ELEMENTARY DATA STRUCTURES AND LOGICAL THINKING

ASSIGNMENT: QUESTION 8

NAME: D.SUSHMITHA

ROLL NO.: ME24B1020 (Batch 5)

Objective:

To coordinate a team of rescue robots during an earthquake emergency using structured data management. The program allows:

- Assignment and prioritization of tasks.
- Tracking urgent situations.
- Monitoring damaged and repaired robots.
- Organizing redeployment in a circular priority fashion.
- Logging important mission events into the rescue log.

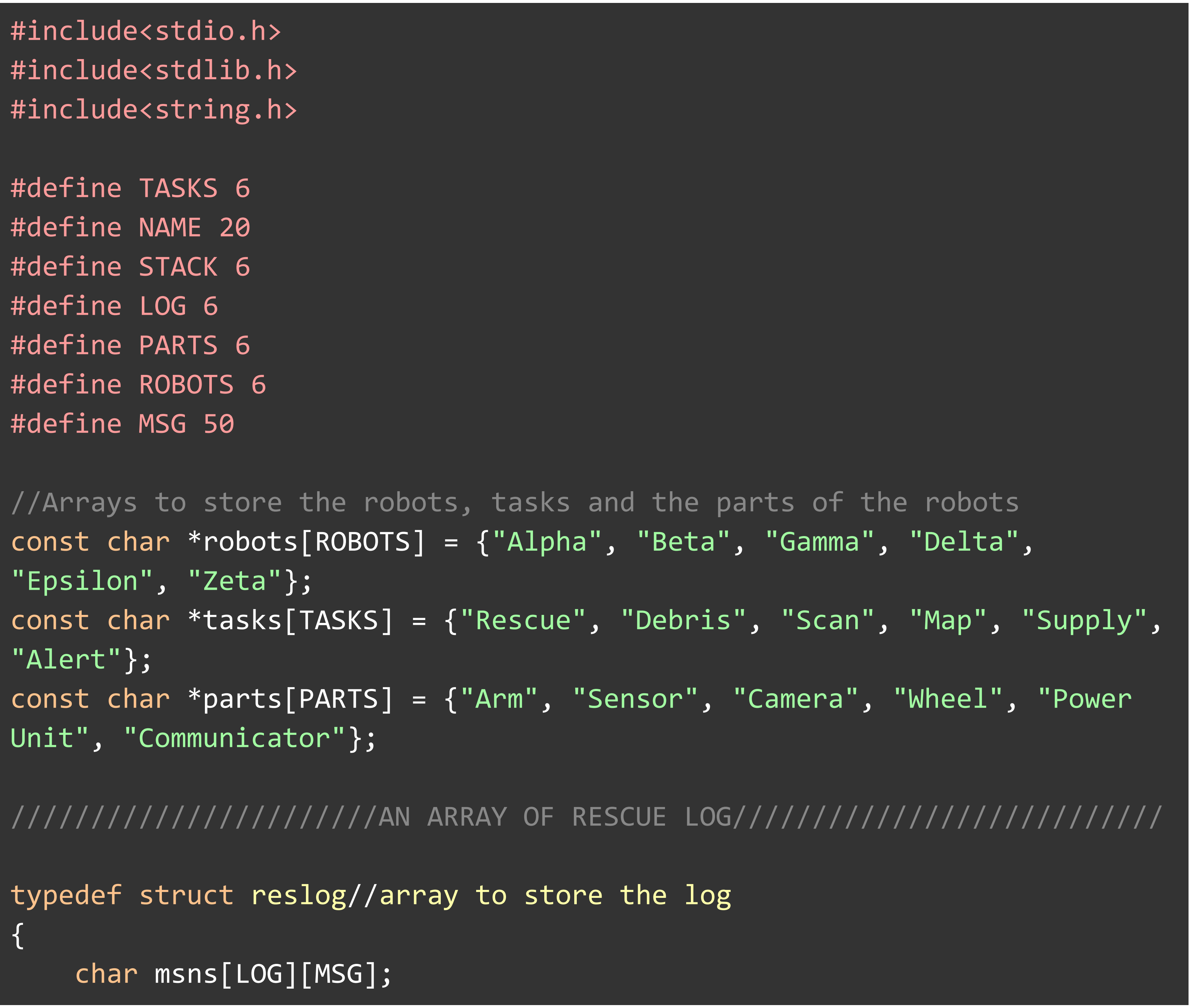
Data structures and their roles:

Variable name	Data structure	Role and Justification
queue	Circular queue	● Holds regular mission tasks ● Efficient task rotation ● Avoid memory flow
stack	Stack	● Holds urgent tasks in LIFO order ● Urgent tasks to be prioritized and processed first as per LIFO order
reslog	Circular array	● A log of mission events(overwrites if full) ● Maintains fixed size ● Replaces the oldest entry if full
dmgd	Singly Linked List	● Tracks damaged robots along with their faulty parts ● Easy insertion/removal of damaged robot entries
rpd	Doubly Linked List	● Tracks repaired robots. Displayed both forward and backward. ● Navigate repairs in dual directions
cpr	Circular Linked List	● Maintains robots that are ready for circular priority redeployment ● Ensures continuous and fair redeployment

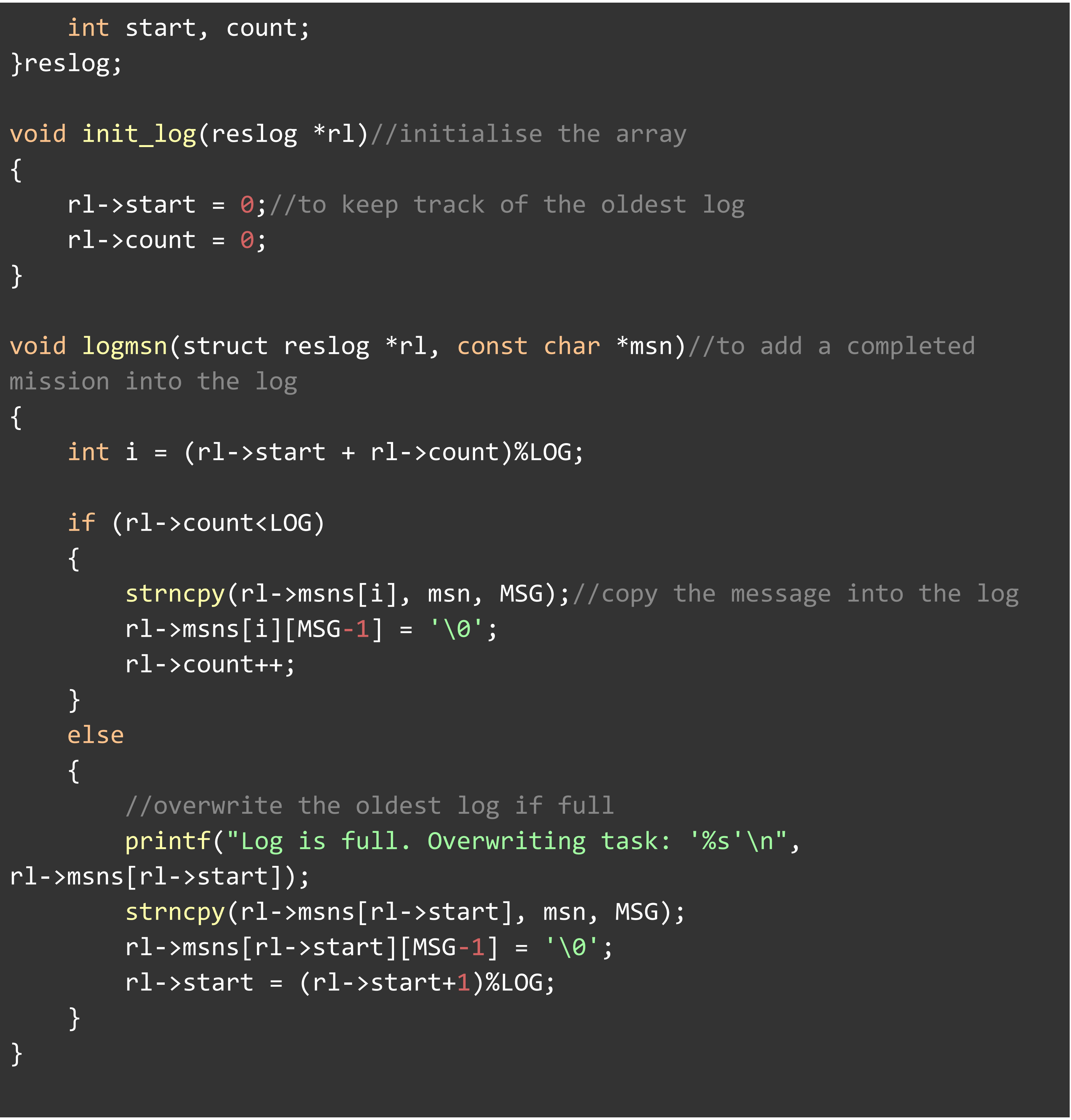
Main Features:

- **Add & View Tasks:** Queue for normal missions, stack for urgent ones.
- **Urgent Task Handling:** Move tasks from queue to stack and pop them with logs.
- **Rescue Log:** Logs all significant events (task assignment, damage, repairs, etc.).
- **Robot Damage Management:** Mark robots with damaged parts and later repair them.
- **Repaired Robots Tracking:** Maintain and view a list of robots that are fixed.
- **Redeployment System:** Circular list to rotate redeployment of fixed robots.
- **Interactive Console UI:** Menu-driven interface using `scanf` and `printf`.

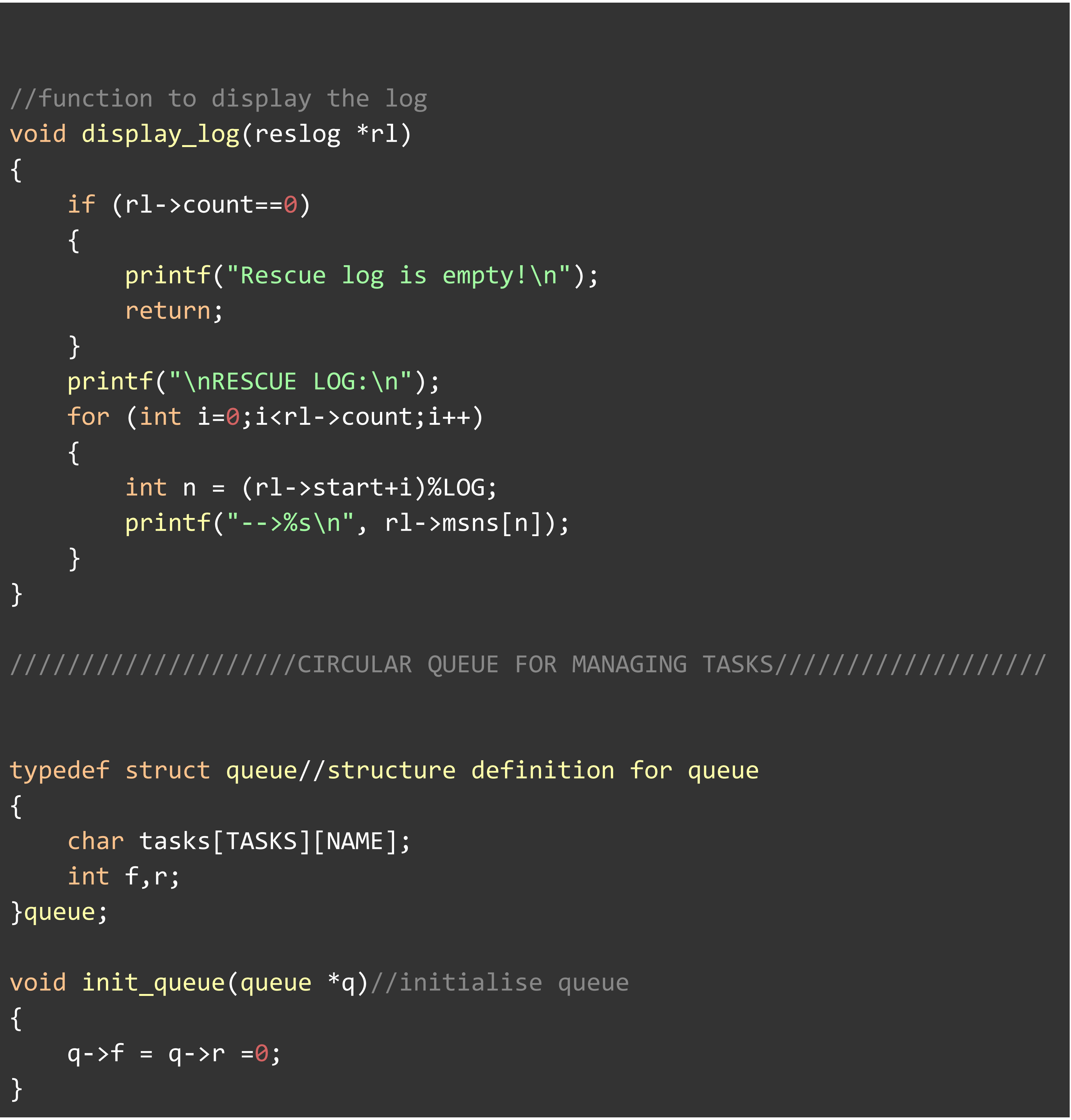
Source code in C:



