/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## NAME : N.VAMSI

## ROLL NO : ME24I1032

## Date : 15 TH ,APRIL

## Purpose : Simulate rescue coordination with queue, stack, logs,

## and robot repair tracking (linked lists)

## Title : Earthquake Rescue Robot Coordinator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define slots 6

// Queue structure for mission queue

struct Queue {

char\* Mission;

struct Queue\* next;

};

struct Queue \*front = NULL, \*rear = NULL;

// Stack structure for urgent missions

struct Stack {

char\* Mission;

struct Stack\* next;

};

struct Stack \*top = NULL;

// Singly Linked List for damaged robots

struct Singlynode {

char\* name;

struct Singlynode\* next;

};

struct Singlynode\* damagehead = NULL;

// Doubly Linked List for repaired robots

struct Doublynode {

char\* name;

struct Doublynode\* next;

struct Doublynode\* prev;

};

struct Doublynode \*repairhead = NULL, \*repairtail = NULL;

// Circular Linked List for redeployment robots

struct Circularnode {

char\* name;

struct Circularnode\* next;

};

struct Circularnode \*circularhead = NULL, \*circulartail = NULL;

// Circular Log Array

char\* logarray[SLOTS];

int logindex = 0;

// Queue Operations

void enqueue(char\* missionName) {

struct Queue\* newNode = (struct Queue\*)malloc(sizeof(struct Queue));

newNode->Mission = strdup(missionName);

newNode->next = NULL;

if (rear == NULL) {

front = rear = newNode;

return;

}

rear->next = newNode;

rear = newNode;

}

char\* dequeue() {

if (front == NULL) {

printf("Queue is empty\n");

return NULL;

}

struct Queue\* temp = front;

char\* missionName = strdup(front->Mission);

front = front->next;

if (front == NULL) rear = NULL;

free(temp->Mission);

free(temp);

return missionName;

}

void display() {

struct Queue\* temp = front;

if (!temp) {

printf("Queue is empty\n");

return;

}

printf("Queue elements:\n");

while (temp) {

printf("%s <-> ", temp->Mission);

temp = temp->next;

}

printf("\n");

}

// Stack Operations

void push(char\* missionName) {

struct Stack\* newNode = (struct Stack\*)malloc(sizeof(struct Stack));

newNode->Mission = strdup(missionName);

newNode->next = top;

top = newNode;

}

char\* pop() {

if (!top) {

printf("Stack is empty\n");

return NULL;

}

struct Stack\* temp = top;

char\* missionName = strdup(temp->Mission);

top = top->next;

free(temp->Mission);

free(temp);

return missionName;

}

void displayStack() {

if (!top) {

printf("Stack is empty\n");

return;

}

struct Stack\* temp = top;

printf("Stack elements:\n");

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

// Mission Log Array

void logMission(char\* missionName) {

if (logarray[logindex]) {

printf("Overwriting: %s\n", logarray[logindex]);

free(logarray[logindex]);

}

logarray[logindex] = strdup(missionName);

printf("Logged: %s\n", missionName);

logindex = (logindex + 1) % SLOTS;

}

void displayLog() {

printf("Log of mission names:\n");

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

if (logarray[i]) {

printf("%s <-> ", logarray[i]);

}

}

printf("\n");

}

// Singly Linked List for Damaged Robots

void insertDamagedRobot(char\* robotName) {

struct Singlynode\* newNode = (struct Singlynode\*)malloc(sizeof(struct Singlynode));

newNode->name = strdup(robotName);

newNode->next = NULL;

if (!damagehead) {

damagehead = newNode;

return;

}

struct Singlynode\* temp = damagehead;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

char\* deleteDamagedRobot(char\* robotName) {

struct Singlynode \*temp = damagehead, \*prev = NULL;

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

prev = temp;

temp = temp->next;

}

if (!temp) {

printf("Robot not found\n");

return NULL;

}

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

else prev->next = temp->next;

char\* name = strdup(temp->name);

free(temp->name);

free(temp);

return name;

}

void displayDamaged() {

if (!damagehead) {

printf("No damaged robots\n");

return;

}

struct Singlynode\* temp = damagehead;

printf("Damaged robots:\n");

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

// Doubly Linked List for Repaired Robots

void insertRepairedRobot(char\* robotName) {

struct Doublynode\* newNode = (struct Doublynode\*)malloc(sizeof(struct Doublynode));

newNode->name = strdup(robotName);

newNode->next = NULL;

newNode->prev = repairtail;

if (repairtail) repairtail->next = newNode;

else repairhead = newNode;

repairtail = newNode;

