**NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY**

**EMPLOYEE SHIFT SCHEDULER**

**COURSE:** DATA STRUCTURES

**DEPARTMENT:** COMPUTER SCIENCE

**SECTION:** E

**YEAR:** 1

**SEMESTER:** 1

**DATE:** MAY 01 2025

**SUBMITTED TO:** ASHOK SELVA KUMAR E

**SUBMITTED BY:**

1. G. HANEESH1 11(24KB1A05F6)11
2. G. BHANU PRAKASH 1(24KB1A05K2)11
3. VENKATA YASWANTH 1 1(24KB1A05Q2)1
4. K. JAGAN (24KB1A05U0)1

# Acknowledgments

I would like to express my sincere gratitude to my mentor, Prof. A. Sharma, for their continuous support and guidance throughout the project. I also thank my classmates and family for their encouragement and motivation.

**Introduction**

The Employee Shift Scheduler is developed to streamline the task of shift planning in small organizations. The idea came from observing manual and inconsistent shift allocation practices, which led to miscommunication and inefficiencies.

Abstract

This project is a console-based Employee Shift Scheduler that allows a manager to assign and view weekly shifts for up to 10 employees. It supports shift allocation, day-wise and week-wise display, and saving schedules to a file.

# Objectives

1. To allow shift assignment for each day of the week.  
2. To display shift schedules per employee or per day.  
3. To store shift data for later use.  
4. To ensure easy and quick access to weekly shift plans.

System Requirements

Software: C Compiler (e.g., GCC), Text Editor (e.g., VS Code)  
Hardware: Minimum 2GB RAM, 1GHz Processor

Methodology

1. Define structures and constants.  
2. Implement functions for adding employees and assigning shifts.  
3. Display shifts by day or employee.  
4. Save the schedule to a file.  
5. Test with various inputs and scenarios.

# Project Description

Problem Statement:  
Manual shift planning is error-prone and inefficient.  
  
Proposed Solution:  
A C program to manage weekly shift schedules for up to 10 employees.  
  
Key Features:  
- Assign daily shifts  
- View schedules per day or employee  
- Save schedules to file

Algorithm

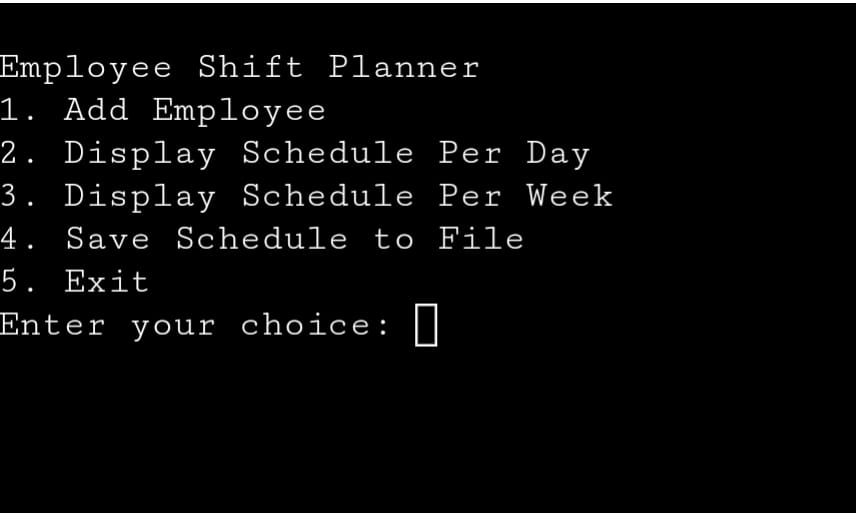
1. Start  
2. Display menu  
3. Based on user choice:  
 - Add employee and assign shifts  
 - Display schedule by day or by employee  
 - Save schedule to file  
4. Repeat until user exits  
5. End

