C Program for Smart Irrigation System

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#define MAX\_QUEUE\_SIZE 6  
#define MAX\_STACK\_SIZE 6  
#define MAX\_USAGE\_LOG 5  
  
// Queue for irrigation requests  
typedef struct {  
 char\* items[MAX\_QUEUE\_SIZE];  
 int front, rear;  
} Queue;  
  
void initializeQueue(Queue\* q) {  
 q->front = q->rear = -1;  
}  
  
int isQueueEmpty(Queue\* q) {  
 return q->front == -1;  
}  
  
int isQueueFull(Queue\* q) {  
 return (q->rear + 1) % MAX\_QUEUE\_SIZE == q->front;  
}  
  
void enqueue(Queue\* q, char\* component) {  
 if (isQueueFull(q)) {  
 printf("Queue is full!\n");  
 return;  
 }  
 if (isQueueEmpty(q)) {  
 q->front = q->rear = 0;  
 } else {  
 q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;  
 }  
 q->items[q->rear] = component;  
}  
  
char\* dequeue(Queue\* q) {  
 if (isQueueEmpty(q)) return NULL;  
 char\* item = q->items[q->front];  
 if (q->front == q->rear) {  
 q->front = q->rear = -1;  
 } else {  
 q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;  
 }  
 return item;  
}  
  
// Stack for emergency watering  
typedef struct {  
 char\* items[MAX\_STACK\_SIZE];  
 int top;  
} Stack;  
  
void initializeStack(Stack\* s) {  
 s->top = -1;  
}  
  
int isStackEmpty(Stack\* s) {  
 return s->top == -1;  
}  
  
int isStackFull(Stack\* s) {  
 return s->top == MAX\_STACK\_SIZE - 1;  
}  
  
void push(Stack\* s, char\* component) {  
 if (isStackFull(s)) {  
 printf("Stack is full!\n");  
 return;  
 }  
 s->items[++s->top] = component;  
}  
  
char\* pop(Stack\* s) {  
 if (isStackEmpty(s)) return NULL;  
 return s->items[s->top--];  
}  
  
// Water usage log  
char\* waterUsageLog[MAX\_USAGE\_LOG];  
  
void logUsage(char\* usage, int index) {  
 waterUsageLog[index] = strdup(usage);  
}  
  
// Singly Linked List for leaky components  
typedef struct SinglyNode {  
 char\* component;  
 struct SinglyNode\* next;  
} SinglyNode;  
  
SinglyNode\* insertSingly(SinglyNode\* head, char\* component) {  
 SinglyNode\* node = (SinglyNode\*)malloc(sizeof(SinglyNode));  
 node->component = strdup(component);  
 node->next = head;  
 return node;  
}  
  
SinglyNode\* deleteSingly(SinglyNode\* head, char\* component) {  
 SinglyNode \*curr = head, \*prev = NULL;  
 while (curr) {  
 if (strcmp(curr->component, component) == 0) {  
 if (prev) prev->next = curr->next;  
 else head = curr->next;  
 free(curr->component);  
 free(curr);  
 break;  
 }  
 prev = curr;  
 curr = curr->next;  
 }  
 return head;  
}  
  
// Doubly Linked List for fixed components  
typedef struct DoublyNode {  
 char\* component;  
 struct DoublyNode\* next;  
 struct DoublyNode\* prev;  
} DoublyNode;  
  
DoublyNode\* insertDoubly(DoublyNode\* head, char\* component) {  
 DoublyNode\* node = (DoublyNode\*)malloc(sizeof(DoublyNode));  
 node->component = strdup(component);  
 node->next = head;  
 node->prev = NULL;  
 if (head) head->prev = node;  
 return node;  
}  
  
void traverseForward(DoublyNode\* head) {  
 DoublyNode\* temp = head;  
 while (temp) {  
 printf("Fixed Component (FWD): %s\n", temp->component);  
 temp = temp->next;  
 }  
}  
  
void traverseBackward(DoublyNode\* tail) {  
 DoublyNode\* temp = tail;  
 while (temp && temp->next) temp = temp->next;  
 while (temp) {  
 printf("Fixed Component (BWD): %s\n", temp->component);  
 temp = temp->prev;  
 }  
}  
  
