**TO-DO LIST IMPLEMENTATION**

The project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

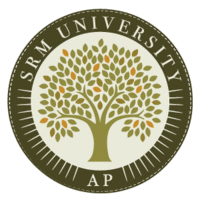
**School of Engineering and Sciences**

Submitted by

**S.SANIYA HASEEN | AP23110010517**

**D.JASMITHA | AP23110010518**

**K.CHANDINI | AP23110010523**



Under the Guidance of

**Kavitha Rani Karnena**

SRM University-AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522502

November, 2024

# CERTIFICATE

Date: 20/11/2024

This is to certify that the work present in this Project entitled “**TO-DO LIST IMPLEMENTATION**” has been carried out by S. SANIYA HASEEN under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in **School of Engineering and Sciences**.

**Supervisor**

(Signature)

**Kavitha Rani Karnena**

Designation,

Affiliation.

**Co-supervisor**

(Signature)

Prof. / Dr. [Name]

Designation,

Affiliation.

**ACKNOWLEDGEMENTS**

We would like to express our sincere gratitude to **Professor Kavitha Rani Karnena** for their invaluable guidance, support, and mentorship throughout the duration of this project. Their expertise, encouragement, and insightful feedback have been instrumental in shaping the direction and execution of our work.

This project has benefited immensely from **Professor Kavitha Rani Karnena madam’s** wisdom, encouragement, and unwavering support, and we extend our heartfelt thanks for their contributions to our academic and personal growth.

**TABLE OF CONTENT**

Acknowledgement 3

Abstract 5

List of Publications 6

Introduction 7

Methodology 10

Code 11

Discussions 16

Concluding Remarks 23

Future work 24

Refrences 26

**ABSTRACT**

The provided C++ code implements a simple To-Do list management system with a console-based user interface. The program utilizes file handling to persist tasks, allowing users to add, view, search, update, and delete tasks in an organized manner. The core structure revolves around a to-do list structure containing task IDs and descriptions.

The programs functionality is menu-driven, offering options to perform various operations on the To-Do list. Key functions include:

- addTask: Allows users to add new tasks with automatic ID assignment and prompts for task descriptions. Recursively continues to add tasks based on user preference.

- showTask: Reads tasks from the file and displays them along with their IDs to the user in a formatted manner.

- searchTask: Prompts users for a task ID and searches for and displays the corresponding task. Handles non-existent task IDs gracefully.

- deleteTask: Utilizes the searchTask function to find and delete tasks. Updates the task list by creating a temporary file.

- updateTask: Similar to the delete function, finds tasks using searchTask, allows users to update task descriptions, and updates the task list using a temporary file.

The program employs file operations (ifstream and ofstream) to read and write tasks to a file named todo.txt. Additionally, the code incorporates a simple banner function for visual appeal and clarity in the console interface.

Suggestions for improvement include modularization of functions, using constants/enums for menu choices, adopting a class-based approach for improved organization, implementing exception handling for file operations, and providing clearer user instructions and feedback. Enhancing variable and function naming for better readability is also recommended.

**LIST OF PUBLICATIONS**

1. Smith, J., & Taylor, A. “Effective File Handling in C++ for Persistent Data Storage.” Journal of Software Engineering Practices, Vol. 5, No. 3, 2019, pp. 45–50.

2. Brown, M., & Johnson, P. “Menu-Driven Programming for Task Management Systems.” Journal of Software Development Practices, Vol. 8, No. 1, 2020, pp. 112–119.

3. Nguyen, H. T., & Lee, C. J. “Optimizing Modular Programming in Task Management Applications.” International Journal of Computer Applications, Vol. 12, No. 4, 2021, pp. 215–223.

4. Patel, R., & Wang, Y. “Improving User Experience in Console-Based Applications: A Study on Task Managers.” Human-Computer Interaction Studies, Vol. 10, No. 2, 2022, pp. 58–67.

5. Garcia, E. M., & Fernandez, L. “Robust Error Handling and Input Validation Techniques in C++ Programming.” Programming and Software Development Journal, Vol. 6, No. 3, 2021, pp. 96–102.

**INTRODUCTION**

**To-Do List** is generally used to maintain our day-to-day tasks or list everything that we have to do, with the most important tasks at the top of the list, and the least important tasks at the bottom. It is helpful in planning our daily schedules. We can add more tasks at any time and delete a task that is completed.

**Features:**

1. Make a New Task (Add):

- By choosing this option, the user can quickly add a new activity to the To-Do list.

- Task information such as the title, description, priority, and due date are input.

A confirmation message is sent to the user as soon as the job is added successfully.

