A Project Report

On

To Do List

For The Course

“Software Development Project-I”

By

Sumi Akter
ID-IT23055

Supervised by

Dr Ziaur Rahman

Associate Professor,

Department of ICT, MBSTU.

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

SANTOSH, TANGAIL-1902, Dhaka, Bangladesh

Declaration

This is to certify that the candidate worked on the project under the direction of Professor Dr.Ziaur Rahman at department of information and communication technology in the MBSTU, Tangail, Bangladesh. The fact that none of these projects has ever been presented elsewhere for a degree or diploma is also declared. A list of references is provided, and information taken from both published and unpublished works of others is recognized in the text.

Signature of Supervisor

Dr. Ziaur Rahman

Professor

Dept of ICT MBSTU

Introduction of Project

Project Overview:

The **To-Do List Application** is a command-line program developed in C++ to help users manage and organize tasks. It includes functionalities to **add, display, delete, edit, and mark tasks as completed**, storing them in a file for persistence. This ensures that the user's task list remains accessible even after the program exits, making it a reliable tool for basic task management.

Objectives:

1. **Implement Task Management**: Develop functions for creating, viewing, editing, deleting, and marking tasks as completed.
2. **Enable Data Persistence**: Ensure that tasks are stored in a file, so data persists across sessions.
3. **Provide an Intuitive Interface**: Use a menu-based system to guide users through the available task management options.

Target Audience:

- **General Audience**: The to-do list application is designed to appeal to users of all ages and backgrounds who are interested in organizing their daily tasks and improving productivity in a simple, accessible way.
- **Students**: The application can be an essential tool for students to manage their study schedules, keep track of assignments, prioritize tasks, and maintain a balanced workload. It encourages good time management and helps reinforce organizational skills.

C ++ Programming Language

C++ is a programming language that is an extension of an earlier language, C. For the most part, we will use the C subset of C++ in this course because it provides the tools that we need to explore physical data structures. A few of the language features that we will use are part of C++ but not of C. These notes make no attempt to offer a com

Most Important Features of C Language:

- Simple.
- Abstract Data types.
- Machine Independent or Portable.
- Mid-level programming language.
- Structured programming language.
- Rich Library.
- Memory Management.
- Quicker Compilation.

Advantages:

- Object-Oriented. C++ is an object-oriented programming language which means that the main focus is on objects and manipulations around these objects. ...
- Speed. ...
- Compiled. ...
- Rich Library Support. ...
- Pointer Support. ...
- Closer to Hardware.

Disadvantages:

- Object-orientated programming languages have several security issues which means that programs written in C++ aren't as safe as others.
- The pointers that are used in C++ take up a lot of memory which is not always suitable for some devices.
- Cannot support built-in code threads.

2.3 IDE details

Code::Blocks:

Code::Blocks is a free, open-source cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C, C++, and Fortran. It has a custom build system and optional Make support.

Features

Compilers

Code::Blocks supports multiple compilers, including GCC, MinGW, Digital Mars, Microsoft Visual C++, Borland C++, LLVM Clang, Watcom, LCC and the Intel C++ compiler. Although the IDE was designed for the C++ language, there is some support for other languages, including Fortran and D. A plug-in system is included to support other programming languages.

Code editor

The IDE features syntax highlighting and code folding (through its Scintilla editor component), C++ code completion, class browser, a hex editor and many other utilities. Opened files are organized into tabs. The code editor supports font and font size selection and personalized syntax highlighting colors.

Debugger

The Code::Blocks debugger has full breakpoint support. It also allows the user to debug their program by having access to the local function symbol and argument

display, user-defined watches, call stack, disassembly, custom memory dump, thread switching, CPU registers and GNU Debugger Interface.

Code Structure and Implementation

The program consists of **several key components** to handle task operations and file I/O, ensuring efficient management of tasks. Below are the main components, data structures, and functions that were implemented.

1. Data Structure

- **Task Struct**: A `Task` struct is used to store the essential information for each task:

```
struct Task {
    string name;         // Stores the name or description of the task
    bool completed;      // Boolean value indicating completion status
};
```

This structure allows us to encapsulate each task's name and its completion status (`completed` is `true` if the task is completed and `false` otherwise).

- **Tasks Array**: The program uses a static array, `tasks[MAX_TASKS]`, to hold up to 100 tasks:

2. Function Declarations

The program includes the following function prototypes, each designed to handle a specific aspect of task management:

```
void displayMenu();                              // Displays the main menu
void addTask(Task tasks[], int& taskCount);  // Adds a new task to the list
void displayTasks(const Task tasks[], int taskCount); // Displays all tasks
void deleteTask(Task tasks[], int& taskCount); // Deletes a task by number
void markTaskCompleted(Task tasks[], int taskCount);  // Marks a task as completed
void editTask(Task tasks[], int taskCount);   // Edits the name of a specific task
void loadTasksFromFile(Task tasks[], int& taskCount); // Loads tasks from the file
void saveTasksToFile(const Task tasks[], int taskCount); // Saves tasks to the file
```

Each function operates on the tasks array and taskCount, modifying or displaying task data as needed.

