**数据结构实验报告**

# 实验二：栈和队列

专 业： 网络工程

班 级： 1501

姓 名： 邓泽先

学 号： 150200325

指导教师： 王梅

日 期： 2016-11-02

**一、实验目的**

（1）掌握栈、队列的思想及其存储实现。

（2）掌握栈、队列的常见算法的程序实现。

（3）明确栈、队列均是特殊的线性表。

**二、实验内容**

**1．栈的实现及应用**

（1）采用链式存储实现栈的初始化、入栈、出栈操作。

（2）采用顺序存储实现栈的初始化、入栈、出栈操作。

（3）设表达式以字符形式已存入数组E[n]中，‘#’为表达式的结束符，试写出判断表达式中括号‘（’、‘）’、‘[’、‘]’是否配对的C语言描述算法。

（4）在主函数中设计一个简单的菜单，分别测试上述算法。

**2. 队列的实现**

（1）采用链式存储实现队列的初始化、入队、出队操作。

（2）采用顺序存储实现循环队列的初始化、入队、出队操作。

（3）在主函数中设计一个简单的菜单，分别测试上述算法。

**3.迷宫问题**

【问题描述】

以一个m\*n的长方阵表示迷宫，0和1分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

【基本要求】

首先实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得的通路以三元组(i,j,d)的形式输出，其中(i,j)指示迷宫中的一个坐标，d表示走到下一坐标的方向。如:对于下列数据的迷宫，输出的一条通路为(1,1,1),(1,2,2),(2,2,2)，(3,2,3),(3,1,2),…。

【选作内容】

以方阵形式输出迷宫及其通路。

**三、需求分析**

**1.实现链式储存和顺序储存的栈的初始化、入栈和出栈操作，并实现可以检验输入的字符中括号‘（’、‘）’、‘[’、‘]’是否配对的算法。**

**2.实现链式储存和顺序储存的队列的初始化、入队和出队操作。**

**3.通过文件读取获取迷宫，得到迷宫的一条通路。**迷宫中‘1’代表不可通过的地方，‘0’代表可以行走的地方。‘\*’代表通路上的路径，‘#’代表走过的但是不能到达终点的地方。

**四、概要设计**

1.栈

(1) 本程序包含的函数

主函数：int main (void)

链式存储的栈：

初始化函数：SqStackList \* InitStackList(SqStackList \* s)

清除元素函数：int ClearStackList(SqStackList \* s)

消除栈函数：int DestoryStackList(SqStackList \* s)

入栈函数：void PushList(SqStackList \* s, char ch)

出栈函数：int PopList(SqStackList \* s, char \* ch)

检验括号是否配对函数：int isEqualList(SqStackList \* s)

顺序储存的栈：

初始化函数：SqStackOrder \* InitStackOrder(SqStackOrder \* s)

清除元素函数：int ClearStackOrder(SqStackOrder \* s)

消除栈函数：int DestoryStackOrder(SqStackOrder \* s)

入栈函数：void PushOrder(SqStackOrder \* s, char ch)

出栈函数：int PopOrder(SqStackOrder \* s, char \* ch)

检验括号是否配对函数：int isEqualOrder(SqStackOrder \* s)

(2)各函数的功能设计为：

int main (void)

调试各函数。

SqStackList \* InitStackList(SqStackList \* s)

接收一个链式存储的栈的指针，初始化这个栈，并返回这个栈指针。

int ClearStackList(SqStackList \* s)

接收一个链式存储的栈的指针，将该指针指向的栈中的元素清除。若栈本来就没有元素，返回0；成功清除后返回1。

int DestoryStackList(SqStackList \* s)

接收一个链式存储的栈的指针，清除该指针指向的栈。

void PushList(SqStackList \* s, char ch)

接收一个链式存储的栈的指针和一个字符，将字符入栈。

int PopList(SqStackList \* s, char \* ch)

接收一个链式储存的栈的指针和一个字符指针，将栈顶元素出栈，ch获取栈顶元素的值的地址。

int isEqualList(SqStackList \* s)

接收一个链式存储的栈的指针，如何提示用户输入字符并检验输入的字符中 ()[] 字符是否匹配。

SqStackOrder \* InitStackOrder(SqStackOrder \* s)

接收一个顺序存储的栈的指针，将其初始化并返回该指针。

void PushOrder(SqStackOrder \* s, char ch)

接收一个顺序存储的栈的指针和一个字符，将字符入栈。

int PopOrder(SqStackOrder \* s, char \* ch)

接收一个顺序储存的栈的指针和一个字符指针，将栈顶元素出栈，ch获取栈顶元素的值的地址。

int isEqualOrder(SqStackOrder \* s)