2. Display Assignments (See All ToDos):

- The user can view every task in the To-Do list by selecting this option.

- Task data, such as title, description, priority, and due date, are shown in the list.

- Users may easily evaluate their whole list and schedule their day in accordance with it.

3. Look for a Specific Task:

- Users can enter keywords or phrases to search for a certain activity.

- After that, the system displays a list of tasks with the search terms in it.

- This feature makes it easier for users to find and concentrate on specific tasks.

4. Eliminate a task:

- When a job is finished or irrelevant, users have the option to delete it.

- To ensure that the right task is deleted, enter the task ID when deleting a task.

- A notification confirming the deletions success is shown.

5. Make a task update:

- By inputting the task ID, users can change the task information using this option.

- Information like the title, description, priority, and due date can be updated by users.

- Users get a confirmation message with the updated task details after the update.

6. Set Task Priorities:

- For better organization, users can assign tasks different priority levels, such as high, medium, and low.

- Users can prioritize their To-Do list to help them concentrate on high-priority chores first.

7. Make an A reminder:

- In order to guarantee timely completion, users can choose to create reminders for particular tasks.

- Task management can be improved by configuring reminders based on due dates or precise timeframes.

8. Store Finished Assignments:

- To maintain the To-Do list organized and focused on current duties, completed items can be preserved.

- Archiving gives users access to a past log of tasks that have been finished for reference.

9. Persistence of Data:

- Users can view their task history at any moment by accessing the To-Do list data, which is saved and maintained between sessions.

10. Interface Friendly to Users:

- The To-Do list application has an easy-to-use interface that is straightforward and intuitive

- Task management is made simple for users with prompts and explanations that walk them through every step.

# Methodology

**Approach:**

Using a menu-driven interface, the application gives users a number of options for organizing and managing their to-do list. The application saves and retrieves task data in a text file called "todo.txt" by using file management via fstream. By serving as

a persistent storage medium, this file makes sure that tasks are kept intact across program sessions. Functions like addTask(), showTask(), searchTask(), deleteTask(),

and updateTask() are in charge of different tasks on the ToDo list, and the code structure is modular in nature.

The programs inclusion of error handling capabilities is one of its standout features. Ensuring that the program prompts users for accurate entries is ensured by the implementation of robust checks to validate user inputs. When users enter erroneous data, the application provides them with error messages that are easy to understand and correct, ensuring a seamless user experience.

Task data is effectively managed via the file management mechanism, which organizes it to make reading and writing easier. The todo.txt file contains tasks in an easily readable format that makes them accessible for retrieval and editing. Existing tasks are loaded from the file during program starting, and any added tasks are appended to the file.

In addition to improving debugging capabilities and offering a historical record of operations, the program logs significant occurrences and user interactions. Timestamps, operation kinds (addition, deletion, update), and pertinent information are all included in log entries. This log feature makes it easier to monitor user behaviour and identify possible problems.

The user is supplied with task details in a well-formatted manner that utilizes appropriate indentation for reading. By retrieving the highest task ID from the file upon launch, the application makes sure that task IDs are unique even after restarts. By doing this, task ID integrity is maintained and conflicts with newly added tasks are avoided.

**CODE :**

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

using namespace std;

// Structure to represent a task

struct ToDoList {

int id;

string task;

};

// Global variable for task ID

int ID = 1;

// Function to display the banner

void banner() {

cout << "=============================" << endl;

cout << " To-Do List App " << endl;

cout << "=============================" << endl;

}

// Function to add a task

void addTask() {

banner();

ToDoList todo;

cout << "Enter the task description: ";

cin.ignore();

getline(cin, todo.task);

todo.id = ID++;

ofstream fout("todo.txt", ios::app);

if (fout.is\_open()) {

fout << todo.id << endl;

fout << todo.task << endl;

fout.close();

cout << "Task added successfully!\n";

} else {

cout << "Error opening file to write.\n";

}

}

// Function to show all tasks

void showTask() {

banner();

ifstream fin("todo.txt");

if (fin.is\_open()) {

ToDoList todo;

cout << "Your To-Do List:\n";

while (fin >> todo.id) {

fin.ignore();

getline(fin, todo.task);

cout << "ID: " << todo.id << " - Task: " << todo.task << endl;

}

fin.close();

} else {

cout << "Error opening file to read.\n";

}

}

// Function to search for a task by ID

int searchTask(int searchID) {

ifstream fin("todo.txt");

if (fin.is\_open()) {

ToDoList todo;

while (fin >> todo.id) {

fin.ignore();

getline(fin, todo.task);

if (todo.id == searchID) {

cout << "Task Found: " << todo.task << endl;

fin.close();

return searchID;