}

void forwardTraversal() {

if (!repairhead) {

printf("No repaired robots\n");

return;

}

struct Doublynode\* temp = repairhead;

printf("Repaired robots (forward):\n");

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

void backwardTraversal() {

if (!repairtail) {

printf("No repaired robots\n");

return;

}

struct Doublynode\* temp = repairtail;

printf("Repaired robots (backward):\n");

while (temp) {

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

temp = temp->prev;

}

printf("\n");

}

// Circular Linked List for Redeployment

void redeployDamagedRobot(char\* robotName) {

struct Circularnode\* newNode = (struct Circularnode\*)malloc(sizeof(struct Circularnode));

newNode->name = strdup(robotName);

newNode->next = NULL;

if (!circularhead) {

circularhead = circulartail = newNode;

newNode->next = circularhead;

} else {

circulartail->next = newNode;

circulartail = newNode;

circulartail->next = circularhead;

}

}

void traverseCircular() {

if (!circularhead) {

printf("No urgent redeployment robots.\n");

return;

}

struct Circularnode\* temp = circularhead;

int count = 0;

printf("Urgent Redeployment Cycle (2x):\n");

do {

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

temp = temp->next;

count++;

} while (temp != circularhead && count < 2);

while (count < 4) {

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

temp = temp->next;

count++;

}

printf("Back to start\n");

}

void freeMemory() {

struct Circularnode\* temp = circularhead;

if (!temp) return;

struct Circularnode\* nextNode;

do {

nextNode = temp->next;

free(temp->name);

free(temp);

temp = nextNode;

} while (temp != circularhead);

}

// Main Function

int main() {

int choice, n, i, damagedCount, redeployCount;

char robotName[100], \*task, \*urgentTask, \*repairedRobot;

for (i = 0; i < SLOTS; i++) logarray[i] = NULL;

do {

printf("\n=========== Earthquake Rescue Coordinator ===========\n");

printf("1. Enqueue Missions\n2. Dequeue All and Push to Stack\n3. Display Urgency Missions (LIFO)\n");

printf("4. Log Missions\n5. Insert Damaged Robots\n6. Repair Robot\n7. Display Repaired Robots (Forward)\n");

printf("8. Display Repaired Robots (Backward)\n9. Urgent Redeployment\n10. Traverse Redeployment List\n11. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter number of missions to enqueue: ");

scanf("%d", &n);

for (i = 0; i < n; i++) {

printf("Enter mission name %d: ", i + 1);

scanf(" %[^\n]", robotName);

enqueue(robotName);

}

display();

break;

case 2:

while ((task = dequeue()) != NULL) {

push(task);

free(task);

}

break;

case 3:

while ((urgentTask = pop()) != NULL) {

printf("%s > ", urgentTask);

free(urgentTask);

}

printf("\n");

break;

case 4:

printf("Enter number of missions to log: ");

scanf("%d", &n);

for (i = 0; i < n; i++) {

printf("Enter mission name %d: ", i + 1);

scanf(" %[^\n]", robotName);

logMission(robotName);

}

displayLog();

break;

case 5:

printf("Enter number of damaged robots: ");

scanf("%d", &damagedCount);

for (i = 0; i < damagedCount; i++) {

printf("Enter damaged robot name %d: ", i + 1);

scanf("%s", robotName);

insertDamagedRobot(robotName);

}

displayDamaged();

break;

case 6:

printf("Enter robot name to repair: ");

scanf("%s", robotName);

repairedRobot = deleteDamagedRobot(robotName);

if (repairedRobot) {

insertRepairedRobot(repairedRobot);

free(repairedRobot);

}

break;

case 7:

forwardTraversal();

break;

case 8:

backwardTraversal();

break;

case 9:

printf("Enter number of urgent redeployments: ");

scanf("%d", &redeployCount);

for (i = 0; i < redeployCount; i++) {

printf("Enter robot name %d: ", i + 1);

scanf("%s", robotName);

redeployDamagedRobot(robotName);

}

break;

case 10:

traverseCircular();

break;

case 11:

printf("Exiting...\n");

break;

default:

printf("Invalid choice! Try again.\n");

}

} while (choice != 11);

// Free all dynamic memory

for (i = 0; i < SLOTS; i++) if (logarray[i]) free(logarray[i]);

struct Singlynode\* temp1 = damagehead;

while (temp1) {

struct Singlynode\* next = temp1->next;

free(temp1->name);

free(temp1);

temp1 = next;

}

struct Doublynode\* temp2 = repairhead;

while (temp2) {

struct Doublynode\* next = temp2->next;

free(temp2->name);

free(temp2);

temp2 = next;

}

freeMemory();

return 0;

}