3. Functional Descriptions

Here's an overview of each function:

1. **displayMenu()**:
    - Presents the menu to the user, allowing them to select from available actions.

```
void displayMenu() {
    cout << "\n---------- To-Do List Menu ----------\n";
    cout << "1. Add Task\n";
    cout << "2. Display Tasks\n";
    cout << "3. Delete Task\n";
    cout << "4. Mark Task as Completed\n";
    cout << "5. Edit Task\n";
    cout << "6. Exit\n";
    cout << "-------------------------------------\n";
}
```

2. **addTask()**:
    - Adds a new task by asking the user for a name and setting its completed status to false initially.
3. **displayTasks()**:
    - Iterates through the tasks array and prints each task, showing its name and whether it is completed or not.
4. **deleteTask()**:
    1. Deletes a task by shifting all tasks after the deleted task one position up in the array. It then decreases the taskCount by one.
5. **markTaskCompleted()**:
    1. Allows users to mark a specific task as completed by setting its completed status to true.
6. **editTask()**:
    1. Enables users to update the name of a selected task.
7. **loadTasksFromFile()**:
    1. Reads each line from tasks.txt, extracting the task name and status, then populates the tasks array with this data.
8. **saveTasksToFile()**:
    1. Writes the current list of tasks to tasks.txt in a format that includes the task name and its completion status.

Source Code

```
#include <iostream>

#include <string>

#include <fstream>


using namespace std;


const int MAX_TASKS = 100; // Maximum number of tasks

const string FILENAME = "tasks.txt"; // File to store tasks


// Structure to store a task

struct Task

{

   string name;

   bool completed;

};


// Function prototypes

void displayMenu();

void addTask(Task tasks[], int& taskCount);

void displayTasks(const Task tasks[], int taskCount);

void deleteTask(Task tasks[], int& taskCount);
```

```
void markTaskCompleted(Task tasks[], int taskCount);
void editTask(Task tasks[], int taskCount);
void loadTasksFromFile(Task tasks[], int& taskCount);
void saveTasksToFile(const Task tasks[], int taskCount);

int main()
{
    Task tasks[MAX_TASKS];
    int taskCount = 0;

    // Load tasks from the file when the program starts
    loadTasksFromFile(tasks, taskCount);

    int choice;
    do
    {
        displayMenu();
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
```

```
            addTask(tasks, taskCount);
            break;
        case 2:
            displayTasks(tasks, taskCount);
            break;
        case 3:
            deleteTask(tasks, taskCount);displayTasks(tasks, taskCount);
            break;
        case 4:
            markTaskCompleted(tasks, taskCount);
            break;
        case 5:
            editTask(tasks, taskCount);
            break;
        case 6:
            cout << "Exiting program. Bye!" << endl;
            break;
        default:
            cout << "Invalid choice. Please try again!" << endl;
        }
    }
    while (choice != 6);
```

```
    return 0;
}

void displayMenu()
{
    cout << "\n---------- To-Do List Menu ----------\n";
    cout << "1. Add Task\n";
    cout << "2. Display Tasks\n";
    cout << "3. Delete Task\n";
    cout << "4. Mark Task as Completed\n";
    cout << "5. Edit Task\n";
    cout << "6. Exit\n";
    cout << "-------------------------------------\n";
}

void addTask(Task tasks[], int& taskCount)
{
    if (taskCount >= MAX_TASKS)
    {
        cout << "Task list is full! Cannot add more tasks." << endl;
        return;
    }
```

```
    cout << "Enter New Task: ";
    cin.ignore(); // Ignore leftover newline from previous input
    getline(cin, tasks[taskCount].name);
    tasks[taskCount].completed = false;

    taskCount++;
    cout << "Task added successfully!" << endl;
    saveTasksToFile(tasks, taskCount);
}
void displayTasks(const Task tasks[], int taskCount)
{
    if (taskCount == 0)
    {
        cout << "No tasks to display!" << endl;
        return;
    }

    cout << "Tasks:" << endl;
    for (int i = 0; i < taskCount; ++i)
    {
        cout << i + 1 << ". " << tasks[i].name << " ("
            << (tasks[i].completed ? "Complete" : "Incomplete") << ")" << endl;
    }
```

```
}
void deleteTask(Task tasks[], int& taskCount)
{
    if (taskCount == 0)
    {
        cout << "No tasks to delete!" << endl;
        return;
    }

    displayTasks(tasks, taskCount);
    cout << "Enter the task number to delete: ";
    int taskNumber;
    cin >> taskNumber;

    if (taskNumber >= 1 && taskNumber <= taskCount)
    {
        for (int i = taskNumber - 1; i < taskCount - 1; ++i)
        {
            tasks[i] = tasks[i + 1];
        }
        taskCount--;
        cout << "Task deleted successfully!" << endl;
        saveTasksToFile(tasks, taskCount);
```

```
    }
    else
    {
        cout << "Invalid task number!" << endl;
    }
}



void markTaskCompleted(Task tasks[], int taskCount)
{
    if (taskCount == 0)
    {
        cout << "No tasks to mark as completed!" << endl;
        return;
    }

    displayTasks(tasks, taskCount);
    cout << "Enter the task number to mark as completed: ";
    int taskNumber;
    cin >> taskNumber;

    if (taskNumber >= 1 && taskNumber <= taskCount)
```

```
    {
        tasks[taskNumber - 1].completed = true;
        cout << "Task marked as completed!" << endl;
        saveTasksToFile(tasks, taskCount);
    }
    else
    {
        cout << "Invalid task number!" << endl;
    }
}

void editTask(Task tasks[], int taskCount)
{
    if (taskCount == 0)
    {
        cout << "No tasks to edit!" << endl;
        return;
    }

    displayTasks(tasks, taskCount);
    cout << "Enter the task number to edit: ";
    int taskNumber;
    cin >> taskNumber;
```

```
    if (taskNumber >= 1 && taskNumber <= taskCount)
    {
        cout << "Enter new task name (current: " << tasks[taskNumber - 1].name << "): ";
        cin.ignore(); // Ignore leftover newline from previous input
        getline(cin, tasks[taskNumber - 1].name);

        cout << "Task updated successfully!" << endl;
        saveTasksToFile(tasks, taskCount);
    }
    else
    {
        cout << "Invalid task number!" << endl;
    }
}

void loadTasksFromFile(Task tasks[], int& taskCount)
{
    ifstream file(FILENAME);
    if (file.is_open())
    {
        string line;
        while (getline(file, line) && taskCount < MAX_TASKS)
```

```
        {
            size_t delimiterPos = line.find(" | ");
            if (delimiterPos != string::npos)
            {
                string name = line.substr(0, delimiterPos);
                string status = line.substr(delimiterPos + 3);
                tasks[taskCount].name = name;
                tasks[taskCount].completed = (status == "complete");
                taskCount++;
            }
        }
        file.close();
    }
}

void saveTasksToFile(const Task tasks[], int taskCount)
{
    ofstream file(FILENAME);
    if (file.is_open())
    {
        for (int i = 0; i < taskCount; ++i)
        {
            file << tasks[i].name << " | "
```

```
            << (tasks[i].completed ? "complete" : "incomplete") << endl;
        }
        file.close();
    }
}
```

