信息科学与技术学院，20级电子信息类专业

**程 序 设 计 实 践 报 告**

学 号：\_\_\_\_\_\_\_\_200910816\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

姓 名：\_\_\_\_\_\_\_\_\_\_陈展鸿\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

设计题目：

1.\_\_\_\_\_\_\_\_\_\_\_\_\_停车场管理系统\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.\_\_\_\_\_\_\_\_\_\_\_\_\_求二叉树上结点的路径\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.\_\_\_\_\_\_\_\_\_\_\_\_\_迷宫问题求解\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

指导教师：\_\_\_\_\_\_\_\_谢颖华 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

成 绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

完成日期：\_\_\_\_\_\_\_\_2021.06.17\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 设计题目一

停车场管理系统

# 任务：

设计一个停车场管理系统，模拟停车场的运作，此程序具备以下功能：

（1）若车辆到达，则显示汽车在停车场内或者便道上的停车位置；

（2）若车辆离去，则显示汽车在停车场内停留的时间和应缴纳的费用（在便道上停留的时间不收费）

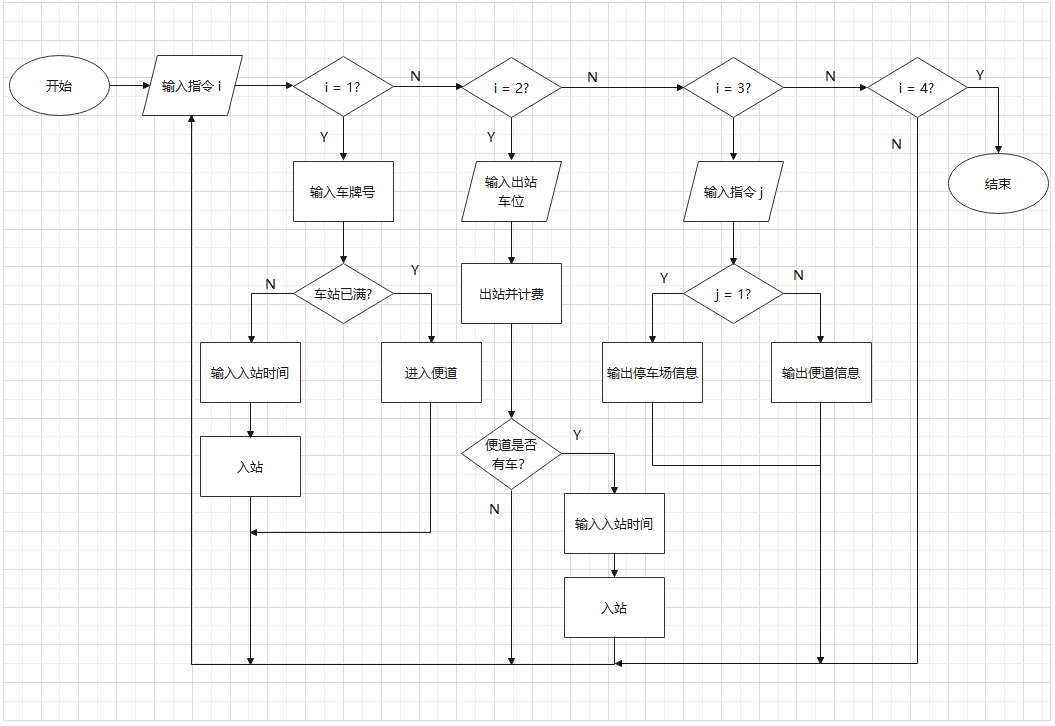
# 功能要求：

（1）要求以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入和输入数据序列进行模拟管理。

（2）要求处理的数据元素包括三个数据项：汽车“到达”或“离去”信息，汽车牌照号码及到达或离去的时刻。

（3）要求栈以顺序结构实现，队列以链表实现。

# 需求分析：



这个程序关键是对车进站和出站时顺序栈相关的操作，以及处理车在便道时与在停车场里时相关关系。对停车场进行模拟时只能使用顺序栈，因此初步实现停车场只能采取先进后出的形式。而对便道进行模拟使用了链队列，方便进入停车场。当停车场满载，车子就进入便道，且因未入站不收费。如果车子驶出停车场，则便道上的车进入停车场并开始收费。同时设置选项用于显示停车相关信息。后续采用双栈解决顺序栈“先进后出”的与实际不符的问题。

# 概要设计

#include <stdio.h>

#include <malloc.h>

#include <process.h>

//最大容车量

#define MAX 1

//费用

#define PRICE\_PER\_HOUR 6

#define PRICE\_PER\_MIN 0.2

//时间

typedef struct

{

int hour;

int min;

} Time;

//车

typedef struct

{

//车牌号

char num[10];

Time in\_time;

Time out\_time;

} Car;

//车栈

#define STACK\_INIT\_SIZE 10

#define STACKINCREMENT 2

typedef struct

{

Car car[STACK\_INIT\_SIZE];

int top;

} CarSqStack;

//便道

typedef struct Car\_p

{

Car data;

struct Car\_p \*next;

} Car\_p;

typedef struct

{

Car\_p \*front;

Car\_p \*rear;

} Passway;

void InitStack (CarSqStack \*);

void Push(CarSqStack \*,Car);

void Pop(CarSqStack \*,Car \*);

void InitQueue(Passway \*);

void EnQueue(Passway \*,Car);

void DeQueue(Passway \*,Car \*);

void arrival(CarSqStack \*,Passway\*);

void leave(CarSqStack \*,CarSqStack \*,Passway\*);

void info(CarSqStack\*,CarSqStack\*,Passway\*);

void print(Car \*);