// Circular Linked List for priority fixes  
typedef struct CircularNode {  
 char\* component;  
 struct CircularNode\* next;  
} CircularNode;  
  
CircularNode\* insertCircular(CircularNode\* tail, char\* component) {  
 CircularNode\* node = (CircularNode\*)malloc(sizeof(CircularNode));  
 node->component = strdup(component);  
 if (!tail) {  
 node->next = node;  
 return node;  
 } else {  
 node->next = tail->next;  
 tail->next = node;  
 return node;  
 }  
}  
  
void traverseCircular(CircularNode\* tail) {  
 if (!tail) return;  
 CircularNode\* current = tail->next;  
 do {  
 printf("Priority Fix: %s\n", current->component);  
 current = current->next;  
 } while (current != tail->next);  
}  
  
// Main function  
int main() {  
 Queue irrigationQueue;  
 Stack emergencyStack;  
 initializeQueue(&irrigationQueue);  
 initializeStack(&emergencyStack);  
  
 // a) Irrigation requests and emergency stack  
 char\* components[] = {"Pump", "Pipe", "Valve", "Sensor", "Sprinkler", "Filter"};  
 for (int i = 0; i < 6; i++) {  
 enqueue(&irrigationQueue, components[i]);  
 }  
  
 while (!isQueueEmpty(&irrigationQueue)) {  
 char\* comp = dequeue(&irrigationQueue);  
 push(&emergencyStack, comp);  
 }  
  
 while (!isStackEmpty(&emergencyStack)) {  
 char\* comp = pop(&emergencyStack);  
 printf("Watering component: %s\n", comp);  
 }  
  
 printf("\n");  
  
 // b) Water Usage Log  
 for (int i = 0; i < MAX\_USAGE\_LOG; i++) {  
 char usage[10];  
 sprintf(usage, "Use%d", i + 1);  
 logUsage(usage, i);  
 }  
  
 for (int i = 5; i < 7; i++) {  
 for (int j = 0; j < MAX\_USAGE\_LOG - 1; j++) {  
 free(waterUsageLog[j]);  
 waterUsageLog[j] = strdup(waterUsageLog[j + 1]);  
 }  
 char usage[10];  
 sprintf(usage, "Use%d", i + 1);  
 free(waterUsageLog[MAX\_USAGE\_LOG - 1]);  
 waterUsageLog[MAX\_USAGE\_LOG - 1] = strdup(usage);  
 }  
  
 for (int i = 0; i < MAX\_USAGE\_LOG; i++) {  
 printf("Log[%d]: %s\n", i, waterUsageLog[i]);  
 }  
  
 printf("\n");  
  
 // c) Leaky and Fixed Components  
 SinglyNode\* leakyHead = NULL;  
 DoublyNode\* fixedHead = NULL;  
  
 leakyHead = insertSingly(leakyHead, "Pipe");  
 leakyHead = insertSingly(leakyHead, "Sprinkler");  
  
 leakyHead = deleteSingly(leakyHead, "Pipe");  
 fixedHead = insertDoubly(fixedHead, "Pipe");  
  
 traverseForward(fixedHead);  
 traverseBackward(fixedHead);  
  
 printf("\n");  
  
 // d) Priority Fixes  
 CircularNode\* tail = NULL;  
 tail = insertCircular(tail, "Pump");  
 tail = insertCircular(tail, "Valve");  
  
 printf("First traversal:\n");  
 traverseCircular(tail);  
 printf("Second traversal:\n");  
 traverseCircular(tail);  
  
 return 0;  
}