Assignment edsa

Name :suryaprabha

Roll no:me4i1021

Date :15/04/2025

1.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Task a: Flow and Emergency (Queue and Stack)

#define MAX\_QUEUE 6

#define MAX\_STACK 6

typedef struct {

char\* items[MAX\_QUEUE];

int front, rear;

} Queue;

typedef struct {

char\* items[MAX\_STACK];

int top;

} Stack;

void initQueue(Queue\* q) {

q->front = 0;

q->rear = -1;

}

void enqueue(Queue\* q, char\* item) {

if (q->rear < MAX\_QUEUE - 1) {

q->items[++q->rear] = item;

}

}

char\* dequeue(Queue\* q) {

if (q->front > q->rear) return NULL;

return q->items[q->front++];

}

void initStack(Stack\* s) {

s->top = -1;

}

void push(Stack\* s, char\* item) {

if (s->top < MAX\_STACK - 1) {

s->items[++s->top] = item;

}

}

char\* pop(Stack\* s) {

if (s->top >= 0) {

return s->items[s->top--];

}

return NULL;

}

// Task b: Power Output Log (Array)

#define LOG\_SIZE 5

typedef struct {

char\* outputs[LOG\_SIZE];

int index;

} PowerLog;

void initPowerLog(PowerLog\* log) {

log->index = 0;

for (int i = 0; i < LOG\_SIZE; i++) {

log->outputs[i] = NULL;

}

}

void insertPower(PowerLog\* log, char\* output) {

if (log->index < LOG\_SIZE) {

log->outputs[log->index++] = output;

} else {

printf("Transmitting: %s\n", log->outputs[0]);

for (int i = 0; i < LOG\_SIZE - 1; i++) {

log->outputs[i] = log->outputs[i + 1];

}

log->outputs[LOG\_SIZE - 1] = output;

}

}

void printPowerLog(PowerLog\* log) {

printf("Current Log: [");

for (int i = 0; i < LOG\_SIZE; i++) {

printf("%s", log->outputs[i] ? log->outputs[i] : "null");

if (i < LOG\_SIZE - 1) printf(", ");

}

printf("]\n");

}

// Task c: Worn Component Tracker (Singly and Doubly Linked Lists)

typedef struct SinglyNode {

char\* data;

struct SinglyNode\* next;

} SinglyNode;

typedef struct DoublyNode {

char\* data;

struct DoublyNode\* prev;

struct DoublyNode\* next;

} DoublyNode;

typedef struct {

SinglyNode\* head;

} SinglyLinkedList;

typedef struct {

DoublyNode\* head;

} DoublyLinkedList;

void initSinglyList(SinglyLinkedList\* list) {

list->head = NULL;

}

void insertSingly(SinglyLinkedList\* list, char\* data) {

SinglyNode\* newNode = (SinglyNode\*)malloc(sizeof(SinglyNode));

newNode->data = data;

newNode->next = list->head;

list->head = newNode;

}

void deleteSingly(SinglyLinkedList\* list, char\* data) {

SinglyNode\* current = list->head;

SinglyNode\* prev = NULL;

if (current && strcmp(current->data, data) == 0) {

list->head = current->next;

free(current);

return;

}

while (current && strcmp(current->data, data) != 0) {

prev = current;

current = current->next;

}

if (current) {

prev->next = current->next;

free(current);

}

}

void initDoublyList(DoublyLinkedList\* list) {

list->head = NULL;

}

void insertDoubly(DoublyLinkedList\* list, char\* data) {

DoublyNode\* newNode = (DoublyNode\*)malloc(sizeof(DoublyNode));

newNode->data = data;

newNode->prev = NULL;

newNode->next = list->head;

if (list->head) {

list->head->prev = newNode;

}

list->head = newNode;

}

void traverseForward(DoublyLinkedList\* list) {

DoublyNode\* current = list->head;

printf("Forward: ");

while (current) {

printf("%s ", current->data);

current = current->next;

}

printf("\n");

}

void traverseBackward(DoublyLinkedList\* list) {

DoublyNode\* current = list->head;

if (!current) return;

while (current->next) {

current = current->next;

}

printf("Backward: ");

while (current) {

printf("%s ", current->data);

current = current->prev;

}

printf("\n");

}

// Task d: Priority Tuning (Circular Linked List)

typedef struct CircularNode {

char\* data;

struct CircularNode\* next;

} CircularNode;

typedef struct {

CircularNode\* head;

} CircularLinkedList;

void initCircularList(CircularLinkedList\* list) {

list->head = NULL;

}

void insertCircular(CircularLinkedList\* list, char\* data) {

CircularNode\* newNode = (CircularNode\*)malloc(sizeof(CircularNode));

newNode->data = data;

if (!list->head) {

list->head = newNode;

newNode->next = newNode;

} else {

CircularNode\* current = list->head;

while (current->next != list->head) {

current = current->next;

}

current->next = newNode;

newNode->next = list->head;

}

}

void traverseCircular(CircularLinkedList\* list, int count) {

if (!list->head) return;

CircularNode\* current = list->head;

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

CircularNode\* start = current;

do {

printf("%s ", current->data);

current = current->next;

} while (current != start);

}

printf("\n");

}

int main() {

// Task a: Flow and Emergency

printf("Task a: Flow and Emergency\n");

Queue q;

Stack s;

initQueue(&q);

initStack(&s);

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

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

enqueue(&q, requests[i]);

}

while (q.front <= q.rear) {

char\* item = dequeue(&q);

push(&s, item);

}

printf("Emergency Adjustment Order:\n");

while (s.top >= 0) {

printf("%s\n", pop(&s));

}

printf("\n");

// Task b: Power Output Log

printf("Task b: Power Output Log\n");

PowerLog log;

initPowerLog(&log);

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

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

insertPower(&log, outputs[i]);

printPowerLog(&log);

}

printf("\n");

// Task c: Worn Component Tracker

printf("Task c: Worn Component Tracker\n");

SinglyLinkedList singlyList;

DoublyLinkedList doublyList;

initSinglyList(&singlyList);

initDoublyList(&doublyList);

insertSingly(&singlyList, "Turbine");

insertSingly(&singlyList, "Pump");

deleteSingly(&singlyList, "Turbine");

insertDoubly(&doublyList, "Turbine");

traverseForward(&doublyList);

traverseBackward(&doublyList);

printf("\n");

// Task d: Priority Tuning

printf("Task d: Priority Tuning\n");

CircularLinkedList circularList;

initCircularList(&circularList);

insertCircular(&circularList, "Gate");

insertCircular(&circularList, "Spillway");

traverseCircular(&circularList, 2);

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. |