Sample Interactions

1. **Menu Display**:

```
---------- To-Do List Menu -----------
1. Add Task
2. Display Tasks
3. Delete Task
4. Mark Task as Completed
5. Edit Task
6. Exit
---------------------------------------
```

2. **Adding a Task**:

```
Enter New Task: Buy groceries
Task added successfully!
```

3. **Marking a Task as Completed**:

```
Enter the task number to mark as completed: 1
Task marked as completed!
```

Editing a Task:

```
Enter the task number to edit: 1
Enter new task name (current: Buy groceries): Buy fruits and vegetables
Task updated successfully!
```

File Format and Persistence

The tasks are saved in tasks.txt in the format:

```
Task Name | Status
```

where Status is either complete or incomplete. This format allows easy parsing and ensures data consistency.

Testing and Validation

The program was tested extensively for correctness:

1. **Functionality Testing**: Verified each function independently to ensure they perform as expected.
2. **File Handling**: Confirmed that data is saved and loaded correctly by modifying tasks, then restarting the program to check for data persistence.
3. **Edge Cases**: Ensured robustness by handling cases such as adding up to MAX_TASKS, attempting to delete non-existent tasks, and managing invalid inputs.

Potential Enhancements

1. **Dynamic Memory Allocation**: Replace the fixed-size array with a dynamically allocated structure, such as std::vector, to allow for an unlimited number of tasks.

2. **Improved UI**: A GUI could improve user experience by providing a more interactive task management interface.
3. **Task Sorting and Searching**: Adding functions to sort tasks by name or search for tasks by keywords.
4. **Task Prioritization**: Include priority levels, allowing tasks to be sorted by importance.

Conclusion

This project successfully achieved the objectives, implementing a basic yet functional to-do list application in C++. By handling task data through a simple menu and file persistence, the application provides a straightforward way for users to manage tasks effectively. This codebase could be extended with more features or a graphical interface to make it even more user-friendly and powerful in the future.