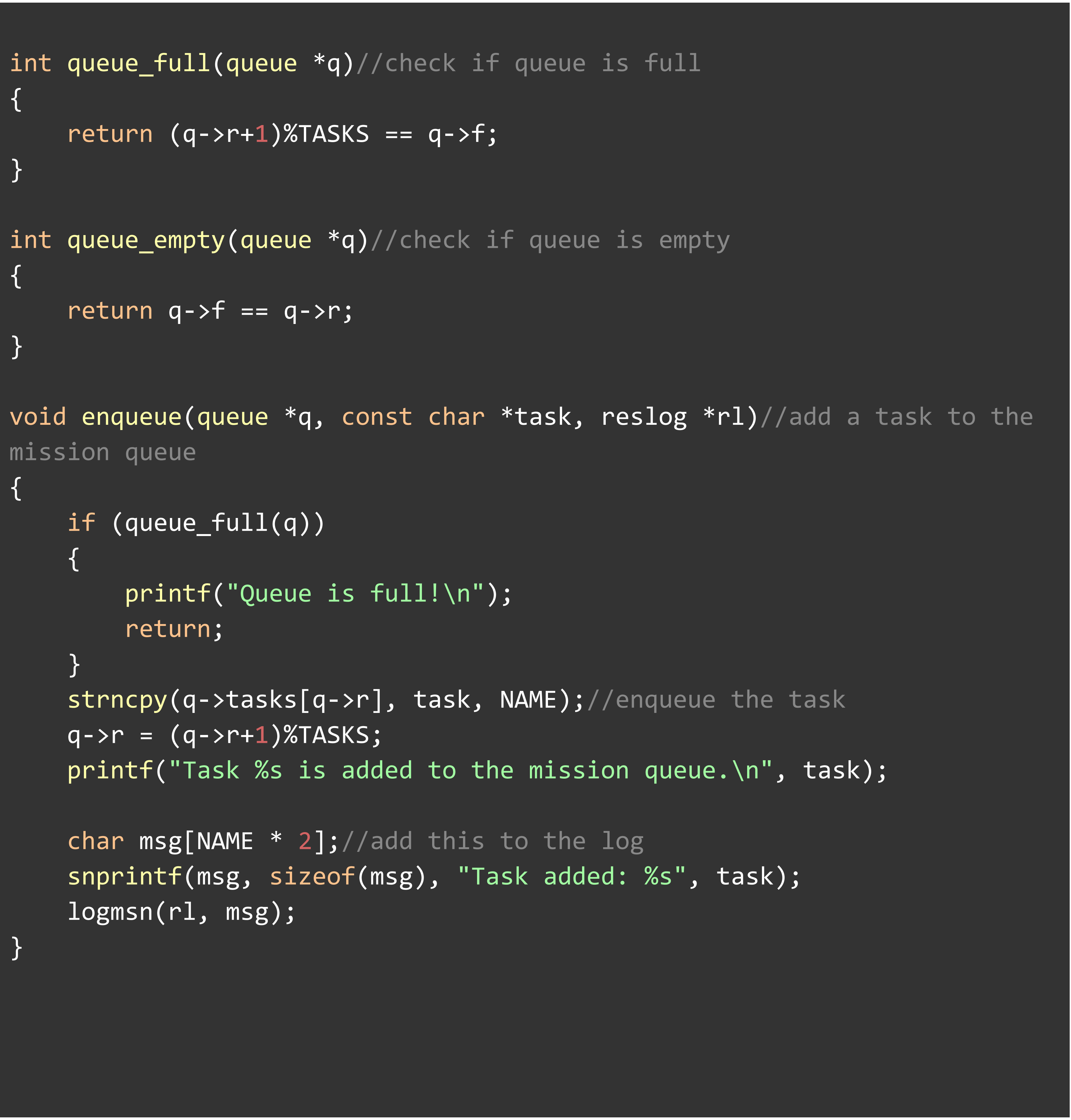
```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define TASKS 6
#define NAME 20
#define STACK 6
#define LOG 6
#define PARTS 6
#define ROBOTS 6
#define MSG 50

//Arrays to store the robots, tasks and the parts of the robots
const char *robots[ROBOTS] = {"Alpha", "Beta", "Gamma", "Delta",
"Epsilon", "Zeta"};
const char *tasks[TASKS] = {"Rescue", "Debris", "Scan", "Map", "Supply",
"Alert"};
const char *parts[PARTS] = {"Arm", "Sensor", "Camera", "Wheel", "Power
Unit", "Communicator"};

///////////////////////AN ARRAY OF RESCUE LOG//////////////////////////

typedef struct reslog//array to store the log
{
    char msns[LOG][MSG];
```



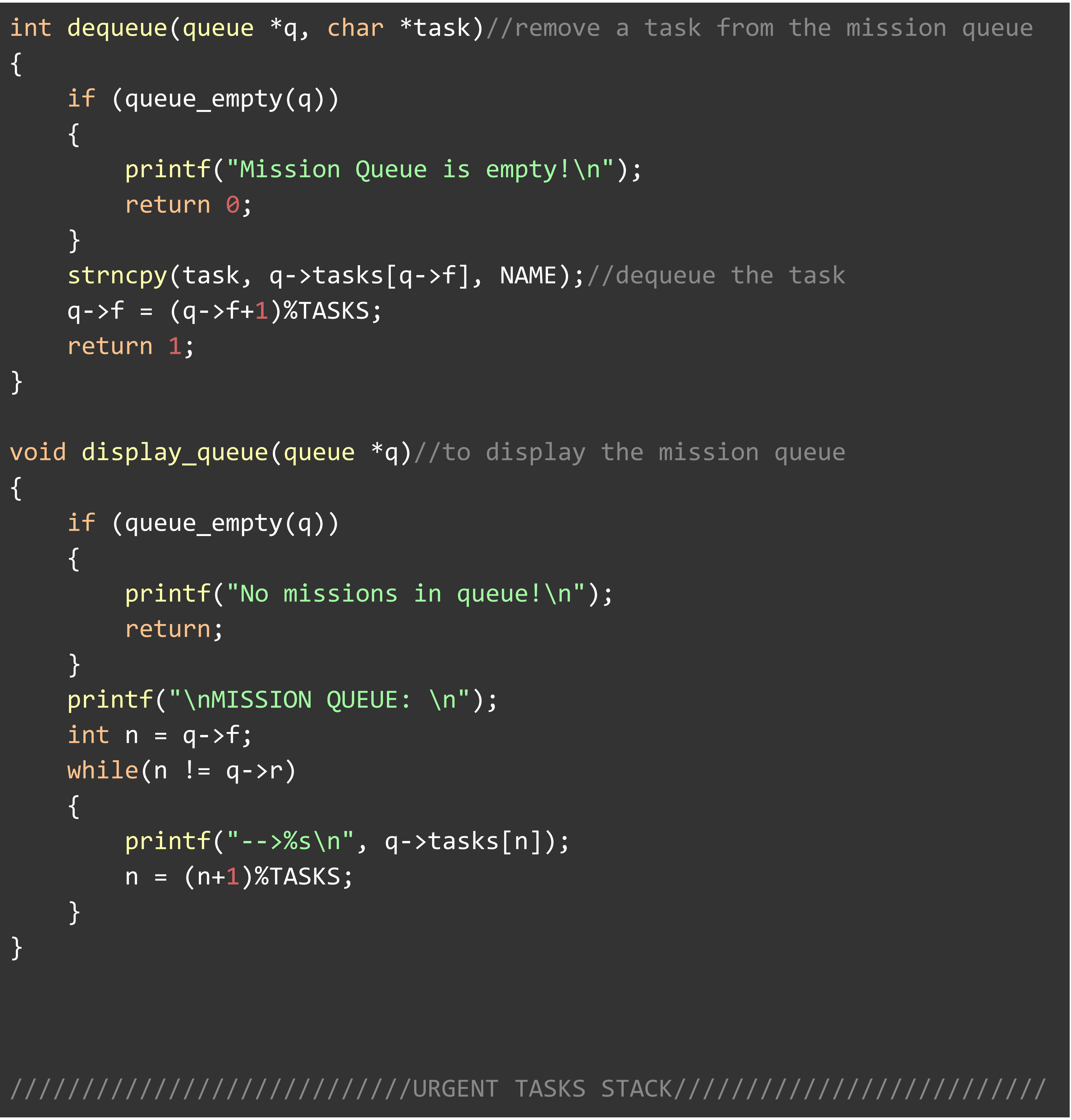
```
    int start, count;
}reslog;

void init_log(reslog *rl)//initialise the array
{
    rl->start = 0;//to keep track of the oldest log
    rl->count = 0;
}

void logmsn(struct reslog *rl, const char *msn)//to add a completed
mission into the log
{
    int i = (rl->start + rl->count)%LOG;

    if (rl->count<LOG)
    {
        strncpy(rl->msns[i], msn, MSG);//copy the message into the log
        rl->msns[i][MSG-1] = '\0';
        rl->count++;
    }
    else
    {
        //overwrite the oldest log if full
        printf("Log is full. Overwriting task: '%s'\n",
rl->msns[rl->start]);
        strncpy(rl->msns[rl->start], msn, MSG);
        rl->msns[rl->start][MSG-1] = '\0';
        rl->start = (rl->start+1)%LOG;
    }
}
```