接收一个顺序存储的栈的指针，如何提示用户输入字符并检验输入的字符中 ()[] 字符是否匹配。

int DestoryStackOrder(SqStackOrder \* s)

接收一个顺序存储的栈的指针，清除该指针指向的栈。

int ClearStackOrder(SqStackOrder \* s)

接收一个顺序存储的栈的指针，将该指针指向的栈中的元素清除。若栈本来就没有元素，返回0；成功清除后返回1。

2.队列

(1) 本程序包含的函数

主函数：int main (void)

链式存储的队列：

初始化函数：LinkQueueList \* InitQueueList(LinkQueueList \* q)

入队函数：void EnQueueList(LinkQueueList \* q, int data)

出队函数：int DeQueueList(LinkQueueList \* q)

消除队列函数：void DestoryQueueList(LinkQueueList \* q)

清除元素函数：int ClearQueueList(LinkQueueList \* q)

顺序存储的队列：

初始化函数：void InitQueueOrder(LinkQueueOrder \* q)

入队函数：int EnQueueOrder(LinkQueueOrder \* q, int e)

出队函数：int DeQueueOrder(LinkQueueOrder \* q)

消除队列函数：void DestoryQueueOrder(LinkQueueOrder \* q)

清除元素函数：void ClearQueueOrder(LinkQueueOrder \* q)

(2)各函数的功能设计为：

LinkQueueList \* InitQueueList(LinkQueueList \* q)

接收一个链式存储的队列的指针，初始化这个队列，并返回这个队列的指针。

void EnQueueList(LinkQueueList \* q, int data)

接收一个链式存储的队列的指针和int类型的数据，将该数据入队。

int DeQueueList(LinkQueueList \* q)

接收一个链式存储的队列的指针，将队头数据出队，并返回这个出队的数据。

void DestoryQueueList(LinkQueueList \* q)

接收一个链式存储的队列的指针，清除该指针指向的队列。

int ClearQueueList(LinkQueueList \* q)

接收一个链式存储的队列的指针，将该指针指向的队列中的元素清除。若队列本来就没有元素，返回0；成功清除后返回1。

void InitQueueOrder(LinkQueueOrder \* q)

接收一个顺序存储的队列的指针，初始化这个队列。

int EnQueueOrder(LinkQueueOrder \* q, int e)

接收一个顺序存储的队列的指针和int类型的数据，将该数据入队。

int DeQueueOrder(LinkQueueOrder \* q)

接收一个顺序存储的队列的指针，将队头数据出队，并返回这个出队的数据。

void DestoryQueueOrder(LinkQueueOrder \* q)

接收一个顺序存储的队列的指针，清除该指针指向的队列。

void ClearQueueOrder(LinkQueueOrder \* q)

接收一个顺序存储的队列的指针，将该指针指向的队列中的元素清除。

3.迷宫

(1) 本程序包含的函数

主函数：int main (void)

栈初始化函数：Stack \* InitStack(Stack \* s)

清除栈内元素函数：int ClearStack(Stack \* s)

清除栈函数：int DestoryStack(Stack \* s)

栈长度函数：int StackLength(Stack \* s)

检验栈是否为空的函数：int StackEmpty(Stack \* s)

得到栈顶元素函数：void GetTop(Stack \* s, ElemType \* e)

入栈函数：void Push(Stack \* s, ElemType e)

出栈函数：void Pop(Stack \* s, ElemType \* e)

留下可以通过的标志的函数：void FootPrint(MazeType \* maze, PosType pos)

留下不可以通过的标志的函数：void BlockedPrint(MazeType \* maze, PosType pos)

得到下一个位置的函数：PosType NextPos(PosType pos, int di)

得到迷宫的函数：MazeType \* GetMaze(MazeType \* maze)

初始化迷宫函数：MazeType \* InitMaze(MazeType \* maze)

是否可通函数：int Pass(MazeType \* maze, PosType pos)

计算通路函数：int MazePath(Stack \* s, MazeType \* maze, PosType start, PosType en)

输出通路坐标和方向函数：void PrintMazePosition(Stack \* s, MazeType \* maze)

输出迷宫和通路方阵：void PrintMaze(MazeType \* maze)

(2)各函数的功能设计为：

Stack \* InitStack(Stack \* s)

接收一个栈的指针，初始化该栈，并返回栈的指针。

int ClearStack(Stack \* s)

接收一个栈的指针，将该指针指向的栈中的元素清除。若栈中本身没有元素，返回0；清除成功后返回1。

int DestoryStack(Stack \* s)

接收一个栈的指针，清除栈指针指向的内容，释放内存。清除成功后返回1。

int StackLength(Stack \* s)

接收一个栈的指针，获取栈的长度并返回。

int StackEmpty(Stack \* s)

接收一个栈的指针，若栈为空则返回1；否则返回0。

void GetTop(Stack \* s, ElemType \* e)

