《数据结构（A）》第 3 章设计型作业

栈与队的应用

专　业：计算机科学与技术

班　级：_____

学生姓名：_____

学　号：_____

北京交通大学计算机与信息技术学院

2021 年 10 月 10 日

《数据结构（A)》第 3 章设计型作业[①]

提醒同学们在进行作业工作之前，认真阅读教材第 3 章“栈和队列”，以及《数据结构题集（C 语言版)》，第 72 页至第 75 页，并努力按照其要求来完成本章的设计型作业题目。

设计型作业题目[②]

3.3 （要求学号末位为奇数的学生必作，学号末位为偶数的学生选作。可以参考《数据结构题集（C 语言版)》，第 96 页，实习 2 栈和队列及其应用，第 2.1 题：停车场管理，难度系数为 3。题目有些改变）某商场有一个 100 个车位的停车场，当车位未满时，等待的车辆可以进入并计时；当车位已满时，必须有车辆离开，等待的车辆才能进入；当车辆离开时计算停留的时间，并且按照每小时 10 元收费。

汽车的输入信息格式可以是（进入/离开，车牌号，进入/离开时间)，要求可以随时显示停车场内的车辆信息以及收费历史记录。

3.4 （要求学号末位为偶数的学生必作，学号末位为奇数的学生选作。这个题目不是《数据结构题集（C 语言版)》，第 100 页，实习 2 栈和队列及其应用，第 2.6 题：银行业务模拟，习题集上题目的难度系数为 5。留作研究型题目吧！）某银行营业厅共有 6 个营业窗口，设有排队系统语音叫号。该银行的业务分为公积金、个人卡、企业卡等 3 种。公积金业务指定 1 号窗口，个人卡业务指定 2、3、4 号窗口，企业卡业务指定 5、6 号窗口。但如果 5、6 号窗口全忙，而 2、3、4 号窗口有空闲时，企业卡业务也可以在空闲的 2、3、4 号窗

---

① 这是《数据结构（A)》第 3 章的设计型作业题目。提交截止日期是 2021 年 10 月 20 日。我们的课程是计算机类专业最重要的课程，作业比较多呀。

② 本章设计型作业题目的序号接着本章“基本作业”题目的序号进行编排。

口之一办理。

客户领号、业务完成可以作为输入信息，要求可以随时显示 6 个营业窗口的状态。

3.5 （所有学生必作，《数据结构题集（C 语言版）》，第 26 页，第 3.32 题的简化版：原题难度系数为 4。参考本章课堂幻灯片。）

原第 3.32 题（参考本章幻灯片）：

试利用循环队列编写求 k 阶 Fibonacci 序列中前 n+1 项（f0, f1 , f2 ,… f_n)的算法，要求满足 $f_n \leqslant max$ 而 $f_{n+1} > max$，其中 max 为某个约定的常数。（注意本题所用循环队列的容量仅为 k,则在算法执行结束时，留在循环队列中的元素应是 k 阶斐波那契序列中的最后 k 项 f_{n-k+1}, … f_n)。

本题具体化为下列要求：

已知 4 阶 Fibonacci 斐波那契序列如下：f0=f1=f2=0, f3= 1, … , $f_i = f_{i-1}+f_{i-2}+f_{i-3}+f_{i-4}$，利用容量为 k=4 的循环队列，构造序列的前 n+1 项（f0, f1 , f2 ,… f_n），要求满足 $f_n \leqslant 200$ 而 $f_{n+1} > 200$。

3.6 （选作题，八皇后问题，Eight Queens Problem）：设 8 皇后问题的解为(x1, x2, x3, …,x8),约束条件为：在 8x8 的棋盘上，其中任意两个 xi 和 xj 不能位于棋盘的同行、同列及同对角线。要求用一位数组进行存储，输出所有可能的排列。

3.7 （所有学生必作。《数据结构题集（C 语言版）》，第 105 页，实习 2 栈和队列及其应用，第 2.9 题：迷宫问题，Maze Problem，题目难度系数为 4。请同学们务必阅读第 105 页至第 115 页，并严格按照习题集之上的要求撰写设计报告。）