程序调用关系如下：

停车场管理系统

小车入站

退出系统

小车信息

小车出站

# 详细设计

#include <stdio.h>

#include <malloc.h>

#include <process.h>

//最大容车量

#define MAX 1

//费用

#define PRICE\_PER\_HOUR 6

#define PRICE\_PER\_MIN 0.2

//时间

typedef struct

{

int hour;

int min;

} Time;

//车

typedef struct

{

//车牌号

char num[10];

Time in\_time;

Time out\_time;

} Car;

//车栈

#define STACK\_INIT\_SIZE 10

#define STACKINCREMENT 2

typedef struct

{

Car car[STACK\_INIT\_SIZE];

int top;

} CarSqStack;

//便道

typedef struct Car\_p

{

Car data;

struct Car\_p \*next;

} Car\_p;

typedef struct

{

Car\_p \*front;

Car\_p \*rear;

} Passway;

void InitStack (CarSqStack \*);

void Push(CarSqStack \*,Car);

void Pop(CarSqStack \*,Car \*);

void InitQueue(Passway \*);

void EnQueue(Passway \*,Car);

void DeQueue(Passway \*,Car \*);

void arrival(CarSqStack \*,Passway\*);

void leave(CarSqStack \*,CarSqStack \*,Passway\*);

void info(CarSqStack\*,CarSqStack\*,Passway\*);

void print(Car \*);

int main ()

{

int item;

CarSqStack barn,temp;

Passway passway;

InitStack(&barn);

InitStack(&temp);

InitQueue(&passway);

while(1)

{

fflush(stdin);

printf("\n\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*欢迎使用停车场系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.车辆到达\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.车辆离开\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.车辆信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4.退出系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*欢迎使用停车场系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n 请选择服务，请输入1/2/3/4:\n");

while(1)

{

scanf("%d",&item);

if(item>4)

printf("输入无效数字，请重新输入！\n");

else

break;

}

switch(item)

{

case 1:

arrival(&barn,&passway);

break;

case 2:

leave(&barn,&temp,&passway);

break;

case 3:

info(&barn,&temp,&passway);

break;

case 4:

exit(0);

default:

break;

}

}

return 0;

}

//初始化栈

void InitStack (CarSqStack \*s)

{

int i;

s->top=0;

// for(i=0; i<STACK\_INIT\_SIZE; i++)

// {

// s->car[i]=NULL;

// }

}

//压栈

void Push(CarSqStack \*s,Car e)

{

if(s->top>=STACK\_INIT\_SIZE)

{

printf("ERROR!");

return;

}

s->car[s->top] = e;

s->top++;

}

//出栈

void Pop(CarSqStack \*s,Car \*e)

{

if(s->top>=STACK\_INIT\_SIZE)

{

printf("ERROR\n");

return;

}

s->top--;

\*e = s->car[s->top];

}

//队列初始化

void InitQueue(Passway \*q)

{

q->front=(Car\_p \*)malloc(sizeof(Car\_p));

if(q->front!=NULL)

{

q->front->next=NULL;

q->rear=q->front;

}

}

//入队

void EnQueue(Passway \*Q,Car e)

{

Car\_p \*car\_p=(Car\_p \*)malloc(sizeof(Car\_p));

car\_p->data=e;

car\_p->next=NULL;

if(Q->front==Q->rear)

{

Q->front->next=car\_p;

Q->rear=car\_p;

}

else

{

Q->rear->next=car\_p;

Q->rear=car\_p;

}

}

//出队

void DeQueue(Passway \*Q,Car \*e)

{

if(Q->front==Q->rear)

{

printf("ERROR!");

return;

}

\*e=Q->front->next->data;

Q->front->next=Q->front->next->next;

if(Q->front->next==NULL)

{

Q->rear=Q->front;

}

}

void arrival(CarSqStack \*s,Passway \*q)

{

Car \*p;

fflush(stdin);

printf("请输入车牌号:");

gets(p->num );

if(s->top<MAX)

{

printf("车辆在停车场中的位置为 %d:\n",s->top+1);

printf("请输入到达的时间(hh:mm): ");

scanf("%d:%d",&(p->in\_time.hour),&(p->in\_time.min));

Push(s,\*p);

}

else

{

printf("\n\n----------------------停车场已满正在进入便道！-----------------\n");

EnQueue(q,\*p);

}

}

void print(Car \*p)

{

int t1,t2;

printf("请输入离开的时间(hh:mm): ");

scanf("%d:%d",&(p->out\_time.hour),&(p->out\_time .min));

printf("离开车辆的车牌号码:" );

puts(p->num);

printf("离开车辆的进入时间为%2d:%d\n",p->in\_time.hour ,p->in\_time .min );

printf("离开车辆的离开时间为%2d:%d\n",p->out\_time.hour ,p->out\_time .min );

t1=(p->out\_time.hour)-(p->in\_time.hour);

t2=(p->out\_time.min)-(p->in\_time.min);

printf("应缴纳费用%f元\n\n",t1\*PRICE\_PER\_HOUR+t2\*PRICE\_PER\_MIN);

}

void leave(CarSqStack \*s,CarSqStack \*temp,Passway \*q)

{

int room;

Car \*p;

//判断停车场内是否有车；

if(s->top>0)

{

while(1)

