Course Work Assignment

CS/2020/004

U.T.A. Perera

To-Do List Application

Introduction:

The To-Do List Application is a user-friendly C program designed to help you manage your tasks efficiently, whether it's organizing daily chores, tracking work assignments, or jotting down important reminders. With a straightforward interface, you can add new tasks, mark them as completed when finished, and view the list at any time. The program provides a clean and organized display of your tasks, allowing you to focus on what's important without the hassle of remembering everything in your busy schedule. To use the application, select option "1" to add a new task, enter a brief description of the task, mark it as complete by providing the index, and view all completed tasks and their status. The application can add up to 100 tasks to your list, and the application will inform you if the task limit has been reached. Start managing your tasks effectively and conquer your day with ease with the To-Do List Application.

Purpose:

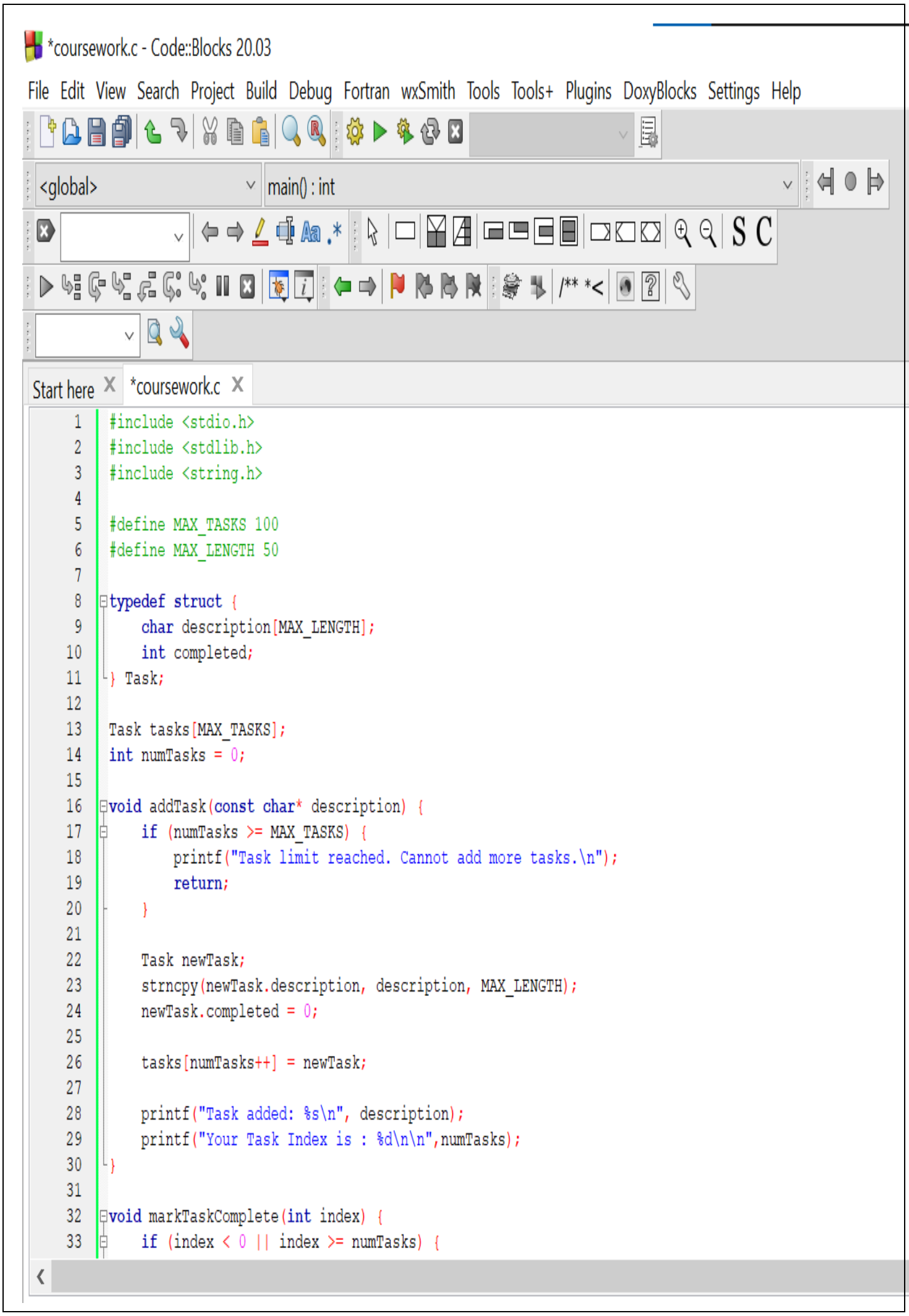
This code implements a simple C To-Do List application, allowing users to manage their tasks efficiently. The program offers various actions, including adding tasks, marking tasks as completed, printing tasks, and exiting the application. Users can add new tasks by entering a brief description, store them in an array, mark tasks as completed by providing the index, view their to-do list, and exit when done. The code uses a `struct` to represent each task, with a description and completion status flag. The main function provides a simple menu-driven interface, allowing users to choose the desired action by entering corresponding options. The To-Do List application serves as a starting point for task management and can be expanded and improved to suit specific requirements.