迷宫求解：用二维矩阵表示迷宫，自动生成或者直接输入迷宫的格局，确定迷宫是否能走通，如果能走通，输出行走路线。

设计型作业题目解答

【第 3.3 题解答】

思路：

猜测等待的车辆以队列结构排列，可以使用宏变量进行调整停车场的容量，方便测试。

只需要在有车辆出场时，把等待车辆中队列第一个结点插入到停车场中并开始计时。当出场时，停止计时，计算 fee，然后加入到计费信息中。

```
01:#include <iostream>
02:#include <time.h>
03:#include <string.h>
04:#define time_speed 3600
05:#define parking_count 100
06:// 时间测试中，车辆入场出场时间太短，这里由宏定义修改时间倍率进行调整
07:// 这里以一秒为一小时计算， 现实中倍率为1
08:// parking_count 为了方便测试用 当其比较小时方便测试
09:
10:struct vehicle{
11:    char id[10];
12:    time_t start_time;
13:    time_t end_time;
14:    int duration;
15:    int fee;
16:    int count;
17:    vehicle* next;
18:
19:};
20://三个车辆链表 停车进行中的 出场的 等待的
21:struct parking_lot{
22:    vehicle* vehicle_ongoing;
23:    vehicle* vehicle_over;
24:    vehicle* waiting_vehicle;
25:};
26:void waiting_vehicle_add(parking_lot*lot, vehicle*p){
27:    vehicle*q= (vehicle*)malloc(sizeof(vehicle));
28:    vehicle*r = lot->waiting_vehicle;
29:
30:    while(r->next != NULL){
```

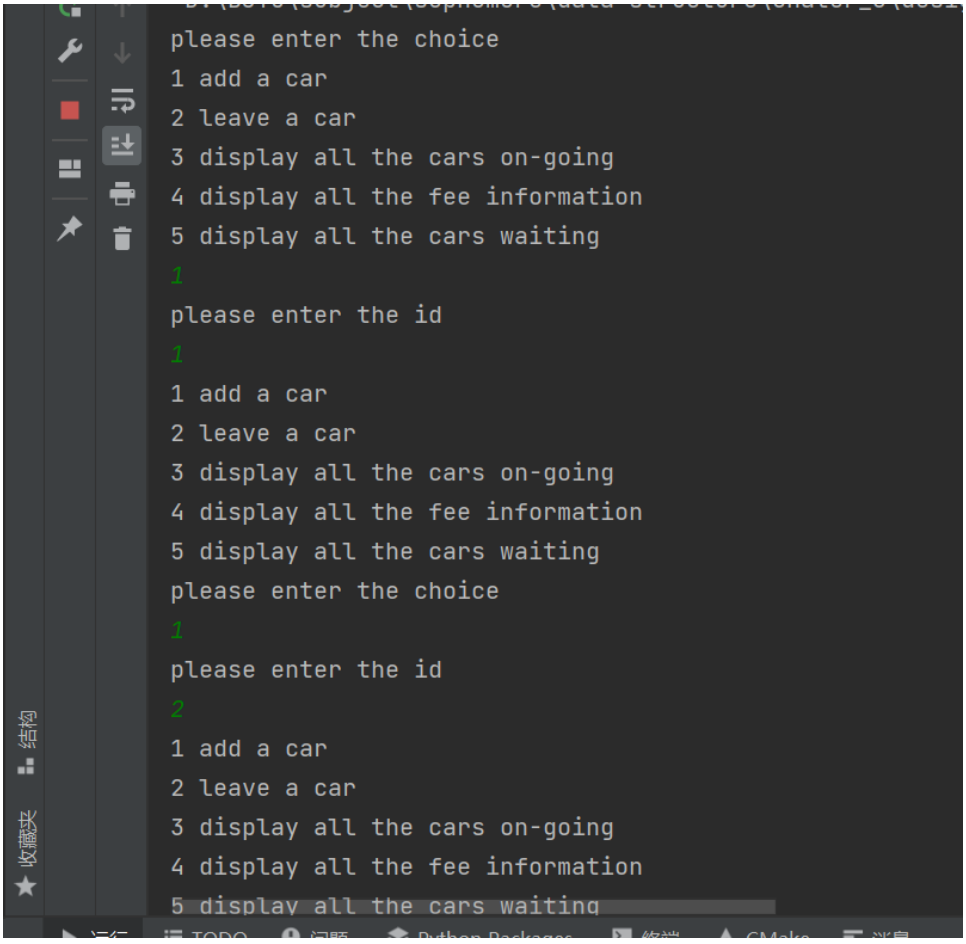
```
31:         r = r->next;
32:     }
33:
34:     *q = *p;
35:     r->next = q;
36:}
37://加入停车场 开始计费
38:void add_vehicle(parking_lot*l, vehicle*p){
39:     if(l->vehicle_ongoing->count >=parking_count){
40:         printf("the parking lot is full, please wait\n");
41:     }
42:     else{
43:         vehicle*q= (vehicle*)malloc(sizeof(vehicle));
44:         *q = *p;
45:         q->next= l->vehicle_ongoing->next;
46:         l->vehicle_ongoing->next = q;
47:
48:         l->vehicle_ongoing->count++;
49:     }
50:}
51:离开停车场  从计费场中剔除 完成计费加入完成场
52:int leave_vehicle(parking_lot*l, char*s){
53:     vehicle *p = l->vehicle_ongoing->next;
54:     vehicle *q = (vehicle*) malloc(sizeof(vehicle));
55:     vehicle *r = l->vehicle_ongoing;
56:
57:     if(l->vehicle_ongoing->count==0) return -1;
58:     while(p!=NULL){
59:         if(strcmp(s, p->id) == 0){
60:             p->end_time = time(0);
61:             p->duration = difftime(p->end_time, p->start_time);
62:             p->fee = p->duration * time_speed / 3600.0 * 10;//进行倍率调试
63:             printf("leave success\n");
64:
65:             *q= *p;
66:             q->next=l->vehicle_over->next;
67:             l->vehicle_over->next = q;
68:
69:             r->next = p->next;
70:             free(p);
71:
72:
73:             l->vehicle_ongoing->count--;
```

