停车场管理

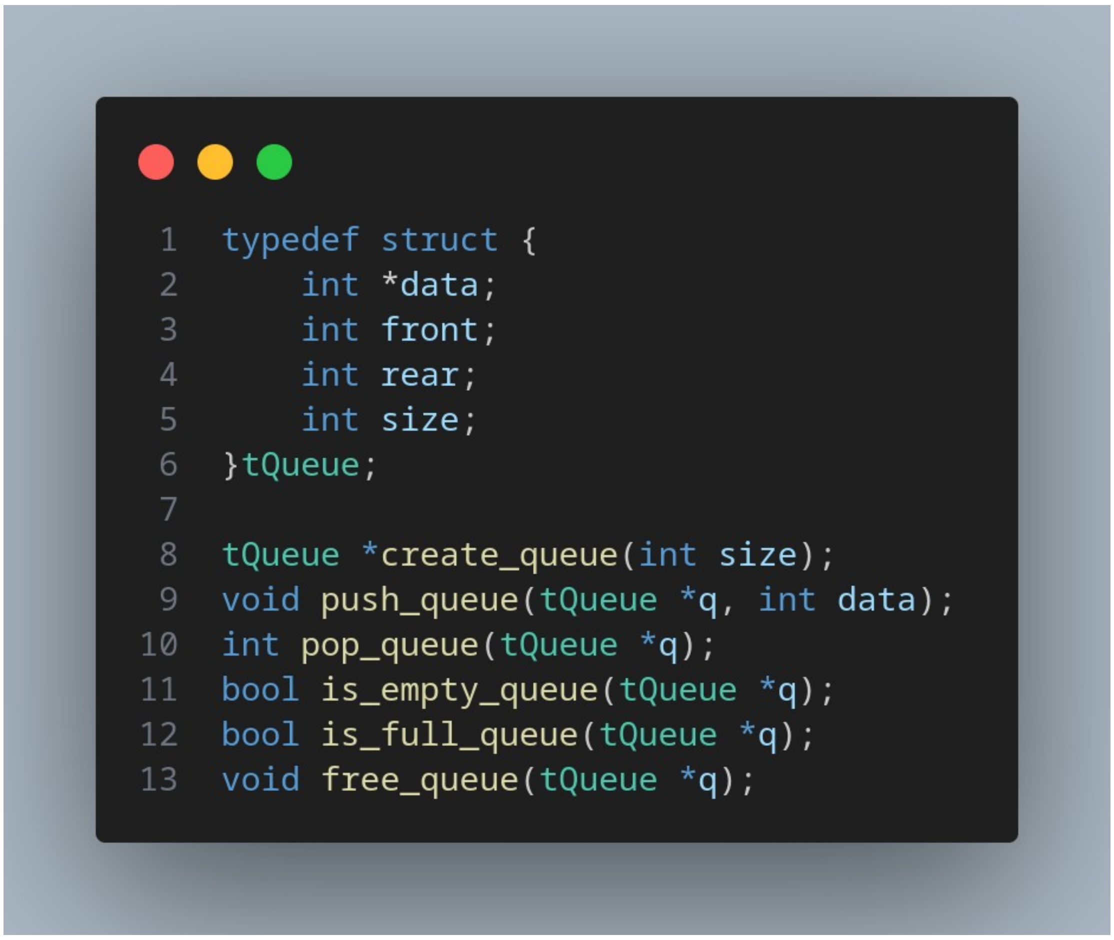
专业：计算机科学与技术 班级：23052322 姓名：张逸轩学号：23051214