Code Structured:

The code is structured as a C program with several functions and a main function to provide the functionality of the To-Do List application. The code includes standard header files for input/output operations, memory allocation, and string handling. The macros define the maximum number of tasks and the maximum length of each task description. The task structure is created using a typedef struct, which represents a single task in the to-do list. The program has two global variables: `tasks` and `Num Tasks`. Function definitions are provided to perform various actions, such as adding a new task, marking a task as complete, and printing the list of tasks along with their completion status. The main function is the entry point, providing a user-friendly menu for users to interact with the application. The code's modular structure improves readability and maintainability, as each function is responsible for a specific task. The available menu options include Add task ('1'), Mark task complete ('2'), Print tasks ('3'), and Exit ('0'). The code's modular structure improves readability and maintainability, as each function is responsible for a specific task. However, there are potential areas for improvement, such as input validation and handling larger task lists, which can enhance the program's robustness and usability.

Execution:

The To-Do List application code works by initializing the ‘tasks’ array and setting `Num Tasks` to 0. The program then enters a `do-while` loop, which continues until the user chooses option '0'. Inside the loop, the application displays a menu with options to add tasks, mark tasks complete, print tasks, or exit. The user inputs their choice and presses Enter. The program performs actions based on the user's choice, such as adding a task, marking tasks complete, printing tasks, and exiting. The '1' option adds a task to the `tasks` array, while the '2' option marks a task as completed. The '3' option prints the list of tasks and their completion status. If no tasks are found, an error message is displayed. The '0' option exits the loop, allowing the user to perform further actions or exit the application. The loop continues until the user chooses option '0', which breaks the loop and prints an "Exiting..." message before terminating. The program keeps track of the added tasks and their completion status, enabling users to effectively manage their to-do list. To use the To-Do List application, follow the instructions displayed in the menu and enter the corresponding option numbers. Users can add new tasks, mark them as complete, and view the list of tasks at any time, with the program guiding them through the process.


*coursework.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
main() : int
Start here
*coursework.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_TASKS 100
#define MAX_LENGTH 50
typedef struct {
char description[MAX_LENGTH];
int completed;
} Task;
Task tasks[MAX_TASKS];
int numTasks = 0;
void addTask(const char* description) {
if (numTasks >= MAX_TASKS) {
printf("Task limit reached. Cannot add more tasks.\n");
return;
}
Task newTask;
strncpy(newTask.description, description, MAX_LENGTH);
newTask.completed = 0;
tasks[numTasks++] = newTask;
printf("Task added: %s\n", description);
printf("Your Task Index is : %d\n\n",numTasks);
}
void markTaskComplete(int index) {
if (index < 0 || index >= numTasks) {

*coursework.c - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global> main() : int