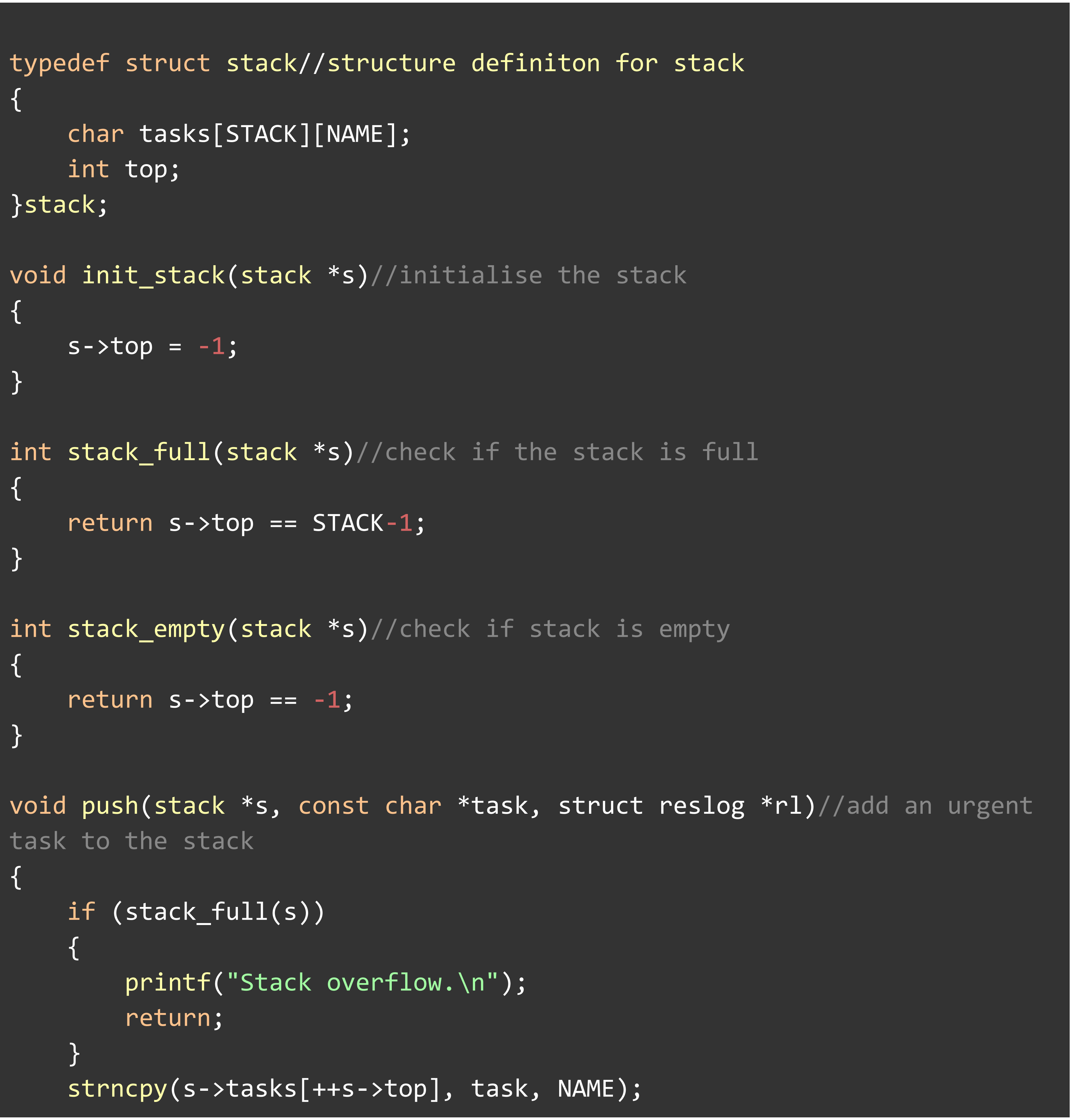
```
//function to display the log
void display_log(reslog *rl)
{
    if (rl->count==0)
    {
        printf("Rescue log is empty!\n");
        return;
    }
    printf("\nRESCUE LOG:\n");
    for (int i=0;i<rl->count;i++)
    {
        int n = (rl->start+i)%LOG;
        printf("-->%s\n", rl->msns[n]);
    }
}

///////////////////CIRCULAR QUEUE FOR MANAGING TASKS//////////////////


typedef struct queue//structure definition for queue
{
    char tasks[TASKS][NAME];
    int f,r;
}queue;

void init_queue(queue *q)//initialise queue
{
    q->f = q->r =0;
}
```



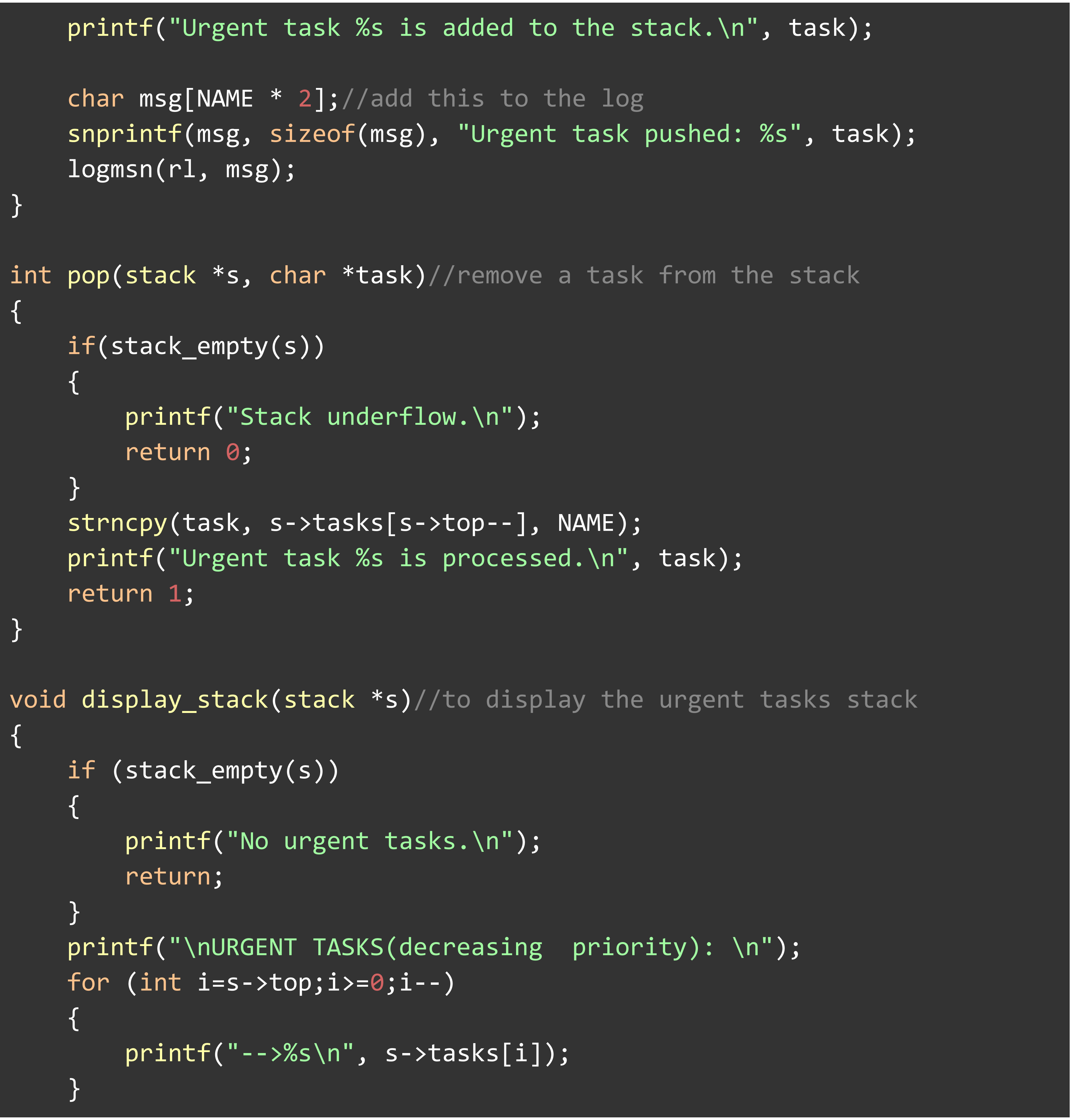
```c
int queue_full(queue *q)//check if queue is full
{
    return (q->r+1)%TASKS == q->f;
}

int queue_empty(queue *q)//check if queue is empty
{
    return q->f == q->r;
}

void enqueue(queue *q, const char *task, reslog *rl)//add a task to the
mission queue
{
    if (queue_full(q))
    {
        printf("Queue is full!\n");
        return;
    }
    strncpy(q->tasks[q->r], task, NAME);//enqueue the task
    q->r = (q->r+1)%TASKS;
    printf("Task %s is added to the mission queue.\n", task);

    char msg[NAME * 2];//add this to the log
    snprintf(msg, sizeof(msg), "Task added: %s", task);
    logmsn(rl, msg);
}
```



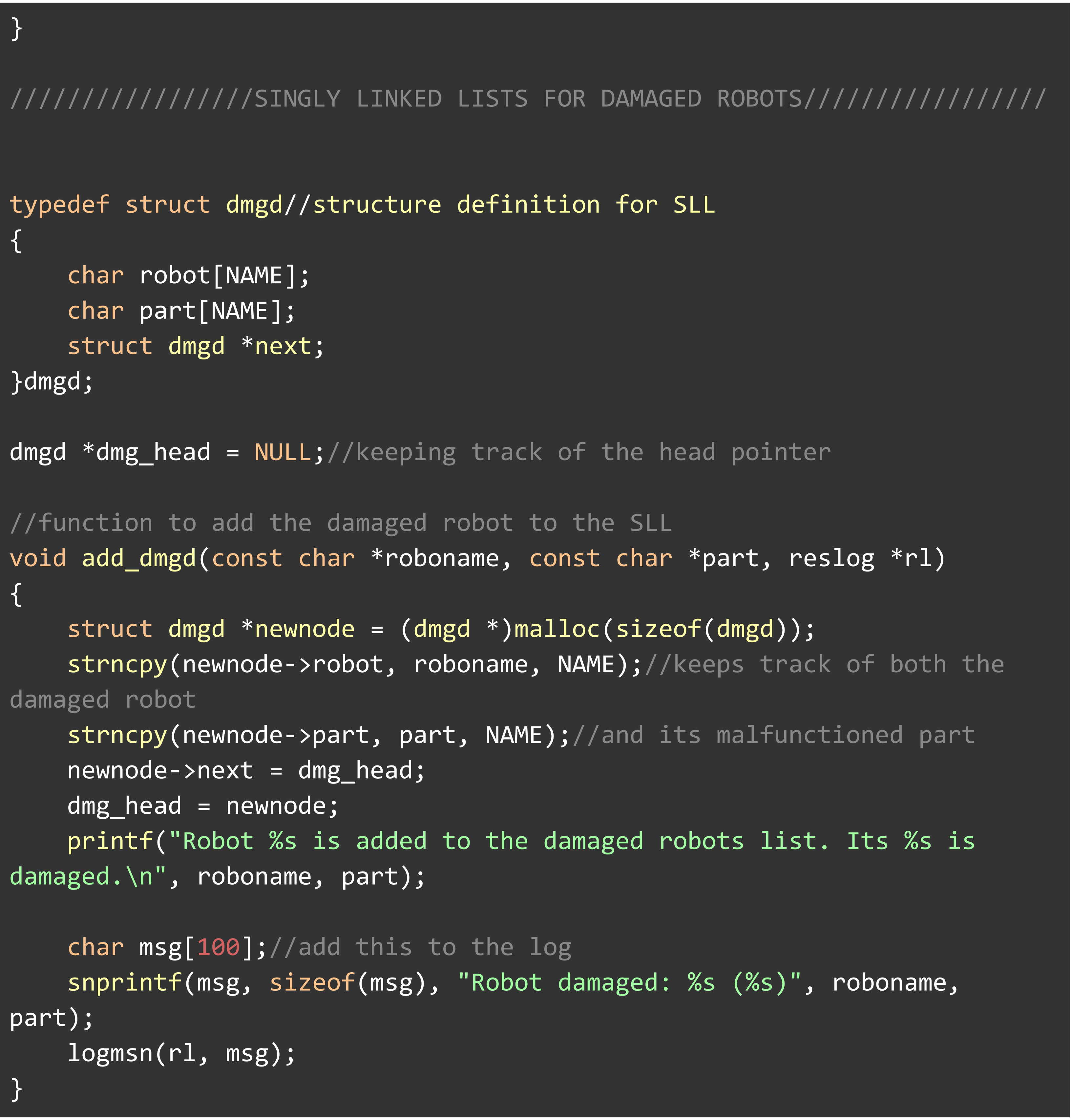
```
int dequeue(queue *q, char *task)//remove a task from the mission queue
{
    if (queue_empty(q))
    {
        printf("Mission Queue is empty!\n");
        return 0;
    }
    strncpy(task, q->tasks[q->f], NAME);//dequeue the task
    q->f = (q->f+1)%TASKS;
    return 1;
}

void display_queue(queue *q)//to display the mission queue
{
    if (queue_empty(q))
    {
        printf("No missions in queue!\n");
        return;
    }
    printf("\nMISSION QUEUE: \n");
    int n = q->f;
    while(n != q->r)
    {
        printf("-->%s\n", q->tasks[n]);
        n = (n+1)%TASKS;
    }
}



////////////////////////////URGENT TASKS STACK//////////////////////////
```