接收一个栈的指针和一个元素类型指针，获取栈顶元素，将元素类型指针指向栈顶元素。

void Push(Stack \* s, ElemType e)

接收一个栈的指针和一个元素，将该元素入栈。

void Pop(Stack \* s, ElemType \* e)

接收一个栈的指针和一个元素类型指针，将栈顶元素出栈，将元素类型指针指向该栈顶元素。

void FootPrint(MazeType \* maze, PosType pos)

接收一个迷宫的指针和一个位置，将迷宫上该位置留下足迹（这里用’\*’号代表足迹）。

void BlockedPrint(MazeType \* maze, PosType pos)

接收一个迷宫的指针和一个位置，将迷宫上该位置留下不能通过的痕迹（这里用’#’代表阻塞的痕迹）。

PosType NextPos(PosType pos, int di)

接收一个位置和方向值，获取该位置在该方向下的下一个位置，并将其返回。

MazeType \* GetMaze(MazeType \* maze)

接收一个迷宫的指针，将该指针指向一个“迷宫”，返回该指针。

MazeType \* InitMaze(MazeType \* maze)

接收一个迷宫的指针，初始化迷宫类型元素。

int Pass(MazeType \* maze, PosType pos)

接收一个迷宫的指针和一个位置，判断该位置在迷宫上是否是可行的。

int MazePath(Stack \* s, MazeType \* maze, PosType start, PosType en)

接收一个栈、一个迷宫的指针、迷宫的开始位置和结束位置。若迷宫是可以通过的则返回1；否则返回0。在迷宫上会留下通路的痕迹。

void PrintMazePosition(Stack \* s, MazeType \* maze)

接收一个栈、一个迷宫的指针，输出通路每个位置的坐标和方向。

void PrintMaze(MazeType \* maze)

接收一个迷宫的指针，输出该迷宫和通路路径的方阵。

**五、详细设计**

(1)结构体类型、结点类型和指针类型:

**栈：**

struct SElemTypeList{

char elem;

struct SElemTypeList \* pro;

};

typedef struct{

struct SElemTypeList \* base;

struct SElemTypeList \* top;

}SqStackList;

typedef struct{

char \* base;

char \* top;

int stacksize;

}SqStackOrder;

**队列：**

typedef struct QNode \* QLink;

struct QNode{

int data;

QLink next;

};

typedef struct{

QLink qfront;

QLink qrear;

}LinkQueueList;

typedef struct{

int qfront;

int qrear;

int qbase[SIZE];

}LinkQueueOrder;

**迷宫：**

typedef struct {

int r;

int c;

} PosType;

typedef struct{

int step;

int rMax, cMax;

char \*\*ch;

}MazeType;

typedef struct{

PosType pos;

int di;

}ElemType;

typedef struct nodetype \*LinkType;

typedef struct nodetype{

ElemType data;

LinkType next;

}NodeType;

typedef struct{

LinkType base;

LinkType top;

}Stack;

(2)函数中的复杂操作：

int isEqualOrder(SqStackOrder \* s)和 int isEqualList(SqStackList \* s)：

这两个的函数中的大致算法一样：就是将字符先放在一个数组里，如何检验数组中的数据，如果是’(’、’[’就入栈，如果是’]’、’)’就检验栈顶中的括号是否与之配对，如果不配对就输出“表达式中括号'(', ')', '[', ']'不配对!!”对并返回0；如果配对就输出“表达式中括号'(', ')', '[', ']'配对!!”并返回1。

int MazePath(Stack \* s, MazeType \* maze, PosType start, PosType en)：

伪码算法:

设定当前位置的初值为入口位置；

do{

若当前位置可通，

则{将当前位置插入栈顶； //纳入路径

若该位置是出口位置，则结束；//求得路径存放在栈中

否则切换当前位置的东邻方块为新的当前位置；

}

否则{

若栈不空且栈顶位置尚有其他方向未被探索，

则设定新的当前位置为沿顺时针方向旋转找到的栈顶位置的下一相邻块；

若栈不空但栈顶位置的四周均不可通，

则{删去栈顶位置； //后退一步，从路径中删去该通道块，

若栈不空，则重新测试新的栈顶位置，

直至找到一个可通的相邻块或出栈至栈空；

}

}

}while(栈不空);

{栈空说明没有路径存在}

**六、调试分析**

调试结果比较麻烦。主要调试的功夫在迷宫中算法和括号的配对。

**七、使用说明**

**栈：**

栈运行后会有以下菜单：

请选择您的操作(输入0结束):