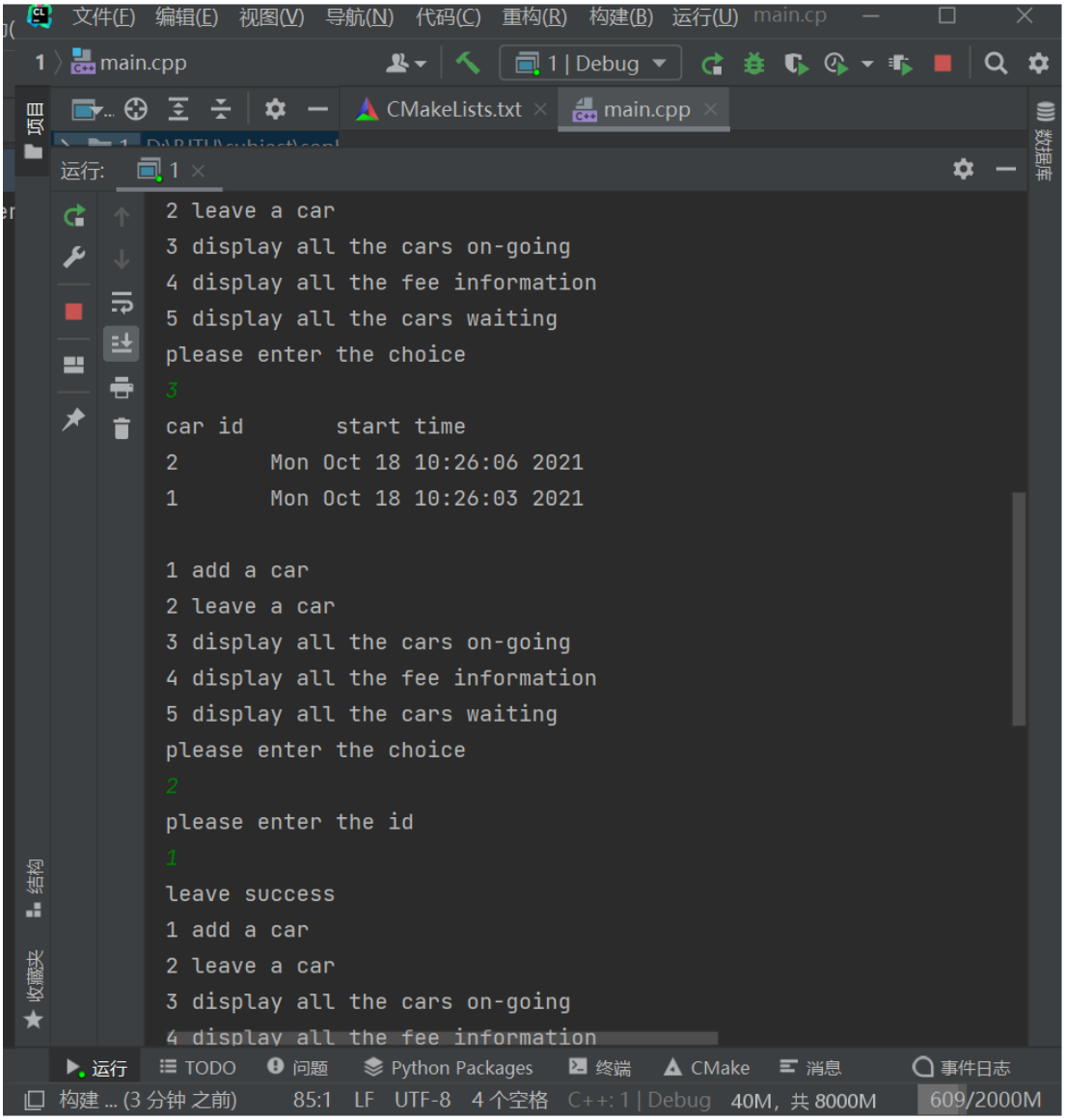
```
 74:            l->vehicle_over->count++;
 75:            break;
 76:        }
 77:        else{
 78:            p=p->next;
 79:            r=r->next;
 80:        }
 81:    }
 82:    return 0;
 83:}
 84:
 85:int on_going_display(parking_lot*l){
 86:    vehicle*p = l->vehicle_ongoing->next;
 87:
 88:    printf("car id\t     start time\n");
 89:    if(p==NULL) return -1;
 90:    while(p->next!=NULL){
 91:        printf("%s\t%s", p->id, ctime(&(p->start_time)));
 92:        p= p->next;
 93:    }
 94:    printf("%s\t%s\n", p->id, ctime(&(p->start_time)));
 95:    p= p->next;
 96:    return 0;
 97:}
 98:
 99:int waiting_display(parking_lot*l){
100:    vehicle*p = l->waiting_vehicle->next;
101:
102:    printf("car id\t     start time\n");
103:    if(p==NULL) return -1;
104:    while(p->next!=NULL){
105:        printf("%s\t%s", p->id, ctime(&(p->start_time)));
106:        p= p->next;
107:    }
108:    printf("%s\t%s\n", p->id, ctime(&(p->start_time)));
109:    p= p->next;
110:    return 0;
111:}
112:char* delete_final_character(char*t2){
113:    char*t = (char*) malloc(sizeof(char)*25);
114:    strcpy(t, t2);
115:    t[strlen(t)-1]='\0';
116:    return t;
```

```
117:}
118:
119:int over_display(parking_lot*l){
120:    vehicle*p = l->vehicle_over->next;
121:    if(p ==NULL) return -1;
122:    printf("car id\t         star time\t        end time\t\t duration\t
    fee\n");
123:    while(p->next!=NULL){
124:        printf("%s \t%s  %s  \t     %d   \t %d\n", p-
    >id,delete_final_character(ctime(&(p->start_time))),
125:               delete_final_character(ctime(&(p->end_time))), p->duration,
    p->fee);
126:        p = p->next;
127:    }
128:    printf("%s \t%s  %s  \t     %d   \t %d\n", p->id,
    delete_final_character(ctime(&(p->start_time))),
129:           delete_final_character(ctime(&(p->end_time))), p->duration, p-
    >fee);
130:    p= p->next;
131:    return 0;
132:}
133:
134:
135:
136:int main(){
137:    parking_lot *pl = (parking_lot*) malloc(sizeof(parking_lot));
138:    pl->vehicle_over =(vehicle*) malloc(sizeof(vehicle));
139:    pl->vehicle_ongoing = (vehicle*) malloc(sizeof(vehicle));
140:    pl->vehicle_ongoing->count = pl->vehicle_over->count =0;
141:    pl->waiting_vehicle = (vehicle*) malloc(sizeof(vehicle));
142:// 初始化 链表
143:    pl->vehicle_over->next = NULL;
144:    pl->vehicle_ongoing->next = NULL;
