TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A Final Report On Todo List Application in C++

Submitted By:

Nawed Ahmed	(079bel049)
Neeraj Bari	(079bel050)
Prem Shah	(079bel063)
Sandeep K.Yadav	(079bel078)

Submitted To:
Department of Electronics and Computer Engineering
Pulchowk Campus
Lalitpur, Nepal

Aug 2, 2024

Acknowledgments

We would like to express our sincere gratitude to the **Department of Electronics and Computer Engineering** for providing us with the resources and support necessary for the successful completion of our Todo List application project. The department's facilities and infrastructure were instrumental in enabling us to conduct our research and development work effectively.

We extend our heartfelt thanks to **Bibha Sthapit** Ma'am, our lab instructor, for her continuous guidance and support. Her expertise and insightful feedback played a crucial role in helping us navigate the complexities of the project. Her encouragement and patience were invaluable, and we are deeply appreciative of her contributions.

We also want to acknowledge the efforts and collaboration of our team members, **Nawed Ahmed, Neeraj Bari, Prem Shah, and Sandeep Kumar Yadav**. Each member's dedication, creativity, and teamwork made this project a fulfilling and successful endeavor. Working together, we were able to overcome challenges and achieve our project objectives.

Thank you all for your contributions and support, which made this project a rewarding experience.

Nawed Ahmed	(079BEL049)	____________
Neeraj Bari	(079BEL050)	____________
Prem Shah	(079BEL063)	____________
Sandeep Kumar Yadav	(079BEL078)	____________

DATE: August 2,2024

Abstract

This project report presents the design, development, and implementation of a Todo List application written in C++. The primary objective of this project is to create a simple, efficient, and user-friendly tool for managing tasks. The application allows users to add, update, delete, and filter tasks based on various criteria, such as completion status and priority.

To achieve these objectives, the project utilizes fundamental concepts of object-oriented programming (OOP) in C++. The system is built around two main classes: **Task** and **TodoList**. The **Task** class encapsulates individual task attributes and behaviors, while the **TodoList** class manages the collection of tasks and provides methods for user interaction. Data persistence is implemented using a text file to store task details, ensuring that user data is retained between sessions.

Key features of the application include:

Task prioritization

Due date tracking

Ability to mark tasks as completed

The results of the project demonstrate that the Todo List application meets the initial objectives, providing a functional and reliable tool for task management. While the application successfully implements core functionalities, the report also discusses certain limitations, such as the absence of a graphical user interface (GUI) and advanced features like task categorization. The project concludes with suggestions for future enhancements, including the integration of a GUI and additional features to improve user interaction and task organization.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Background

A Todo List application is a simple yet powerful tool designed to help individuals and teams manage their tasks and responsibilities. The primary function of such an application is to enable users to create, organize, and track their tasks, thereby enhancing productivity and time management. In today's fast-paced world, where managing multiple commitments and deadlines has become a norm, a Todo List application provides a centralized platform to streamline these activities.

The relevance of a Todo List application lies in its ability to assist users in prioritizing tasks based on urgency and importance, setting deadlines, and marking tasks as completed. This not only helps in reducing the cognitive load of remembering tasks but also fosters a sense of accomplishment as tasks are completed. Moreover, it serves as a motivational tool, allowing users to visualize their progress and stay organized.

In the context of software development, a Todo List application provides an excellent opportunity to apply and demonstrate various programming concepts. For this project, we developed the Todo List application using C++, leveraging object-oriented programming (OOP) principles. The application features functionalities such as adding, updating, and deleting tasks, along with filtering tasks by priority and completion status. By implementing this project, we aimed to create a functional, user-friendly tool that could be used by anyone looking to manage their daily tasks efficiently.

In summary, the Todo List application is a relevant and practical tool for task management, offering a structured way to handle both personal and professional tasks. This project not only addresses a common need but also serves as a comprehensive exercise in applying programming knowledge and skills.

