Assignment edsa

Name :suryaprabha

Roll no:me4i1021

Date :15/04/2025

1.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 6

#define LOG\_SIZE 5

// Queue and Stack

char\* flowQueue[SIZE], \*emergencyStack[SIZE];

int front = 0, rear = 0, top = -1;

// Log (no circular buffer now)

char\* powerLog[LOG\_SIZE];

int logCount = 0;

// Singly and Doubly Linked Lists

typedef struct SNode {

char\* name;

struct SNode\* next;

} SNode;

typedef struct DNode {

char\* name;

struct DNode \*prev, \*next;

} DNode;

SNode\* wornHead = NULL;

DNode \*repairedHead = NULL, \*repairedTail = NULL;

// Circular List

typedef struct CNode {

char\* name;

struct CNode\* next;

} CNode;

CNode\* circularHead = NULL;

// Task A: Queue → Stack → Print

void task\_a() {

char\* components[] = {"Turbine", "Gate", "Spillway", "Reservoir", "Pump", "Valve"};

for (int i = 0; i < SIZE; i++) flowQueue[rear++] = components[i];

while (front < rear) emergencyStack[++top] = flowQueue[front++];

while (top >= 0) printf("Adjusting: %s\n", emergencyStack[top--]);

}

// Task B: Log without circular buffer

void logPower(char\* power) {

if (logCount < LOG\_SIZE) {

powerLog[logCount++] = power;

} else {

printf("Log Full. Skipping: %s\n", power);

}

}

void task\_b() {

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

for (int i = 0; i < 7; i++) logPower(readings[i]);

printf("Current Power Log: ");

for (int i = 0; i < logCount; i++) printf("%s ", powerLog[i]);

printf("\n");

}

// Task C: Singly → Doubly Linked List

void insertWorn(char\* name) {

SNode\* node = malloc(sizeof(SNode));

node->name = name;

node->next = wornHead;

wornHead = node;

}

void moveToRepaired(char\* name) {

SNode \*curr = wornHead, \*prev = NULL;

while (curr && strcmp(curr->name, name) != 0) {

prev = curr;

curr = curr->next;

}

if (!curr) return;

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

else wornHead = curr->next;

free(curr);

DNode\* node = malloc(sizeof(DNode));

node->name = name;

node->prev = repairedTail;

node->next = NULL;

if (repairedTail) repairedTail->next = node;

else repairedHead = node;

repairedTail = node;

}

void task\_c() {

insertWorn("Turbine");

insertWorn("Pump");

moveToRepaired("Turbine");

printf("Forward: ");

for (DNode\* temp = repairedHead; temp; temp = temp->next) printf("%s ", temp->name);

printf("\nBackward: ");

for (DNode\* temp = repairedTail; temp; temp = temp->prev) printf("%s ", temp->name);

printf("\n");

}

// Task D: Circular Linked List

void insertCircular(char\* name) {

CNode\* node = malloc(sizeof(CNode));

node->name = name;

if (!circularHead) {

node->next = node;

circularHead = node;

} else {

node->next = circularHead->next;

circularHead->next = node;

circularHead = node;

}

}

void task\_d() {

insertCircular("Gate");

insertCircular("Spillway");

if (!circularHead) return;

CNode\* temp = circularHead->next;

int count = 0;

printf("Circular Traversal (2 rounds): ");

do {

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

temp = temp->next;

count++;

} while (temp != circularHead->next || count < 2);

printf("\n");

}

// Main

int main() {

printf("=== Task A ===\n"); task\_a();

printf("\n=== Task B ===\n"); task\_b();

printf("\n=== Task C ===\n"); task\_c();

printf("\n=== Task D ===\n"); task\_d();

return 0;

}

2. What Problem Are You Solving?

 Managing incoming water flow requests.

 Prioritizing emergency adjustments.

 Logging generated power efficiently.

 Tracking worn and repaired components.

 Handling high-priority components under constant monitoring.

3. **Design Explanation**

**Why These Data Structures?**

* **Queue (Flow Requests):** First-come, first-served processing of flow inputs from sensors.
* **Stack (Emergency Adjustments):** LIFO structure for urgent backtracking and prioritizing last-arrived critical events.
* **Array (Power Log):** Fixed-size, indexed logging of generated power, with oldest data sent when full.
* **Singly Linked List (Worn Components):** Efficient for sequential additions with minimal overhead.
* **Doubly Linked List (Repaired Components):** Allows review (forward/backward traversal) by technicians.
* **Circular Linked List (Urgent Tuning):** Continuous monitoring loop for priority parts.

**Efficiency Gains:**

* Real-time simulation.
* Logical flow mimicking real-world dam operations.
* Reduced time complexity for insertions and deletions in lists.
* Continuous monitoring with circular structure.

## 4.Logic of the Code (Step by Step)

**a) Flow and Emergency Adjustments**

1. **Enqueue** all 6 component requests into a queue.
2. **Dequeue** each one and **push** it onto a stack.
3. **Pop** each item from the stack to simulate emergency response order.

**b) Power Output Log (5-Slot Array)**

1. Insert first 5 power outputs (Pow1 to Pow5) into an array.
2. When adding Pow6 and Pow7, **transmit and replace** the oldest entries using a **circular buffer** approach.

**c) Worn and Repaired Components**

1. Add "Turbine" and "Pump" to a **singly linked list** (worn components).
2. Move "Turbine" to a **doubly linked list** after repair.
3. **Traverse forward and backward** to simulate review.

**d) Priority Tuning (Circular Linked List)**

1. Add "Gate" and "Spillway" to a circular linked list.
2. Traverse the list **twice** to simulate repeated checks.

5. Variables and Functions Used

| **Name** | **Type / Structure** |  | **Purpose** |
| --- | --- | --- | --- |
| flowQueue | Queue |  | Holds sensor-based flow adjustment requests. |
| emergencyStack | Stack |  | Stores emergency items in LIFO order. |
| powerLog[5] | Array |  | Circular buffer for latest 5 power logs. |
| start, end | Integers |  | Track front and rear positions in power log. |
| wornList | Singly Linked List |  | Tracks worn components. |
| repairedList | Doubly Linked List |  | Stores repaired components for review. |
| priorityList | Circular Linked List |  | Continuous loop for high-priority component tuning. |