ELEMENTARY DATA STRUCTURES ASSIGNMENT 1

NAME : CH VAMSI PRANEETH

ROLL NO : ME24I1005

BATCH : 4

QUESTION NO : 5

CONCEPTS USED : ARRAYS,FOR LOOPS,QUEUE,STACK AND LINKED LISTS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 6

#define NAME\_SIZE 20

// QUEUE for Delivery Requests

typedef struct {

char items[MAX][NAME\_SIZE];

int front, rear;

} Queue;

void initQueue(Queue \*q) {

q->front = q->rear = -1;

}

int isQueueEmpty(Queue \*q) {

return q->front == -1 || q->front > q->rear;

}

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

if (q->rear == MAX - 1) {

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

return;

}

if (q->front == -1) q->front = 0;

q->rear++;

strcpy(q->items[q->rear], item);

}

char\* dequeue(Queue \*q) {

if (isQueueEmpty(q)) {

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

return NULL;

}

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

}

// STACK for Urgent Dispatch

typedef struct {

char items[MAX][NAME\_SIZE];

int top;

} Stack;

void initStack(Stack \*s) {

s->top = -1;

}

int isStackEmpty(Stack \*s) {

return s->top == -1;

}

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

if (s->top == MAX - 1) {

printf("Stack is full!\n");

return;

}

s->top++;

strcpy(s->items[s->top], item);

}

char\* pop(Stack \*s) {

if (isStackEmpty(s)) {

printf("Stack is empty!\n");

return NULL;

}

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

}

// ARRAY for Flight Log

char flightlog[MAX][NAME\_SIZE];

int logCount = 0;

void logDelivery(char item[]) {

if (logCount < MAX) {

strcpy(flightlog[logCount], item);

logCount++;

} else {

// Remove the oldest and shift left

for (int I = 1; I < MAX; I++) {

strcpy(flightlog[I - 1], flightlog[I]);

}

strcpy(flightlog[MAX - 1], item);

}

}

void displayFlightLog() {

printf("\nFlight Log:\n");

for (int I = 0; I < logCount; I++) {

printf("%s\n", flightlog[I]);

}

}

// SINGLY LINKED LIST for Overloaded Drones

typedef struct SLLNode {

char name[NAME\_SIZE];

struct SLLNode \*next;

} SLLNode;

SLLNode\* insertSLL(SLLNode \*head, char name[]) {

SLLNode newNode = (SLLNode)malloc(sizeof(SLLNode));

strcpy(newNode->name, name);

newNode->next = head;

return newNode;

}

SLLNode\* deleteSLL(SLLNode \*head, char name[]) {

SLLNode \*temp = head, \*prev = NULL;

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

prev = temp;

temp = temp->next;

}

if (!temp) return head;

if (!prev) head = temp->next;

else prev->next = temp->next;

free(temp);

return head;

}

// DOUBLY LINKED LIST for Serviced Drones

typedef struct DLLNode {

char name[NAME\_SIZE];

struct DLLNode \*prev, \*next;

} DLLNode;

DLLNode\* insertDLL(DLLNode \*head, char name[]) {

DLLNode newNode = (DLLNode)malloc(sizeof(DLLNode));

strcpy(newNode->name, name);

newNode->prev = NULL;

newNode->next = head;

if (head != NULL) head->prev = newNode;

return newNode;

}

void traverseDLL(DLLNode \*head) {

DLLNode \*temp = head;

printf("\nServiced Drones (Forward):\n");

while (temp != NULL) {

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

if (temp->next == NULL) break;

temp = temp->next;

}

printf("Serviced Drones (Backward):\n");

while (temp != NULL) {

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

temp = temp->prev;

}

}

// CIRCULAR LINKED LIST for Emergency Drones

typedef struct CLLNode {

char name[NAME\_SIZE];

struct CLLNode \*next;

} CLLNode;

CLLNode\* insertCLL(CLLNode \*tail, char name[]) {

CLLNode newNode = (CLLNode)malloc(sizeof(CLLNode));

strcpy(newNode->name, name);

if (tail == NULL) {

newNode->next = newNode;

return newNode;

}

newNode->next = tail->next;

tail->next = newNode;

return newNode;

}

void traverseCLL(CLLNode \*tail) {

if (tail == NULL) return;

CLLNode \*start = tail->next;

printf("\nEmergency Drones (Circular List x2):\n");

for (int round = 0; round < 2; round++) {

CLLNode \*temp = start;

do {

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

temp = temp->next;

} while (temp != start);

}

// MAIN FUNCTION

int main() {

Queue requestQueue;

Stack priorityStack;

SLLNode \*overloaded = NULL;

DLLNode \*serviced = NULL;

CLLNode \*emergency = NULL;

initQueue(&requestQueue);

initStack(&priorityStack);

// a) Request and Dispatch

printf("=== Delivery Requests ===\n");

char requests[][NAME\_SIZE] = {"Food", "Medicine", "Tools", "Water", "Parts", "Fuel"};

for (int I = 0; I < MAX; I++) {

enqueue(&requestQueue, requests[I]);

}

while (!isQueueEmpty(&requestQueue)) {

char \*req = dequeue(&requestQueue);

push(&priorityStack, req);

}

printf("\n=== Dispatch Order (LIFO) ===\n");

while (!isStackEmpty(&priorityStack)) {

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

}

// b) Flight Log

char deliveries[][NAME\_SIZE] = {"Del1", "Del2", "Del3", "Del4", "Del5", "Del6", "Del7", "Del8"};

for (int I = 0; I < 8; I++) {

logDelivery(deliveries[I]);

}

displayFlightLog();

// c) Overloaded and Serviced

printf("\n=== Overloaded Drones ===\n");

overloaded = insertSLL(overloaded, "Drone6");

overloaded = insertSLL(overloaded, "Drone3");

overloaded = deleteSLL(overloaded, "Drone3");

serviced = insertDLL(serviced, "Drone3");

traverseDLL(serviced);

// d) Emergency Rerouting

emergency = insertCLL(emergency, "Drone1");

emergency = insertCLL(emergency, "Drone4");

traverseCLL(emergency);

return 0;

}