1.2. Problem Statement

The problem addressed by this project is the lack of an efficient and user-friendly system for managing daily tasks and responsibilities. Individuals and teams often face challenges in keeping track of multiple tasks, leading to disorganization, missed deadlines, and inefficient prioritization. Traditional methods, such as handwritten notes or mental lists, are inadequate for managing the complexity and volume of tasks in a fast-paced environment. This project aims to develop a Todo List application that offers a centralized platform for:

- Adding, updating, deleting, and prioritizing tasks.
- Tracking task status and due dates.
- Providing a clear and organized overview of responsibilities.

By addressing these issues, the project seeks to enhance productivity and ensure effective task management.

1.3. Objectives

The primary goals of this Todo List application project are:

- **Create an Efficient Task Management System:**
  - Develop a software tool that enables users to efficiently manage their tasks by adding, updating, deleting, and tracking them.
  - Support the organization of tasks in a clear and accessible manner.
- **Prioritize Tasks:**
  - Implement a prioritization feature that allows users to categorize tasks based on their urgency and importance (Low, Medium, High).
  - Help users focus on the most critical tasks first, ensuring efficient use of time and resources.
- **Track Task Status and Due Dates:**
  - Provide functionality to monitor the completion status of tasks and set due dates.
  - Assist users in keeping track of their progress and staying on schedule.
- **User-Friendly Interface:**
  - Design a simple and intuitive command-line interface that ensures ease of use for all users, regardless of their technical expertise.
  - Facilitate quick and easy navigation and interaction with the application.
- **Data Persistence:**

  - Implement a method for data persistence, such as storing tasks in a text file.
  - Ensure that user data is retained between sessions, allowing access to their task list at any time without losing information.

- **Scalability and Flexibility:**
  - Build the application in a modular and scalable manner, allowing for future enhancements and the addition of new features.
  - Consider features such as task categorization or integration with other tools.

1.4. Scope

The scope of this Todo List application project outlines the features that are included and those that are excluded, setting clear boundaries for the project's development and functionality.

- **Task Creation and Management:**
  - Users can add new tasks with details such as descriptions, due dates, and priority levels.
  - They can also update existing tasks, mark them as completed, or delete them.
- **Task Prioritization:**
  - Tasks can be categorized based on their priority levels (Low, Medium, High).
  - This allows users to organize and manage their workload effectively.
- **Status Tracking:**
  - The application enables users to track the completion status of tasks.
  - It distinguishes between completed and pending tasks.
- **Due Date Management:**
  - Users can set and modify due dates for tasks.
  - This helps them meet deadlines and manage time efficiently.
- **Data Persistence:**
  - The application saves task data to a text file.

  - This ensures that user information is retained between sessions.

- **Search and Filter Functionality:**
  - Users can search for tasks by ID.
  - They can also filter tasks based on criteria such as completion status and priority.

1.5. Significance

The Todo List application project holds significant value for both individual and team productivity, addressing common challenges in task management and organization. Its importance lies in several key areas:

- **Enhanced Productivity:** By providing a structured platform for task management, the application helps users efficiently organize their daily activities. Users can prioritize tasks, track progress, and manage deadlines, leading to improved time management and increased productivity.
- **Reduced Cognitive Load:** The application alleviates the cognitive burden of remembering tasks and deadlines. Users can offload these responsibilities to the system, allowing them to focus more on executing tasks rather than remembering them.
- **Improved Organization:** With features like task prioritization and due date tracking, the application helps users maintain a clear overview of their tasks. This organization aids in preventing missed deadlines and ensures that important tasks are addressed in a timely manner.
- **User Empowerment:** The application empowers users by providing them with control over their task management. By allowing customization and easy access to task details, users can tailor the application to fit their specific needs and preferences.
- **Educational Value:** For students and developers, the project serves as an excellent exercise in applying programming concepts such as object-oriented design, file handling, and user interface development. It provides practical experience in building a real-world application and solving common software engineering problems.
- **Foundation for Future Enhancements:** The project establishes a solid foundation for potential future developments, such as integrating a graphical user interface (GUI), adding advanced features like notifications and task categorization, or expanding the application to web and mobile platforms.