145:    pl->waiting_vehicle->next = NULL;
146:
147:
148:    vehicle*tem = (vehicle*) malloc(sizeof(vehicle));
149:    vehicle* tem1;
150:    tem->next=NULL;
151:
152:    int choice;
153:    char* s;
154:    printf("please enter the choice\n");
```

```
155:
156:    printf("1 add a car\n");
157:    printf("2 leave a car\n");
158:    printf("3 display all the cars on-going\n");
159:    printf("4 display all the fee information\n");
160:    printf("5 display all the cars waiting\n");
161://提示信息
162:    scanf("%d", &choice);
163:    while(choice!=0){
164:
165:        switch(choice){
166:            case 1:
167:                printf("please enter the id\n");
168:                scanf("%s",&(tem->id));
169:                tem->start_time = time(0);
170:                if(pl->vehicle_ongoing->count>=parking_count){
171:                    printf("the parking lot is full, please wait\n");
172:                    waiting_vehicle_add(pl, tem);
173:                }
174:                else{
175:                    add_vehicle(pl, tem);
176:                }
177:                break;
178:            case 2:
179:                printf("please enter the id\n");
180:                scanf("%s", &(tem->id));
181:                leave_vehicle(pl, tem->id);
182:// 有车离开则可以把等待场中的车辆计入到计费场中
183:                if(pl->vehicle_ongoing->count < parking_count){
184:                    if(pl->waiting_vehicle->next!=NULL){
185:                        tem1 = pl->waiting_vehicle->next;
186:                        add_vehicle(pl, tem1);
187:                        pl->waiting_vehicle->next = pl->waiting_vehicle-
188:>next->next;
189:                        free(tem1);
190:                    }
191:                }
192:                break;
193:            case 3:
194:                on_going_display(pl);
195:                break;
196:
197:            case 4:
```

```
198:                over_display(pl);
199:                break;
200:            case 5:
201:                waiting_display(pl);
202:                break;
203:        }
204:        printf("1 add a car\n");
205:        printf("2 leave a car\n");
206:        printf("3 display all the cars on-going\n");
207:        printf("4 display all the fee information\n");
208:        printf("5 display all the cars waiting\n");
209:
210:        printf("please enter the choice\n");
211:        scanf("%d", &choice);
212:
213:    }
214:
215:};
```