1. 实验目的

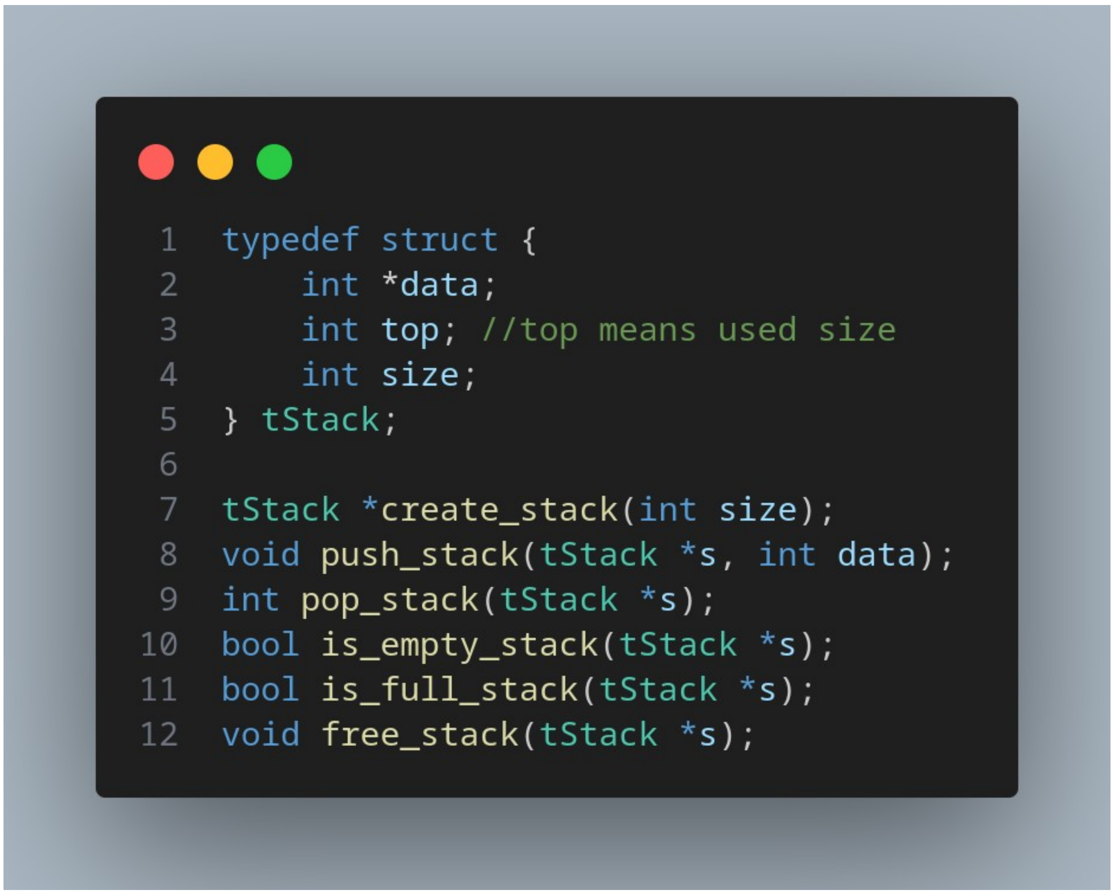
实际实现栈和队列，并且用于模拟。增加对栈和队列的认知。

2. 实验过程(实验方案、流程、程序等)(参考书上的格式需要写详细)