Start here | *coursework.c

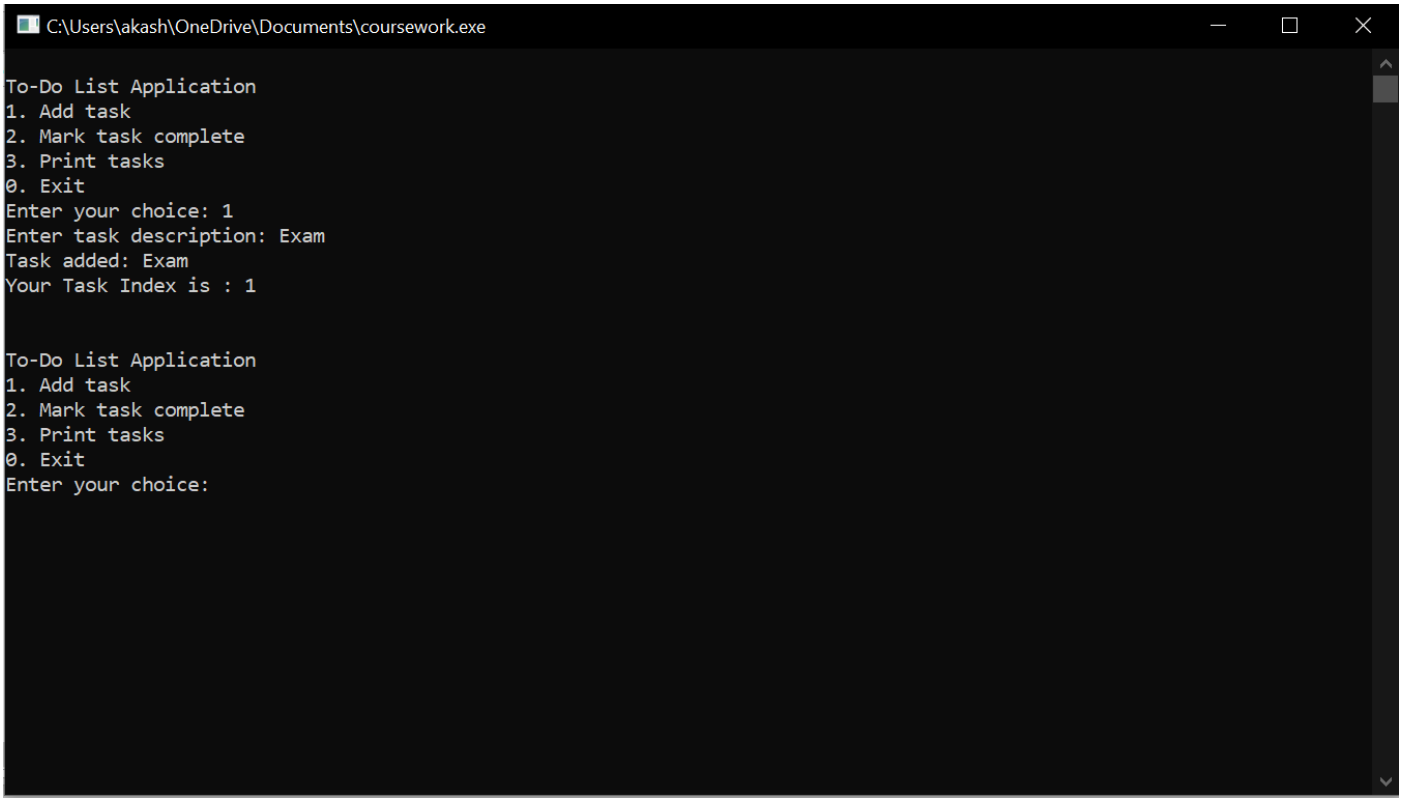
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_TASKS 100
#define MAX_LENGTH 50

typedef struct {
    char description[MAX_LENGTH];
    int completed;
} Task;

