# \*\*Task Manager Program - Comprehensive Report\*\*

This report provides a detailed analysis of the C# Task Manager program, including its structure, components, functions, and their purposes.

---

## \*\*1. Program Overview\*\*

The Task Manager is a console application that helps users:

- Create and manage tasks

- Organize tasks by priority and due date

- Track completed tasks

- Handle high-priority tasks separately

---

## \*\*2. Data Structures & Classes\*\*

### \*\*A. TaskItem Class\*\*

Represents a single task with these properties:

- `Title` (string): Task name/description

- `Importance` (int): Priority level (1-3, where 3 is highest)

- `DueDate` (DateTime): Task deadline

### \*\*B. FinishedTaskNode Class\*\*

Used for storing completed tasks in a \*\*linked list\*\*:

- `Item` (TaskItem): The completed task

- `Next` (FinishedTaskNode): Reference to the next completed task

### \*\*C. Main Data Containers\*\*

1. \*\*`tasks` array\*\* (TaskItem[]): Stores active tasks (max 100)

2. \*\*`completedTasks` linked list\*\* (FinishedTaskNode): Tracks finished tasks

3. \*\*`priorityQueue`\*\* (Queue<TaskItem>): Handles urgent tasks (FIFO)

---

## \*\*3. Core Functions\*\*

### \*\*A. Main Menu Functions\*\*

| Function | Description |

|----------|-------------|

| `DisplayMenu()` | Shows the interactive menu |

| `Main()` | Program entry point and main loop |

### \*\*B. Task Management Functions\*\*

| Function | Description | Key Operations |

|----------|-------------|----------------|

| `CreateNewTask()` | Adds a new task | - User input<br>- Array insertion |

| `DisplayAllTasks()` | Lists all active tasks | - Array iteration |

| `RemoveTask()` | Deletes a task | - Array shifting |

| `MarkAsComplete()` | Moves task to completed list | - Array removal<br>- Linked list insertion |

### \*\*C. Sorting Functions\*\*

| Function | Description | Algorithm |

|----------|-------------|-----------|

| `OrganizeByPriority()` | Sorts tasks by importance | Bubble Sort |

| `OrganizeByDate()` | Sorts tasks by due date | \*\*QuickSort\*\* (recursive) |

| `DateQuickSort()` | Helper for date sorting | Partitioning |

| `DatePartition()` | QuickSort pivot handling | Swapping |

### \*\*D. Special Features\*\*

| Function | Description | Data Structure |

|----------|-------------|----------------|

| `AddHighPriorityTask()` | Adds urgent tasks | \*\*Queue\*\* (FIFO) |

| `DisplayHighPriorityTasks()` | Shows urgent tasks | Queue iteration |

| `ShowFinishedTasks()` | Displays completed tasks | Linked list traversal |

---

## \*\*4. Key Algorithms & Techniques\*\*

1. \*\*Bubble Sort\*\*

- Used in `OrganizeByPriority()` for simple priority-based sorting.

2. \*\*QuickSort\*\*

- Efficient O(n log n) sorting in `OrganizeByDate()`.

3. \*\*Linked List Operations\*\*

- LIFO insertion in `MarkAsComplete()` (like a stack).

4. \*\*Queue Operations\*\*

- `Enqueue()`/iteration for handling urgent tasks.

---

## \*\*5. Error Handling\*\*

- Basic validation for:

- Task limit (`MAXIMUM\_TASKS`)

- Array bounds checking

- User input parsing

\*\*Improvement Opportunity\*\*: Add `try-catch` for robust input handling.

---

## \*\*6. Potential Enhancements\*\*

1. \*\*Data Persistence\*\*

- Save tasks to a file/database.

2. \*\*Search Functionality\*\*

- Filter tasks by name/date.

3. \*\*Priority Queue\*\*

- Proper heap-based implementation for urgent tasks.

4. \*\*User Interface\*\*

- Console colors or GUI version.

---

## \*\*Conclusion\*\*

This program demonstrates:

✅ Core C# concepts (OOP, arrays, collections)

✅ Sorting algorithms (BubbleSort, QuickSort)

✅ Data structures (Linked List, Queue)

✅ Basic task management logic

It serves as a solid foundation for expansion into a more sophisticated productivity tool.

using System;

using System.Collections.Generic;

public class TaskItem

{

public string Title { get; set; }

public int Importance { get; set; }

public DateTime DueDate { get; set; }

}

public class FinishedTaskNode

{

public TaskItem Item { get; set; }

public FinishedTaskNode NextItem { get; set; }

}

class TaskManager

{

private const int MAXIMUM\_TASKS = 100;

private static TaskItem[] taskList = new TaskItem[MAXIMUM\_TASKS];

private static int currentTaskCount = 0;

private static FinishedTaskNode finishedTasksHead = null;

private static Queue<TaskItem> highPriorityTasks = new Queue<TaskItem>();

static void Main(string[] args)

{

bool isRunning = true;

while (isRunning)

{

DisplayMenu();

string choice = Console.ReadLine();

switch (choice)

{

case "1":

CreateNewTask();

break;

case "2":

DisplayAllTasks();

break;

case "3":

RemoveTask();

break;

case "4":

OrganizeByPriority();

break;

case "5":

OrganizeByDate();

break;

case "6":

MarkAsComplete();

break;

case "7":

ShowFinishedTasks();

break;

case "8":