1.链式存储栈的初始化

2.链式存储栈的入栈

3.链式存储栈的出栈

4.链式存储栈的字符匹配检验

5.顺序存储栈的初始化

6.顺序存储栈的入栈

7.顺序存储栈的出栈

8.链式存储栈的字符匹配检验

**队列：**

队列运行后会有以下菜单：

请选择您的操作(输入0结束):

1.链式存储队列的初始化

2.链式存储队列的入队

3.链式存储队列的出队

4.顺序存储队列的初始化

5.顺序存储队列的入队

6.顺序存储队列的出队

**迷宫：**

迷宫运行后会提示用户输入迷宫文件

Please input filename:

**注意：栈和队列在测试时每次入栈操作、入队操作和字符检验操作前最好先初始化一次栈和队列以防出现异常。**

**八、测试结果**

**栈：**

1. 链式存储栈的初始化

输入1即可

1. 链式存储栈的入栈

输入2后输出了：请输入要入栈的字符（以'\n'结束）

然后输入了abcdefg

1. 链式存储栈的出栈

输入3后控制台输出了：

栈中的内容为（出栈）：

gfedcba

1. 链式存储栈的字符匹配检验

输入4后输出了：输入一组表达式，判断表达式中括号'(', ')', '[', ']'是否配对。'#'为表达式的结束符

然后输入：[(asd)][]#

控制台输出：表达式中括号'(', ')', '[', ']'配对!!!

1. 顺序存储栈的初始化

输入5即可。

1. 顺序存储栈的入栈

输入6后输出了：请输入要入栈的字符（以'\n'结束）

然后输入：123456789

1. 顺序存储栈的出栈

输入7后输出了：

栈中的内容为（出栈）：

987654321

1. 链式存储栈的字符匹配检验

输入4后输出了：输入一组表达式，判断表达式中括号'(', ')', '[', ']'是否配对。'#'为表达式的结束符

然后输入：[(asd)][(])#

控制台输出：表达式中括号'(', ')', '[', ']'不配对!!!

**队列：**

1. 链式存储队列的初始化

输入1即可

1. 链式存储队列的入队

输入2后输出：请输入入队的int值(值为-1结束)：

然后输入了数据：1 2 3 4 5 6 -1

1. 链式存储队列的出队

输入3后输出了：

队列的值为：

1 2 3 4 5 6

1. 顺序存储队列的初始化

输入4即可

1. 顺序存储队列的入队

输入5后输出：请输入入队的int值(值为-1结束)：

然后输入了数据：6 8 2 3 5 -1

6.顺序存储队列的出队

输入6输出了：

队列的值为：

6 8 2 3 5

**迷宫：**

注：迷宫中‘1’代表不可通过的地方，‘0’代表可以行走的地方。‘\*’代表通路上的路径，‘#’代表走过的但是不能到达终点的地方。

示例迷宫为：

**11111111111**

**10010001001**

**10010001011**

**10000110011**

**10111000011**

**10001000011**

**10100010001**

**10111011001**

**11000000001**

**11111111111**

输入文件名：maze.txt后输出了如下内容：

The path is:

(1, 1, 1) (1, 2, 2) (2, 2, 2) (3, 2, 3) (3, 1, 2) (4, 1, 2) (5, 1, 1) (5, 2, 1) (5, 3, 2) (6, 3, 1) (6, 4, 1) (6, 5, 2) (7, 5, 2) (8, 5, 1) (8, 6, 1) (8, 7, 1) (8, 8, 1)

The path is:

1111111111

1\*\*1###101

10\*1###101

1\*\*##11001

1\*11100001

1\*\*\*100001

101\*\*\*1001

10111\*1101

11000\*\*\*\*1

1111111111

**九、附录：部分代码**

**栈：**

#include<stdio.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

struct SElemTypeList{

char elem;

struct SElemTypeList \* pro;

};

typedef struct{

struct SElemTypeList \* base;

struct SElemTypeList \* top;

}SqStackList;

typedef struct{

char \* base;

char \* top;

int stacksize;

}SqStackOrder;

SqStackList \* InitStackList(SqStackList \* s);

int ClearStackList(SqStackList \* s);

int DestoryStackList(SqStackList \* s);

void PushList(SqStackList \* s, char ch);

int PopList(SqStackList \* s, char \* ch);

int isEqualList(SqStackList \* s);

SqStackOrder \* InitStackOrder(SqStackOrder \* s);

void PushOrder(SqStackOrder \* s, char ch);

int PopOrder(SqStackOrder \* s, char \* ch);

int isEqualOrder(SqStackOrder \* s);

int DestoryStackOrder(SqStackOrder \* s);

int ClearStackOrder(SqStackOrder \* s);

int main (void)

{

SqStackList \* slist = NULL;