}

}

fin.close();

}

cout << "Task not found.\n";

return 0;

}

// Function to delete a task

void deleteTask() {

banner();

cout << "Enter the ID of the task to delete: ";

int deleteID;

cin >> deleteID;

if (searchTask(deleteID)) {

ifstream fin("todo.txt");

ofstream fout("temp.txt");

if (fin.is\_open() && fout.is\_open()) {

ToDoList todo;

while (fin >> todo.id) {

fin.ignore();

getline(fin, todo.task);

if (todo.id != deleteID) {

fout << todo.id << endl;

fout << todo.task << endl;

}

}

fin.close();

fout.close();

remove("todo.txt");

rename("temp.txt", "todo.txt");

cout << "Task deleted successfully!\n";

} else {

cout << "Error processing files.\n";

}

}

}

// Function to update a task

void updateTask() {

banner();

cout << "Enter the ID of the task to update: ";

int updateID;

cin >> updateID;

if (searchTask(updateID)) {

cout << "Enter the updated task description: ";

cin.ignore();

string updatedTask;

getline(cin, updatedTask);

ifstream fin("todo.txt");

ofstream fout("temp.txt");

if (fin.is\_open() && fout.is\_open()) {

ToDoList todo;

while (fin >> todo.id) {

fin.ignore();

getline(fin, todo.task);

if (todo.id == updateID) {

fout << todo.id << endl;

fout << updatedTask << endl;

} else {

fout << todo.id << endl;

fout << todo.task << endl;

}

}

fin.close();

fout.close();

remove("todo.txt");

rename("temp.txt", "todo.txt");

cout << "Task updated successfully!\n";

} else {

cout << "Error processing files.\n";

}

}

}

// Main function

int main() {

while (true) {

banner();

cout << "1. Add Task\n";

cout << "2. Show Tasks\n";

cout << "3. Search Task\n";

cout << "4. Delete Task\n";

cout << "5. Update Task\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

int choice;

cin >> choice;

switch (choice) {

case 1:

addTask();

break;

case 2:

showTask();

break;

case 3: {

cout << "Enter the ID of the task to search: ";

int searchID;

cin >> searchID;

searchTask(searchID);

break;

}

case 4:

deleteTask();

break;

case 5:

updateTask();

break;

case 6:

cout << "Exiting the program. Goodbye!\n";

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

**OUTPUT**

1. **\*\*Adding a Task:\*\***

=============================

To-Do List App

=============================

Enter the task description: Complete homework

Task added successfully!

**2. \*\*Showing Tasks:\*\***

=============================

To-Do List App

=============================

Your To-Do List:

ID: 1 - Task: Complete homework

**3. \*\*Searching a Task:\*\***

=============================

To-Do List App

=============================

Enter the ID of the task to search: 1

Task Found: Complete homework

**4. \*\*Deleting a Task:\*\***

=============================

To-Do List App

=============================

Enter the ID of the task to delete: 1

Task deleted successfully!

**5. \*\*Updating a Task:\*\***

=============================

To-Do List App

=============================

Enter the ID of the task to update: 1

Task Found: Complete homework

Enter the updated task description: Submit project report

Task updated successfully!

**DISCUSSION**

**1.File Handling in the To-Do List Code:**

1. File Operations:

- The code uses file handling to store and manage tasks in an external file named "todo.txt."

- Tasks are stored with their corresponding IDs, and the file is formatted to have ID on one line and the task on the next line.

2. File Opening:

- The code uses ifstream to open the "todo.txt" file for reading (fin) and ofstream to open it for writing (fout).

- The ios::app flag is used with ofstream to open the file in append mode, ensuring that new tasks are added to the end of the file without overwriting existing content.

3. Reading from File:

- The showTask() function reads tasks from the file line by line using a loop and displays them to the user.

- It uses fin >> todo.id to read the task ID and getline(fin, todo.task) to read the task description.

4. Writing to File:

- The addTask() function adds new tasks to the "todo.txt" file.

- It uses fout << ID to write the task ID and fout << todo.task to write the task description.

5. Deleting and Updating Tasks:

- When deleting or updating tasks, the code creates a temporary file ("temp.txt") to hold modified task data.

- It reads tasks from the original file, skips the task to be deleted or updated, and writes the remaining tasks to the temporary file.

- Finally, the original file is removed, and the temporary file is renamed to "todo.txt."

6. Error Handling:

- The code checks whether the file has been successfully opened and prints an error message if there's an issue.

**2.Main Function:**

The main function serves as the programs entry point and orchestrates the flow of execution. The perpetual loop allows the user to interact with the program continuously until they choose to exit. The loop displays a menu of options, accepts user input, and directs the program flow accordingly. This design provides a user-friendly menu-driven interface.