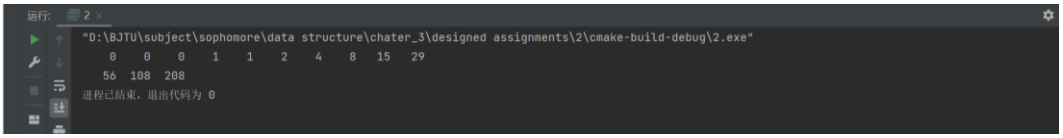

文件(F) 编辑(E) 视图(V) 导航(N) 代码(C) 重构(R) 构建(B) 运行(U) main.cp
main.cpp
1 | Debug
CMakeLists.txt
main.cpp
运行:
1
2 leave a car
3 display all the cars on-going
4 display all the fee information
5 display all the cars waiting
please enter the choice
3
car id start time
2 Mon Oct 18 10:26:06 2021
1 Mon Oct 18 10:26:03 2021
1 add a car
2 leave a car
3 display all the cars on-going
4 display all the fee information
5 display all the cars waiting
please enter the choice
2
please enter the id
1
leave success
1 add a car
2 leave a car
3 display all the cars on-going
4 display all the fee information
运行
TODO
问题
Python Packages
终端
CMake
消息
事件日志
构建 ... (3 分钟 之前)
85:1 LF UTF-8 4 个空格 C++: 1 | Debug 40M，共 8000M
609/2000M