SqStackOrder \* sorder = NULL;

int flag;

printf("请选择您的操作(输入0结束):\n");

printf("1.链式存储栈的初始化\n");

printf("2.链式存储栈的入栈\n");

printf("3.链式存储栈的出栈\n");

printf("4.链式存储栈的字符匹配检验\n");

printf("5.顺序存储栈的初始化\n");

printf("6.顺序存储栈的入栈\n");

printf("7.顺序存储栈的出栈\n");

printf("8.顺序存储栈的字符匹配检验\n");

scanf("%d", &flag);

while(flag)

{

char ch;

switch(flag)

{

case 1:

DestoryStackList(slist);

slist = InitStackList(slist);

break;

case 2:

printf("请输入要入栈的字符（以'\\n'结束）\n");

getchar();

while((ch = getchar()) != '\n')

{

PushList(slist, ch);

}

break;

case 3:

printf("栈中的内容为（出栈）：\n");

while(slist->top != NULL)

{

PopList(slist, &ch);

putchar(ch);

}

putchar('\n');

break;

case 4:

isEqualList(slist);

break;

case 5:

DestoryStackOrder(sorder);

sorder = InitStackOrder(sorder);

break;

case 6:

printf("请输入要入栈的字符（以'\\n'结束）\n");

getchar();

while((ch = getchar()) != '\n')

{

PushOrder(sorder, ch);

}

break;

case 7:

printf("栈中的内容为（出栈）：\n");

while(sorder->top > sorder->base)

{

PopOrder(sorder, &ch);

putchar(ch);

}

putchar('\n');

break;

case 8:

isEqualOrder(sorder);

break;

default:

break;

}

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请选择您的操作(输入0结束):\n");

printf("1.链式存储栈的初始化\n");

printf("2.链式存储栈的入栈\n");

printf("3.链式存储栈的出栈\n");

printf("4.链式存储栈的字符匹配检验\n");

printf("5.顺序存储栈的初始化\n");

printf("6.顺序存储栈的入栈\n");

printf("7.顺序存储栈的出栈\n");

printf("8.顺序存储栈的字符匹配检验\n");

scanf("%d", &flag);

}

DestoryStackList(slist);

DestoryStackOrder(sorder);

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

return 0;

}

SqStackList \* InitStackList(SqStackList \* s)

{

s = (SqStackList\*) malloc (sizeof(SqStackList));

s->top = s->base = NULL;

return s;

}

int ClearStackList(SqStackList \* s)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

if(s->top == NULL)

{

return 0;

}

while(s->top->pro != NULL)

{

struct SElemTypeList \* l = s->top->pro;

free(s->top);

s->top = l;

}

free(s->top);

return 1;

}

int DestoryStackList(SqStackList \* s)

{

if(s == NULL)

{

return 0;

}

if(s->top == NULL)

{

free(s);

return 0;

}

ClearStackList(s);

free(s);

return 1;

}

void PushList(SqStackList \* s, char ch)

{

struct SElemTypeList \* e = (struct SElemTypeList \*) malloc (sizeof(struct SElemTypeList));

e->elem = ch;

e->pro = s->top;

if(s->top == NULL)

{

s->base = e;

e->pro = NULL;

}

s->top = e;

}

int PopList(SqStackList \* s, char \* ch)

{

struct SElemTypeList \* e;

if(s->top == NULL)

{

s->base = NULL;

printf("栈为空！！\n");

return 0;

}

e = s->top;

\*ch = e->elem;

s->top = s->top->pro;

free(e);

if(s->top == NULL)

{

s->base = NULL;

}

return 1;

}

int isEqualList(SqStackList \* s)

{

char ch, E[100]="";

int i = 0;

printf("输入一组表达式，判断表达式中括号'(', ')', '[', ']'是否配对。'#'为表达式的结束符\n");

while((ch = getchar()) != '#' && i < 100)

{

E[i++] = ch;

}

int j = 0;

while(j < i)

{

if(E[j] == '(' || E[j] == '[')

{

PushList(s, E[j]);

}

else if(E[j] == ')' || E[j] == ']')

{

if(E[j] == ')' && s->top->elem == '(')

{

PopList(s, &ch);

}

else if(E[j] == ']' && s->top->elem == '[')

{

PopList(s, &ch);

}

else

{

if(s->base == s->top)

{

printf("表达式中括号'(', ')', '[', ']'不配对!!\n");

return 0;

}

}

}

j++;

}

if(s->top != NULL)

{

printf("表达式中括号'(', ')', '[', ']'不配对!!!\n");

return 0;

}

printf("表达式中括号'(', ')', '[', ']'配对!!!\n");

return 1;

}

SqStackOrder \* InitStackOrder(SqStackOrder \* s)

{

s = (SqStackOrder \*) malloc (sizeof(SqStackOrder));

s->base = (char \*) malloc (STACK\_INIT\_SIZE \* sizeof(char));

if(!s->base)