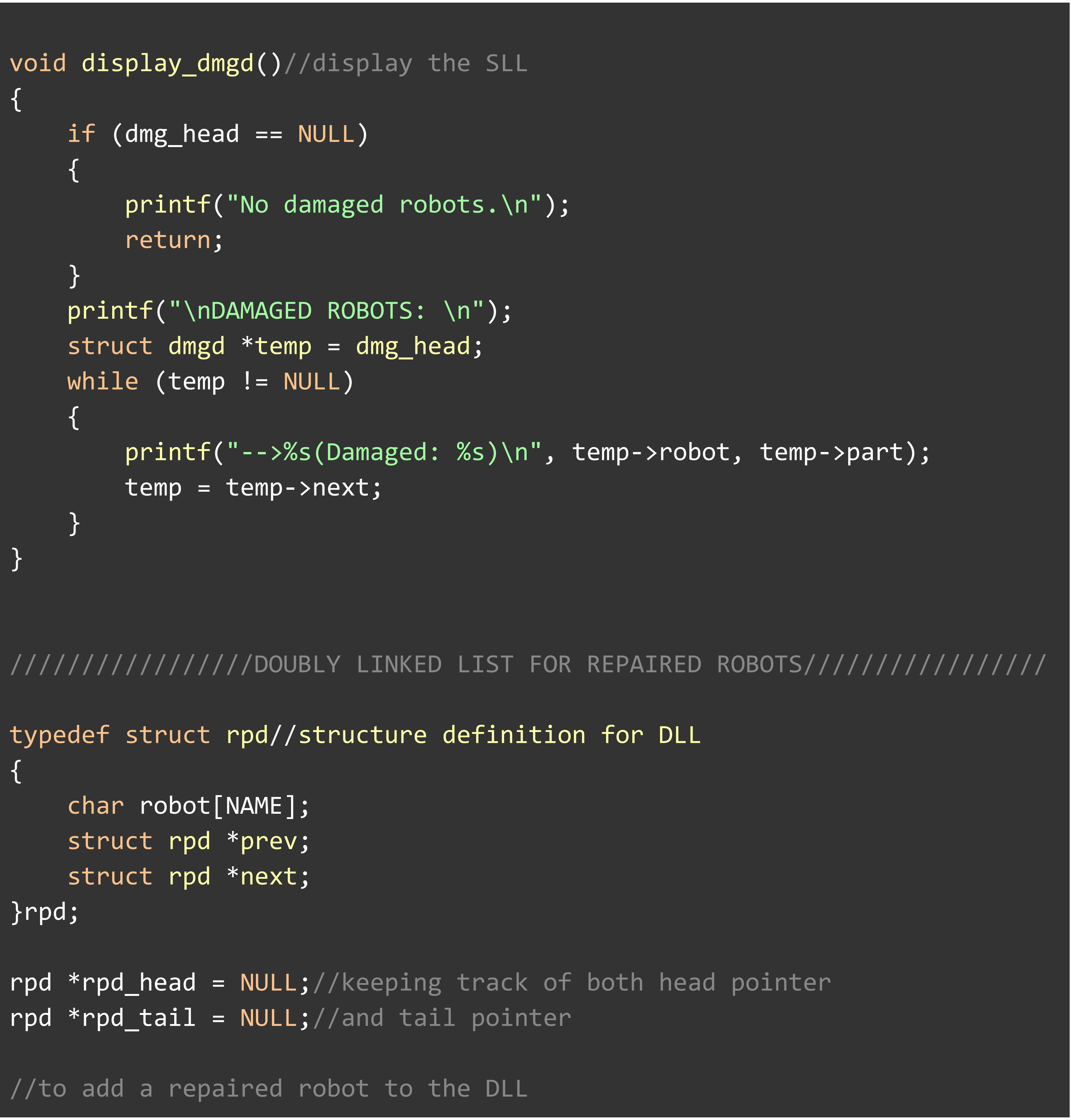
```
typedef struct stack//structure definiton for stack
{
    char tasks[STACK][NAME];
    int top;
}stack;

void init_stack(stack *s)//initialise the stack
{
    s->top = -1;
}

int stack_full(stack *s)//check if the stack is full
{
    return s->top == STACK-1;
}

int stack_empty(stack *s)//check if stack is empty
{
    return s->top == -1;
}

void push(stack *s, const char *task, struct reslog *rl)//add an urgent task to the stack
{
    if (stack_full(s))
    {
        printf("Stack overflow.\n");
        return;
    }
    strncpy(s->tasks[++s->top], task, NAME);
```



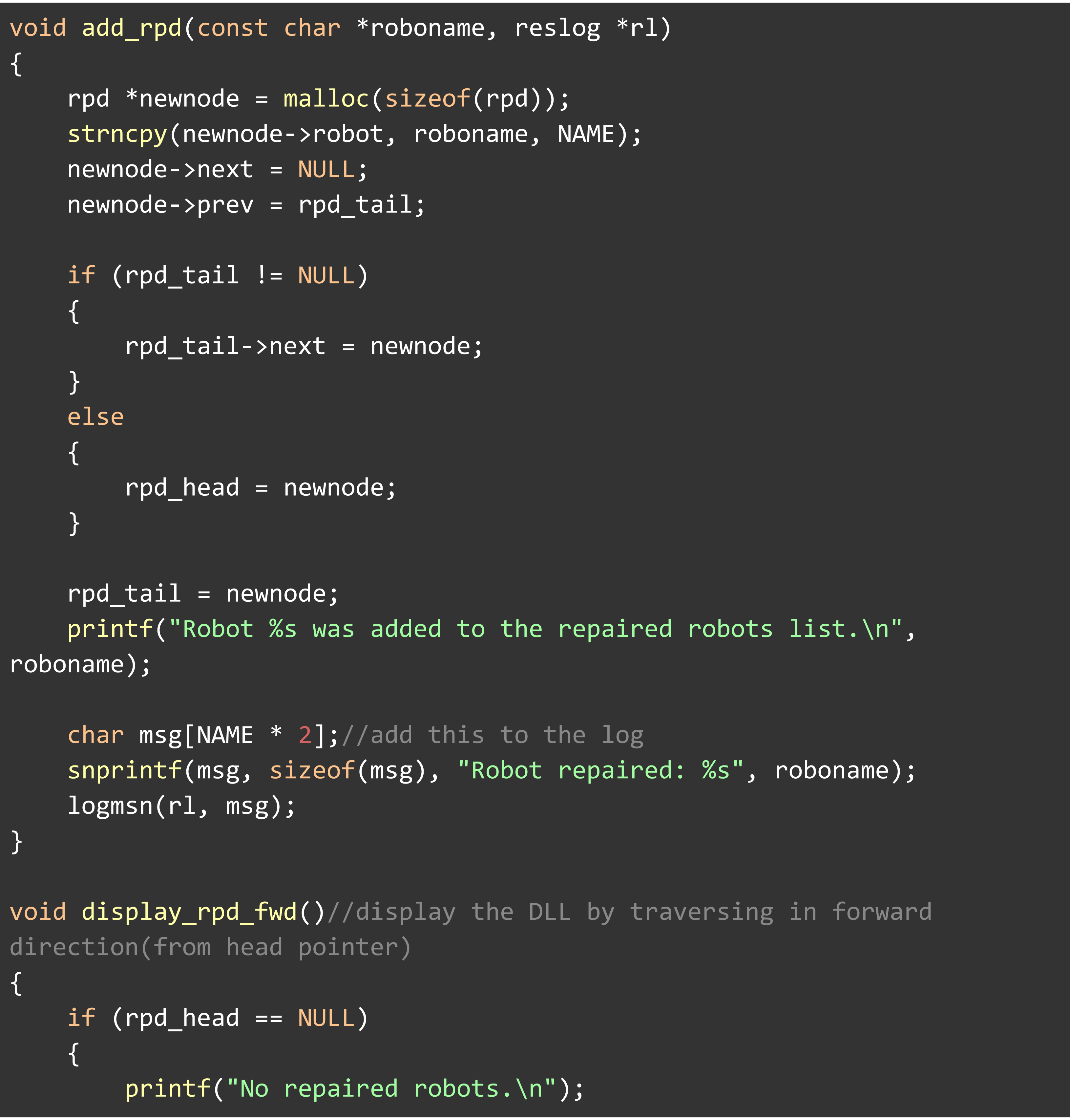
```c
    printf("Urgent task %s is added to the stack.\n", task);

    char msg[NAME * 2];//add this to the log
    snprintf(msg, sizeof(msg), "Urgent task pushed: %s", task);
    logmsn(rl, msg);
}

int pop(stack *s, char *task)//remove a task from the stack
{
    if(stack_empty(s))
    {
        printf("Stack underflow.\n");
        return 0;
    }
    strncpy(task, s->tasks[s->top--], NAME);
    printf("Urgent task %s is processed.\n", task);
    return 1;
}

void display_stack(stack *s)//to display the urgent tasks stack
{
    if (stack_empty(s))
    {
        printf("No urgent tasks.\n");
        return;
    }
    printf("\nURGENT TASKS(decreasing  priority): \n");
    for (int i=s->top;i>=0;i--)
    {
        printf("-->%s\n", s->tasks[i]);
    }
```



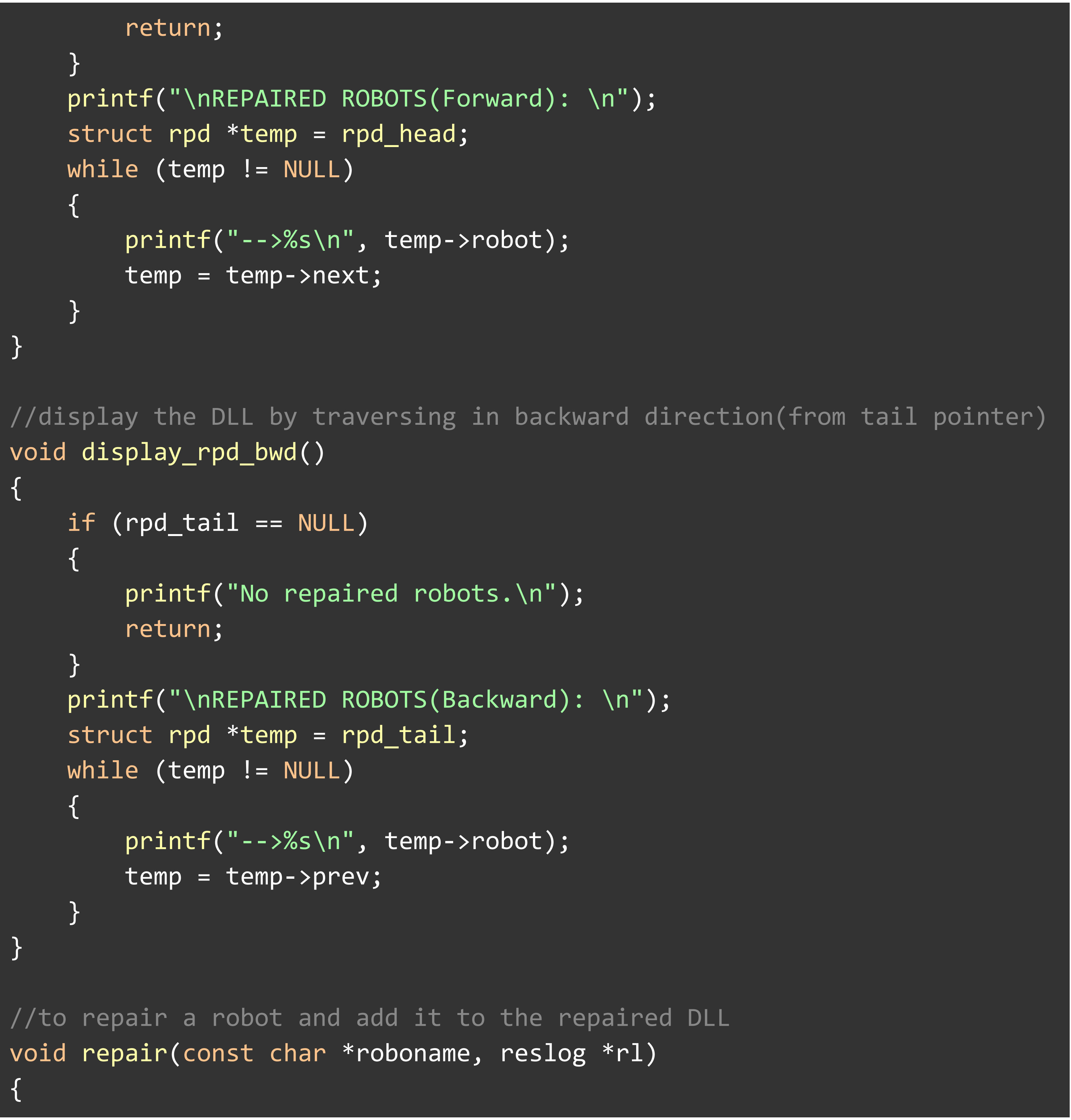
```
}

/////////////////SINGLY LINKED LISTS FOR DAMAGED ROBOTS/////////////////


typedef struct dmgd//structure definition for SLL
{
    char robot[NAME];
    char part[NAME];
    struct dmgd *next;
}dmgd;

dmgd *dmg_head = NULL;//keeping track of the head pointer

//function to add the damaged robot to the SLL
void add_dmgd(const char *roboname, const char *part, reslog *rl)
{
    struct dmgd *newnode = (dmgd *)malloc(sizeof(dmgd));
    strncpy(newnode->robot, roboname, NAME);//keeps track of both the
damaged robot
    strncpy(newnode->part, part, NAME);//and its malfunctioned part
    newnode->next = dmg_head;
    dmg_head = newnode;
    printf("Robot %s is added to the damaged robots list. Its %s is
damaged.\n", roboname, part);

    char msg[100];//add this to the log
    snprintf(msg, sizeof(msg), "Robot damaged: %s (%s)", roboname,
part);
    logmsn(rl, msg);
}
```



