HYDROELECTRIC DAM MANAGER

Name:P.Ritesh

Roll.No:ME24B1007

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Flow Request System (Queue)

char\* flowQueue[6];

int front = 0, rear = 0;

void enqueue(char\* component) {

if (rear < 6) {

flowQueue[rear++] = component;

printf("[Sensor] Flow request from %s enqueued!\n", component);

} else {

printf("[Warning] Flow queue is full!\n");

}

}

char\* dequeue() {

if (front < rear) {

return flowQueue[front++];

}

return NULL;

}

// Emergency Adjuster (Stack)

char\* emergencyStack[6];

int top = -1;

void push(char\* component) {

if (top < 5) {

emergencyStack[++top] = component;

printf("[Emergency] %s pushed to emergency stack!\n", component);

}

}

char\* pop() {

if (top >= 0) {

return emergencyStack[top--];

}

return NULL;

}

// Power Output Log (Array)

char\* powerLog[5];

int powerIndex = 0;

void logPower(char\* power) {

if (powerIndex < 5) {

powerLog[powerIndex++] = power;

} else {

// Simulate transmission of the oldest

printf("[Transmission] Oldest power data '%s' transmitted.\n", powerLog[0]);

for (int i = 1; i < 5; i++) {

powerLog[i - 1] = powerLog[i];

}

powerLog[4] = power;

}

printf("[Log] Power output '%s' logged.\n", power);

}

// Maintenance Tracker (Linked Lists)

typedef struct Component {

char name[20];

struct Component\* next;

struct Component\* prev;

} Component;

Component\* wornHead = NULL;

Component\* repairedHead = NULL;

void addWorn(char\* name) {

Component\* newComp = (Component\*)malloc(sizeof(Component));

strcpy(newComp->name, name);

newComp->next = wornHead;

newComp->prev = NULL;

wornHead = newComp;

printf("[Maintenance] Worn component '%s' added to check list.\n", name);

}

void repairComponent(char\* name) {

Component\* curr = wornHead;

Component\* prev = NULL;

while (curr) {

if (strcmp(curr->name, name) == 0) {

if (prev) prev->next = curr->next;

else wornHead = curr->next;

break;

}

prev = curr;

curr = curr->next;

}

curr->next = repairedHead;

curr->prev = NULL;

if (repairedHead) repairedHead->prev = curr;

repairedHead = curr;

printf("[Maintenance] '%s' repaired and moved to reviewed list.\n", name);

}

void traverseRepairedForward() {

Component\* temp = repairedHead;

printf("[Inspect] Repaired Components (Forward): ");

while (temp) {

printf("%s -> ", temp->name);

temp = temp->next;

}

printf("NULL\n");

}

void traverseRepairedBackward() {

Component\* temp = repairedHead;

while (temp && temp->next) temp = temp->next;

printf("[Inspect] Repaired Components (Backward): ");

while (temp) {

printf("%s -> ", temp->name);

temp = temp->prev;

}

printf("NULL\n");

}

// Priority Tuning (Circular Linked List)

Component\* priorityTail = NULL;

void addPriority(char\* name) {

Component\* newComp = (Component\*)malloc(sizeof(Component));

strcpy(newComp->name, name);

if (!priorityTail) {

newComp->next = newComp;

priorityTail = newComp;

} else {

newComp->next = priorityTail->next;

priorityTail->next = newComp;

priorityTail = newComp;

}

printf("[Tuning] Component '%s' added to high-priority cycle.\n", name);

}

void traversePriority(int cycles) {

if (!priorityTail) return;

Component\* temp = priorityTail->next;

printf("[Tuning] High-priority tuning cycle:\n");

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

printf(" Round %d: %s\n", i + 1, temp->name);

temp = temp->next;

}

}

int main() {

printf("\n--- Hydroelectric Dam Flow Manager ---\n\n");

// a) Flow and Emergency

enqueue("Turbine"); enqueue("Gate"); enqueue("Spillway");

enqueue("Reservoir"); enqueue("Pump"); enqueue("Valve");

for (int i = 0; i < 6; i++) {

char\* component = dequeue();

push(component);

}

printf("\n[Emergency Response Order - LIFO]:\n");

for (int i = 0; i < 6; i++) {

printf(" Responding to: %s\n", pop());

}

printf("\n> LIFO fits well here: The 'Valve' must be adjusted last to precisely fine-tune water pressure after bulk flow is managed.\n");

// b) Power Output Log

char\* powerOutputs[] = {"Pow1", "Pow2", "Pow3", "Pow4", "Pow5", "Pow6", "Pow7"};

for (int i = 0; i < 7; i++) {

logPower(powerOutputs[i]);

}

printf("\n> Transmission ensures no overflow and timely updates during high grid demand spikes.\n");

// c) Maintenance Tracker

addWorn("Turbine");

addWorn("Pump");

repairComponent("Turbine");

repairComponent("Pump");

traverseRepairedForward();

traverseRepairedBackward();

printf("\n> Turbine eroded by sediment? Repaired by ultrasonic bots polishing blade edges!\n");

// d) Priority Tuning

addPriority("Gate");

addPriority("Spillway");

traversePriority(4);

printf("\n> Spillway upgraded with smart sensors to self-adjust during storm surge.\n");

printf("\n--- Simulation Complete! Thanks for keeping the dam safe! ---\n");

return 0;

}

Sample Output:

