BY :IKHTAJ HASSAN

Cs 3rd (A)

To-Do List Program Documentation

Overview:

This program allows the user to manage a simple to-do list by performing tasks like adding tasks, viewing tasks, and deleting tasks. The program is implemented in C++ and provides a basic text-based menu for interaction.

Features:

- **Add Task**: Allows users to add new tasks to the list.
- **View Tasks**: Displays all the tasks currently stored in the to-do list.
- **Delete Task**: Allows the user to remove a specific task from the list based on its index (task number).
- **Exit**: Terminates the program.

Program Structure:

1. **Class: `ToDoList`**
   This class contains the core functionality of the to-do list management.
   - **Private Member:**
     - `vector<string> tasks`: A dynamic array (vector) that stores the list of tasks.
   - **Public Methods:**
     - **`addTask(const string& task)`**: Adds a new task to the `tasks` list.
     - **`viewTasks() const`**: Displays all the tasks in the list. If the list is empty, it notifies the user.
     - **`deleteTask(int taskNumber)`**: Deletes a task at a specified index (1-based) from the `tasks` list. If the task number is invalid, it shows an error message.
2. **Main Function:**
   - Displays a menu with four choices: add a task, view tasks, delete a task, or exit.
   - Based on user input, the appropriate method from the `ToDoList` class is invoked.
   - The loop continues until the user selects "Exit" to terminate the program.

Menu Options:

- **1. Add Task**: Prompts the user to enter a task and adds it to the list.

- **2. View Tasks**: Displays all tasks in the to-do list.
- **3. Delete Task**: Prompts the user to specify the task number to delete.
- **4. Exit**: Ends the program.

Detailed Functionality:

1. **Add Task**:
    - The program prompts the user to input the task description.
    - The task is added to the `tasks` vector.
    - The program confirms the addition with a message: `"Task added!"`.
2. **View Tasks**:
    - The program checks if there are any tasks in the list.
    - If there are tasks, it displays each task with its number (index + 1).
    - If there are no tasks, it displays the message `"No tasks to display!"`.
3. **Delete Task**:
    - The user is asked to input the task number they want to delete.
    - If the task number is valid, it is removed from the list, and a confirmation message is shown: `"Task deleted!"`.
    - If the task number is invalid (outside the valid range), an error message is displayed: `"Invalid task number!"`.
4. **Exit**:
    - The program ends when the user selects the "Exit" option from the menu.

Code Flow:

1. The program starts by initializing a `ToDoList` object named `myList`.
2. It enters a `do-while` loop to display the menu until the user chooses to exit.
3. Depending on the user's menu choice, the program performs the respective actions like adding, viewing, deleting tasks, or exiting.

Error Handling:

- **Invalid Task Number**: If the user enters a task number that doesn't exist in the list, the program will print: `"Invalid task number!"`.
- **Invalid Menu Choice**: If the user enters a choice that is not between 1 and 4, the program will prompt: `"Invalid choice, please try again."`.

Conclusion:

This simple to-do list program allows users to manage their tasks interactively through a text-based interface. It demonstrates the use of the `vector` container to store tasks and basic menu-driven control flow for task management.

-