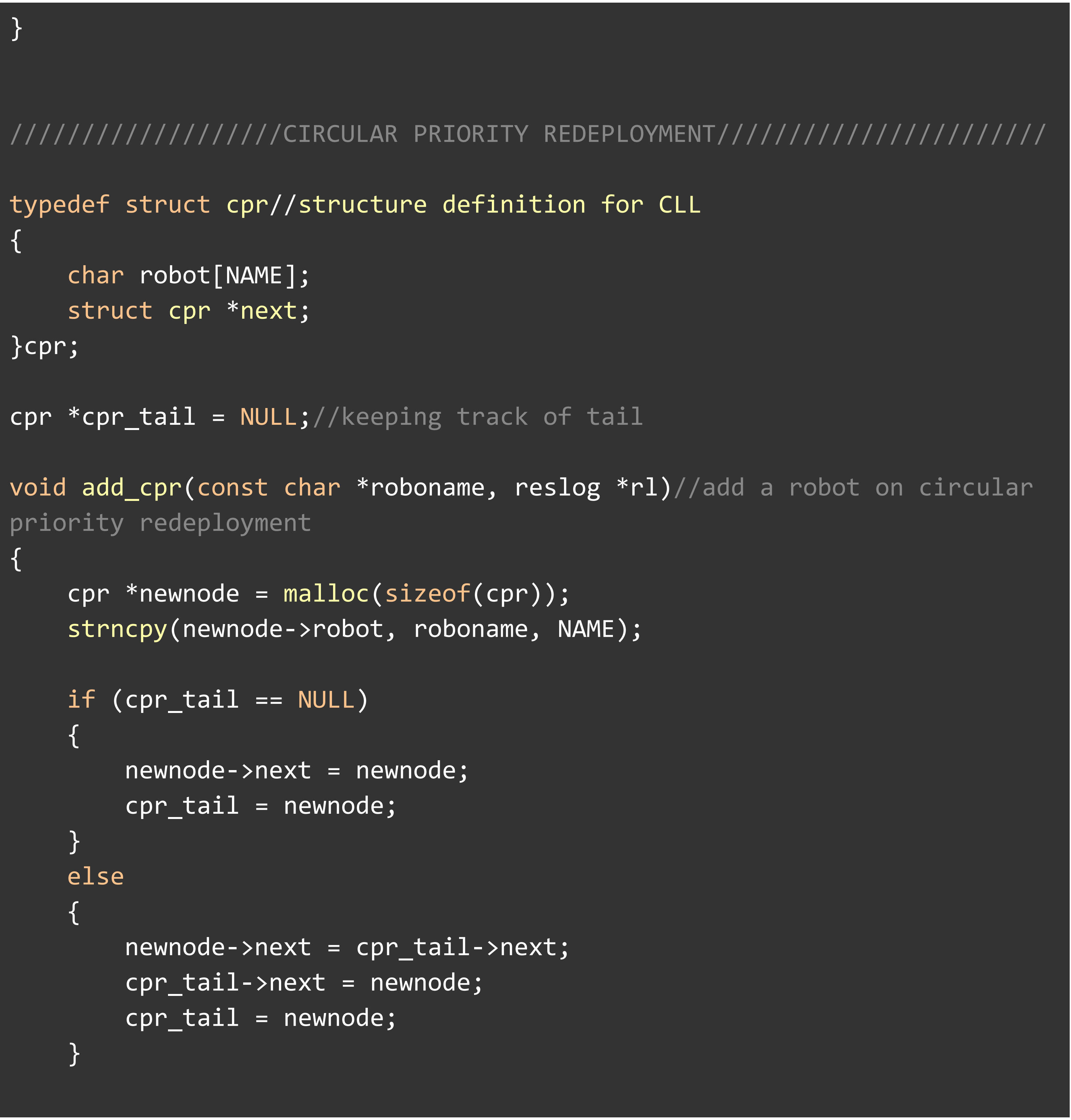
```
void display_dmgd()//display the SLL
{
    if (dmg_head == NULL)
    {
        printf("No damaged robots.\n");
        return;
    }
    printf("\nDAMAGED ROBOTS: \n");
    struct dmgd *temp = dmg_head;
    while (temp != NULL)
    {
        printf("-->%s(Damaged: %s)\n", temp->robot, temp->part);
        temp = temp->next;
    }
}


/////////////////DOUBLY LINKED LIST FOR REPAIRED ROBOTS/////////////////

typedef struct rpd//structure definition for DLL
{
    char robot[NAME];
    struct rpd *prev;
    struct rpd *next;
}rpd;

rpd *rpd_head = NULL;//keeping track of both head pointer
rpd *rpd_tail = NULL;//and tail pointer

//to add a repaired robot to the DLL
```



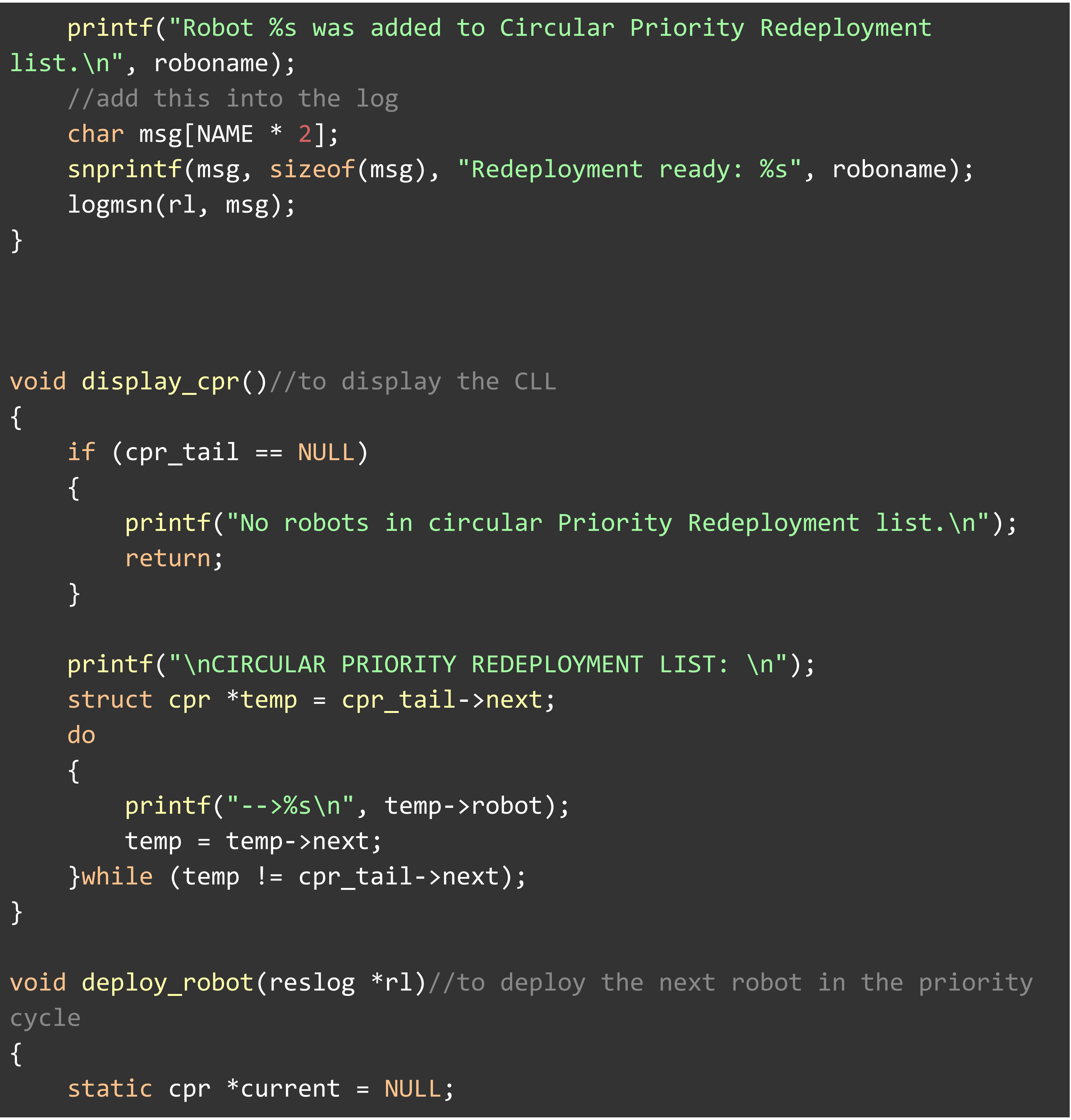
```c
void add_rpd(const char *roboname, reslog *rl)
{
    rpd *newnode = malloc(sizeof(rpd));
    strncpy(newnode->robot, roboname, NAME);
    newnode->next = NULL;
    newnode->prev = rpd_tail;

    if (rpd_tail != NULL)
    {
        rpd_tail->next = newnode;
    }
    else
    {
        rpd_head = newnode;
    }

    rpd_tail = newnode;
    printf("Robot %s was added to the repaired robots list.\n", 
roboname);

    char msg[NAME * 2];//add this to the log
    snprintf(msg, sizeof(msg), "Robot repaired: %s", roboname);
    logmsn(rl, msg);
}

void display_rpd_fwd()//display the DLL by traversing in forward 
direction(from head pointer)
{
    if (rpd_head == NULL)
    {
        printf("No repaired robots.\n");
```