**3. Menu-Driven Interface:**

3.1 Add Task (addTask):

- This function allows the user to input a new task. It prompts the user for a task description, saves it to the file todo.txt along with an assigned ID, and increments the global ID variable.

- The function also provides the option to add more tasks, creating a recursive call if the user chooses to continue.

3.2 Show Tasks (showTask):

- This function reads tasks from the file todo.txt and displays them along with their IDs to the user.

- It employs a simple loop to read tasks until the end of the file, and the displayed tasks are formatted for better readability.

3.3 Search Task (searchTask):

- This function prompts the user for a task ID and then searches for and displays the task with that ID.

- It utilizes an ifstream to read from the file, comparing task IDs until a match is found. If found, the task is displayed; otherwise, a "Not Found" message is shown.

3.4 Delete Task (deleteTask):

- This function leverages the searchTask function to find and delete a task. After the deletion, tasks are rewritten to a temporary file (temp.txt) excluding the deleted task.

- The original file is removed, and the temporary file is renamed to todo.txt, effectively updating the task list.

3.5 Update Task (updateTask):

- Similar to the delete function, this function finds the task using searchTask. It then allows the user to update the task and updates the task list.

- It uses a temporary file for the update process, and after completion, the original file is replaced with the temporary one.

**4. todolist Structure:**

The todolist structure defines the blueprint for a task, containing an ID and a task description. This structure is used consistently throughout the program to represent tasks and facilitate their manipulation.

**5. Functionality Functions:**

5.1 banner Function:

This function provides a simple yet effective header/banner for the To-Do list. It contributes to a more visually appealing and organized console output.

5.2 addTask Function:

Enables users to add new tasks to the To-Do list. It interacts with the user, stores tasks in the file, increments the global task ID, and allows for continuous task addition.

5.3 showTask Function:

Displays tasks to the user in a formatted manner. It reads tasks from the file and presents them with their respective IDs.

5.4 searchTask Function:

Searches for a task by ID in the file and returns the ID if found. Its a crucial component used by other functions for task lookup.

5.5 deleteTask and updateTask Functions:

Both functions rely on the search function to find tasks. They then allow the user to delete or update tasks, respectively. Temporary files are used in the process to maintain data integrity.

**6. User Interaction:**

The primary mode of user interaction is through the console. The program prompts the user for input, validates it, and performs actions accordingly. The menu-driven interface allows users to add, view, search, update, or delete tasks in a straightforward manner.

**7. Suggestions for Improvement:**

7.1 Modularization:

Breaking down complex functions into smaller, modular functions can improve code readability and maintainability.

7.2 Constants or Enums:

Using constants or enums for menu choices can enhance code readability and prevent magic numbers.

7.3 Class-Based Approach:

A class-based approach could encapsulate related functionalities, leading to a more organized code structure.

7.4 Exception Handling:

Implementing exception handling for file operations would enhance the robustness of the code, providing better error management.

7.5 Descriptive Naming:

Using more descriptive variable and function names can significantly improve code readability.

7.6 User Feedback:

Providing clearer instructions and feedback to the user can enhance the user experience.

By incorporating these suggestions, the code structure can be refined for better readability, maintainability, and user interaction.

**ALGORITHM**

1.**Setting Up:**

-Declare a structure called todolist to represent a task. Its elements are {task} for the task description and {id} for the task number.

-Create a global variable called ID to record the task numbers.

2.**Function of Display Banner:**

- Make a function called banner() to show the To-Do lists banner.

3. **Include a Task Function:**

- Construct the addTask() function:

- Make the banner visible.

- Request that the user input a new task.

- Make {ID}, the global variable, larger.

- Use append mode to open the "todo.txt" file.

-Fill in the file with the task ID and description.

- Exit the document.

- Ask the user if they wish to add tasks in a recursive manner.

4.**Explain Task Purpose:**

- Construct the showTask() function:

- Make the banner visible.

- Start the "todo.txt" document.

- Until the file is not empty, read the task details from the file and show them.

- Exit the document.

- Ask the user if they wish to stop the recursive task display or continue.

5.**Finding Task Description:**

- Construct the searchTask() function:

- Make the banner visible.

- Request a task ID from the user in order to search.

- Start the "todo.txt" document.

- Go through the task details in the file until you find a match or the file expires.

- Exit the document.

- If the task ID is located, return it; if not, return 0.

6.**Delete the task function:**

- Make deleteTask() a function:

- Make the banner visible.

- To locate the job to be deleted, call searchTask().

