect results. Implementing comprehensive input validation and error messages would significantly improve the program's reliability, helping users understand what went wrong and guiding them to rectify their mistakes.

1. **Platform Limitations**:

Currently, the calculator is designed to run exclusively in a console environment, which may not be suitable for all users. Many individuals prefer to access applications via mobile devices or web browsers. By limiting the tool to a console-based application, we restrict its potential user base and accessibility. Future development should consider adapting the calculator for web and mobile platforms to enhance its reach and usability.

1. **Lack of Multilingual Support**:

The application is only available in English, which may exclude non-English speakers from fully benefiting from the tool. In a globalized world, providing multilingual support would not only broaden the user base but also improve accessibility and user engagement. Implementing language localization could help cater to a diverse audience.

**5.3 Future Scope of the Project:**

The Conversion Calculator has significant potential for future enhancements and expansions. Here are several directions in which the project can evolve:

**5.3.1. Enhanced Conversion Categories**:

Expanding the range of conversion types is essential to meet user needs. Future updates could include conversions for time (e.g., hours to minutes), speed (e.g., kilometers per hour to meters per second), area (e.g., acres to hectares), and currency (e.g., exchange rates between different currencies). By offering a more comprehensive list of conversions, the calculator can serve a wider audience and cater to various use cases.

* + 1. **Multilingual Support**:

Incorporating multilingual support is vital for broadening the application's user base. By localizing the user interface and instructions into multiple languages, the calculator can cater to non-English speakers and enhance accessibility. This could involve collaborating with translators or using localization frameworks to manage language variations effectively.

**5.4 Conclusion**

In conclusion, while the Conversion Calculator project successfully addresses the basic needs for unit conversions, it faces several limitations that hinder its overall performance and user experience. By acknowledging these constraints, we can lay the groundwork for future improvements and enhancements. The proposed future developments aim to expand the calculator's capabilities, enhance usability, and make the tool accessible to a broader audience. By focusing on these areas, we can transform the Conversion Calculator into a more robust, versatile, and user-friendly application, ultimately improving its value and effectiveness in meeting user needs. The insights gained from this analysis will guide future development efforts, ensuring that the application remains relevant and useful in an ever-evolving technological landscape.

**Chapter 6**

**Conclusion**

**6.1 Summary of the Project:**

The Conversion Calculator project provides a user-friendly tool to perform various types of conversions, including length, weight, temperature, and volume. It is designed to support users with a streamlined interface, enabling quick and accurate calculations across multiple units. The program is implemented in C, leveraging fundamental programming concepts, and offers interactive options that cater to both novice and experienced users.

One of the key highlights of the project is the inclusion of a history feature, allowing users to view and retrieve previous conversion entries. This feature provides two modes: searching history by a specific date or displaying all recorded entries. For easier management, the clear history functionality allows users to erase stored records as needed. Each conversion entry is stored with a timestamp, giving users a chronological record of their activities.

Additionally, this project emphasizes modular programming with separate functions for each conversion type, improving code readability and maintainability. Overall, the Conversion Calculator is a practical and educational tool that showcases fundamental C programming skills, file handling for persistent data storage, and a structured user experience.

**6.2 Lessons Learned:**

Developing the Conversion Calculator project provided several key learning experiences, both technical and practical. Through this project, I deepened my understanding of C programming fundamentals, especially in areas such as modular function design, file handling, and user input validation. Breaking down the code into separate functions for each type of conversion taught me the importance of modularity and reusability, which made the code more organized and easier to troubleshoot or expand with new features.

Working with file handling was particularly insightful, as it allowed me to implement a persistent history feature. This required me to learn how to open, write to, read from, and close files in C, as well as how to structure data for easy retrieval and display. Additionally, implementing a history search by date enhanced my familiarity with string manipulation functions, such as strstr, and the careful handling of user inputs to avoid errors.

Building the user interface taught me the importance of clear prompts and consistent navigation, particularly for an interactive tool where user experience is key. I also learned the importance of robust input validation to ensure smooth operation and prevent incorrect entries from disrupting program flow. This experience reinforced my ability to design with the end-user in mind, ensuring that each feature added value and was intuitive to use.

Overall, this project has been a practical learning experience that has equipped me with essential programming and problem-solving skills, alongside a better appreciation for user-centric design in software development.

**6.3 Final Remarks on the Conversion Calculator:**

The Conversion Calculator project demonstrates the power of combining practical utility with ease of use, resulting in a tool that simplifies everyday tasks involving unit conversions. Throughout its development, this project highlighted the importance of accuracy, user-centered design, and data management. Each module—whether for length, weight, temperature, or volume—was structured to be accessible and efficient, supporting users in quickly obtaining reliable results.

The addition of history and clearing functions provides users with a more personalized experience, enabling them to retrieve or reset previous conversions as needed. This feature reflects the project’s commitment to not only meet basic conversion needs but also to offer a way to manage and track information, adding value for users who may rely on repeated or historical data for their work or studies.

In sum, the Conversion Calculator offers a solid foundation that could be expanded further. Its modular structure makes it adaptable for adding more units or even advanced functionalities. As it stands, it fulfills its core purpose effectively, serving as a dependable, user-friendly tool for a variety of everyday and professional applications. This project underscores the potential for even simple tools to have a meaningful impact in helping users achieve precision and efficiency in their tasks.

**6.4 Conclusion:**

The Conversion Calculator project has successfully established itself as a practical tool designed to address the widespread need for unit conversions across various measurement categories. By providing a user-friendly interface and diverse conversion options, the project effectively simplifies the process of converting units, which is essential in today’s globalized world where multiple measurement systems are in use.

In summary, the Conversion Calculator project has not only fulfilled its initial objectives but also highlighted the critical role of effective programming in solving real-world problems. By facilitating accurate and efficient unit conversions, this project contributes to clearer communication and understanding in various contexts. Ultimately, the Conversion Calculator stands as a valuable resource that promotes accuracy and efficiency, making it an essential tool for anyone navigating the complexities of different measurement systems.

**References**

1. “Conversion Calculator,” *Calculator.net*, 2019. <https://www.calculator.net/conversion-calculator.html>
2. “Basics of File Handling in C - GeeksforGeeks,” *GeeksforGeeks*, Oct. 27, 2017. <https://www.geeksforgeeks.org/basics-file-handling-c/>
3. Programiz, “C Files I/O: Opening, Reading, Writing and Closing a file,” *Programiz.com*, 2019. <https://www.programiz.com/c-programming/c-file-input-output>
4. “Search string in a file in c,” *Stack Overflow*. <https://stackoverflow.com/questions/29429384/search-string-in-a-file-in-c>
5. “C Tutorial – Searching for Strings in a Text File» CodingUnit Programming Tutorials.” <https://www.codingunit.com/c-tutorial-searching-for-strings-in-a-text-file>
6. “C Program to Print Date and Time,” *codescracker.com*. <https://codescracker.com/c/program/c-program-print-date.htm>
7. “Conversion of Units (Mathematical Conversions) - Tables & Examples,” *BYJUS*. <https://byjus.com/maths/conversion-of-units/>