Task tasks[MAX_TASKS];
int numTasks = 0;

void addTask(const char* description) {
    if (numTasks >= MAX_TASKS) {
        printf("Task limit reached. Cannot add more tasks.\n");
        return;
    }

    Task newTask;
    strncpy(newTask.description, description, MAX_LENGTH);
    newTask.completed = 0;

    tasks[numTasks++] = newTask;

    printf("Task added: %s\n", description);
    printf("Your Task Index is : %d\n\n",numTasks);
}

void markTaskComplete(int index) {
    if (index < 0 || index >= numTasks) {
```

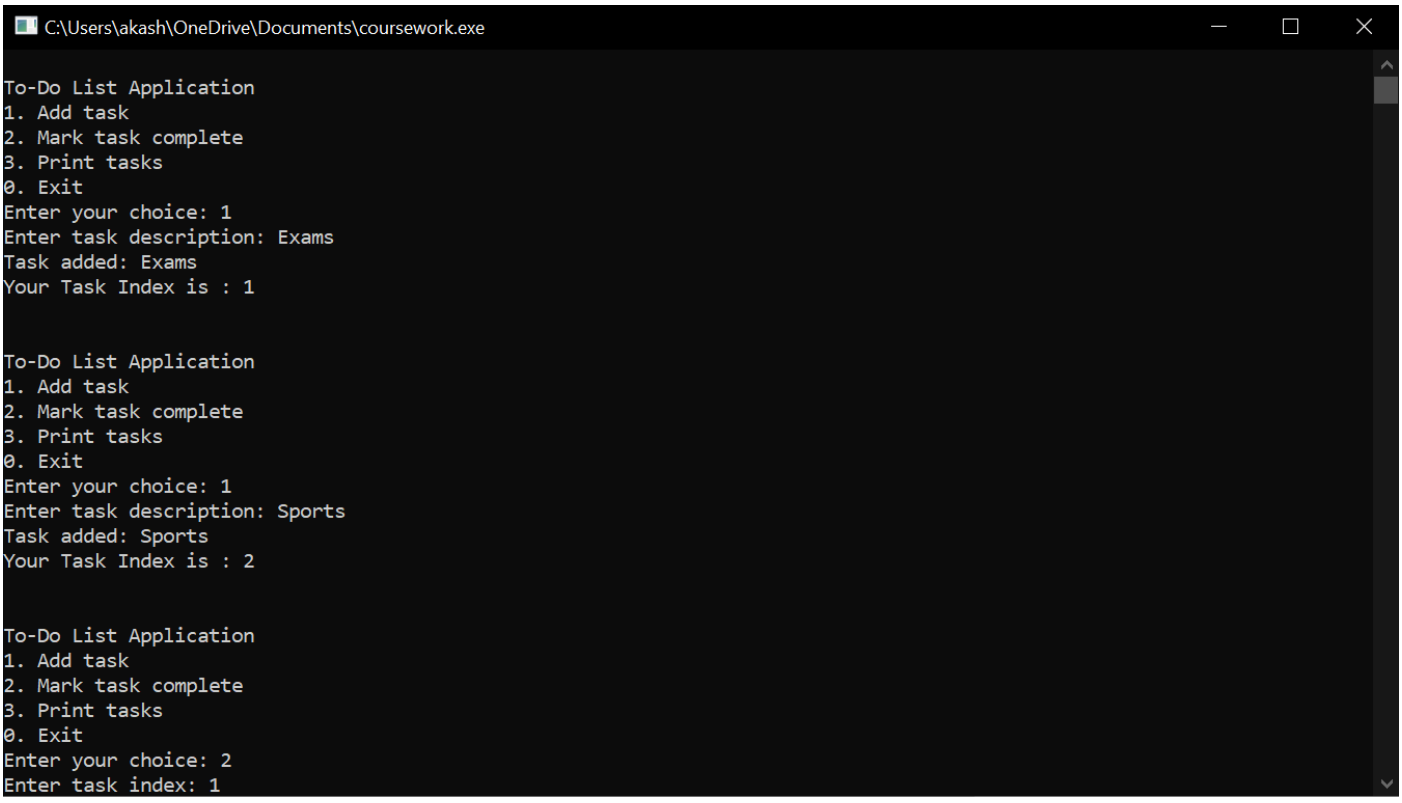
- Given the text file name in the location of source code.

C:\Users\akash\OneDrive\Documents\coursework.exe