{

printf("停车场里停车总数:%d\n",s->top);

printf("请输入停车场中要离开的车的位置:\n");

fflush(stdin);

scanf("%d",&room);

if(room>=1&&room<=s->top)

break;

}

while(s->top>room)

{

Pop(s,p);

Push(temp,\*p);

}

Pop(s,p);

print(p);

//通道是否有车

if((q->front!=q->rear) &&s->top<MAX)

{

DeQueue(q,p);

printf("现在请便道%s车进入车库%d位置\n",p->num,s->top);

printf("请输入到达的时间(hh:mm): ");

fflush(stdin);

scanf("%d:%d",&(p->in\_time.hour),&(p->in\_time.min));

Push(s,\*p);

}

else

printf("通道里没有车\n");

while(temp->top>1)

{

Pop(temp,p);

Push(s,\*p);

}

}

else

printf("\n目前车库里没有车\n");

}

void info(CarSqStack \*s,CarSqStack \*temp,Passway \*q)

{

Car\_p \*p;

Car \*c = (Car \*) malloc(sizeof(Car));

int i;

int item;

printf("\n\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.查看车库中车的信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.查看便道中车的信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.返回\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*请选择所需的服务\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fflush(stdin);

scanf("%d",&item);

switch(item)

{

case 1:

if(s->top>0)

{

printf("\n\n");

printf("---------------------停车场信息-------------\n");

printf("停车位置 到达时间 车牌号\n");

while((s->top)>0)

{

Pop(s,c);

Push(temp,\*c);

}

for(i=0; temp->top>0; i++)

{

printf("%6d",i+1);

Pop(temp,c);

Push(s,\*c);

printf("%12d:%2d",c->in\_time.hour,c->in\_time.min);

printf(" ");

puts(c->num);

}

printf("----------------------------------------------------\n\n");

}

else

printf("车库里没有车\n");

break;

case 2:

p=(Car\_p \*)malloc(sizeof(Car\_p));

//p=q->head->next;

p=q->front->next;

if(q->front!=q->rear)

{

printf("\n\n------------------通道-------------------\n");

printf(" 等待车辆的车牌号码为: \n");

while(p!=NULL)

{

puts(p->data.num);

p=p->next;

}

printf("\n-----------------------------------------------------");

}

else

printf("\n\n---------------------便道里没有车-------------\n\n\n");

break;

case 3:

break;

default:

printf("输入错误 请重新输入:\n");

break;

}

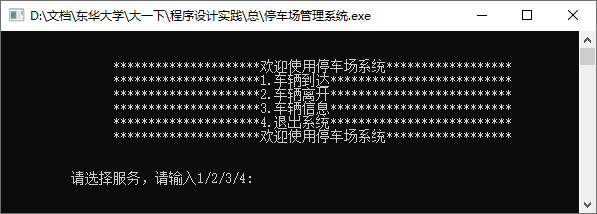
}

# 调试分析

1. 对顺序栈和链栈不够理解，导致前期写栈相关实现函数时出现太多莫名其妙的Bug. 后来重新了解栈，才把栈相关的东西做好。
2. 跟栈一样，对队列不够理解。没有弄清楚队列front指针和rear指针的指向区别，导致入队出队函数出错。结果也是重新了解了队列相关内容。
3. 对栈“先进后出”的优化有思路上的错误。一开始想的是用一个数组临时存储小车让指定位置车子出站，结果在处理数组大小时出现了越界等奇怪的问题。后来在调试过程中想到用一个临时栈更好用，临时栈同样拥有“先进后出”的特点，结果发现这样做特别简便。

# 用户手册

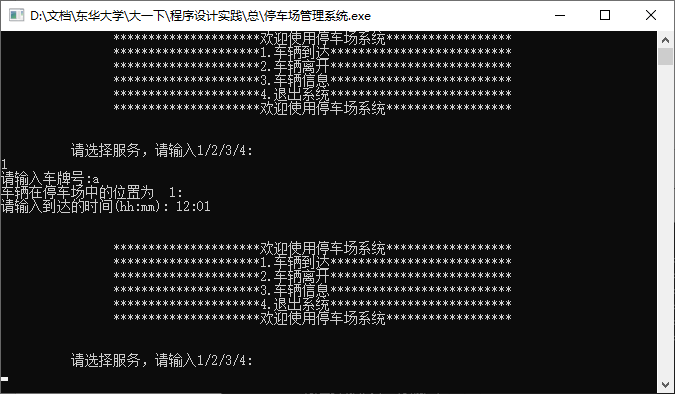
1. 演示程序的运行环境为Windows Vista系统，Microsoft Visual Studio 6.0中的Microsoft Visual C++ 6.0中运。执行文件为：停车场管理系统.exe
2. 进入演示程序后即显示DOS形式的界面：