2. LITERATURE REVIEW

2.1. Existing Solutions

Various Todo List applications and task management tools are available, each with unique features and limitations:

- **Microsoft To Do** offers integration with Microsoft products and cross-platform access. However, its customization options are limited, and it lacks advanced task analytics.
- **Todoist** excels with its prioritization and clean interface but is hindered by its premium pricing for advanced features and potential complexity for new users.
- **Asana** provides comprehensive project management and collaboration features, making it ideal for teams. Nonetheless, it has a steep learning curve and can be overly complex for individual users.
- **Trello** uses a visual board-based system that is highly flexible but may lack detailed task tracking and complex integrations.
- **Google Keep** offers simplicity and Google integration but lacks advanced task management features and detailed organization options.
- **Remember The Milk** is feature-rich with task management capabilities but has an outdated interface and can be overwhelming.

2.2. Technological Background

- **Data Structures**: Essential for organizing and managing tasks efficiently. Common structures include arrays, vectors, and linked lists, which are used to store and manipulate task data.
- **File Handling in C++**: Involves reading from and writing to files, crucial for data persistence. Techniques include using `ifstream` for reading and `ofstream` for writing task data to text files.
- **Object-Oriented Programming (OOP)**: C++ supports OOP principles such as encapsulation, inheritance, and polymorphism, which are used to design and implement the Todo List application. Classes and objects represent tasks and manage their attributes and behaviors.

2.3. Gap Analysis

Existing Todo List applications often have limitations such as lack of customization, advanced task analytics, or overly complex interfaces. Our project aims to address these gaps by offering a simple, user-friendly command-line interface with essential features for task management, including task prioritization, due date tracking, and data persistence. By focusing on a streamlined, easy-to-use design, we provide a solution that meets basic task management needs while laying the groundwork for future enhancements.

3. METHODOLOGY

The methodology section outlines the systematic approach taken to develop the Todo List application, covering the project lifecycle from planning to implementation and testing.

3.1. Planning and Requirements Gathering

The initial phase involved gathering requirements and setting clear objectives for the project. Key features were identified, such as task creation, management, prioritization, and data persistence. The team conducted research on existing solutions to understand their strengths and weaknesses, which informed the project's scope and feature set.

3.2. System Design and Architecture

A comprehensive design phase was undertaken, focusing on the overall system architecture. Key components included:

- **Class Design:** Using object-oriented principles, the `Task` and `TodoList` classes were designed to encapsulate task data and functionalities.
- **Data Flow:** Data flow diagrams were created to illustrate the interaction between different system components, particularly the flow of task data from user input to storage.

3.3. Development Environment Setup

The development environment was set up using the following tools:

- **Integrated Development Environment (IDE):** Visual Studio Code was used for code writing and debugging.
- **Compiler:** GCC compiler was chosen for compiling the C++ code.
- **Libraries:** Standard Template Library (STL) was utilized for data structures.

3.4. Implementation

The implementation phase involved coding the application, with the following steps:

- **Class Implementations:** The `Task` and `TodoList` classes were implemented with methods for task management, such as `addTask`, `deleteTask`, and `updateTask`.
- **File Handling:** File I/O operations were coded to ensure data persistence. Tasks were saved to and loaded from a text file.
- **Command-Line Interface:** A simple and user-friendly CLI was developed for user interactions, allowing task management through commands.

3.5. Testing and Debugging

Extensive testing was conducted to ensure the application's functionality and reliability. The testing process included:

- **Unit Testing:** Individual functions and classes were tested to verify correctness.
- **Integration Testing:** The interaction between different components was tested to ensure seamless functionality.
- **User Testing:** A few users tested the application to provide feedback on usability and functionality.

3.6. Documentation

Documentation was created for both the technical and user aspects of the application. The technical documentation covered the code structure and implementation details, while the user manual provided a guide on how to use the application.

3.7. Deployment and Future Enhancements

The final phase involved preparing the application for deployment and considering future enhancements. Potential improvements included adding a graphical user interface (GUI), expanding task categorization, and integrating with other platforms.

4. IMPLEMENTATION

Integrated Development Environment (IDE): Microsoft Visual Studio Code was used for code editing and development. These IDEs provide robust support for C++ development, including syntax highlighting, debugging, and project management.

Compiler: GCC (GNU Compiler Collection) was used for compiling the C++ code. GCC is known for its efficiency and compatibility with various platforms.

