#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_TASKS 100

typedef struct {

    int id;

    char name[50];

    int priority; // Lower number means higher priority

    int deadline; // Deadline in days

} Task;

typedef struct {

    Task tasks[MAX\_TASKS];

    int size;

} PriorityQueue;

// Function prototypes

void initQueue(PriorityQueue \*pq);

void insertTask(PriorityQueue \*pq, Task task);

Task extractMin(PriorityQueue \*pq);

void printTasks(PriorityQueue \*pq);

void scheduleTasks(PriorityQueue \*pq);

void addDependency(int graph[MAX\_TASKS][MAX\_TASKS], int task1, int task2);

void printDependencies(int graph[MAX\_TASKS][MAX\_TASKS], int numTasks);

int main() {

    PriorityQueue pq;

    initQueue(&pq);

    // Sample tasks

    Task task1 = {1, "Task 1", 1, 5};

    Task task2 = {2, "Task 2", 2, 3};

    Task task3 = {3, "Task 3", 1, 2};

    Task task4 = {4, "Task 4", 3, 1};

    insertTask(&pq, task1);

    insertTask(&pq, task2);

    insertTask(&pq, task3);

    insertTask(&pq, task4);

    printf("Tasks in priority queue:\n");

    printTasks(&pq);

    printf("\nScheduled Tasks:\n");

    scheduleTasks(&pq);

    // Example of task dependencies

    int graph[MAX\_TASKS][MAX\_TASKS] = {0};

    addDependency(graph, 1, 2); // Task 1 must be completed before Task 2

    addDependency(graph, 1, 3); // Task 1 must be completed before Task 3

    printf("\nTask Dependencies:\n");

    printDependencies(graph, 4);

    return 0;

}

// Initialize the priority queue

void initQueue(PriorityQueue \*pq) {

    pq->size = 0;

}

// Insert a task into the priority queue

void insertTask(PriorityQueue \*pq, Task task) {

    if (pq->size < MAX\_TASKS) {

        pq->tasks[pq->size++] = task;

        // Simple bubble up to maintain heap property

        for (int i = pq->size - 1; i > 0; i--) {

            if (pq->tasks[i].priority < pq->tasks[(i - 1) / 2].priority) {

                Task temp = pq->tasks[i];

                pq->tasks[i] = pq->tasks[(i - 1) / 2];

                pq->tasks[(i - 1) / 2] = temp;

            } else {

                break;

            }

        }

    } else {

        printf("Priority Queue is full!\n");

    }

}

// Extract the task with the highest priority (lowest number)

Task extractMin(PriorityQueue \*pq) {

    if (pq->size == 0) {

        printf("Priority Queue is empty!\n");

        exit(EXIT\_FAILURE);

    }

    Task minTask = pq->tasks[0];

    pq->tasks[0] = pq->tasks[--pq->size];

    // Bubble down to maintain heap property

    int i = 0;

    while (i < pq->size / 2) {

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        int smallest = left;

        if (right < pq->size && pq->tasks[right].priority < pq->tasks[left].priority) {

            smallest = right;

        }

        if (pq->tasks[i].priority <= pq->tasks[smallest].priority) {

            break;

        }

        Task temp = pq->tasks[i];

        pq->tasks[i] = pq->tasks[smallest];

        pq->tasks[smallest] = temp;

        i = smallest;

    }

    return minTask;

}

// Print all tasks in the priority queue

void printTasks(PriorityQueue \*pq) {

    for (int i = 0; i < pq->size; i++) {

        printf("ID: %d, Name: %s, Priority: %d, Deadline: %d\n",

               pq->tasks[i].id, pq->tasks[i].name, pq->tasks[i].priority, pq->tasks[i].deadline);

    }

}

// Schedule tasks based on priority and deadline

void scheduleTasks(PriorityQueue \*pq) {

    while (pq->size > 0) {

        Task task = extractMin(pq);

        printf("Executing Task: ID: %d, Name: %s, Priority: %d, Deadline: %d\n",

               task.id, task.name, task.priority, task.deadline);

    }

}

// Add a dependency between two tasks in the graph

void addDependency(int graph[MAX\_TASKS][MAX\_TASKS], int task1, int task2) {

    graph[task1 - 1][task2 - 1] = 1; // Marking task1 as a prerequisite for task2

}

// Print task dependencies

void printDependencies(int graph[MAX\_TASKS][MAX\_TASKS], int numTasks) {

    for (int i = 0; i < numTasks; i++) {

        printf("Task %d dependencies: ", i + 1);

        for (int j = 0; j < numTasks; j++) {

            if (graph[i][j]) {

                printf("%d ", j + 1);

            }

        }

        printf("\n");

    }

}