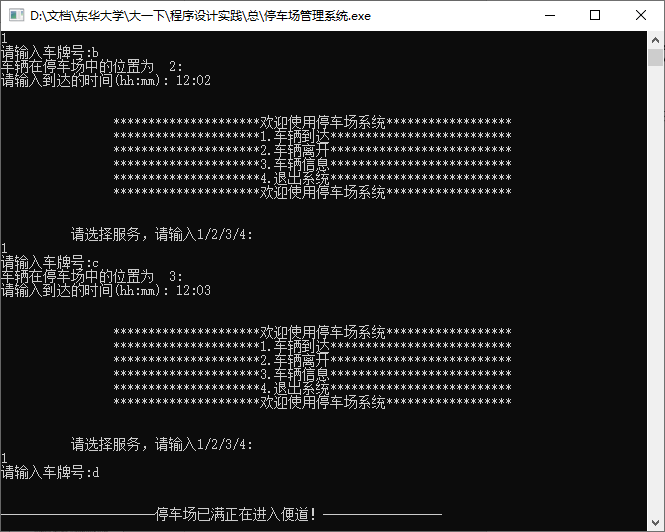
1. 输入指定命令则可执行相应的操作，来模拟停车场管理

# 测试结果

输入指令1，让车辆入站



继续让车辆入站，直到停车场满了为止（方便测试，停车场仅能停3辆车，宏定义可改）



可以看到，当停车场满了之后，车辆会进入便道

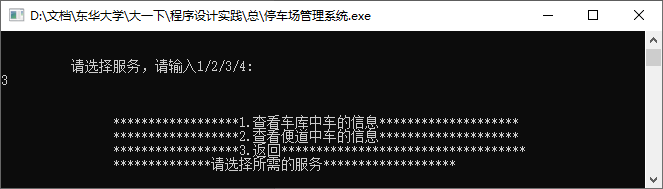
输入指令2，让车辆离开



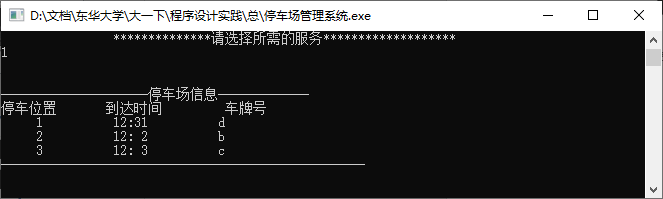
我们让1号位车辆离开，并输入离开时间

可以看到，1号车位离开，且便道中的车子入站

输入指令3，查看信息

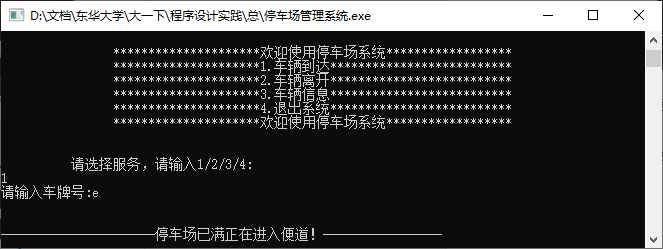


先查看车库信息，输入指令1

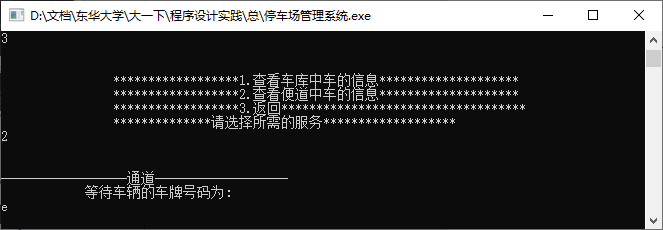


当前车库中所有的车信息显示如上

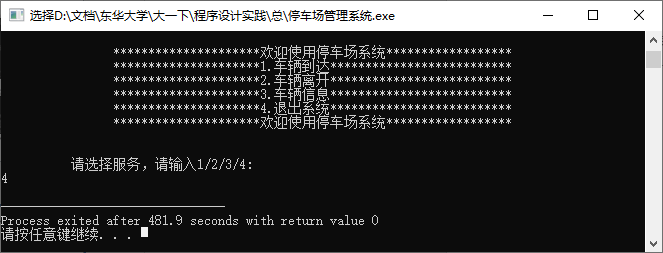
再让车驶入一辆，此时车库是满的，驶入便道以供查看



此时便道有一辆车



输入指令4就是退出系统了



演示结束

# 总结与分析：

最好的学习方法莫过于实践。这次的课程设计使我们真正的能运用我们所学知识来做实际的事情，虽然现实中的停车场管理系统远比这个复杂，但我还是从中学到了不少的东西。程序设计实践其实就是数据结构的延伸，数据结构主讲算法，这个课程主于实践。事实证明，即使数据结构课程拿到了还行的成绩，但能真正运用到的知识很少。直到开始尝试完成课程设计任务，才知道以前掌握的知识有多表层。用到顺序栈的时候，把顺序栈和链栈搞混，忘记链队列头尾指针的指向及其意义。当然，经过这次的练习，我算是真正开始掌握了栈和队列的知识。工科的学习本身就是要和实践相结合的，以后的学习也应该多多动手，不要只顾着看知识点。

# 设计题目二

求二叉树上节点的路径

# 任务：

建立一棵二叉树，编程实现从根结点到给定结点之间的路径。

# 功能要求：

建立一棵以二叉链表存储的二叉树，以bt指向根节点、p指向任一给定结点，编程实现“以字符形式输出从根结点到给定结点之间的路径”。

# 需求分析：

该程序需要对用户输入的二叉树结点数据进行识别和处理，从而使用递归方法创建二叉树，再通过非递归后序遍历的方式得出根结点到指定结点之间的路径

# 概要设计

#include<stdio.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 10

#define STACKINCREASEMENT 5

typedef struct BiTNode

{

char data;

struct BiTNode\* lchild, \* rchild;

} BiTNode, \* BiTree;

BiTree bt;

BiTNode \*p;

typedef struct

{

BiTree bn[STACK\_INIT\_SIZE];

int top;

int stacksize;

} SqStack;