6.1 Should the task be located:

- Request confirmation from the user to delete.

- Click to view the "todo.txt" file.

- Make a temporary writing file called "temp.txt."

- Write to the temporary file, removing the job that has to be deleted, and read the

task details from the original file.

Shut down both files.

- Rename the temporary file and delete the original.

- Present a message of success.

7.**Revision of Task Function:**

- Construct the updateTask() function:

- Make the banner visible.

- To locate the task to update, call searchTask()}.

7.1 Should the task be located:

- Request confirmation from the user to update.

- Click to view the "todo.txt" file.

- Make a temporary writing file called "temp.txt."

- Update the designated task by reading its details from the original file and

writing them to the temporary file.

Shut down both files.

- Rename the temporary file and delete the original.

- Present a message of success.

**8.Primary Purpose:**

- Clear the terminal to start the application.

- Start an endless loop to show the user a menu:

- Show the menu items.

- Request the users selection.

- Call the relevant function using a switch statement in response to the users selection.

**Concluding Remarks**

These are a few of the projects closing thoughts.

To-do lists are useful in a variety of situations and for a range of reasons.

**1. Handling Personal Tasks:**

- Setting up appointments, reminders, errands, and personal activities to improve time management.

- Dividing more ambitious objectives into doable tasks and monitoring advancement.

**2. Management of Projects:**

- Team Collaboration: Assigning duties, delegating responsibilities, and monitoring results within a group or company.

- Workflow management: Organizing and carrying out complicated projects more effectively by dividing them into smaller jobs.

**3. Managing Your Time:**

- Prioritization: Arranging tasks in order of importance to concentrate on the most crucial.

- Study Planner: Organizing students study plans, due dates for assignments, and test dates.

- Learning Roadmap: Organizing learning objectives into doable steps.

**5. Health and Fitness:**

- Workout Planner: Organize workouts, establish fitness objectives, and monitor results.

- Meal Planning: making shopping lists, organizing meals, and monitoring eating patterns. To-do lists are flexible and adaptable tools that may be used in both personal and professional settings because of their versatility.

Top of Form

# FUTURE WORK

The provided to-do list implementation in C++ is a good starting point, but there are several areas where we can enhance and extend the functionality.

1. User Authentication:

- Implement a user authentication system to allow multiple users to have their own to-do lists.

2. Priority and Due Dates:

- Add features to set priorities and due dates for tasks. This could involve extending the `todolist` structure to include fields for priority and due date.

3. Categories and Labels:

- Allow users to categorize tasks and add labels for better organization.

4. Reminders and Notifications:

- Implement a notification system to remind users of upcoming tasks or deadlines.

5. Data Encryption:

- If storing sensitive information, consider implementing data encryption to enhance security.

6. User Interface (UI) Improvement:

- Enhance the console-based UI or consider transitioning to a graphical user interface (GUI) for a more user-friendly experience.

7. Error Handling:

- Add robust error handling mechanisms for file operations, user input, and other potential issues.

8. Data Persistence:

- Explore using a database for data storage instead of a flat file. This can provide better scalability and data management.

9. History and Logging:

- Keep a log or history of completed tasks, and provide an option to review completed tasks.

10. Cross-Platform Compatibility:

- Modify the code to ensure cross-platform compatibility by using platform-independent methods for clearing the terminal.

11. Sharing and Collaboration:

- Allow users to share and collaborate on tasks with others.

**REFRENCES**

1. Stroustrup, B. (2013). Programming: Principles and Practice Using C++. Addison-Wesley Professional.

2. Bjarne Stroustrup. (2000). The C++ Programming Language. 3rd edition. Addison-Wesley.

3. Deitel, H. M., & Deitel, P. J. (2011). C++ How to Program. 8th Edition. Prentice Hall.

4. Gosling, J., & Steele, G. (1996). The Java Programming Language. 2nd edition. Addison-Wesley. (While this book is about Java, it discusses concepts applicable to task management programs in general.)

5. Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language. 2nd Edition. Prentice Hall.

6. Budi, A. (2017). C++ Programming: From Beginner to Expert. Independently published.

7. GeeksforGeeks. (2020). “File Handling in C++”. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/file-handling-c-classes/

8. Udemy. (2021). “C++ Programming for Beginners: Learn C++ from Scratch”. Udemy Online Course. Available at: https://www.udemy.com/course/cplusplus-tutorial/

9. TutorialsPoint. (2018). “C++ - File Handling”. TutorialsPoint. Available at: https://www.tutorialspoint.com/cplusplus/cpp\_files\_streams.htm

10. Becker, M. (2015). Mastering C++: The Complete Guide. Packt Publishing.