//ME24B1041

//RATHOD JAISH

//Assignment-1

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 6

// Queue

char queue[SIZE][20] = {"Scanner", "Digger", "Lift", "Light", "Drone", "Cutter"};

int front = -1, rear = -1;

// Stack

char stack[SIZE][20];

int top = -1;

// Log (renamed from 'log' to avoid conflict with math.h function)

char missionLog[SIZE][20];

int logCount = 0;

//Linked List (Damaged robots)

typedef struct Node {

char name[20];

struct Node\* next;

struct Node\* prev;

} Node;

Node \*damagedHead = NULL;

Node \*repairedHead = NULL;

Node \*repairedTail = NULL;

// Circular Linked List

typedef struct CNode {

char name[20];

struct CNode\* next;

} CNode;

CNode\* ctail = NULL;

// Queue Functions

void enqueue(char task[]) {

if (rear == SIZE - 1) {

printf("full, Can't add task,: %s\n", task); //Overflow condition

return;

}

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

strcpy(queue[++rear], task);

}

char\* dequeue() {

if (front == -1 ) {

return NULL;

}

return queue[front++];

}

// Stack Functions

void push(char task[]) {

if (top == SIZE - 1) {

printf(" full. Can't push: %s\n", task); //Overflow condition

return;

}

strcpy(stack[++top], task);

}

void popStack() {

while (top != -1) {

printf("Urgent task being handled: %s\n", stack[top--]);

}

}

// completed missions into the array//

void logMission(char mission[]) {

if (logCount < SIZE) {

strcpy(missionLog[logCount++], mission);

} else {

printf("Log's full. Deleting the oldest mission: %s\n", missionLog[0]);

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

strcpy(missionLog[i - 1], missionLog[i]);

}

strcpy(missionLog[SIZE - 1], mission);

}

printf("Mission '%s' has been logged.\n", mission);

}

// Damaged to Repaired

void insertDamaged(char name[]) {

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

strcpy(newNode->name, name);

newNode->next = damagedHead;

damagedHead = newNode;

}

void moveToRepaired(char name[]) {

Node \*temp = damagedHead, \*prev = NULL;

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

prev = temp;

temp = temp->next;

}

if (!temp) return;

if (prev == NULL) damagedHead = temp->next;

else prev->next = temp->next;

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

strcpy(newNode->name, name);

newNode->next = NULL;

newNode->prev = repairedTail;

if (repairedTail) repairedTail->next = newNode;

else repairedHead = newNode;

repairedTail = newNode;

}

void traverseRepairedForward() {

Node\* temp = repairedHead;

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

void traverseRepairedBackward() {

Node\* temp = repairedTail;

while (temp) {

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

temp = temp->prev;

}

printf("\n");

}

// Circular Linked List

void insertCircular(char name[]) {

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

strcpy(newNode->name, name);

if (!ctail) {

newNode->next = newNode;

ctail = newNode;

} else {

newNode->next = ctail->next;

ctail->next = newNode;

ctail = newNode;

}

}

void traverseCircular(int rounds) {

if (!ctail) return;

CNode\* current = ctail->next;

for (int r = 0; r < rounds; r++) {

CNode\* start = current;

do {

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

current = current->next;

} while (current != start);

printf("(loop %d)\n", r + 1);

}

}

int main() {

int n;

// Enqueue predefined tasks//

printf("Let's add the 6 tasks to the queue:\n");

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

enqueue(queue[i]);

}

// Dequeue and push to stack//

while ((n = (front != -1 && front <= rear))) {

char\* task = dequeue();

if (task) push(task);

}

// Urgent Task Execution

printf("\n--- Urgent Task Execution ---\n");

popStack();

// Log predefined missions//

char missions[8][20] = {"Mis1", "Mis2", "Mis3", "Mis4", "Mis5", "Mis6", "Mis7", "Mis8"};

printf("\nLogging 8 missions:\n");

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

logMission(missions[i]);

}

// Insert predefined damaged robots//

char damagedRobots[2][20] = {"Digger", "Drone"};

printf("\nInserting damaged robots:\n");

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

insertDamaged(damagedRobots[i]);

}

// Move a robot to repaired list//

char repairedRobot[1][20] = {"Digger"};

printf("\nMoving repaired robot:\n");

moveToRepaired(repairedRobot[0]);

// Display repaired robots//

printf("\nRepaired Robots (Forward): ");

traverseRepairedForward();

printf("Repaired Robots (Backward): ");

traverseRepairedBackward();

// Insert predefined priority robots//

char priorityRobots[2][20] = {"Scanner", "Lift"};

printf("\nInserting priority robots for circular list:\n");

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

insertCircular(priorityRobots[i]);

}

// Redeployment traversal//

printf("\nHigh Priority Robot Redeployment:\n");

traverseCircular(2);

// Why LIFO fits for urgent tasks//

printf("LIFO works best for urgent tasks like 'Cutter' because the last task to be added is the one that needs immediate action (e.g., 'Cutter' to clear debris).\n");

// Reason for logging missions//

printf("\nLogging missions is essential as it helps track progress and survivor status. If a task is full, removing the oldest keeps the log updated.\n");

// Damage and repair example//

printf("\n'Digger' was damaged when its arm was crushed, but it was rebuilt and moved to the repaired list for inspection.\n");

// Redeployment tweak//

printf("\nFor the redeployment, 'Lift' is now equipped with a thermal camera to help locate trapped survivors more effectively.\n");

return 0;

}