调试成功

【第 3.5 题解答】

思路：使用容量为 4 的循环队列，因为生成的数列一定会把队列填满，所以不需要判满，即 front 和 rear 没有意义。只需要保持循环队列灵魂的取模转头即可。

```
01:#include <stdio.h>
02:
03:// 永远是满队列 其实可以不用front和rear进行判定
04:struct SqQueue{
05:    int date[4];
06:    int count;
07:    int rear;
08:};
09:
10:int main() {
11:    SqQueue q;
12:    q.count = 4;
13:    q.date[0] = q.date[1] = q.date[2] = 0;
14:    q.date[3]=1;
15:    q.rear = 0;
16:    int tem=0;
17:    printf("%5d%5d%5d%5d", 0, 0, 0, 1);
18:    while(tem<=200){
19:        tem = q.date[0] + q.date[1] + q.date[2] + q.date[3];
20:        q.date[q.rear]= tem;
21:        printf("%5d", tem);
22:        q.count++;
23://保留循环队列最经典的取模 进行转圈
24:        q.rear = (q.rear+1)%4;
25:        if(q.count%10== 0){
26:            printf("\n");
27:        }
28:    }
29:
30:
31:}
```



【第 3.7 题解答】

思路：

以栈进行回溯操作，寻找路的操作为从东边开始顺时针旋转。找路时，要避免不是来路，即与已入栈的结点的 data 域的坐标进行对比。无路回撤是要与已入栈的结点的 data 域的坐标进行对比，从上次已探索的方向开始继续探索，避免死循环。

```
01:#include <stdio.h>
02:#include <random>
03:#include <stdlib.h>
04:struct Maze{
05:    int data[8][8];
06:};
07:
08:struct Node{
09:    int a;
10:    int b;
11:    Node* next;
12:};
13:
14:struct Map_stack{
15:    Node* head;
16:};
17:
18:Maze*create_maze(){
19:
20:    // 创建迷宫 可以用随机数创建 但是迷宫满足要求的几率不高 遂自定义生成
21:    Maze*maze=(Maze*) malloc(sizeof(Maze));
22:    int array[8][8] ={
```

```
23:         { 0,0, 0, 0, 0, 0, 0 ,0 },
24:         { 0,1, 1, 1, 1, 1, 0, 0 },
25:         { 0,1, 0, 1, 0, 1, 0, 0 },
26:         { 0,1, 1, 1, 1, 1, 0, 0 },
27:         { 0,1, 0, 0, 0, 1, 1, 0 },
28:         { 0,1, 1, 1, 1, 0, 0, 0 },
29:         { 0, 1, 1, 0, 1, 1, 1 ,0},
30:         { 0,0, 0, 0, 0, 0, 0 ,0 },
31:     };//外带一圈0 避免穿墙
32:     for(int i= 0; i<8; i++) {
33:         for (int j = 0; j < 8; j++) {
34:             maze->data[i][j] = array[i][j];
35:         }
36:     }
37:
38:     return maze;
39:}
40:// 压入栈中
41:
42:void push(Map_stack*mapStack, Node*node){
43:     Node* tem = (Node*) malloc(sizeof (Node));
44:     *tem = *node;
45:     tem->next = mapStack->head->next;
46:     mapStack->head->next = tem;
47:
48:}
49://出栈回溯
50:void pop(Map_stack*mapStack){
51:     Node*tem = mapStack->head->next;
52:     mapStack->head->next = mapStack->head->next->next;
53:     free(tem);
54:}
55:
56:Maze* find_map(Map_stack*mapStack, Maze*maze) {
57:     int x = 1;
58:     int y = 1;
59:     int choice = 1;
60:
61:     Node *tem = (Node*) malloc(sizeof(Node));
62:     tem->a = x;
63:     tem->b = y;
64:     //
65:     tem->next = NULL;
```

```
    push(mapStack, tem);


    while (x != 6 || y != 6) {

        if(mapStack->head->next == NULL){
            printf("no way\n");
            break;
        }

        switch (choice){
            case 1:
                // east
// 能走到则入账 即判断是否为来路时需 回溯两个 这是避免第一个结点被回溯无两个结
//点 遂保留两个空结点
                 if (maze->data[x][y+1] == 1 && (x != (mapStack->head-
>next->next->a) || (y+1) != mapStack->head->next->next->b)){
                    tem->a = x;
                    tem->b = ++y;
                    push(mapStack, tem);
                    choice = 1;
                    continue;
                }

            case 2:
                // south
                if(maze->data[x+1][y] == 1 && ((x+1) != mapStack->head-
>next->next->a || y != mapStack->head->next->next->b)){
                    tem->a = ++x;
                    tem->b = y;
                    push(mapStack, tem);
                    choice = 1;
                    continue;
                }
            case 3:
                // west
                if (maze->data[x][y-1] == 1 && (x != mapStack->head-
>next->next->a || (y-1) != mapStack->head->next->next->b)){
                    tem->a = x;
                    tem->b = --y;
                    push(mapStack, tem);
                    choice = 1;