Program Code

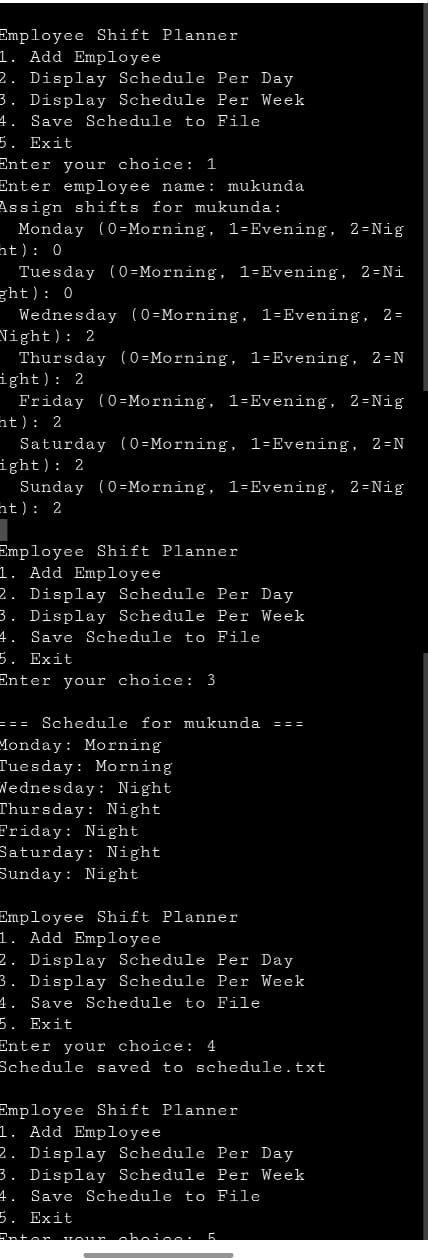
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#define MAX\_EMPLOYEES 10  
#define MAX\_NAME\_LEN 50  
#define DAYS\_IN\_WEEK 7  
  
const char \*days[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};  
const char \*shifts[] = {"Morning", "Evening", "Night"};  
  
typedef struct {  
 char name[MAX\_NAME\_LEN];  
 int shift[DAYS\_IN\_WEEK];  
} Employee;  
  
Employee employees[MAX\_EMPLOYEES];  
int employee\_count = 0;  
  
void addEmployee() {  
 if (employee\_count >= MAX\_EMPLOYEES) {  
 printf("Max employees reached.\n");  
 return;  
 }  
 printf("Enter employee name: ");  
 scanf(" %[^  
]", employees[employee\_count].name);  
 printf("Assign shifts for %s:\n", employees[employee\_count].name);  
 for (int i = 0; i < DAYS\_IN\_WEEK; i++) {  
 int shift\_choice;  
 printf(" %s (0=Morning, 1=Evening, 2=Night): ", days[i]);  
 scanf("%d", &shift\_choice);  
 if (shift\_choice >= 0 && shift\_choice <= 2)  
 employees[employee\_count].shift[i] = shift\_choice;  
 else {  
 printf("Invalid shift. Defaulting to Morning.\n");  
 employees[employee\_count].shift[i] = 0;  
 }  
 }  
 employee\_count++;  
}  
  
void displayPerDay() {  
 for (int day = 0; day < DAYS\_IN\_WEEK; day++) {  
 printf("\n--- %s ---\n", days[day]);  
 for (int i = 0; i < employee\_count; i++) {  
 printf("%s: %s\n", employees[i].name, shifts[employees[i].shift[day]]);  
 }  
 }  
}  
  
void displayPerWeek() {  
 for (int i = 0; i < employee\_count; i++) {  
 printf("\n=== Schedule for %s ===\n", employees[i].name);  
 for (int day = 0; day < DAYS\_IN\_WEEK; day++) {  
 printf("%s: %s\n", days[day], shifts[employees[i].shift[day]]);  
 }  
 }  
}  
  
void saveToFile(const char \*filename) {  
 FILE \*file = fopen(filename, "w");  
 if (!file) {  
 printf("Error opening file.\n");  
 return;  
 }  
 for (int i = 0; i < employee\_count; i++) {  
 fprintf(file, "%s\n", employees[i].name);  
 for (int day = 0; day < DAYS\_IN\_WEEK; day++) {  
 fprintf(file, "%s: %s\n", days[day], shifts[employees[i].shift[day]]);  
 }  
 fprintf(file, "\n");  
 }  
 fclose(file);  
 printf("Schedule saved to %s\n", filename);  
}  
  