Libraries: The project primarily relies on the C++ Standard Library, which provides essential features such as input/output operations, file handling, and data structures. No external libraries were used.

5. PROBLEM FACED AND SOLUTION

During the development of the Todo List application, several challenges were encountered. Each of these issues was addressed with targeted solutions to ensure the application's functionality and user experience. Below is a detailed account of the problems faced and the solutions implemented:

- **Data Persistence:**
  Issue: Ensuring that tasks were saved and loaded correctly between sessions was challenging. The initial implementation struggled with file handling, leading to data loss or corruption.
  Solution: Implemented robust file handling using C++'s `ifstream` and `ofstream` for reading and writing task data. Added error-checking mechanisms to handle file operations gracefully and employed a simple text file format for storing tasks to ensure data integrity.
- **User Interface Complexity:**
  Issue: Designing a user-friendly command-line interface (CLI) proved difficult, especially for users unfamiliar with command-line tools. The initial interface was not intuitive, leading to confusion.
  Solution: Simplified the CLI by providing clear instructions and feedback messages.

Implemented a menu-driven approach with concise commands and options to guide users through task management operations effectively.

- **Task Prioritization and Sorting:**
  Issue: The implementation of task prioritization and sorting based on priority and due dates was initially flawed. Tasks were not sorted correctly, and priority levels were not consistently applied.
  Solution: Refined the sorting algorithms to ensure accurate prioritization. Used standard C++ sorting functions and data structures like vectors to manage and sort tasks efficiently. Added validation checks to ensure priority levels were correctly assigned and displayed.

- **Error Handling:**
  Issue: The application encountered issues when handling invalid user inputs, such as incorrect task IDs or invalid commands, leading to crashes or unexpected behavior.
  Solution: Improved error handling by adding input validation and exception handling. Implemented user prompts for correct input formats and provided meaningful error messages to guide users in correcting their mistakes.

- **Task Filtering and Searching:**
  Issue: The initial filtering and searching functionalities were inefficient and slow, especially with a large number of tasks. Filtering options were also limited.
  Solution: Optimized the filtering and searching algorithms to enhance performance. Utilized efficient data structures and algorithms to speed up task searches and filtering. Expanded the filtering options to include various criteria for better task management.

- **Scalability Concerns:**
  Issue: The application was not designed to handle a large number of tasks efficiently, leading to performance issues as the task list grew.
  Solution: Designed the application to be modular and scalable. Improved data structures and algorithms to handle larger datasets efficiently. Ensured that the application could be easily extended with additional features in the future.

- **Lack of Advanced Features:**
  Issue: The initial version lacked advanced features such as task categorization and notifications, which users often find useful in more complex task management systems.
  Solution: Designed the application with a modular architecture to facilitate future enhancements. Identified key features for future development, including task categorization and notifications, and planned their integration into the application.

6. LIMITATION AND FUTURE ENHANCEMENTS

Throughout the development of the Todo List application, several limitations were identified. These limitations provide areas for potential future enhancements, which could improve the application's functionality and user experience. The following points outline the current limitations and suggest possible enhancements:

- **Limited User Interface:**
  Limitation: The application currently features a basic command-line interface (CLI), which may not be intuitive for all users, especially those unfamiliar with text-based interfaces.
  Future Enhancement: Develop a graphical user interface (GUI) to enhance user interaction. A GUI could offer a more visually appealing and user-friendly experience, making it easier for users to manage tasks.
- **Lack of Real-time Notifications:**
  Limitation: The application does not support real-time notifications or reminders for upcoming tasks, which limits its effectiveness in reminding users of deadlines.
  Future Enhancement: Implement a notification system to alert users of upcoming due dates and overdue tasks. This feature could include customizable alerts and integration with external notification services.
- **Basic Task Categorization:**
  Limitation: Task categorization is limited to priority levels without additional organizational structures such as tags or categories.
  Future Enhancement: Introduce task categorization by tags, labels, or projects. This would allow users to group tasks more flexibly and organize them according to specific contexts or themes.
- **Single-User Focus:**
  Limitation: The application is designed for individual use, lacking features for team collaboration or multi-user access.
  Future Enhancement: Develop features that support multi-user environments, including shared task lists, collaborative editing, and user roles. This would extend the application's utility to team-based task management.
- **Limited Data Analysis and Reporting:**
  Limitation: The application provides basic task management without advanced analytics or reporting features.