使用 c 语言，先设计栈和队列这两个数据结构如下：

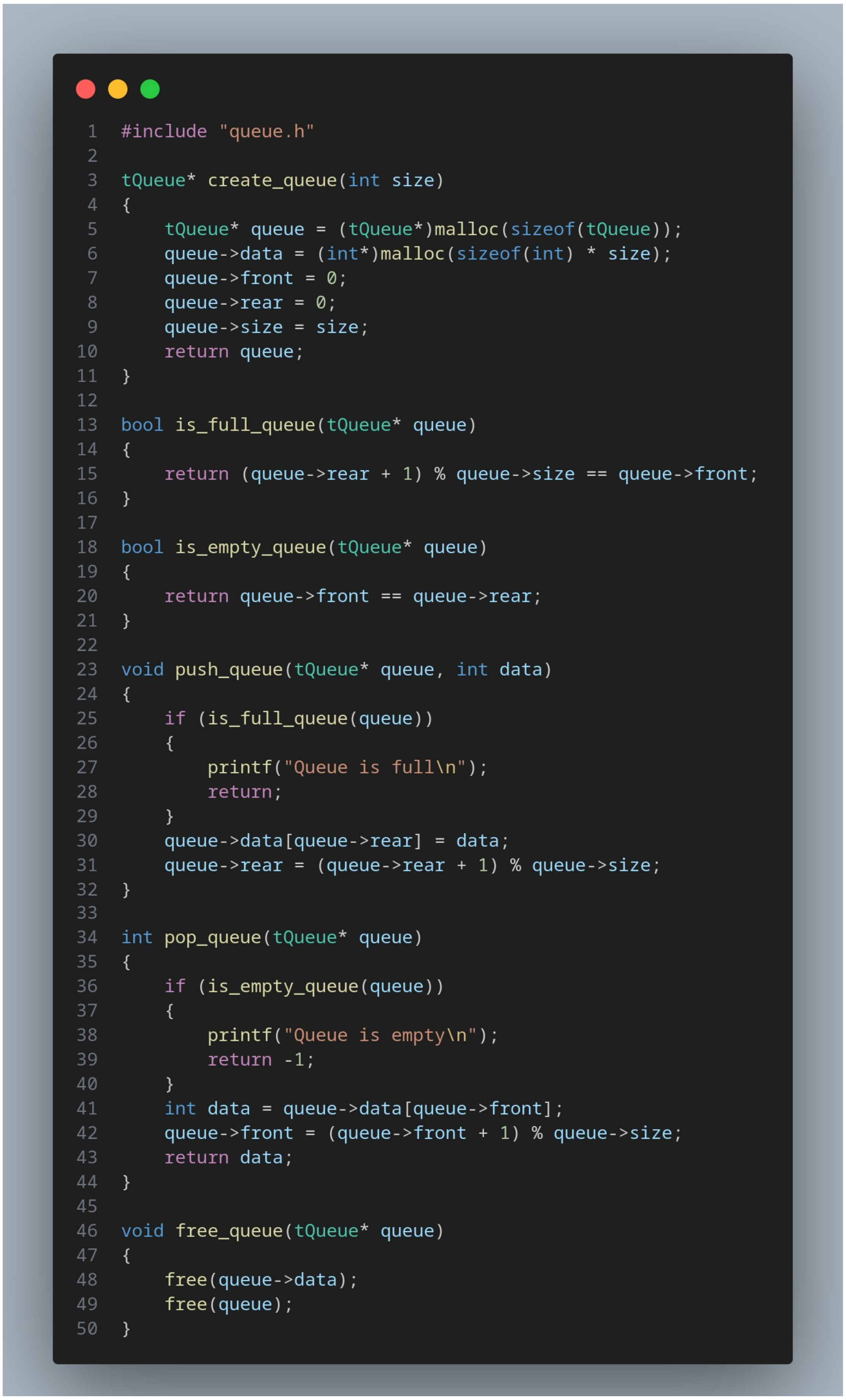


```c
typedef struct {
    int *data;
    int front;
    int rear;
    int size;
}tQueue;

tQueue *create_queue(int size);
void push_queue(tQueue *q, int data);
int pop_queue(tQueue *q);
bool is_empty_queue(tQueue *q);
bool is_full_queue(tQueue *q);
void free_queue(tQueue *q);
```

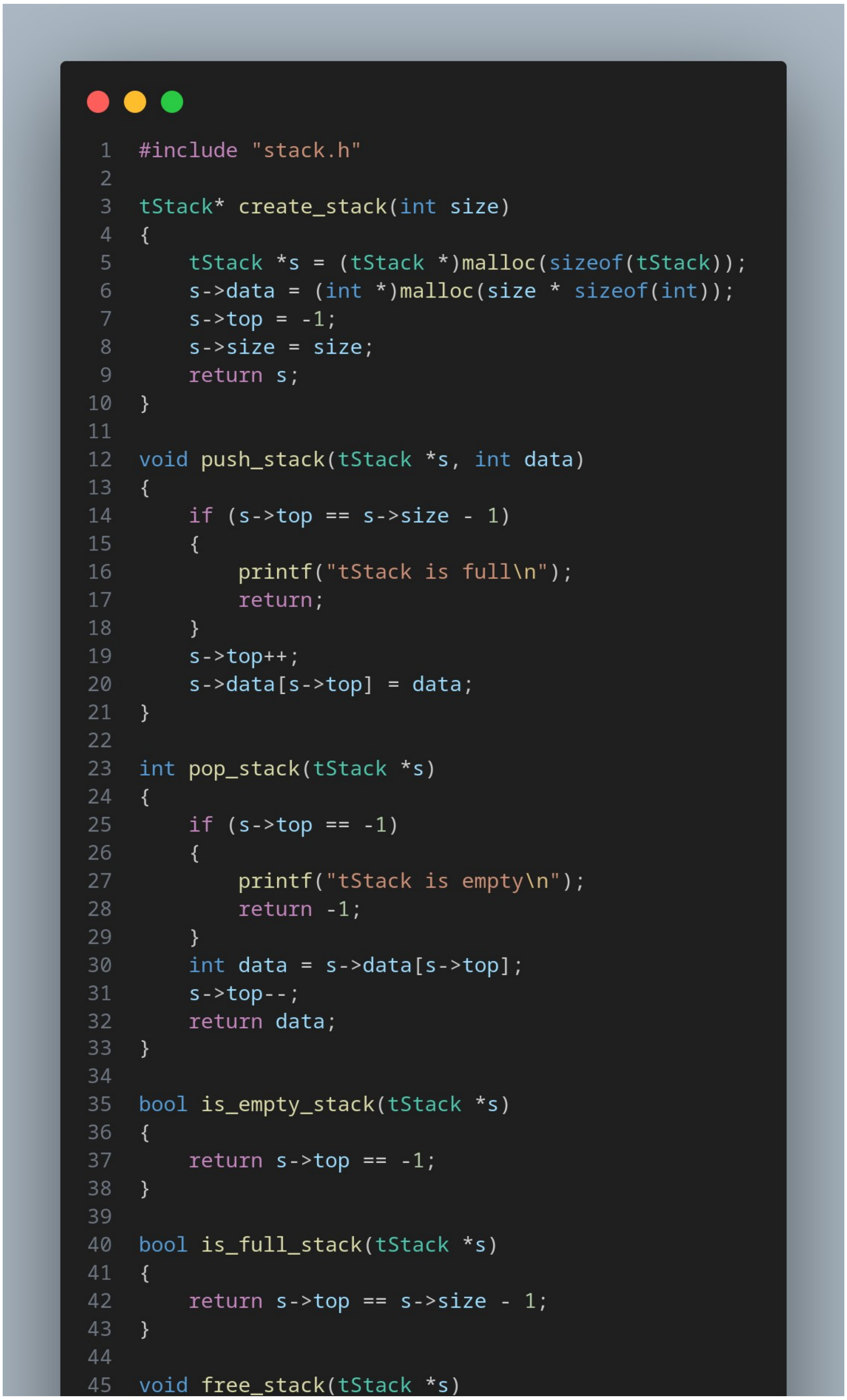


```c
typedef struct {
    int *data;
    int top; //top means used size
    int size;
} tStack;

tStack *create_stack(int size);
void push_stack(tStack *s, int data);
int pop_stack(tStack *s);
bool is_empty_stack(tStack *s);
bool is_full_stack(tStack *s);
void free_stack(tStack *s);
```