```
To-Do List Application
1. Add task
2. Mark task complete
3. Print tasks
0. Exit
Enter your choice:
```

- First you choce number
- After you enter task description
- Next you can see your Task and Task index number





Improvements:

The To-Do List application code is a simple and functional implementation, but there are potential improvements that can enhance its robustness, usability, and features. Some suggestions include improving input validation, dynamic memory allocation, task editing and deletion, error handling, file I/O, user interface improvement, task sorting, command-line arguments, undo/redo functionality, data persistence, user authentication, and GUI implementation. Input validation should handle erroneous or unexpected user inputs, such as valid menu options and task index validation. Dynamic memory allocation can handle an arbitrary number of tasks efficiently, allowing the application to handle a more extensive list without being constrained by a predefined limit (`MAX_TASKS`). Task editing and deletion should be added to allow users more flexibility in managing their tasks. Error handling mechanisms should be improved to provide informative messages when something goes wrong. File I/O should be enabled to save the current task list to a file and load it in subsequent executions. User interface improvements should include more user-friendly messages and clearer instructions, including color-coding or icons to indicate task completion status. Task sorting options should be implemented based on different criteria to make it easier for users to organize their tasks. Command-line arguments

should be added to load task lists from external files at program startup or specify options directly from the command line. Undo/redo functionality should be introduced to revert completed tasks back to their previous state and vice versa. Data persistence features should be added, such as saving the task list to a database or cloud storage for backup and synchronization across devices. User authentication should be implemented for multi-user scenarios and task lists associated with specific user accounts. In conclusion, the scope of improvements to the To-Do List application depends on its intended use and target audience. Carefully considering user needs and requirements when deciding which improvements to implement is crucial for a successful implementation.

Uses:

The To-Do List application code offers various practical uses in both personal and professional settings. It can be used for personal task management, project management, study and academic planning, workplace task organization, shopping lists, event planning, goal tracking, daily routine management, team collaboration, and time management and productivity boost. It helps individuals organize and manage their daily tasks, chores, appointments, and reminders, ensuring no important tasks are overlooked. In a professional context, project managers and team members can create task lists, allocate responsibilities, and monitor task completion. Students can create study plans, track assignment deadlines, and manage academic tasks efficiently, aiding in time management and reducing academic stress. Workplace task organization helps employees prioritize assignments and ensure timely completion of tasks, leading to increased productivity. Shopping lists help users remember necessary items for shopping trips. Event planning helps manage event preparation tasks, while goal tracking helps individuals set and track personal or professional goals. Daily routine management helps establish and manage daily routines, while team collaboration allows team members to share and update task lists, keeping everyone informed of progress and status. In summary, the To-Do List application code is a versatile tool for improving task management and

organization in various aspects of life, promoting better efficiency, time utilization, and task prioritization. It can be adapted and customized to cater to specific needs and preferences, making it a valuable resource for individuals and teams seeking better task management solutions.

Conclusion:

The To-Do List application code offers a user-friendly, efficient solution for managing tasks. Its modular structure allows for easy maintenance and potential future enhancements. The application is useful for personal task management, project tracking, and academic planning. It empowers individuals and teams to stay organized, prioritize tasks, and ensure no important tasks are overlooked. By using the application, users can increase productivity, reduce missed deadlines, and reduce the chances of forgetting essential tasks. However, the current implementation offers a solid foundation for improvement, with potential enhancements like input validation, dynamic memory allocation, task editing, data persistence, and a graphical user interface. Overall, the To-Do List application code is a valuable starting point for streamlining task management and offers potential for customization and enhancement.

Thank you.