Future Enhancement: Add analytical tools to track user productivity, task completion trends, and time management efficiency. Implementing visual reports and dashboards could provide valuable insights for users.

- **No Mobile or Web Integration:**
  Limitation: The current implementation does not include mobile or web versions, limiting accessibility.
  Future Enhancement: Expand the application to include mobile and web platforms, enabling users to access their task lists across multiple devices. This would offer greater flexibility and convenience.

- **Scalability Constraints:**
  Limitation: The current design may struggle with scalability as the number of tasks increases, potentially leading to performance issues.
  Future Enhancement: Optimize the application's architecture and algorithms to handle larger datasets more efficiently. Implementing database support or cloud storage solutions could also improve scalability.

7. SOURCE CODE

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <limits>
#include <algorithm>

using namespace std;

class Task {
public:
    int id;
    string description;
    bool completed;
    string dueDate;
    int priority; // 1 = Low, 2 = Medium, 3 = High

    Task(int id, string desc, string date, int prio)
        : id(id), description(desc), completed(false),
        dueDate(date), priority(prio) {}

    void display() const {
        cout << "| " << setw(4) << id << " | "
             << setw(30) << description.substr(0, 30) << " | "
             << setw(10) << dueDate << " | "
             << setw(10) << (completed ? "Yes" : "No") << " | "
             << setw(8) << (priority == 1 ? "Low" : priority == 2 ?
              "Medium" : "High") << " |" << endl;
    }
};

class TodoList {
private:
    vector<Task> tasks;
    int nextID;

    void loadTasks() {
        ifstream fin("todo.txt");
        if (fin) {
            int id, completed, priority;
            string desc, date;
            while (fin >> id) {
                fin.ignore();
                getline(fin, desc);
                fin >> completed >> ws;
                getline(fin, date);
                fin >> priority;
                tasks.emplace_back(id, desc, date, priority);
                tasks.back().completed = completed;
                nextID = max(nextID, id + 1);
            }
        }
    }
```

```
void saveTasks() const {
    ofstream fout("todo.txt");
    for (const auto& task : tasks) {
        fout << task.id << '\n'
             << task.description << '\n'
             << (task.completed ? 1 : 0) << '\n'
             << task.dueDate << '\n'
             << task.priority << '\n';
    }
}

int getValidatedChoice(int min, int max) const {
    int choice;
    while (true) {
        cin >> choice;
        if (!cin.fail() && choice >= min && choice <= max) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return choice;
        }
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid choice. Please enter a number between "
        << min << " and " << max << ": ";
    }
}

string getValidatedDate() const {
    string date;
    while (true) {
        cin >> date;
        if (isValidDate(date)) {
            return date;
        }
        cout << "Invalid date format. Please enter in YYYY-MM-DD format: ";
    }
}
```

```
    bool isValidDate(const string& date) const {
        if (date.size() != 10 || date[4] != '-' || date[7] != '-') {
            return false;
        }
        for (char ch : date) {
            if (!isdigit(ch) && ch != '-') {
                return false;
            }
        }
        return true;
    }

    void displayHeader() const {
        cout << "+------+--------------------------------+------------+
        cout << "|  ID  |              Task              |  Due Date  |
        cout << "+------+--------------------------------+------------+
    }+------------+----------+" << endl;
     Completed  | Priority |" << endl;
     ------------+----------+" << endl;

    void displayFooter() const {
        cout << "+------+--------------------------------+------------+----------
    }

public:
    TodoList() : nextID(1) {
        loadTasks();
    }

    void banner() const {
        cout << "==============================================" << endl;
        cout << "\t\tMy Todo List" << endl;
        cout << "==============================================" << endl;
    }

    void addTask() {
        system("cls");
        banner();
        string desc, date;
        int priority;

        cout << "Enter task description: ";
        cin.ignore();
        getline(cin, desc);
        cout << "Enter due date (YYYY-MM-DD): ";
        date = getValidatedDate();
        cout << "Enter priority (1 = Low, 2 = Medium, 3 = High): ";
        priority = getValidatedChoice(1, 3);