实现代码如下：

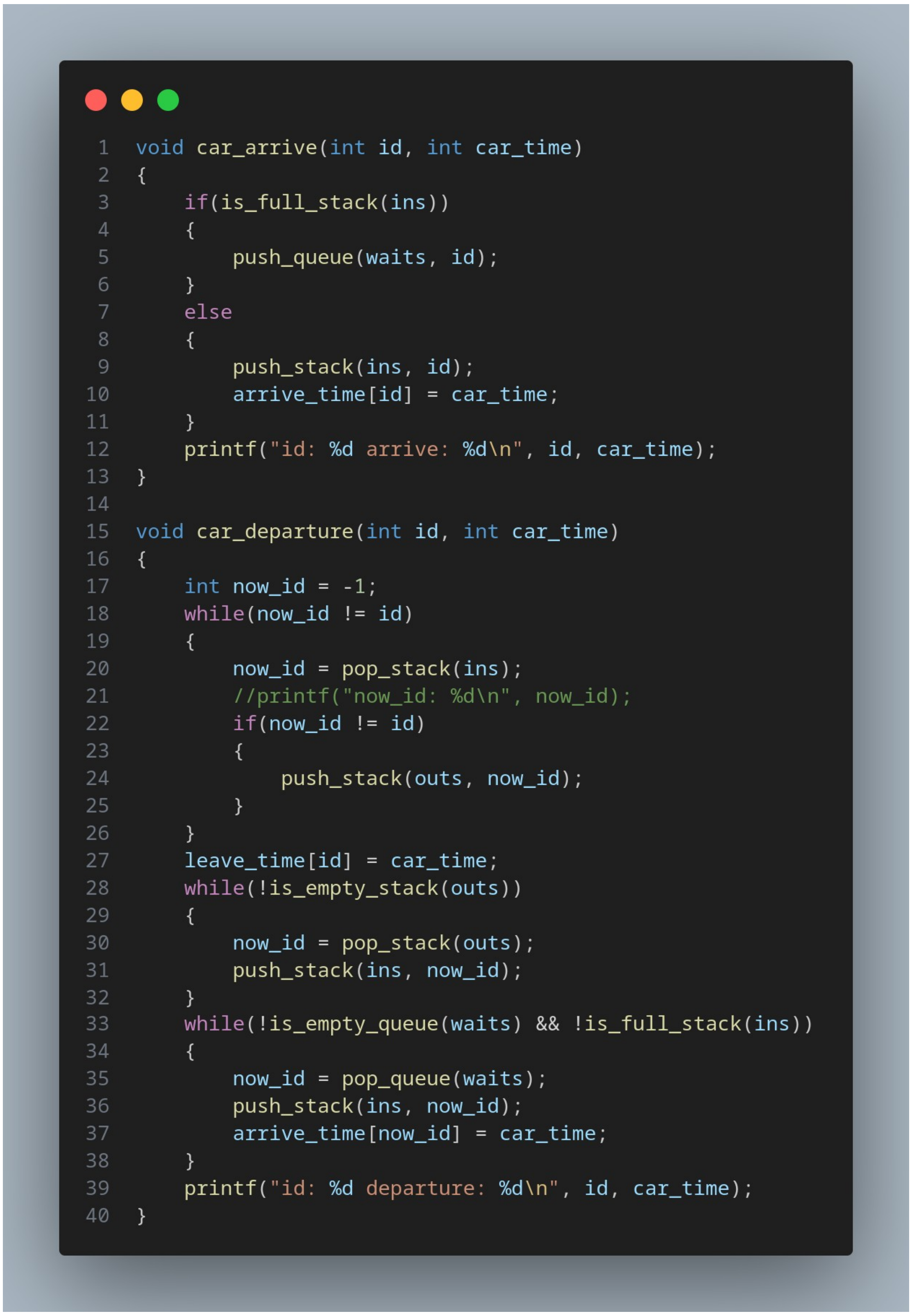


```c
#include "queue.h"

tQueue* create_queue(int size)
{
    tQueue* queue = (tQueue*)malloc(sizeof(tQueue));
    queue->data = (int*)malloc(sizeof(int) * size);
    queue->front = 0;
    queue->rear = 0;
    queue->size = size;
    return queue;
}

bool is_full_queue(tQueue* queue)
{
    return (queue->rear + 1) % queue->size == queue->front;
}

bool is_empty_queue(tQueue* queue)
{
    return queue->front == queue->rear;
}

void push_queue(tQueue* queue, int data)
{
    if (is_full_queue(queue))
    {
        printf("Queue is full\n");
        return;
    }
    queue->data[queue->rear] = data;
    queue->rear = (queue->rear + 1) % queue->size;
}

int pop_queue(tQueue* queue)
{
    if (is_empty_queue(queue))
    {
        printf("Queue is empty\n");
        return -1;
    }
    int data = queue->data[queue->front];
    queue->front = (queue->front + 1) % queue->size;
    return data;
}

void free_queue(tQueue* queue)
{
    free(queue->data);
    free(queue);
}
```