{

printf("内存分配失败！！\n");

exit(-1);

}

s->top = s->base;

s->stacksize = STACK\_INIT\_SIZE;

return s;

}

void PushOrder(SqStackOrder \* s, char ch)

{

if(s->top - s->base >= s->stacksize)

{

s->base = (char \*) realloc (s->base, (s->stacksize + STACKINCREMENT) \* sizeof(char));

if(!s->base)

{

printf("内存分配失败！！\n");

exit(-1);

}

s->top = s->base + s->stacksize;

s->stacksize += STACKINCREMENT;

}

\*s->top++ = ch;

}

int PopOrder(SqStackOrder \* s, char \* ch)

{

if(s->top == s->base)

{

printf("栈为空！！\n");

return 0;

}

\*ch = \* --s->top;

return 1;

}

int isEqualOrder(SqStackOrder \* s)

{

char ch, E[100]="", c;

int i = 0;

printf("输入一组表达式，判断表达式中括号'(', ')', '[', ']'是否配对。'#'为表达式的结束符\n");

while((ch = getchar()) != '#' && i < 100)

{

E[i++] = ch;

}

int j = 0;

while(j < i)

{

if(E[j] == '(' || E[j] == '[')

{

PushOrder(s, E[j]);

}

else if(E[j] == ')' || E[j] == ']')

{

if(E[j] == ')' && \*(s->top - 1) == '(')

{

PopOrder(s, &c);

}

else if(E[j] == ']' && \*(s->top - 1) == '[')

{

PopOrder(s, &c);

}

else

{

if(s->base == s->top)

{

printf("表达式中括号'(', ')', '[', ']'不配对!!\n");

return 0;

}

}

}

j++;

}

if(s->top != s->base)

{

printf("表达式中括号'(', ')', '[', ']'不配对!!!\n");

return 0;

}

printf("表达式中括号'(', ')', '[', ']'配对!!!\n");

return 1;

}

int DestoryStackOrder(SqStackOrder \* s)

{

if(s == NULL)

{

return 0;

}

ClearStackOrder(s);

free(s);

return 1;

}

int ClearStackOrder(SqStackOrder \* s)

{

if(s->base != NULL)

{

free(s->base);

}

return 1;

}

**队列：**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define SIZE 10

typedef struct QNode \* QLink;

struct QNode{

int data;

QLink next;

};

typedef struct{

QLink qfront;

QLink qrear;

}LinkQueueList;

typedef struct{

int qfront;

int qrear;

int qbase[SIZE];

}LinkQueueOrder;

LinkQueueList \* InitQueueList(LinkQueueList \* q);

void EnQueueList(LinkQueueList \* q, int data);

int DeQueueList(LinkQueueList \* q);

void DestoryQueueList(LinkQueueList \* q);

int ClearQueueList(LinkQueueList \* q);

void InitQueueOrder(LinkQueueOrder \* q);

int EnQueueOrder(LinkQueueOrder \* q, int e);

int DeQueueOrder(LinkQueueOrder \* q);

void DestoryQueueOrder(LinkQueueOrder \* q);

void ClearQueueOrder(LinkQueueOrder \* q);

int main(void)

{

LinkQueueList \* q = NULL;

LinkQueueOrder link;

int i, flag;

printf("请选择您的操作(输入0结束):\n");

printf("1.链式存储队列的初始化\n");

printf("2.链式存储队列的入队\n");

printf("3.链式存储队列的出队\n");

printf("4.顺序存储队列的初始化\n");

printf("5.顺序存储队列的入队\n");

printf("6.顺序存储队列的出队\n");

scanf("%d", &flag);

while(flag)

{

switch (flag)

{

case 1:

DestoryQueueList(q);

q = InitQueueList(q);

break;

case 2:

printf("请输入入队的int值(值为-1结束)：\n");

while(scanf("%d", &i), i != -1)

{

EnQueueList(q, i);

}

break;

case 3:

printf("队列的值为：\n");

while(q->qfront != q->qrear)

{

printf("%d ", DeQueueList(q));

}

printf("\n");

break;

case 4:

InitQueueOrder(&link);

break;

case 5:

printf("请输入入队的int值(值为-1结束)：\n");

while(scanf("%d", &i), i != -1)

{

EnQueueOrder(&link, i);

}

break;

case 6:

printf("队列的值为：\n");

while(link.qfront != link.qrear)

{

printf("%d ", DeQueueOrder(&link));

}

printf("\n");

break;

default:

break;

}

printf("请选择您的操作(输入0结束):\n");

printf("1.链式存储队列的初始化\n");

printf("2.链式存储队列的入队\n");

printf("3.链式存储队列的出队\n");

printf("4.顺序存储队列的初始化\n");

printf("5.顺序存储队列的入队\n");

printf("6.顺序存储队列的出队\n");

scanf("%d", &flag);

