**REPORT**

**To-Do List Application Code in C++**  
 **1. Introduction**  
This is a very simple, file-based application implemented in C++. Using the application, the users will be able to enter and view, mark, delete tasks, among which are stored on the files that will exist even if this application terminates.

**2. Code Structure and Class Design**  
There are two classes in this application:  
1. Task - A task class itself. It consists of various attributes like description, due date, and whether or not it's completed. Also, a method exists for printing details about a task.  
2. ToDoList- Handles a collection of instances of the task class; besides, the class contains code for opening and closing the file and saving, respectively, which means it keeps track of tasks in successive sessions.  
  
The basic functionality of the ‘ToDoList’ class is shown as follows:  
 1. Add Task  
 2. Remove Task  
 3. Mark as Task Completed  
 4. Display Tasks  
 5. File Operations (Saving and Reading Work Items)

**3. Class Definition and Functionality Division**

* **Task Class**  
  The Task class is simple with three member variables:  
  - description: It contains a description of the task  
  - dueDate: It includes the due date for a particular task  
  - isCompleted: This Boolean flag tells whether the work item is completed or not.  
  The Task class is further augmented with a function named ‘displayTask()’ which prints task details onto the console.
* **ToDoList Class**  
  The ‘ToDoList’ class controls a list of ‘Task’ objects and uses files to store persistently. Its core methods are as follows:  
    
  **Constructor-**Calls the ‘loadTasks()’ function to load tasks from a file.  
  **Destructor-**  
  Calls the ‘saveTasks()’ function to save tasks to a file when the application closes.

**Core Methods-**  
1. addTask: The task is added in ‘tasks’ vector as it generates a ‘Task’ object and appends this into the ‘tasks’ vector. It then responds back to the user informing the addition of task into ‘tasks’ vector.

2.removeTask: Delete task according to its index position present in the vector "tasks". It makes verification regarding the valid input related to index.  
  
3. markTaskAsCompleted: Marks the ‘isCompleted’ flag for a given task as ‘true’. As with ‘removeTask’, it also checks the index for bounds.  
  
4. viewTasks Prints all tasks in the list on the console. If there are no tasks in the list, it prints a message to the user.  
  
5. saveTasks Saves all the tasks in the ‘tasks.txt’ file with each task's attributes separated by a semicolon. The line format is: ‘description;dueDate;isCompleted’.  
  
6. loadTasks: when the application starts, will read tasks from ‘tasks.txt’. The function scans every line in the file using a semicolon separator that will be used to draw out task data and reform ‘Task’ objects respectively.

**4. File Handling**  
File operation is implemented with the application's ‘saveTasks()’ method and ‘loadTasks()’. The tasks are held in ‘tasks.txt,’ and these are merely simple text, easy for the application to parse or read.  
 - Persistence: Tasks are written out to disk when the program terminates and loaded again at startup.  
 - Efficiency: Simple text files keep the file structure light and quick to read/write.

**5. User Interface**  
 The console-based user interface enables the following operations:  
  
 1. Add Task: It will ask for a description of the task and the due date  
 2. View Tasks: List all tasks, and completion status.  
 3. Mark Task as Completed: This facility allows the user to pick a task to mark completed.  
 4. Delete Task: This facility is used to delete a given task.  
 5. Exit: Saves tasks and exits the program.

**6.Program Flow and Control Logic:**  
The ‘main()’ function performs a menu-driven interface of a ‘do-while’ loop. It generates the menu options, captures the user's choice through the keyboard, and applies the corresponding ‘ToDoList’ methods according to choice. The control logic helps prevent out-of-bounds errors while indexing the list of tasks.  
  
**7. Testing:**  
All the above scenarios were executed for testing the application:  
1. Adding Tasks: Ensured that there was no problem in including tasks in the list.  
2. Remove and Mark Operations: The remove operations should be implemented correctly, along with proper display of status update about completion.  
3. File Operations: Implemented the methods ‘saveTasks()’ and ‘loadTasks()’ ensuring data integrity over multiple runs of the program.  
4. Edge Cases: Implemented the response for invalid input, such as selecting non existent task indices for mark or delete.

**8. Conclusion:**  
This is a to-do list application that will demonstrate usage of classes, vectors, file handling, and also simple console-based user interfaces in C++. The whole code structure is modular with the efficiency of `ToDoList` encapsulating most of the core functionality of this application. File-based persistence is implemented in this code to ensure that one does not lose his work during sessions, thus bringing this into daily use for handling tasks.

**9. Improvements:**  
This application can be developed further as follows:  
1. Sort or Filter by Priority Level: The application can be used to sort or filter tasks based on priority levels.  
2. Reminder of Due Dates: Remind users of tasks whose due dates are near.  
3. Better file format: Use JSON or XML format for more organized data storage.  
4. GUI Integration: The application incorporates a GUI using Qt or a web-based frontend.

NAME: Shravani Dinkar Sonawane.

DEPARTMENT: Cyber security (F.Y)

PRN: 2124UCSF1105