        tasks.emplace_back(nextID++, desc, date, priority);
        saveTasks();

        cout << "\nTask added successfully!" << endl;
        system("pause");
    }

    void showTasks(bool completed = false) const {
        system("cls");
        banner();
        cout << (completed ? "Completed Tasks:" : "All Tasks:") << endl;
        displayHeader();
        for (const auto& task : tasks) {
            if (task.completed == completed) {
                task.display();
            }
        }
        displayFooter();
        system("pause");
    }

    void searchTask() const {
        system("cls");
        banner();
        int id;
        cout << "Enter task ID: ";
        id = getValidatedChoice(1, nextID - 1);
        displayHeader();
        for (const auto& task : tasks) {
            if (task.id == id) {
                task.display();
                displayFooter();
                system("pause");
                return;
            }
        }
        displayFooter();
        cout << "Task not found." << endl;
        system("pause");
    }

    void deleteTask() {
        system("cls");
        banner();
        int id;
        cout << "Enter task ID to delete: ";
        id = getValidatedChoice(1, nextID - 1);

        for (const auto& task : tasks) {
            if (task.id == id) {
                cout << "Task found:\n";
                displayHeader();
                task.display();
                displayFooter();
                cout << "Are you sure you want to delete this task? (yes/no): ";
                string confirmation;
                cin >> confirmation;
                if (confirmation == "yes") {
                    tasks.erase(remove_if(tasks.begin(), tasks.end(),
                    [id](const Task& task) { return task.id == id; }),
                    tasks.end());
                    saveTasks();
                    cout << "\nTask deleted successfully!" << endl;
                } else {
                    cout << "\nTask deletion cancelled." << endl;
                }
                system("pause");
                return;
            }
        }
        cout << "Task not found." << endl;
        system("pause");
    }

    void updateTask() {
        system("cls");
        banner();
        int id;
        cout << "Enter task ID to update: ";
        id = getValidatedChoice(1, nextID - 1);

        for (auto& task : tasks) {
            if (task.id == id) {
                cout << "Task found:\n";
                displayHeader();
                task.display();
                displayFooter();
                cout << "Enter new task description: ";
                cin.ignore();
                getline(cin, task.description);
                cout << "Enter new due date (YYYY-MM-DD): ";
                task.dueDate = getValidatedDate();
                cout << "Enter new priority (1 = Low, 2 = Medium,
                3 = High): ";
                task.priority = getValidatedChoice(1, 3);
                saveTasks();
                cout << "\nTask updated successfully!" << endl;
                system("pause");
                return;
            }
        }
        cout << "Task not found." << endl;
        system("pause");
    }

    void markTaskCompleted() {
        system("cls");
        banner();
        int id;
        cout << "Enter task ID to mark as completed: ";
        id = getValidatedChoice(1, nextID - 1);

        for (auto& task : tasks) {
            if (task.id == id) {
                task.completed = true;
                saveTasks();
                cout << "\nTask marked as completed!" << endl;
                system("pause");
                return;
            }
        }
        cout << "Task not found." << endl;
        system("pause");
    }

    void filterTasks() const {
        system("cls");
        banner();
        cout << "Filter tasks by (1 = Priority, 2 = Due Date): ";
        int filterType = getValidatedChoice(1, 2);

        displayHeader();
        if (filterType == 1) {
            cout << "Enter priority to filter by (1 = Low, 2 = Medium,
            3 = High): ";
            int priority = getValidatedChoice(1, 3);
            for (const auto& task : tasks) {
                if (task.priority == priority) {
                    task.display();
                }
            }
        } else {
            cout << "Enter due date to filter by (YYYY-MM-DD): ";
            string date = getValidatedDate();
            for (const auto& task : tasks) {
                if (task.dueDate == date) {
                    task.display();
                }
            }
        }
        displayFooter();
        system("pause");
    }