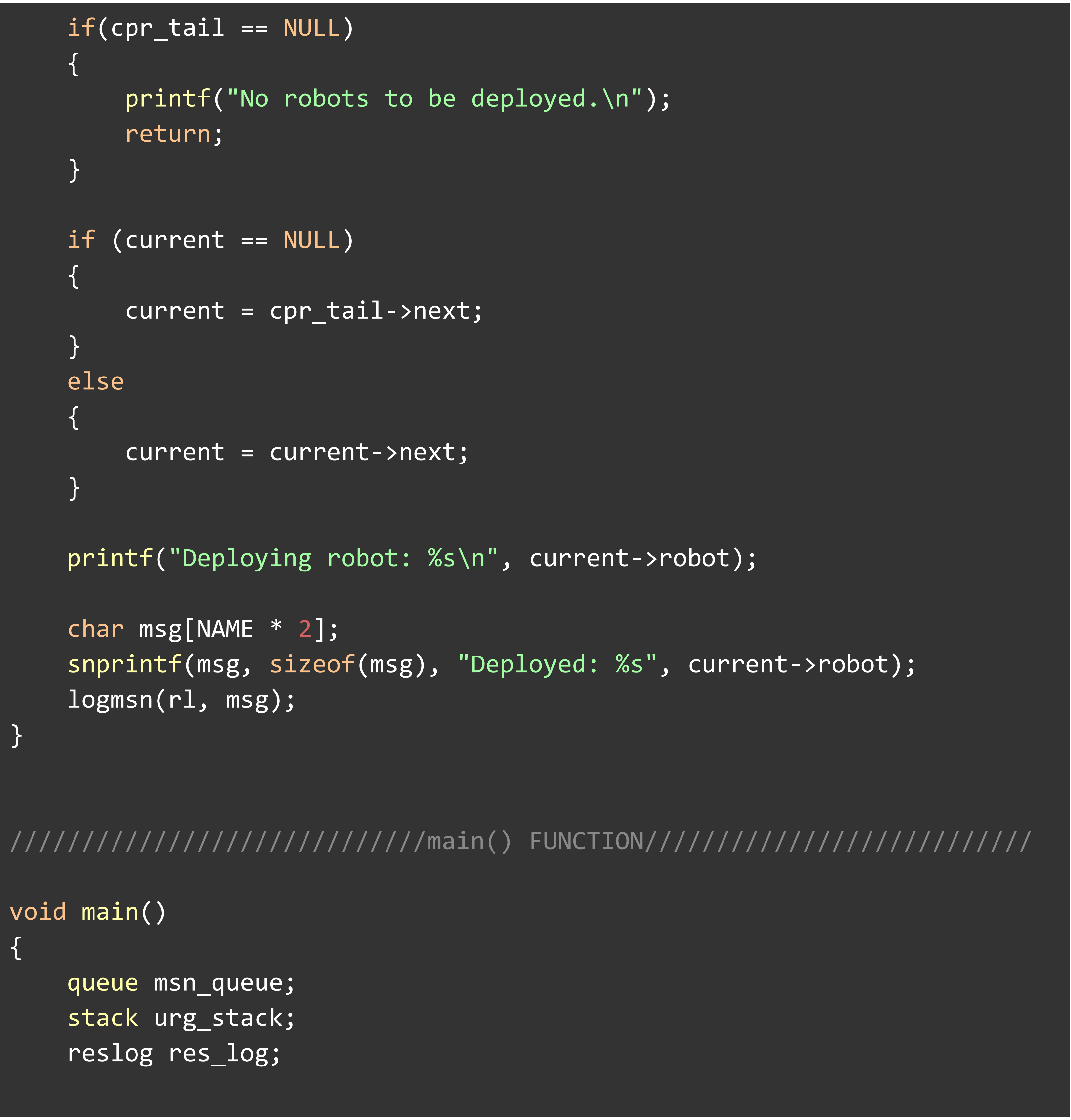
```
        return;
    }
    printf("\nREPAIRED ROBOTS(Forward): \n");
    struct rpd *temp = rpd_head;
    while (temp != NULL)
    {
        printf("-->%s\n", temp->robot);
        temp = temp->next;
    }
}

//display the DLL by traversing in backward direction(from tail pointer)
void display_rpd_bwd()
{
    if (rpd_tail == NULL)
    {
        printf("No repaired robots.\n");
        return;
    }
    printf("\nREPAIRED ROBOTS(Backward): \n");
    struct rpd *temp = rpd_tail;
    while (temp != NULL)
    {
        printf("-->%s\n", temp->robot);
        temp = temp->prev;
    }
}

//to repair a robot and add it to the repaired DLL
void repair(const char *roboname, reslog *rl)
{
    struct dmgd *temp = dmg_head;
    struct dmgd *prev = NULL;

    while (temp != NULL && strcmp(temp->robot, roboname) != 0)
    {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL)
    {
        printf("Robot %s was not found in damaged robots list.\n",
roboname);
        return;
    }

    printf("Repairing %s in robot %s...\n", temp->part, temp->robot);

    if (prev == NULL)
    {
        dmg_head = temp->next;
    }
    else
    {
        prev->next = temp->next;
    }

    add_rpd(roboname, rl);
    printf("Repaired the robot %s and moved it to repaired list.\n",
roboname);
    free(temp);
```



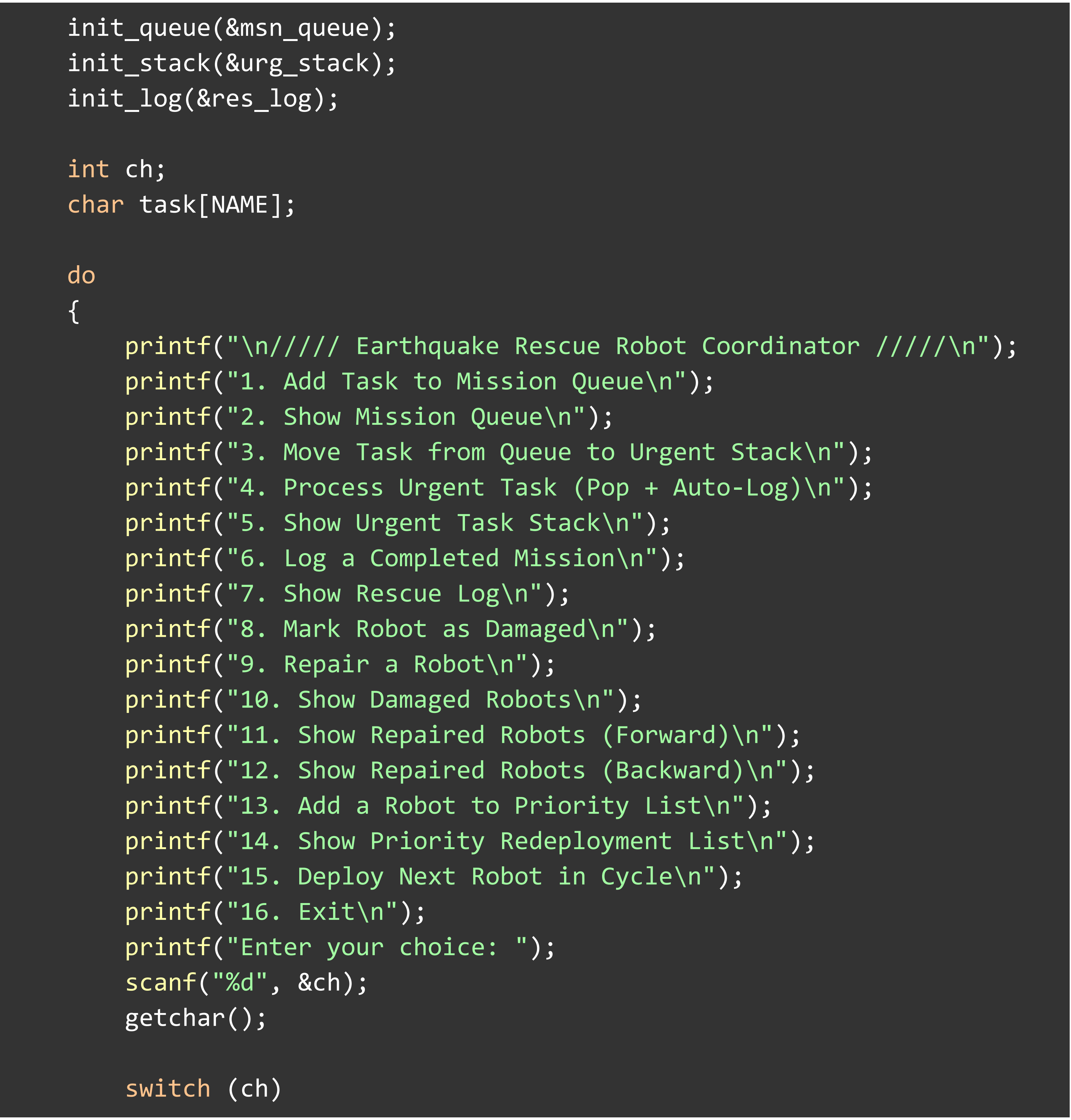
```
}


///////////////////CIRCULAR PRIORITY REDEPLOYMENT///////////////////////

typedef struct cpr//structure definition for CLL
{
    char robot[NAME];
    struct cpr *next;
}cpr;

cpr *cpr_tail = NULL;//keeping track of tail

void add_cpr(const char *roboname, reslog *rl)//add a robot on circular
priority redeployment
{
    cpr *newnode = malloc(sizeof(cpr));
    strncpy(newnode->robot, roboname, NAME);

    if (cpr_tail == NULL)
    {
        newnode->next = newnode;
        cpr_tail = newnode;
    }
    else
    {
        newnode->next = cpr_tail->next;
        cpr_tail->next = newnode;
        cpr_tail = newnode;
    }
```



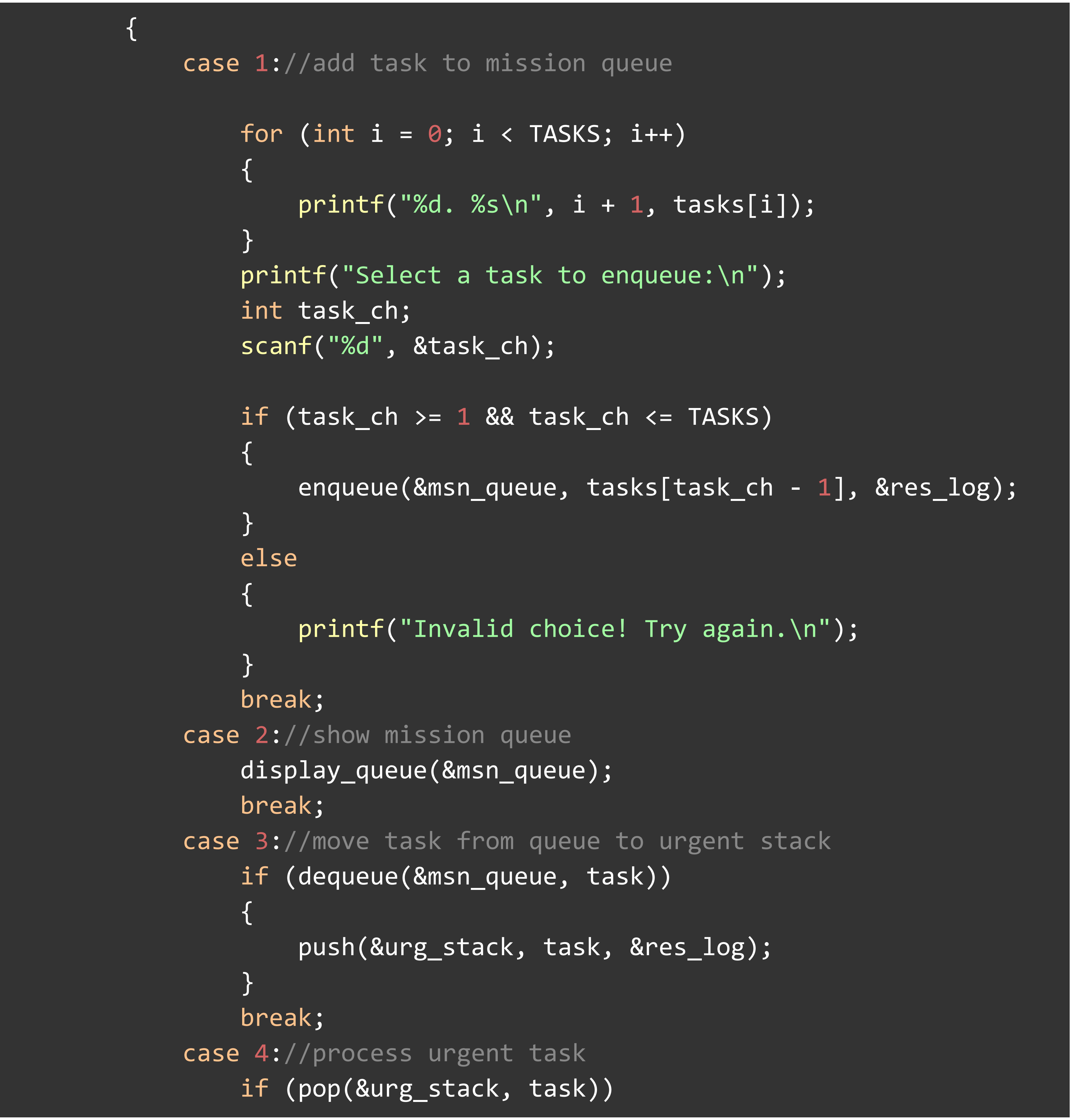
```c
    printf("Robot %s was added to Circular Priority Redeployment list.\n", roboname);
    //add this into the log
    char msg[NAME * 2];
    snprintf(msg, sizeof(msg), "Redeployment ready: %s", roboname);
    logmsn(rl, msg);
}



void display_cpr()//to display the CLL
{
    if (cpr_tail == NULL)
    {
        printf("No robots in circular Priority Redeployment list.\n");
        return;
    }

    printf("\nCIRCULAR PRIORITY REDEPLOYMENT LIST: \n");
    struct cpr *temp = cpr_tail->next;
    do
    {
        printf("-->%s\n", temp->robot);
        temp = temp->next;
    }while (temp != cpr_tail->next);
}

void deploy_robot(reslog *rl)//to deploy the next robot in the priority cycle
{
    static cpr *current = NULL;
```