AddHighPriorityTask();

break;

case "9":

DisplayHighPriorityTasks();

break;

case "10":

isRunning = false;

break;

default:

Console.WriteLine("Invalid selection");

break;

}

}

}

static void DisplayMenu()

{

Console.WriteLine("\nTask Manager Menu:");

Console.WriteLine("1. Create New Task");

Console.WriteLine("2. View All Tasks");

Console.WriteLine("3. Remove Task");

Console.WriteLine("4. Sort by Importance");

Console.WriteLine("5. Sort by Due Date");

Console.WriteLine("6. Complete Task");

Console.WriteLine("7. View Completed Tasks");

Console.WriteLine("8. Add High Priority Task");

Console.WriteLine("9. View High Priority Tasks");

Console.WriteLine("10. Exit");

Console.Write("Please select an option: ");

}

static void CreateNewTask()

{

if (currentTaskCount >= MAXIMUM\_TASKS)

{

Console.WriteLine("Task limit reached!");

return;

}

TaskItem newItem = new TaskItem();

Console.Write("Task Title: ");

newItem.Title = Console.ReadLine();

Console.Write("Importance Level (1-3): ");

newItem.Importance = int.Parse(Console.ReadLine());

Console.Write("Due Date (dd/mm/yyyy): ");

newItem.DueDate = DateTime.Parse(Console.ReadLine());

taskList[currentTaskCount] = newItem;

currentTaskCount++;

Console.WriteLine("Task added successfully!");

}

static void DisplayAllTasks()

{

for (int i = 0; i < currentTaskCount; i++)

{

Console.WriteLine($"{i + 1}. {taskList[i].Title} | Importance: {taskList[i].Importance} | Due: {taskList[i].DueDate:d}");

}

}

static void RemoveTask()

{

DisplayAllTasks();

Console.Write("Enter task number to remove: ");

int selected = int.Parse(Console.ReadLine()) - 1;

if (selected < 0 || selected >= currentTaskCount)

{

Console.WriteLine("Invalid selection");

return;

}

for (int i = selected; i < currentTaskCount - 1; i++)

{

taskList[i] = taskList[i + 1];

}

currentTaskCount--;

Console.WriteLine("Task removed successfully!");

}

static void OrganizeByPriority()

{

for (int i = 0; i < currentTaskCount - 1; i++)

{

for (int j = 0; j < currentTaskCount - i - 1; j++)

{

if (taskList[j].Importance > taskList[j + 1].Importance)

{

TaskItem temp = taskList[j];

taskList[j] = taskList[j + 1];

taskList[j + 1] = temp;

}

}

}

Console.WriteLine("Tasks sorted by importance!");

}

static void OrganizeByDate()

{

DateQuickSort(0, currentTaskCount - 1);

Console.WriteLine("Tasks sorted by due date!");

}

static void DateQuickSort(int start, int end)

{

if (start < end)

{

int partitionIndex = DatePartition(start, end);

DateQuickSort(start, partitionIndex - 1);

DateQuickSort(partitionIndex + 1, end);

}

}

static int DatePartition(int start, int end)

{

DateTime pivotDate = taskList[end].DueDate;

int i = start - 1;

for (int j = start; j < end; j++)

{

if (taskList[j].DueDate < pivotDate)

{

i++;

TaskItem temp = taskList[i];

taskList[i] = taskList[j];

taskList[j] = temp;

}

}

TaskItem temp2 = taskList[i + 1];

taskList[i + 1] = taskList[end];

taskList[end] = temp2;

return i + 1;

}

static void MarkAsComplete()

{

DisplayAllTasks();

Console.Write("Enter task number to complete: ");

int selected = int.Parse(Console.ReadLine()) - 1;

if (selected < 0 || selected >= currentTaskCount)

{

Console.WriteLine("Invalid selection");

return;

}

TaskItem completed = taskList[selected];

for (int i = selected; i < currentTaskCount - 1; i++)

{

taskList[i] = taskList[i + 1];

}

currentTaskCount--;

FinishedTaskNode newNode = new FinishedTaskNode();

newNode.Item = completed;

newNode.NextItem = finishedTasksHead;

finishedTasksHead = newNode;

Console.WriteLine("Task marked as complete!");

}

static void ShowFinishedTasks()

{

FinishedTaskNode current = finishedTasksHead;

int counter = 1;

while (current != null)

{

Console.WriteLine($"{counter}. {current.Item.Title} | Importance: {current.Item.Importance} | Due: {current.Item.DueDate:d}");

current = current.NextItem;

counter++;

}

}

static void AddHighPriorityTask()

{

TaskItem urgentItem = new TaskItem();

Console.Write("High Priority Task Title: ");

urgentItem.Title = Console.ReadLine();

Console.Write("Importance Level (1-3): ");

urgentItem.Importance = int.Parse(Console.ReadLine());

Console.Write("Due Date (dd/mm/yyyy): ");

urgentItem.DueDate = DateTime.Parse(Console.ReadLine());

highPriorityTasks.Enqueue(urgentItem);

Console.WriteLine("High priority task added!");

}

static void DisplayHighPriorityTasks()

{

int position = 1;

foreach (TaskItem item in highPriorityTasks)

{

Console.WriteLine($"{position}. {item.Title} | Importance: {item.Importance} | Due: {item.DueDate:d}");

position++;

}

}

}