SqStack L,Path;

void InitStack (SqStack \*);

void Push(SqStack \*,BiTree);

void Pop(SqStack \*,BiTree \*);

int Equal(BiTree,BiTree);

void CreateBiTree(BiTree\*);

void PostPush(BiTree,char);

程序调用关系如下：

求二叉树节点路径

输出路径

输入待查结点

先序遍历建立二叉树

后序非递归遍历寻路经

# 详细设计

#include<stdio.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 10

#define STACKINCREASEMENT 5

typedef struct BiTNode

{

char data;

struct BiTNode\* lchild, \* rchild;

} BiTNode, \* BiTree;

BiTree bt;

BiTNode \*p;

typedef struct

{

BiTree bn[STACK\_INIT\_SIZE];

int top;

int stacksize;

} SqStack;

SqStack L,Path;

void InitStack (SqStack \*);

void Push(SqStack \*,BiTree);

void Pop(SqStack \*,BiTree \*);

int Equal(BiTree,BiTree);

void CreateBiTree(BiTree\*);

void PostPush(BiTree,char);

int main()

{

InitStack(&L);

InitStack(&Path);

printf("请按照先序遍历顺序输入二叉树的结点(用#表示空结点):\n");

CreateBiTree(&bt);

printf("\n请输入待查找的结点(A-J):");

char a;

fflush(stdin);

scanf("%c",&a);

while(a<'A'||a>'J')

{

printf("\n输入有误，请重新输入:");

scanf("%c",&a);

}

PostPush(bt,a);

return 0;

}

//按先序次序输入二叉树中结点的值，构造二叉链表表示的二叉树

void CreateBiTree(BiTree\* T)

{

char ch;

scanf("%c",&ch);

if(ch=='#') \*T=NULL;

else

{

\*T=(BiTree)malloc(sizeof(BiTNode));

(\*T)->data=ch;

CreateBiTree(&(\*T)->lchild);

CreateBiTree(&(\*T)->rchild);

}

}

void visit(BiTNode e)

{

printf("%c ",e);

}

//初始化栈

void InitStack (SqStack \*s)

{

int i;

s->top=0;

}

//压栈

void Push(SqStack \*s,BiTree e)

{

if(s->top>=STACK\_INIT\_SIZE)

{

printf("ERROR!");

return;

}

s->bn[s->top] = e;

s->top++;

}

//出栈

void Pop(SqStack \*s,BiTree \*e)

{

if(s->top>=STACK\_INIT\_SIZE)

{

printf("ERROR\n");

return;

}

s->top--;

\*e = s->bn[s->top];

}

//判断两结点是否相等

int Equal(BiTree e1,BiTree e2)

{

if(e1==NULL||e2==NULL)

return 0;

if(e1->data == e2->data && e1->lchild == e2->lchild && e1->rchild == e2->rchild)

return 1;

else

return 0;

}

void PostPush(BiTree root,char ch)

{

int i;

BiTree p,q;

p=root;

q=NULL;

while(p!=NULL||L.top!=0)

{

if(p)

{

Push(&L,p);

p=p->lchild;

}

else

{

Pop(&L,&p);

Push(&L,p);

if(p->rchild==NULL||p->rchild==q)

{

if(p->data==ch)

{

while(L.top>0)

{

Pop(&L,&p);

Push(&Path,p);

}

while(Path.top>1)

{

Pop(&Path,&p);

printf("%c->",p->data);

}

printf("%c",ch);

return;

}

else

{

q=p;

Pop(&L,&p);

p=NULL;

}

}

else p=p->rchild;

}

}

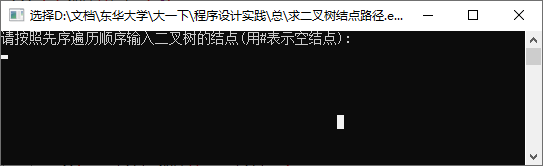
}

# 调试分析

1. 课题的难点在于如何有效遍历二叉树并记录路径，一开始确实没什么思路。后来看到非递归后序遍历的方法，就直接上手。
2. 低估了非递归后序遍历的难度，实现的时候确实没有递归，但却用循环在栈里保存了完整的后序遍历节点。当然得到的路径就不对劲。最后花了不少时间搞懂非递归后序遍历与递归后序遍历的区别。

# 用户手册

1. 演示程序的运行环境为Windows Vista系统，Microsoft Visual Studio 6.0中的Microsoft Visual C++ 6.0中运。执行文件为：求二叉树结点路径.exe
2. 进入演示程序后即显示DOS形式的界面：