```

```
109:                    continue;
110:                }
111:            case 4:
112:                // north
113:                if (maze->data[x-1][y] == 1 && ((x-1) != mapStack-
114:>head->next->next->a || y != mapStack->head->next->next->b)) {
115:                    tem->a = --x;
116:                    tem->b = y;
117:                    push(mapStack, tem);
118:                    continue;
119:                }
120:            default:
121:                // return
122:// 顺时针旋转找出路 避免是来路 当回溯时通过坐标对比判定已经寻找过的方向 由
123://choice 进行指定开始的方向
124:                if (mapStack->head->next->a - mapStack->head->next-
125:>next->a == 1 &&
126:                    mapStack->head->next->b - mapStack->head->next-
127:>next->b == 0)
128:                    choice = 3;
129:                if (mapStack->head->next->a - mapStack->head->next-
130:>next->a == 0 &&
131:                    mapStack->head->next->b - mapStack->head->next-
132:>next->b == -1)
133:                    choice = 4;
134:                if (mapStack->head->next->a - mapStack->head->next-
135:>next->a == 0 &&
136:                    mapStack->head->next->b - mapStack->head->next-
137:>next->b == 1)
138:                    choice = 2;
139:                pop(mapStack);
140:                x = mapStack->head->next->a;
141:                y = mapStack->head->next->b;
142:                continue;
143:        }
144:    }
145:}
146:
147:void mapDisplay(Map_stack*mapStack){
148:    Node* p = mapStack->head->next;
149:    if(mapStack->head->next == NULL)
150:        printf("no way\n");
151:
```

```
152:    while(p->next != NULL){
153:
154:        int n =0;
155:        printf("(%d,%d)<-",p->a, p->b);
156:        n++;
157:        if(n%10 == 0)
158:            printf("\n");
159:        p = p->next;
160:    }
161://    printf("(%d,%d)",p->a, p->b);
162:}
163:
164:
165:
166:
167:void mazeDisplay(Maze*maze){
168:    for(int i=0; i<8; i++){
169:        for(int j=0; j<8; j++){
170:            printf("%3d", maze->data[i][j]);
171:        }
172:        printf("\n");
173:    }
174:
175:}
176:
177:
178:int main() {
179:    Maze*maze = create_maze();
180:    mazeDisplay(maze);
181:
182:    Map_stack*mapStack = (Map_stack*)malloc(sizeof(Map_stack));
183:    mapStack->head = (Node*)malloc(sizeof (Node));
184:    mapStack->head->next = (Node*)malloc(sizeof (Node));
185:    mapStack->head->next->next =NULL;
186:
187:    find_map(mapStack, maze);
188:    mapDisplay(mapStack);
189:
190:}
```