TODO APP: Eleanor Chiu (10942401) CSCI Section B

PROBLEM DESCRIPTION

A TODO app that allows users to organize and track their todo items through a command-line interface. The program will enable users to perform CRUD operations (Create, Read, Update, Delete) on their tasks, with each task containing detailed information including description, priority level, completion status, due date. All task data will be stored persistently in a CSV file, loaded into memory at program startup, manipulated through a vector data structure, and saved back to the file upon program exit or after each operation.

DATA DESCRIPTION

1. UML Class Diagram

```
+----------------------------------+
|            Task                  |
+----------------------------------+
| - id: int                        |
| - description: string            |
| - priority: int                  |
| - isCompleted: bool              |
| - dueDate: time_t                |
| - createdDate: time_t            |
| - lastModified: time_t           |
+----------------------------------+
| + Task()                         |
| + Task(id, desc, priority, due)  |
| + getId(): int                   |
| + setId(int): void               |
| + getDescription(): string       |
| + setDescription(string): void   |
| + getPriority(): int             |
| + setPriority(int): void         |
| + getIsCompleted(): bool         |
| + setIsCompleted(bool): void     |
| + getDueDate(): time_t           |
| + setDueDate(time_t): void       |
| + getCreatedDate(): time_t       |
| + getLastModified(): time_t      |
+----------------------------------+

// Task class represents a single todo item
// Data members store task properties including:
//   - id: unique identifier for each task
//   - description: text describing the task
//   - priority: integer from 1-5 (1=lowest, 5=highest)
//   - isCompleted: tracks completion status
//   - dueDate: unix timestamp for task deadline
//   - createdDate: unix timestamp when task was created
//   - lastModified: unix timestamp of last update time
```

```
// Member functions provide:
//   - Constructors for creating new tasks
//   - Getters and setters for all data members

+---------------------------------+
|         TodoManager             |
+---------------------------------+
| - tasks: vector<Task>           |
| - filename: string              |
+---------------------------------+
| + TodoManager(filename)         |
| + loadFromFile(): bool          |
| + saveToFile(): bool            |
| + createTask(desc, pri, due): void |
| + readAllTasks(): void          |
| + readTaskById(id): Task*       |
| + updateTask(id, ...): bool     |
| + deleteTask(id): bool          |
+---------------------------------+

// TodoManager class manages the collection of tasks
// Data members:
//   - tasks: vector storing all Task objects in memory
//   - filename: path to CSV file for persistence
// Member functions provide:
//   - Constructor to initialize with filename
//   - loadFromFile() reads all tasks from CSV into vector
//   - saveToFile() writes all tasks from vector to CSV
//   - createTask() adds new task to vector
//   - readAllTasks() displays all tasks to user
//   - readTaskById() finds and returns specific task
//   - updateTask() modifies existing task properties
//   - deleteTask() removes task from vector
```

2. Data Structure Usage

The program will use a vector of Task objects (`vector<Task>`) as the primary data structure to store all tasks in memory during program execution. This vector will be a member of the TodoManager class.

3. File I/O Usage

The program will use a CSV file to store task data between program runs. Each line in the CSV file represents one task with fields: id, description, priority, isCompleted, dueDate, createdDate, lastModified

- Read: Load all existing tasks at program startup
- Write: Save all tasks when user exits program or after each CRUD operation
- Both read and write operations will be manually implemented using ifstream and ofstream (no third-party libraries for file I/O)

PROCEDURAL DESCRIPTION

```
1. START PROGRAM and display welcome message

5. MAIN LOOP (repeat until user chooses to exit):
   a. Display menu options:
      - [1] Create new task
      - [2] View all tasks
            - [2.1]* Filter by completion status
            - [2.2]* Sort tasks by priority
            - [2.3]* Sort tasks by due date
      - [3] Update a task
      - [4] Delete a task
      - [8]* View statistics
                - Total number of tasks
                - Number of completed tasks
                - Number of incomplete tasks
                - Completion percentage
                - Number of overdue tasks (if current date > due date)
      - [9] Exit program

   b. Get user input for menu choice

   c. SWITCH on user choice:
      CASE 1 (Create):
         - Prompt user for description, priority (1-5), due date
         - Call createTask() to add new Task to vector
         - Call saveToFile() to persist changes
         - Display success message

      CASE 2 (View all):
         - Call readAllTasks() to display all tasks from vector
         - Format output with columns for ID, description, priority, due
date, status

      CASE 3 (Update):
         - Prompt user for task ID
         - Call readTaskById() to find task in vector
         - If task exists:
            - Display current task details
            - Prompt user for new values (or press Enter to keep current)
            - Call updateTask() to modify task in vector
            - Update lastModified timestamp
            - Call saveToFile() to persist changes
            - Display success message
         - Else display error message

      CASE 4 (Delete):
         - Prompt user for task ID
         - Call deleteTask() to remove task from vector
         - If successful:
            - Call saveToFile() to persist changes
            - Display success message
         - Else display error message
```

```
       CASE 9 (Exit):
          - Display goodbye message
          - Exit main loop

       DEFAULT:
          - Display invalid choice message

6. END PROGRAM
```

Libraries to be used:

- `<iostream>` for console input/output
- `<fstream>` for file I/O operations
- `<vector>` for vector data structure
- `<string>` for string manipulation
- `<ctime>` for timestamp operations
- `<iomanip>` for formatted output display
- `<sstream>` for string parsing (CSV parsing)

Functions TBD

The Below functions (the one with asterisk in the psedo code) are those I'm not sure if I should include in my project proposal. Or even if I later decide not to include them, I figure it might be nice to document them here for future improvement.

```
- [2.1*] Filter by completion status
- [2.2*] Sort tasks by priority
- [2.3*] Sort tasks by due date
- [3] Update a task
- [4] Delete a task
- [8*] View statistics
```

And also some potential future improvements (I drew some ideas from Apple's reminder app)

- **Categories/Tags**: Add a category field to Task class (e.g., Work, Personal, School) and allow filtering by category
- **Reminders**: Display overdue tasks or tasks due soon when program starts
- **Color-coded display**: Use terminal colors to highlight high-priority or overdue tasks

SPECIAL NEEDS/CONCERNS

1. How to generate UUID?
    - Some possible solutions: track the highest ID when loading from file, then use nextId = maxId + 1
    - seek third party library