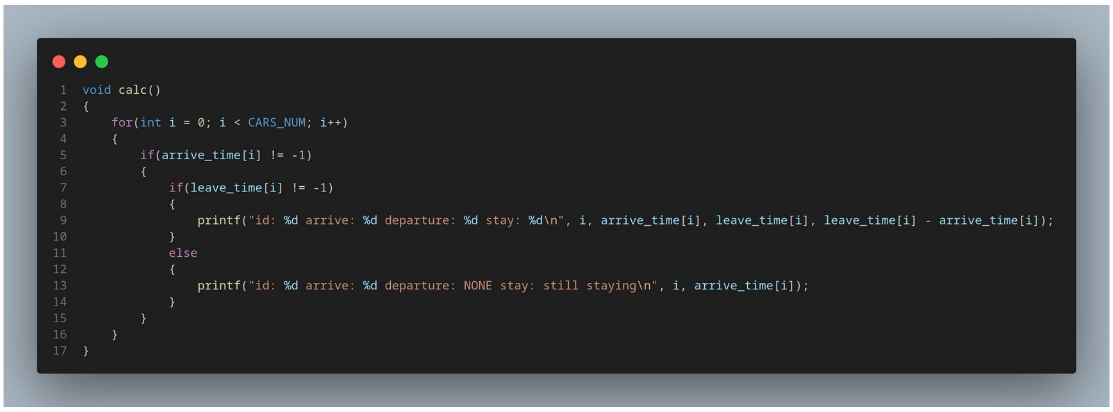
```c
#include "stack.h"

tStack* create_stack(int size)
{
    tStack *s = (tStack *)malloc(sizeof(tStack));
    s->data = (int *)malloc(size * sizeof(int));
    s->top = -1;
    s->size = size;
    return s;
}

void push_stack(tStack *s, int data)
{
    if (s->top == s->size - 1)
    {
        printf("tStack is full\n");
        return;
    }
    s->top++;
    s->data[s->top] = data;
}

int pop_stack(tStack *s)
{
    if (s->top == -1)
    {
        printf("tStack is empty\n");
        return -1;
    }
    int data = s->data[s->top];
    s->top--;
    return data;
}

bool is_empty_stack(tStack *s)
{
    return s->top == -1;
}

bool is_full_stack(tStack *s)
{
    return s->top == s->size - 1;
}

void free_stack(tStack *s)
```