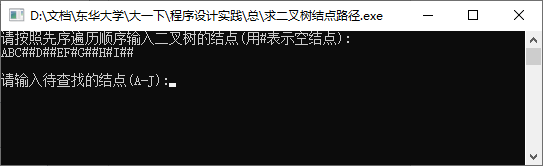


1. 按要求输入任意一个二叉树先序遍历顺序结点即可

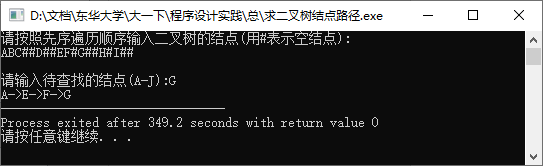
# 测试结果

按照先序遍历顺序输入二叉树结点，并用#表示空节点

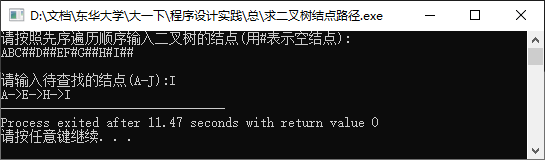
（例如：ABC##D##EF#G##H#I##）



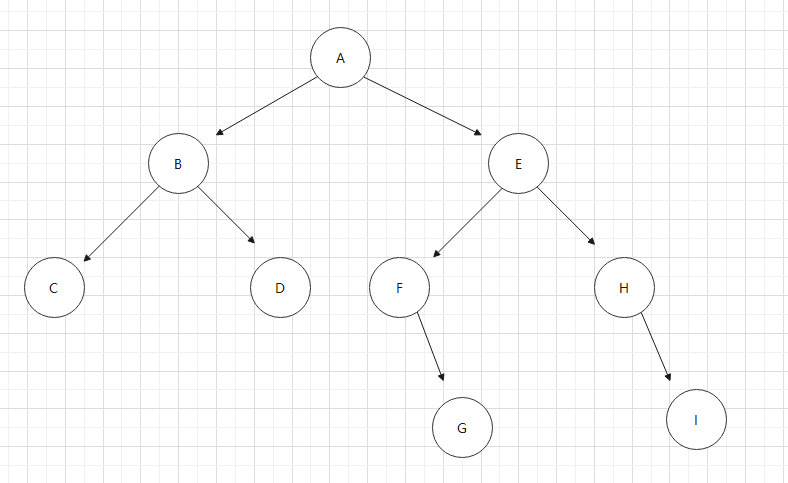
输入一个待查结点（例如：G）



再试一个（例如：I）



以下是测试数据的二叉树图形



# 总结与分析：

通过这次实践，我对二叉链表、顺序栈以及递归的了解又更深一步了。该课题中为我们设置了一个难题：如何不采用递归方法而后序遍历二叉树。最初，我不知从何下手，后来，在查阅了相关资料后，我终于理清了该方法的大致思路，在不断的改进中完成了非递归后序遍历算法。也明白了即使我们认真学习课本知识也还远远不够，比如非递归后序遍历我之前就不懂，但它用来寻找路径是那么多的有效。我们在平时的学习中，也要尝试去接触我们没见过的东西，也许那是我们遇到过的某个问题的一个解决思路。

# 设计题目三

迷宫问题求解

# 任务：

迷宫问题是取自心理学的一个古典实验。实验中，把一只老鼠从一个没有顶的大盒子的门放入，在盒中设置了许多墙，对行进的方向形成了多处阻挡。盒子仅仅有一个出口，在出口处放置了一块奶酪，吸引老鼠在迷宫中寻找道路以到达出口。重复对老鼠进行上述实验，看老鼠能在多久找到出口。

# 功能要求：

请设计一个算法实现迷宫问题求解。

# 需求分析：

该程序需要用户自行绘制迷宫，（也可以自行设置起点和终点），然后通过顺序栈存储经过的结点路径，从而得出迷宫路径。

# 概要设计

#include "stdio.h"

#include<stdio.h>

#include<assert.h>

#include<stdlib.h>

#include<string.h>

#include<windows.h>

#define MAZEROW 10//迷宫行数

#define MAZECOL 10//迷宫列数

#define MAXSIZE 100//最大元素数

//迷宫中的坐标位置

typedef struct Position

{

int x;//行号

int y;//列号

} PosType, \*PPosType;

typedef struct

{

PosType seat;//坐标位置

int dir;//

} SElemType;

typedef struct Stack

{

SElemType data[MAXSIZE];

int top;

} MazeStack, \*pMazeStack;

void InitMazeStack(pMazeStack sm);

void PushStack(pMazeStack sm, SElemType data);

void PopStack(pMazeStack sm, SElemType \*tmp);

int EmptyStack(pMazeStack sm);

void FootPrint(PosType curpos, char maze[MAZEROW][MAZECOL]);

void MarkPrint(PosType curpos, char maze[MAZEROW][MAZECOL]);

PosType NextPos(PosType curpos, int dir);

int IsPass(PosType curpos,char maze[MAZEROW][MAZECOL]);

int MazePath(char maze[MAZEROW][MAZECOL], PosType start, PosType end);

void PrintMaze(char maze[MAZEROW][MAZECOL]);

void PrintPath(char maze[MAZEROW][MAZECOL]);

void PrintSeat(char maze[MAZEROW][MAZECOL]);

void Exploration(char maze[MAZEROW][MAZECOL], PosType start, PosType end);

程序调用关系如下：

迷宫求解

四邻域探索迷宫 用栈存储

输出路径及地图

输出迷宫地图

# 详细设计

#include "stdio.h"

#include<stdio.h>

#include<assert.h>

#include<stdlib.h>

#include<string.h>

#include<windows.h>

#define MAZEROW 10//迷宫行数

#define MAZECOL 10//迷宫列数

#define MAXSIZE 100//最大元素数

//迷宫中的坐标位置

typedef struct Position

{

int x;//行号

int y;//列号

} PosType, \*PPosType;

typedef struct

{

PosType seat;//坐标位置

int dir;//dir = 1,2,3,4分别表示东、南、西、北四个方向

} SElemType;

typedef struct Stack

{

SElemType data[MAXSIZE];

int top;

} MazeStack, \*pMazeStack;

void InitMazeStack(pMazeStack sm);

void PushStack(pMazeStack sm, SElemType data);

void PopStack(pMazeStack sm, SElemType \*tmp);

int EmptyStack(pMazeStack sm);

void FootPrint(PosType curpos, char maze[MAZEROW][MAZECOL]);

void MarkPrint(PosType curpos, char maze[MAZEROW][MAZECOL]);