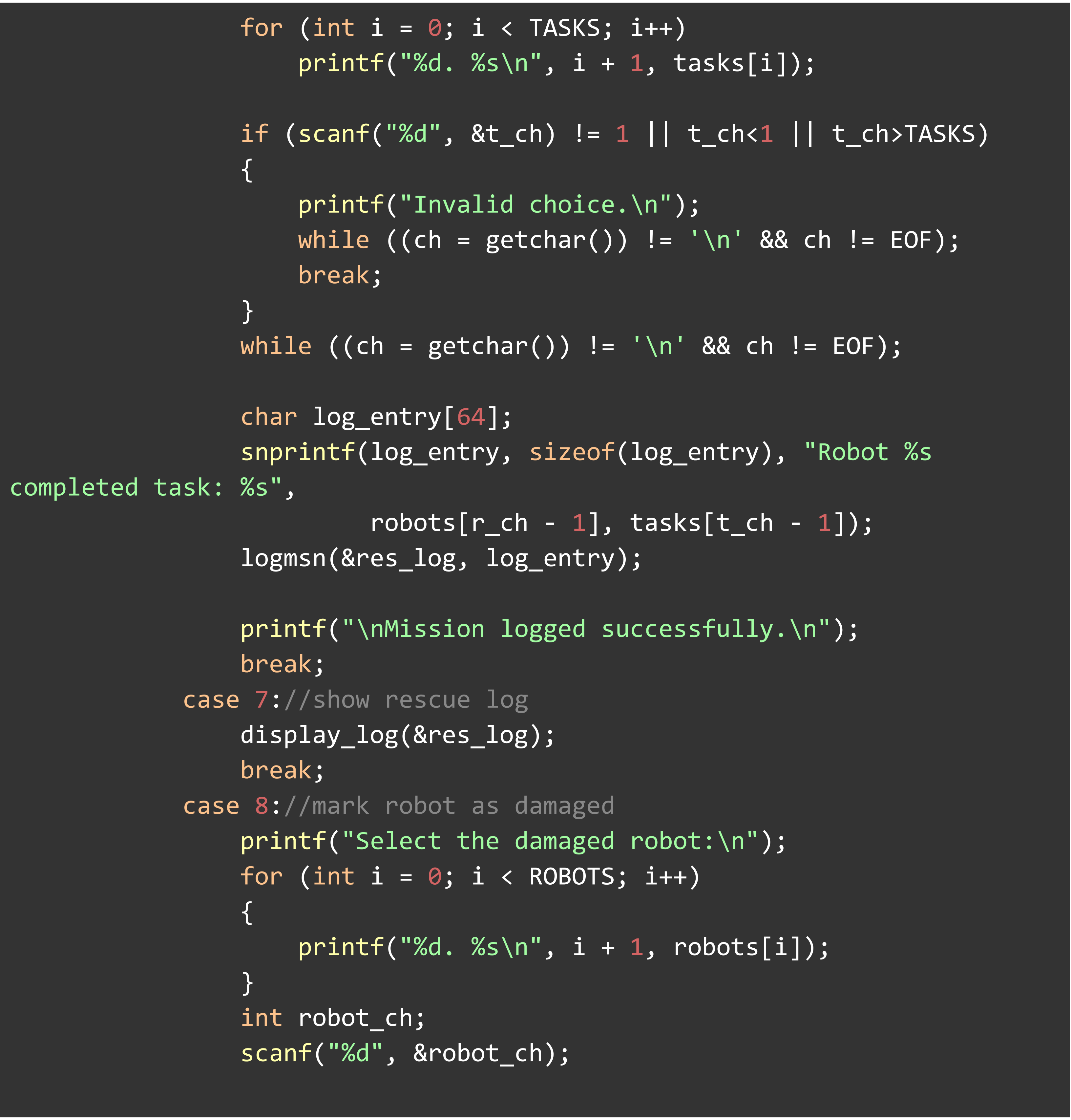
```
    if(cpr_tail == NULL)
    {
        printf("No robots to be deployed.\n");
        return;
    }

    if (current == NULL)
    {
        current = cpr_tail->next;
    }
    else
    {
        current = current->next;
    }

    printf("Deploying robot: %s\n", current->robot);

    char msg[NAME * 2];
    snprintf(msg, sizeof(msg), "Deployed: %s", current->robot);
    logmsn(rl, msg);
}


/////////////////////////////main() FUNCTION///////////////////////////

void main()
{
    queue msn_queue;
    stack urg_stack;
    reslog res_log;
```



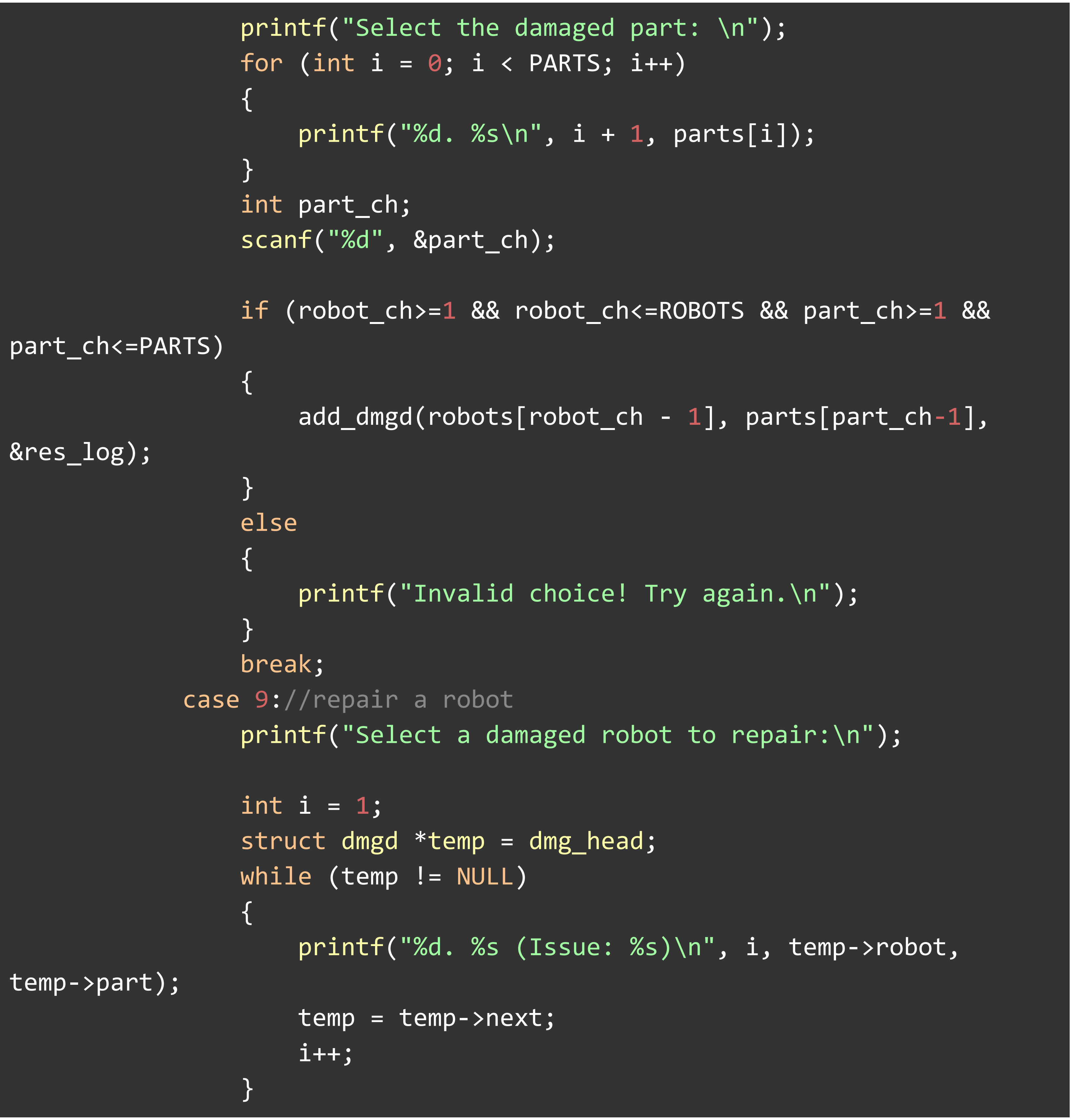
```
    init_queue(&msn_queue);
    init_stack(&urg_stack);
    init_log(&res_log);

    int ch;
    char task[NAME];

    do
    {
        printf("\n///// Earthquake Rescue Robot Coordinator /////\n");
        printf("1. Add Task to Mission Queue\n");
        printf("2. Show Mission Queue\n");
        printf("3. Move Task from Queue to Urgent Stack\n");
        printf("4. Process Urgent Task (Pop + Auto-Log)\n");
        printf("5. Show Urgent Task Stack\n");
        printf("6. Log a Completed Mission\n");
        printf("7. Show Rescue Log\n");
        printf("8. Mark Robot as Damaged\n");
        printf("9. Repair a Robot\n");
        printf("10. Show Damaged Robots\n");
        printf("11. Show Repaired Robots (Forward)\n");
        printf("12. Show Repaired Robots (Backward)\n");
        printf("13. Add a Robot to Priority List\n");
        printf("14. Show Priority Redeployment List\n");
        printf("15. Deploy Next Robot in Cycle\n");
        printf("16. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        getchar();

        switch (ch)
```



