Task Manager

C Project

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

1. **Include Statements:**

    - This section includes three standard C libraries: **stdio.h** for input/output operations, **stdlib.h** for memory allocation functions (**malloc**, **free**), and **string.h** for string manipulation functions (**strcpy**, etc.).

```
struct Task
{
    char title[100];
    char description[500];
    char dueDate[20];
    char status[20];
    struct Task *next;
};
```

2. **Task Structure Definition:**

    - Defines a structure named **Task** that represents a task in the task manager.
    - It has fields like **title**, **description**, **dueDate**, and **status** to store information about the task.
    - The **next** field is a pointer to another **Task**, creating a linked list structure.

```
struct Task* createTask()
{
   struct Task *newTask = (struct Task*)malloc(sizeof(struct Task));
   if (newTask == NULL) {
      printf("Memory allocation failed. Exiting.\n");
      exit(EXIT_FAILURE);
   }
   newTask->next = NULL;
   return newTask;
}
```

3. **Function to Create a Task:**
    - **createTask** function allocates memory for a new task using **malloc**.
    - It checks if the memory allocation was successful.
    - Initializes the **next** pointer to **NULL** and returns a pointer to the new task.

```
void addTask(struct Task **head)
{
    struct Task *newTask = createTask();
    printf("Enter task details:\n");
    printf("Title: ");
    scanf("%s", newTask->title);
    printf("Description: ");
    scanf("%s", newTask->description);
    printf("Due Date: ");
    scanf("%s", newTask->dueDate);
    printf("Status: ");
    scanf("%s", newTask->status);
    // Add the new task to the front of the list
    newTask->next = *head;
    *head = newTask;
    printf("Task added successfully!\n");
}
```

4. **Function to Add a Task:**

    - **addTask** function adds a new task to the linked list.
    - It calls **createTask** to get a new task.
    - Reads task details from the user using **scanf**.
    - Updates the **next** pointer of the new task to point to the current head.
    - Updates the head pointer to the new task.
    - Prints a success message.

```
void viewTasks(struct Task *head)
{
    printf("\nTask List:\n");
    int taskNumber = 1;
    while (head != NULL) {
        printf("%d. Title: %s\n", taskNumber, head->title);
        printf("   Description: %s\n", head->description);
        printf("   Due Date: %s\n", head->dueDate);
        printf("   Status: %s\n", head->status);
        head = head->next;
        taskNumber++;
    }
}
```

5. **Function to View Tasks:**

    - **viewTasks** function displays the list of tasks.

    - Iterates through the linked list, printing details of each task.

```
void markCompleted(struct Task *head)

{

    int taskNumber;

    printf("Enter the task number to mark as completed: ");

    scanf("%d", &taskNumber);

    // Traverse the list to find the specified task

    int i;

    for (i = 1; i < taskNumber && head != NULL; i++) {

        head = head->next;

    }

    if (head != NULL) {

        strcpy(head->status, "Completed");

        printf("Task marked as completed!\n");

    } else {

        printf("Task not found.\n");

    }

}
```

6. **Function to Mark a Task as Completed:**

    - **markCompleted** function marks a task as completed.
    - It prompts the user to enter the task number.
    - It traverses the linked list to find the specified task.
    - If the task is found, it updates the status to "Completed"; otherwise, it prints an error message.

```
void deleteTask(struct Task **head, int taskNumber)
{
    if (*head == NULL) {
        printf("Task list is empty. Cannot delete.\n");
        return;
    }
    struct Task *temp = *head;
    if (taskNumber == 1) {
        *head = (*head)->next;
        free(temp);
        printf("Task deleted successfully!\n");
        return;
    }
    // Traverse the list to find the task before the specified task
    int i;
    for (i = 1; i < taskNumber - 1 && temp->next != NULL; i++) {
        temp = temp->next;
    }
    if (temp->next != NULL) {
        struct Task *toDelete = temp->next;
        temp->next = toDelete->next;
        free(toDelete);
        printf("Task deleted successfully!\n");
    } else {
        printf("Task not found.\n");
    }
}
```

7. **Function to Delete a Task:**

   - **deleteTask** function deletes a task from the linked list.
   - It checks if the list is empty.
   - If the task to be deleted is the first one, it updates the head pointer.
   - Otherwise, it traverses the list to find the task before the specified task and adjusts the pointers.
   - If the task is found, it is deleted and memory is freed; otherwise, an error message is printed.

```
void freeTasks(struct Task *head)

{

    while (head != NULL) {

        struct Task *temp = head;

        head = head->next;

        free(temp);

    }

}
```

8. **Function to Free Memory:**

    - **freeTasks** function frees the memory allocated for each task in the linked list.
    - It iterates through the list, frees the memory for the current task, and moves on to the next one.

```
int main()
{
    struct Task *head = NULL;
    int choice;
    do {
        printf("\nTask Manager\n");
        printf("1. Add Task\n");
        printf("2. View Tasks\n");
        printf("3. Mark Task as Completed\n");
        printf("4. Delete Task\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addTask(&head);
                break;
            case 2:
                viewTasks(head);
                break;
            case 3:
                markCompleted(head);
                break;
            case 4: {
                int taskNumber;
                printf("Enter the task number to delete: ");
                scanf("%d", &taskNumber);
                deleteTask(&head, taskNumber);
                break;
            }
            case 5:
                freeTasks(head);
                printf("Exiting the Task Manager. Goodbye!\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);
    return 0;
}
```

9. **Main Function:**

    - The **main** function is the entry point of the program.
    - It initializes the head pointer to the linked list of tasks.
    - It uses a do-while loop to display a menu of options and process user input until the user chooses to exit (choice 5).
    - Inside the loop, a switch statement handles various user choices, calling the corresponding functions.