PosType NextPos(PosType curpos, int dir);

int IsPass(PosType curpos,char maze[MAZEROW][MAZECOL]);

int MazePath(char maze[MAZEROW][MAZECOL], PosType start, PosType end);

void PrintMaze(char maze[MAZEROW][MAZECOL]);

void PrintPath(char maze[MAZEROW][MAZECOL]);

void PrintSeat(char maze[MAZEROW][MAZECOL]);

void Exploration(char maze[MAZEROW][MAZECOL], PosType start, PosType end);

void menu()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*★欢迎来到迷宫探索★\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 0.退出迷宫 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1.迷宫地图 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2.迷宫探索 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//探索路径

void Exploration(char maze[MAZEROW][MAZECOL], PosType start, PosType end)

{

printf("迷宫地图：\n");

PrintMaze(maze);//打印迷宫初态

if (MazePath(maze,start, end))

{

printf("<--------------------------------------------------------->\n");

printf("探索成功!\n");

printf("<--------------------------------------------------------->\n");

printf("迷宫的现态(包括通路‘$’和已经探索过的位置'@')：\n");

PrintMaze(maze);

printf("探索到迷宫的一条通路：\n");

PrintPath(maze);

printf("迷宫通路坐标：\n");

PrintSeat(maze);

}

else

{

printf("此迷宫不存在通路!\n");

PrintMaze(maze);

}

}

int main()

{

int input = 0;

//入口坐标

PosType start = { 0, 0 };

//出口坐标

PosType end = { 0, 0 };

do

{

//二维数组设置迷宫地图('#'代表墙，'0'代表路)，最外边的一层强是为了防止越界访问

char maze[MAZEROW][MAZECOL] =

{

{ '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' },

{ '#', '0', '0', '#', '0', '0', '0', '#', '0', '#' },

{ '#', '0', '0', '#', '0', '0', '0', '#', '0', '#' },

{ '#', '0', '0', '0', '0', '#', '#', '0', '0', '#' },

{ '#', '0', '#', '#', '#', '0', '0', '0', '0', '#' },

{ '#', '0', '0', '0', '#', '0', '0', '0', '0', '#' },

{ '#', '0', '#', '0', '0', '0', '#', '0', '0', '#' },

{ '#', '0', '#', '#', '#', '0', '#', '#', '0', '#' },

{ '#', '#', '0', '0', '0', '0', '0', '0', '0', '#' },

{ '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' }

};

menu();

printf("请选择:");

scanf("%d", &input);

switch (input)

{

case 1:

{

fflush(stdin);

printf("\n");

printf("迷宫地图：\n");

PrintMaze(maze);

}

break;

case 2:

fflush(stdin);

printf("请输入入口坐标((1,1)-(8,8)):");

scanf("%d,%d", &start.x, &start.y);

printf("请输入出口坐标((1,1)-(8,8)):");

scanf("%d,%d", &end.x, &end.y);

Exploration(maze, start, end);

break;

case 0:

exit(0);

break;

default:

printf("输入有误，请重新输入：\n");

break;

}

}

while (input);

return 0;

}

//初始化栈

void InitMazeStack(pMazeStack sm)

{

sm->top = -1;

}

//压栈

void PushStack(pMazeStack sm, SElemType data)

{

if (sm->top == MAXSIZE - 1)

{

printf("此栈已满，不能入栈\n");

}

else

{

sm->top++;

sm->data[sm->top] = data;

}

}

//出栈

void PopStack(pMazeStack sm, SElemType \*tmp)

{

if (sm->top == -1)

{

printf("栈空，不能出栈\n");

}

else

{

\*tmp = sm->data[sm->top];

sm->top--;

}

}

//判断栈是否为空，为空返回1，非空返回0

int EmptyStack(pMazeStack sm)

{

return sm->top == -1;

}

//走过的地方做标记‘$’

void FootPrint(PosType curpos, char maze[MAZEROW][MAZECOL])

{

maze[curpos.x][curpos.y] = '$';

}

//走不通的地方做标记‘@’

void MarkPrint(PosType curpos, char maze[MAZEROW][MAZECOL])

{

maze[curpos.x][curpos.y] = '@';

}

//找下一个位置作为当前位置，顺时针开始

PosType NextPos(PosType curpos, int dir)

{

switch (dir)

{

case 1:

//‘1’代表东，行号不变，列号加1

curpos.y++;

break;

case 2:

//‘2’代表南，行号加1，列号不变

curpos.x++;

break;

case 3:

//‘3’代表西，行号不变，列号减1

curpos.y--;

break;

case 4:

//‘4’代表北，行号减1，列号不变

curpos.x--;

break;

default:

printf("参数有误!\n");

break;

}

return curpos;

}

//判断当前位置是否可以通过，既为‘0’可以通过。('$'代表已经走过、‘@’代表死胡同)

//返回1可以通过，返回0不能通过

int IsPass(PosType curpos, char maze[MAZEROW][MAZECOL])

{

if (maze[curpos.x][curpos.y] == '0')

{

return 1;

}

else

{

return 0;

}

}

//求一条从入口到出口的路径

int MazePath(char maze[MAZEROW][MAZECOL], PosType start, PosType end)

{

MazeStack sm;

InitMazeStack(&sm);

//设置入口坐标为当前位置

PosType curpos = start;

do

{

//如果当前位置可走

if (IsPass(curpos, maze))