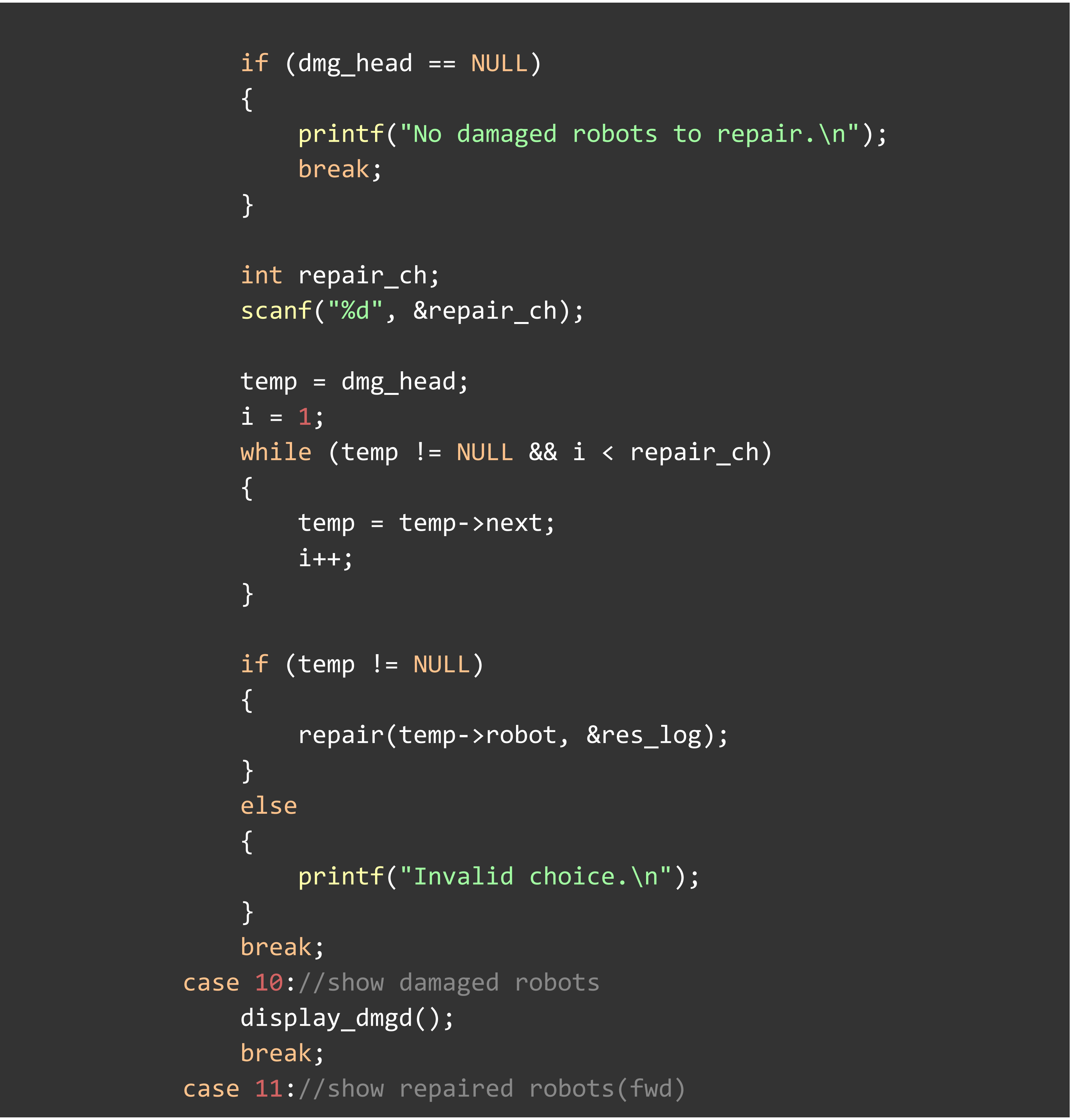
```
        {
            case 1://add task to mission queue

                for (int i = 0; i < TASKS; i++)
                {
                    printf("%d. %s\n", i + 1, tasks[i]);
                }
                printf("Select a task to enqueue:\n");
                int task_ch;
                scanf("%d", &task_ch);

                if (task_ch >= 1 && task_ch <= TASKS)
                {
                    enqueue(&msn_queue, tasks[task_ch - 1], &res_log);
                }
                else
                {
                    printf("Invalid choice! Try again.\n");
                }
                break;
            case 2://show mission queue
                display_queue(&msn_queue);
                break;
            case 3://move task from queue to urgent stack
                if (dequeue(&msn_queue, task))
                {
                    push(&urg_stack, task, &res_log);
                }
                break;
            case 4://process urgent task
                if (pop(&urg_stack, task))
                {
                    char entry[64];
                    snprintf(entry, 64, "Processed urgent task: %s", 
task);
                    logmsn(&res_log, entry);
                }
                break;
            case 5://show urgent task
                display_stack(&urg_stack);
                break;
            case 6://log a completed mission
                int r_ch = 0, t_ch = 0;
                char ch;

                printf("Select robot that completed the mission:\n");
                for (int i = 0; i < ROBOTS; i++)
                {
                    printf("%d. %s\n", i + 1, robots[i]);
                }


                if (scanf("%d", &r_ch) != 1 || r_ch<1 || r_ch>ROBOTS)
                {
                    printf("Invalid choice.\n");
                    while ((ch = getchar()) != '\n' && ch != EOF);
                    break;
                }

                while ((ch = getchar()) != '\n' && ch != EOF);

                printf("Select completed task:\n");
```



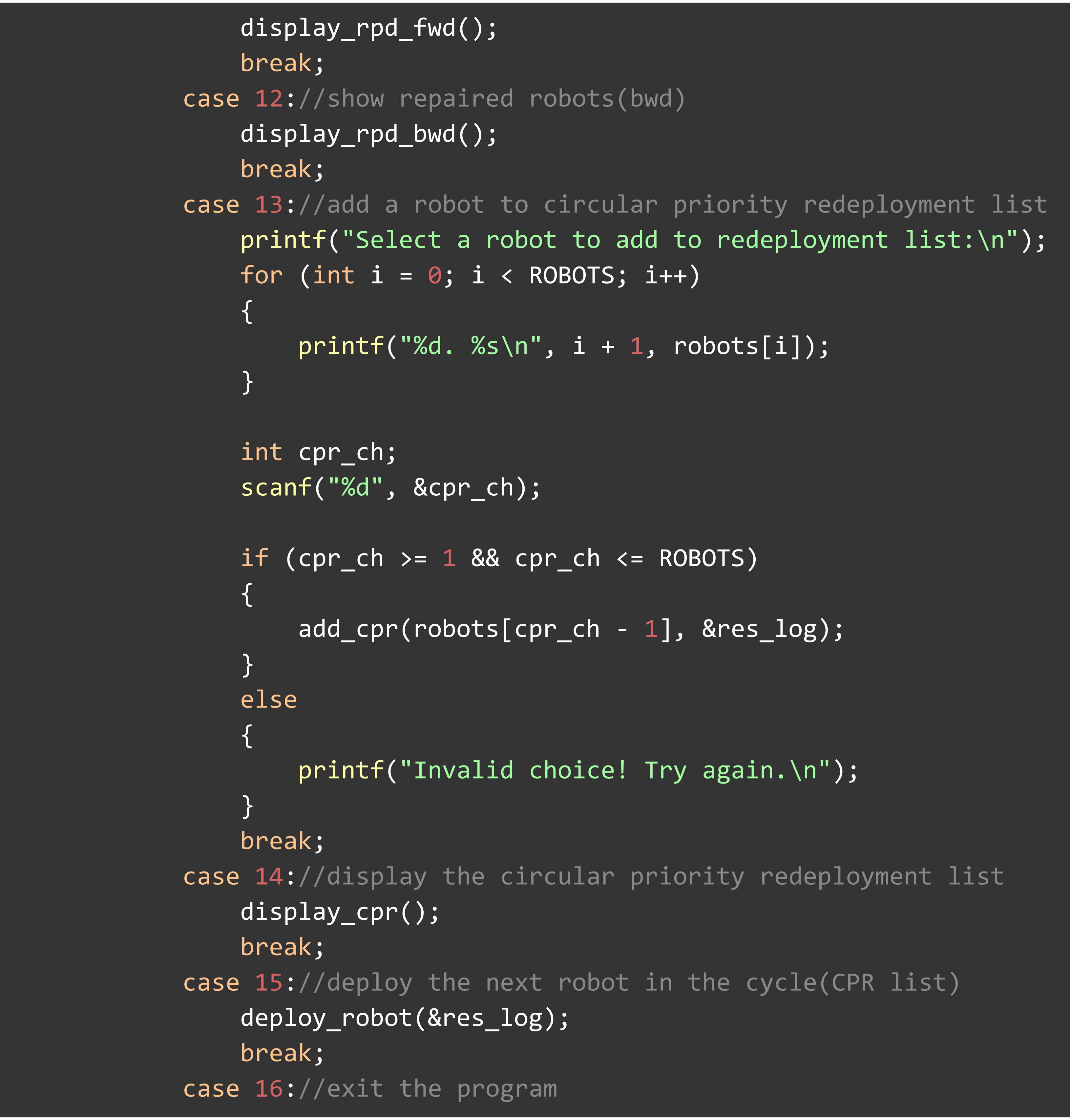
```
                for (int i = 0; i < TASKS; i++)
                    printf("%d. %s\n", i + 1, tasks[i]);

                if (scanf("%d", &t_ch) != 1 || t_ch<1 || t_ch>TASKS)
                {
                    printf("Invalid choice.\n");
                    while ((ch = getchar()) != '\n' && ch != EOF);
                    break;
                }
                while ((ch = getchar()) != '\n' && ch != EOF);

                char log_entry[64];
                snprintf(log_entry, sizeof(log_entry), "Robot %s
completed task: %s",
                         robots[r_ch - 1], tasks[t_ch - 1]);
                logmsn(&res_log, log_entry);

                printf("\nMission logged successfully.\n");
                break;
            case 7://show rescue log
                display_log(&res_log);
                break;
            case 8://mark robot as damaged
                printf("Select the damaged robot:\n");
                for (int i = 0; i < ROBOTS; i++)
                {
                    printf("%d. %s\n", i + 1, robots[i]);
                }
                int robot_ch;
                scanf("%d", &robot_ch);
```



```
                printf("Select the damaged part: \n");
                for (int i = 0; i < PARTS; i++)
                {
                    printf("%d. %s\n", i + 1, parts[i]);
                }
                int part_ch;
                scanf("%d", &part_ch);

                if (robot_ch>=1 && robot_ch<=ROBOTS && part_ch>=1 &&
part_ch<=PARTS)
                {
                    add_dmgd(robots[robot_ch - 1], parts[part_ch-1],
&res_log);
                }
                else
                {
                    printf("Invalid choice! Try again.\n");
                }
                break;
            case 9://repair a robot
                printf("Select a damaged robot to repair:\n");

                int i = 1;
                struct dmgd *temp = dmg_head;
                while (temp != NULL)
                {
                    printf("%d. %s (Issue: %s)\n", i, temp->robot,
temp->part);
                    temp = temp->next;
                    i++;
                }
```



```c
            if (dmg_head == NULL)
            {
                printf("No damaged robots to repair.\n");
                break;
            }

            int repair_ch;
            scanf("%d", &repair_ch);

            temp = dmg_head;
            i = 1;
            while (temp != NULL && i < repair_ch)
            {
                temp = temp->next;
                i++;
            }

            if (temp != NULL)
            {
                repair(temp->robot, &res_log);
            }
            else
            {
                printf("Invalid choice.\n");
            }
            break;
        case 10://show damaged robots
            display_dmgd();
            break;
        case 11://show repaired robots(fwd)
```



```
            display_rpd_fwd();
            break;
        case 12://show repaired robots(bwd)
            display_rpd_bwd();
            break;
        case 13://add a robot to circular priority redeployment list
            printf("Select a robot to add to redeployment list:\n");
            for (int i = 0; i < ROBOTS; i++)
            {
                printf("%d. %s\n", i + 1, robots[i]);
            }

            int cpr_ch;
            scanf("%d", &cpr_ch);

            if (cpr_ch >= 1 && cpr_ch <= ROBOTS)
            {
                add_cpr(robots[cpr_ch - 1], &res_log);
            }
            else
            {
                printf("Invalid choice! Try again.\n");
            }
            break;
        case 14://display the circular priority redeployment list
            display_cpr();
            break;
        case 15://deploy the next robot in the cycle(CPR list)
            deploy_robot(&res_log);
            break;
        case 16://exit the program
```

```
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }while (ch!=16);
}
```