int main() {  
 int choice;  
 while (1) {  
 printf("\nEmployee Shift Planner\n");  
 printf("1. Add Employee\n");  
 printf("2. Display Schedule Per Day\n");  
 printf("3. Display Schedule Per Week\n");  
 printf("4. Save Schedule to File\n");  
 printf("5. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
 switch (choice) {  
 case 1:  
 addEmployee();  
 break;  
 case 2:  
 displayPerDay();  
 break;  
 case 3:  
 displayPerWeek();  
 break;  
 case 4:  
 saveToFile("schedule.txt");  
 break;  
 case 5:  
 exit(0);  
 default:  
 printf("Invalid choice.\n");  
 }  
 }  
 return 0;  
}

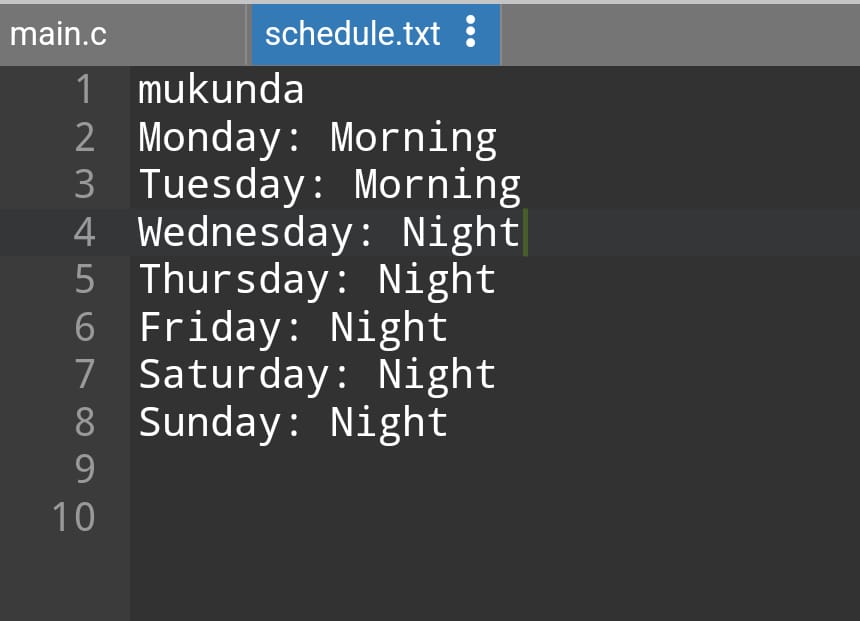
**Output Screenshot**

Below is a sample output generated by the program during execution:



# Testing and Validation





# Limitations

1. Limited to 10 employees.  
2. No data persistence between program runs (except text file output).  
3. No GUI interface.

**Future Enhancements**

1. Increase employee limit.  
2. Add a graphical user interface.  
3. Include feature for editing and deleting entries.  
4. Integrate with databases for persistent storage.

**Conclusion**

This project provided hands-on experience with C programming, especially in using structures and file handling. It also emphasized modular development and user interaction design.

**References**

Here are some references that can help you understand and enhance the code:

1. C Programming - Arrays and Structures:

* Learn about arrays and structures in C programming to better understand how data

is handled, including how structures like Student are used.

2. Sorting Algorithms (Bubble Sort):

* The bubble sort algorithm is used in this code to sort the students by their scores.

You can explore more about bubble sort and other sorting algorithms to optimize the

program.

3. C Programming Input and Output:

* The program uses basic I/O functions such as scanf and printf. Learning about I/O

operations in C will help you understand how to handle user input and output in

your program.

4. C Memory Management (Static vs Dynamic Memory):

* The program uses static memory allocation with the MAX constant. To further

enhance it, you could switch to dynamic memory allocation (e.g., using malloc) to

allow for flexible handling of different numbers of students.

1. Improving sorting efficiency

* Bubble sort is not the most efficient sorting algorithm. You can replace it with more

efficient algorithms like quicksort or merge sort.

6.C Error Handling and Input Validation:

* Error handling and input validation are important in any program to avoid

unexpected behavior. You can improve the program by adding error checks for user

input.