{

FootPrint(curpos, maze);//做一个已经走过的标记

SElemType data;

data.seat = curpos;

data.dir = 1;

PushStack(&sm, data);//将当前位置加入路径，入栈

//如果当前位置是出口，则结束

if (curpos.x == end.x&&curpos.y == end.y)

{

return 1;

}

//否则切换当前位置的东边为下一个位置

curpos = NextPos(curpos, 1);

}

else//当前位置不能通过

{

//栈不为空，还有路径可查

if (!EmptyStack(&sm))

{

SElemType data;

PopStack(&sm, &data);//出栈一次，保留数据

//判断出栈位置的四面是否全被探索，是则标记为死胡同在出栈一次

while (data.dir == 4 && !EmptyStack(&sm))

{

MarkPrint(data.seat, maze);

PopStack(&sm, &data);

}

//上一次出栈的位置必然是其他方向没有探索

//换下一方向探索

if (data.dir < 4)

{

data.dir++;

PushStack(&sm, data);//将当前位置入栈纳为路径

curpos = NextPos(data.seat, data.dir);//切换当前位置为该方向上的东邻位置

}

}

}

printf("<--------------------------------------------------------->\n");

printf("开始探索迷宫：('$'代表路径，'@'代表探索过但是不通的地方)\n");

printf("<--------------------------------------------------------->\n");

PrintMaze(maze);

system("cls");//清屏

}

while (!EmptyStack(&sm)); //栈不为空，一直存在位置探索

return 0;

}

//遍历二维数组，打印迷宫当前的状态

void PrintMaze(char maze[MAZEROW][MAZECOL])

{

int i = 0;

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

{

int j = 0;

for (j = 0; j < MAZECOL; j++)

{

printf("%c ", maze[i][j]);

}

printf("\n");

}

printf("\n");

}

//打印路径

void PrintPath(char maze[MAZEROW][MAZECOL])

{

int i = 0;

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

{

int j = 0;

for (j = 0; j < MAZECOL; j++)

{

if (i == 0 || j == 0 || i == MAZEROW - 1 || j == MAZECOL - 1 || maze[i][j] == '$')

{

printf("%c ", maze[i][j]);

}

else

{

printf(" ");

}

}

printf("\n");

}

printf("\n");

}

//打印通路坐标

void PrintSeat(char maze[MAZEROW][MAZECOL])

{

int i = 0;

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

{

int j = 0;

for (j = 0; j < MAZECOL; j++)

{

if (maze[i][j] == '$')

{

printf("(%d,%d)-->", i, j);

}

}

}

printf("over\n");

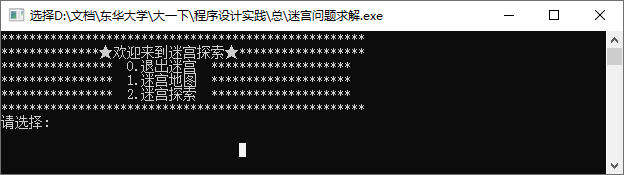
}

# 调试分析

1. 自定义迷宫时，0表示可走通路，#为不可走通路，@为走过的不通路，$为走过的通路
2. 找到通路时，把路径加入栈中并标记路径$，找不到通路则出栈并将出栈位置标记为@
3. 重复2知道到达指定终点

# 用户手册

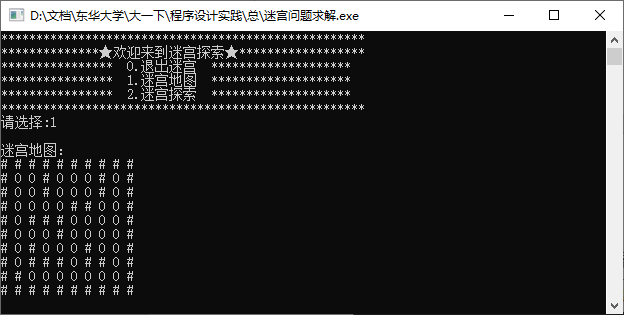
1. 演示程序的运行环境为Windows Vista系统，Microsoft Visual Studio 6.0中的Microsoft Visual C++ 6.0中运。执行文件为：迷宫问题求解.exe
2. 进入演示程序后即显示DOS形式的界面：



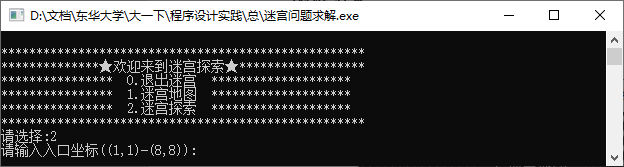
1. 按要求输入相应指令即可

# 测试结果

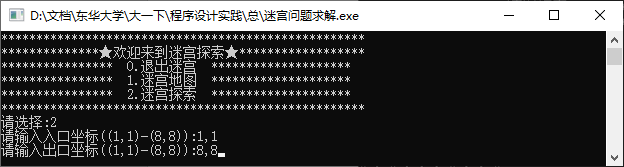
输入指令1，看看迷宫地图



再输入指令2，开始寻找路径



输入入口坐标（1,1）



再输入出口坐标（8,8），回车开始找路径



路径寻找成功

输入0退出程序

# 总结与分析：

通过这次实践，我对顺序栈有了更深的理解。课题限制性很小，因此有很多种方法。而对于路径而言，用栈明显是更好的方式。因为以前用其他语言做过A\*寻路，所以在深入了解栈之后觉得迷宫问题不是特别的难下手。