}

DestoryQueueList(q);

return 0;

}

LinkQueueList \* InitQueueList(LinkQueueList \* q)

{

q = (LinkQueueList \*) malloc (sizeof(LinkQueueList));

q->qfront = q->qrear = (QLink) malloc (sizeof(struct QNode));

if(!q->qfront)

{

printf("内存分配失败！！");

exit(-1);

}

q->qfront->next = NULL;

return q;

}

void EnQueueList(LinkQueueList \* q, int data)

{

QLink p = (QLink) malloc (sizeof(struct QNode));

if(!p)

{

printf("内存分配失败！！\n");

return;

}

p->data = data;

q->qrear->next = p;

q->qrear = p;

q->qrear->next = NULL;

}

int DeQueueList(LinkQueueList \* q)

{

int data;

QLink p;

if(q->qfront == q->qrear)

{

printf("队列为空！！\n");

return -1;

}

p = q->qfront->next;

q->qfront->next = p->next;

data = p->data;

if(q->qrear == p)

{

q->qrear = q->qfront;

}

free(p);

return data;

}

void DestoryQueueList(LinkQueueList \* q)

{

if(q == NULL)

{

return;

}

ClearQueueList(q);

free(q);

}

int ClearQueueList(LinkQueueList \* q)

{

if(q->qfront == q->qrear)

{

return 0;

}

while(q->qfront != q->qrear)

{

QLink link = q->qfront;

q->qfront = link->next;

free(link);

}

free(q->qfront);

return 1;

}

void InitQueueOrder(LinkQueueOrder \* q)

{

int i;

for(i = 0; i < SIZE; i++)

q->qbase[i] = -1;

q->qfront = q->qrear = 0;

}

int EnQueueOrder(LinkQueueOrder \* q, int e)

{

if((q->qrear+1) % SIZE == q->qfront)

{

printf("队列已满！！\n");

return 0;

}

q->qbase[q->qrear] = e;

q->qrear = (q->qrear+1) % SIZE;

return 1;

}

int DeQueueOrder(LinkQueueOrder \* q)

{

if(q->qfront == q->qrear)

{

printf("队列为空！！\n");

return 0;

}

int i = q->qbase[q->qfront];

q->qfront = (q->qfront + 1) % SIZE;

return i;

}

**迷宫：**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

typedef struct {

int r;

int c;

} PosType;

typedef struct{

int step;

int rMax, cMax;

char \*\*ch;

}MazeType;

typedef struct{

PosType pos;

int di;

}ElemType;

typedef struct nodetype \*LinkType;

typedef struct nodetype{

ElemType data;

LinkType next;

}NodeType;

typedef struct{

LinkType base;

LinkType top;

}Stack;

Stack \* InitStack(Stack \* s);

int ClearStack(Stack \* s);

int DestoryStack(Stack \* s);

int StackLength(Stack \* s);

int StackEmpty(Stack \* s);

void GetTop(Stack \* s, ElemType \* e);

void Push(Stack \* s, ElemType e);

void Pop(Stack \* s, ElemType \* e);

void FootPrint(MazeType \* maze, PosType pos);

void BlockedPrint(MazeType \* maze, PosType pos);

PosType NextPos(PosType pos, int di);

MazeType \* GetMaze(MazeType \* maze);

MazeType \* InitMaze(MazeType \* maze);

int Pass(MazeType \* maze, PosType pos);

int MazePath(Stack \* s, MazeType \* maze, PosType start, PosType en);

void PrintMazePosition(Stack \* s, MazeType \* maze);

void PrintMaze(MazeType \* maze);

int main (void)

{

Stack \* s = NULL;

MazeType \* maze = (MazeType \*) malloc (sizeof(MazeType));

s = InitStack(s);

maze = InitMaze(maze);

PosType start = {1, 1};

PosType en = {maze->rMax-2, maze->cMax-2};

MazePath(s, maze, start, en);

PrintMazePosition(s, maze);

PrintMaze(maze);

DestoryStack(s);

free(maze);

return 0;

}

Stack \* InitStack(Stack \* s)

{

s = (Stack \*) malloc (sizeof(Stack));

if(!s)

{

printf("内存分配失败!!\n");

exit(-1);

}

s->top = s->base = NULL;

return s;

}

int ClearStack(Stack \* s)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

if(s->top == NULL)

{

return 0;

}

while(s->top->next != NULL)

{

LinkType l = s->top->next;

free(s->top);

s->top = l;

}

free(s->top);

return 1;

}

int DestoryStack(Stack \* s)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

if(s->top == NULL)

{

free(s);

return 0;

}

ClearStack(s);

free(s);

return 1;