然后参考题目意思，进行两种模拟：



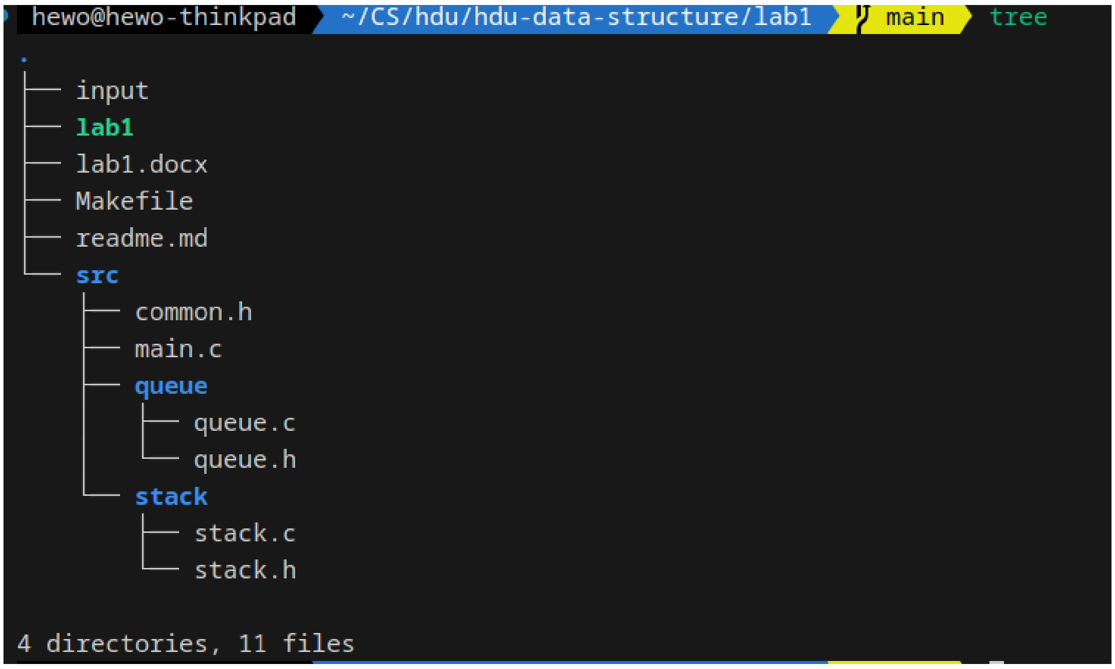
```c
void car_arrive(int id, int car_time)
{
    if(is_full_stack(ins))
    {
        push_queue(waits, id);
    }
    else
    {
        push_stack(ins, id);
        arrive_time[id] = car_time;
    }
    printf("id: %d arrive: %d\n", id, car_time);
}

void car_departure(int id, int car_time)
{
    int now_id = -1;
    while(now_id != id)
    {
        now_id = pop_stack(ins);
        //printf("now_id: %d\n", now_id);
        if(now_id != id)
        {
            push_stack(outs, now_id);
        }
    }
    leave_time[id] = car_time;
    while(!is_empty_stack(outs))
    {
        now_id = pop_stack(outs);
        push_stack(ins, now_id);
    }
    while(!is_empty_queue(waits) && !is_full_stack(ins))
    {
        now_id = pop_queue(waits);
        push_stack(ins, now_id);
        arrive_time[now_id] = car_time;
    }
    printf("id: %d departure: %d\n", id, car_time);
}
```

最后统计结果：



```c
void calc()
{
    for(int i = 0; i < CARS_NUM; i++)
    {
        if(arrive_time[i] != -1)
        {
            if(leave_time[i] != -1)
            {
                printf("id: %d arrive: %d departure: %d stay: %d\n", i, arrive_time[i], leave_time[i], leave_time[i] - arrive_time[i]);
            }
            else
            {
                printf("id: %d arrive: %d departure: NONE stay: still staying\n", i, arrive_time[i]);
            }
        }
    }
}
```