    void countTasks() const {
        system("cls");
        banner();
        int total = tasks.size();
        int completed = count_if(tasks.begin(), tasks.end(), []
        (const Task& task) { return task.completed; });
        int notCompleted = total - completed;
        cout << "Total tasks: " << total << endl;
        cout << "Completed tasks: " << completed << endl;
        cout << "Not completed tasks: " << notCompleted << endl;
        system("pause");
    }

    void showMenu() {
        while (true) {
            system("cls");
            banner();
            cout << "1. Add Task" << endl;
            cout << "2. Show All Tasks" << endl;
            cout << "3. Show Completed Tasks" << endl;
            cout << "4. Search Task" << endl;
            cout << "5. Delete Task" << endl;
            cout << "6. Update Task" << endl;
            cout << "7. Mark Task as Completed" << endl;
```

```
            cout << "8. Filter Tasks" << endl;
            cout << "9. Count Tasks" << endl;
            cout << "10. Exit" << endl;
            cout << "Enter your choice: ";
            int choice = getValidatedChoice(1, 10);

            switch (choice) {
                case 1: addTask(); break;
                case 2: showTasks(); break;
                case 3: showTasks(true); break;
                case 4: searchTask(); break;
                case 5: deleteTask(); break;
                case 6: updateTask(); break;
                case 7: markTaskCompleted(); break;
                case 8: filterTasks(); break;
                case 9: countTasks(); break;
                case 10: return;
                default: cout << "Invalid choice. Please try again."
                << endl; break;
            }
        }
    }
};

int main() {
    TodoList todoList;
    todoList.showMenu();
    return 0;
}
```

8. RESULT AND DISCUSSION

The Todo List application project achieved its main objectives by providing a functional and user-friendly tool for task management. The application includes features such as task addition with unique IDs, descriptions, due dates, and priorities; a clear tabular display of tasks; the ability to update task details; and the option to delete tasks. Users can also mark tasks as completed, search for tasks by ID, filter tasks by priority or due date, and view task counts for better oversight. These features collectively ensure effective task management and organization.

However, there were some limitations. The command-line interface, while functional, could benefit from a graphical user interface (GUI) to enhance usability. Additionally, although basic error handling is implemented, more robust validation could improve user experience. The use of text files for data storage is straightforward but may not scale well for a large number of tasks and lacks the reliability of database systems. Overall, the project successfully met its objectives, but future improvements could include a GUI, better error handling, and a more scalable data management approach.

9. CONCLUSION

The Todo List application project successfully fulfilled its primary objectives by delivering a robust and user-friendly tool for task management. The application allows users to efficiently add, update, delete, and track tasks, providing essential functionalities such as task search, filtering by priority or due date, and counting tasks. These features offer a comprehensive solution for managing personal and professional tasks, making the application a practical asset for users seeking better organization and productivity.

Significant achievements include the successful implementation of core functionalities and the creation of a functional command-line interface that meets the project's goals. The system demonstrates the effectiveness of C++ in handling basic task management operations and provides a solid foundation for further development.

Looking ahead, there are several areas for potential enhancement. Implementing a graphical user interface (GUI) could significantly improve user interaction, while adopting a more sophisticated data storage solution, such as a database, would enhance scalability and reliability. Additionally, incorporating advanced error handling and validation mechanisms could further refine the user experience.

In summary, the project has achieved its core objectives and delivered a valuable tool for task management. Future work should focus on expanding the application's capabilities and improving its overall usability to better meet user needs and expectations.

References

@misccppreference, author = cppreference.com, title = C++ Reference, howpublished = https://en.cppreference.com/w/, year = 2024

@misclearncpp, author = LearnCpp.com, title = Learn C++, howpublished = https://www.learncpp.com/, year = 2024

@miscfilehandling, author = GeeksforGeeks, title = File Handling in C++, howpublished = https://www.geeksforgeeks.org/file-handling-c/, year = 2024

@miscoopcpp, author = TutorialsPoint, title = Object-Oriented Programming in C++, howpublished = https://www.tutorialspoint.com/cplusplus/cpp_object_oriented_programmin
htm, year = 2024

@misctodolist, author = CodeProject, title = Simple Todo List Application in C++, howpublished = https://www.codeproject.com/Articles/82076/Creating-a-Todo-List-Applicatio
year = 2024