}

int StackLength(Stack \* s)

{

if(s == NULL)

{

printf("栈不存在!\n");

exit(-1);

}

int count = 0;

if(s->top == NULL)

{

printf("栈为空！！\n");

return 0;

}

while(s->top != NULL)

{

s->top = s->top->next;

count++;

}

return count;

}

int StackEmpty(Stack \* s)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

if(s->top == NULL)

{

printf("栈为空!!\n");

return 1;

}

return 0;

}

void GetTop(Stack \* s, ElemType \* e)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

else

{

if(s->top == NULL)

{

printf("栈为空!!\n");

}

else

{

\*e = s->top->data;

}

}

}

void Push(Stack \* s, ElemType e)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

LinkType link = (LinkType) malloc (sizeof(NodeType));

if(!link)

{

printf("内存分配失败！！\n");

exit(-1);

}

link->data = e;

link->next = s->top;

if(s->top == NULL)

{

s->base = link;

}

s->top = link;

}

void Pop(Stack \* s, ElemType \* e)

{

if(s == NULL)

{

printf("栈不存在!!\n");

exit(-1);

}

if(s->top == NULL)

{

printf("栈为空！！\n");

}

else

{

\*e = s->top->data;

LinkType link = s->top;

s->top = s->top->next;

free(link);

}

}

void FootPrint(MazeType \* maze, PosType pos)

{

maze->ch[pos.r][pos.c] = '\*';

}

void BlockedPrint(MazeType \* maze, PosType pos)

{

maze->ch[pos.r][pos.c] = '#';

}

PosType NextPos(PosType pos, int di)

{

switch(di)

{

case 1:

pos.c += 1;

break;

case 2:

pos.r += 1;

break;

case 3:

pos.c -= 1;

break;

case 4:

pos.r -= 1;

break;

default:

break;

}

return pos;

}

MazeType \* GetMaze(MazeType \* maze)

{

char ch, filename[100];

int r = 0, c = 0;

FILE \* fp = NULL;

printf("Please input filename:\n");

scanf("%s", filename);

if((fp = fopen(filename, "r")) == NULL)

{

printf("File can't be opened!!\n");

exit(-1);

}

while((ch = getc(fp)) != '\n')

{

c++;

}

while((ch = getc(fp)) != EOF)

{

if(ch != '\n')

r++;

}

r += c;

r /= c;

maze->rMax = r;

maze->cMax = c;

char \*\* arr = (char\*\*) malloc (r \* sizeof(char\*));

while(r--)

{

arr[r] = (char\*) malloc (c \* sizeof(char));

}

r = 0;

if((fp = fopen(filename, "r")) == NULL)

{

printf("File can't be opened!!\n");

exit(-1);

}

while((ch = getc(fp)) != EOF)

{

c = 0;

do

{

arr[r][c++] = ch;

}while((ch = getc(fp)) != '\n' && ch != EOF);

r++;

}

fclose(fp);

maze->ch = arr;

return maze;

}

MazeType \* InitMaze(MazeType \* maze)

{

maze->step = 0;

maze = GetMaze(maze);

return maze;

}

int Pass(MazeType \* maze, PosType pos)

{

if(maze->ch[pos.r][pos.c] == '0')

return 1;

return 0;

}

int MazePath(Stack \* s, MazeType \* maze, PosType start, PosType en)

{

PosType curpos = start;

do

{

if(Pass(maze, curpos))

{

maze->step++;

FootPrint(maze, curpos);

ElemType e;

e.di = 1;

e.pos = curpos;

Push(s, e);

if(curpos.c == en.c && curpos.r == en.r)

return 1;

curpos = NextPos(curpos, 1);

}

else

{

ElemType e;

Pop(s, &e);

while(e.di == 4 && !StackEmpty(s))

{

maze->step--;

BlockedPrint(maze, e.pos);

Pop(s, &e);

}

if(e.di < 4)

{

e.di++;

Push(s, e);

curpos = NextPos(e.pos, e.di);

}

}

}while(!StackEmpty(s));

return 0;

}

void PrintMazePosition(Stack \* s, MazeType \* maze)

{

int i = maze->step;

ElemType e[maze->step];

while(s->top != NULL)

{

e[--i] = s->top->data;

s->top = s->top->next;

}

printf("The path is:\n");

for(i = 0; i < maze->step; i++)

{

printf("(%d, %d, %d) ", e[i].pos.r, e[i].pos.c, e[i].di);

}

printf("\n");

}

void PrintMaze(MazeType \* maze)

{

int r, c;

printf("The path is:\n");

for(r = 0; r < maze->rMax; r++)

{

for(c = 0; c < maze->cMax; c++)

{

putchar(maze->ch[r][c]);

}

putchar('\n');

}

}