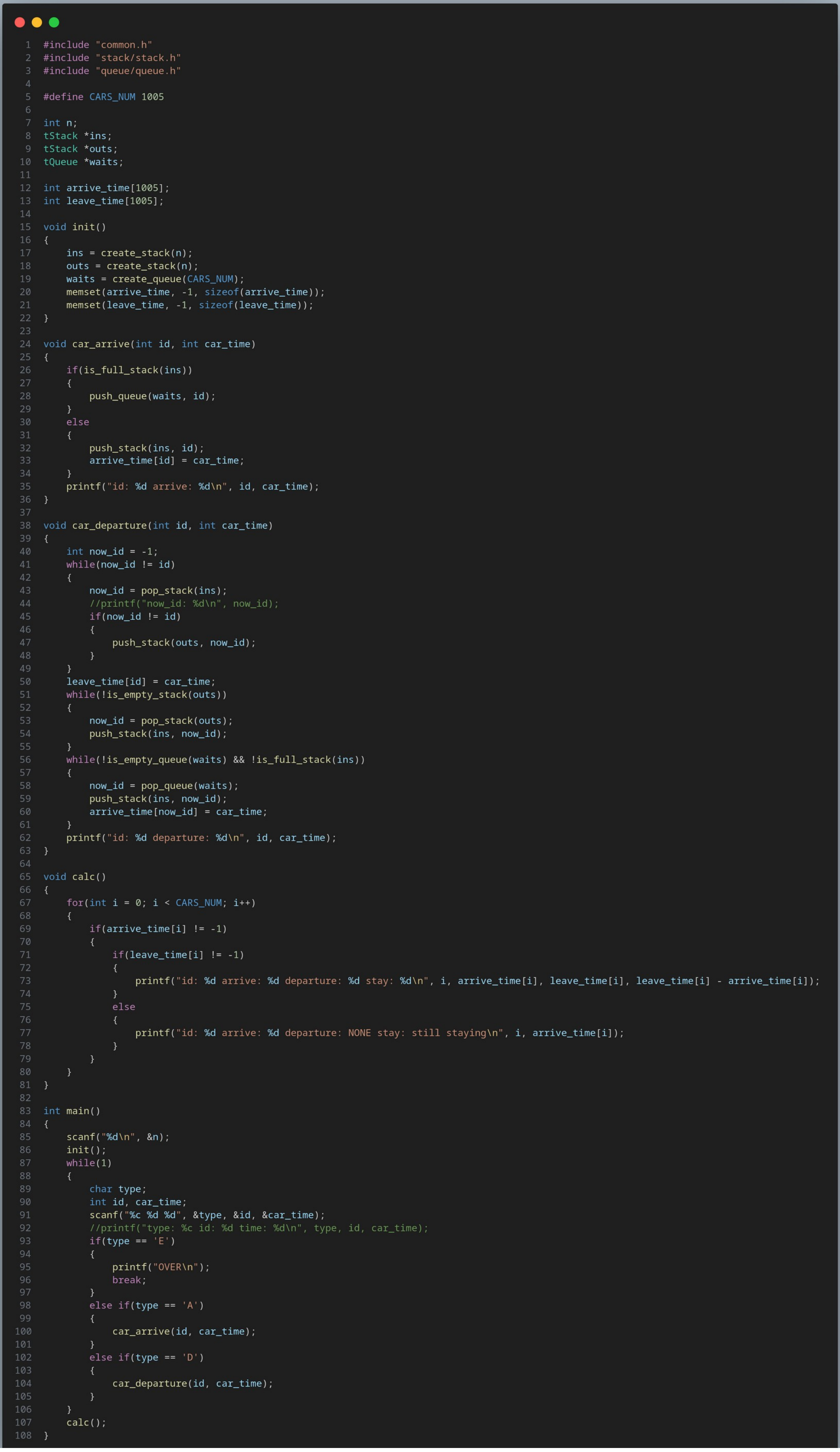
实际项目结构如图：



```
hewo@hewo-thinkpad  ~/CS/hdu/hdu-data-structure/lab1  main  tree
.
├── input
├── lab1
├── lab1.docx
├── Makefile
├── readme.md
└── src
    ├── common.h
    ├── main.c
    ├── queue
    │   ├── queue.c
    │   └── queue.h
    └── stack
        ├── stack.c
        └── stack.h

4 directories, 11 files
```

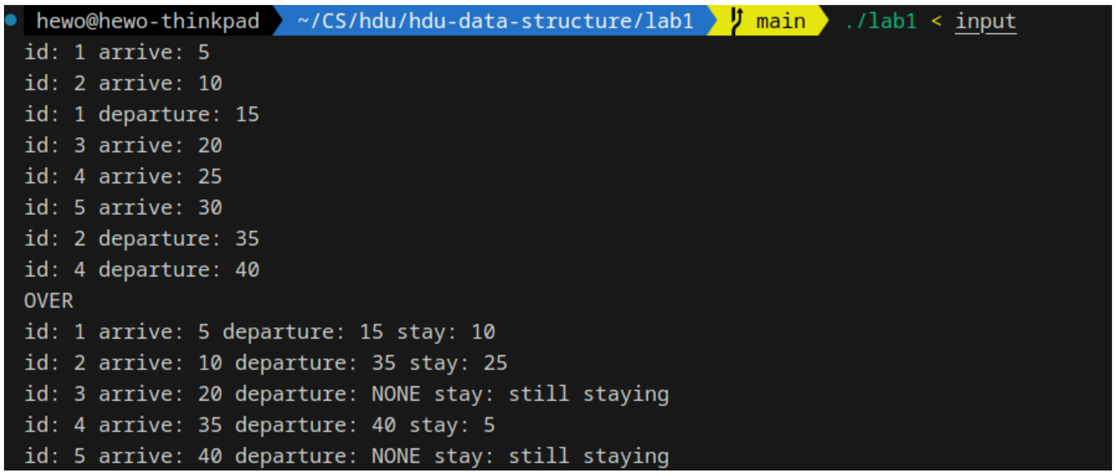
其中 input 提供数据，lab1 为可执行程序。

main.c 如下：



```c
#include "common.h"
#include "stack/stack.h"
#include "queue/queue.h"

#define CARS_NUM 1005

int n;
tStack *ins;
tStack *outs;
tQueue *waits;

int arrive_time[1005];
int leave_time[1005];

void init()
{
    ins = create_stack(n);
    outs = create_stack(n);
    waits = create_queue(CARS_NUM);
    memset(arrive_time, -1, sizeof(arrive_time));
    memset(leave_time, -1, sizeof(leave_time));
}

void car_arrive(int id, int car_time)
{
    if(is_full_stack(ins))
    {
        push_queue(waits, id);
    }
    else
    {
        push_stack(ins, id);
        arrive_time[id] = car_time;
    }
    printf("id: %d arrive: %d\n", id, car_time);
}

void car_departure(int id, int car_time)
{
    int now_id = -1;
    while(now_id != id)
    {
        now_id = pop_stack(ins);
        //printf("now_id: %d\n", now_id);
        if(now_id != id)
        {
            push_stack(outs, now_id);
        }
    }
    leave_time[id] = car_time;
    while(!is_empty_stack(outs))
    {
        now_id = pop_stack(outs);
        push_stack(ins, now_id);
    }
    while(!is_empty_queue(waits) && !is_full_stack(ins))
    {
        now_id = pop_queue(waits);
        push_stack(ins, now_id);
        arrive_time[now_id] = car_time;
    }
    printf("id: %d departure: %d\n", id, car_time);
}

void calc()
{
    for(int i = 0; i < CARS_NUM; i++)
    {
        if(arrive_time[i] != -1)
        {
            if(leave_time[i] != -1)
            {
                printf("id: %d arrive: %d departure: %d stay: %d\n", i, arrive_time[i], leave_time[i], leave_time[i] - arrive_time[i]);
            }
            else
            {
                printf("id: %d arrive: %d departure: NONE stay: still staying\n", i, arrive_time[i]);
            }
        }
    }
}

int main()
{
    scanf("%d\n", &n);
    init();
    while(1)
    {
        char type;
        int id, car_time;
        scanf("%c %d %d", &type, &id, &car_time);
        //printf("type: %c id: %d time: %d\n", type, id, car_time);
        if(type == 'E')
        {
            printf("OVER\n");
            break;
        }
        else if(type == 'A')
        {
            car_arrive(id, car_time);
        }
        else if(type == 'D')
        {
            car_departure(id, car_time);
        }
    }
    calc();
}
```

3.   实验结果及结果分析



```
● hewo@hewo-thinkpad  ~/CS/hdu/hdu-data-structure/lab1  main  ./lab1 < input
id: 1 arrive: 5
id: 2 arrive: 10
id: 1 departure: 15
id: 3 arrive: 20
id: 4 arrive: 25
id: 5 arrive: 30
id: 2 departure: 35
id: 4 departure: 40
OVER
id: 1 arrive: 5 departure: 15 stay: 10
id: 2 arrive: 10 departure: 35 stay: 25
id: 3 arrive: 20 departure: NONE stay: still staying
id: 4 arrive: 35 departure: 40 stay: 5
id: 5 arrive: 40 departure: NONE stay: still staying
```

实际结果如上图，可以实现停车场的模拟，（上文是输入数据的可读化），最后统计每个车在停车场的时间（未离去则告知）

4.   实验总结

学习了数据结构 栈 和队列的实现，实际操作发现了容易出错的点位，同时明白了栈和队列在实际中的用途。