中南大学

数据结构试验报告

题 目 实验二

学生姓名 王云鹏

学 号 8213180228

指导老师 郑 瑾

学 院 计算机学院

专业班级 物联网1802

完成时间 2020.6

指导老师评定 签名

实验二

1. 需求分析

1．栈和队列操作的实现（验证性实验）

问题描述

(1) 建立一个顺序栈。

(2) 建立一个循环顺序队列。

(3) 分别实现栈和队列的基本操作。

2．数制转换问题（设计性实验）

问题描述

十进制数*n*和其他*d*进制数的转换是计算机实现计算的基本问题，其解决方案很多，其中最简单的

方法基于下列原理，即除*d*取余法。例如，(1348)10＝(2504)8。

*N*

*n* div 8

*n* mod 8

1348 168 4

168 21 0

21 2 5

2 0 2

从中可以看出，最先产生的余数4是转换结果的最低位，这正好符合栈先进后出的特性，所以可

以用顺序栈来模拟这个过程。

基本要求

从键盘输入任意一个非负的十进制整数，打印输出与其等值的八进制数。由于上述的计算过程是

从低位到高位顺序产生的八进制数的各个数位，而打印输出一般来说应从高位到低位进行，恰好与计

算过程相反。因此，可以先将计算过程中得到的八进制数的各位进栈，待相对应的八进制数的各位均

产生以后，再使其按顺序出栈，并打印输出。这样就得到了与输入的十进制数相对应的八进制数。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据

5. 停车场管理（综合性实验）

问题描述

设停车场内是一个可停放*n*辆汽车的狭长区域，且只有一个大门可供汽车进出。在停车场内，汽

车按到达时间的先后顺序，依次由北向南排列（假设大门在最南端，最先到达的那辆车停放在车场的

最北端）。若车场内已停满*n*辆汽车，则后来的汽车只能在门外的便道上等候。一旦有车开走，则排在

便道上的第一辆车即可开入。当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它

让路，待该车开出大门外，其他车再按原次序返回停车场。每辆停放在车场的车离开停车场时，必须

按其停留时间的长短交费。试为停车场编制按上述要求进行管理的模拟程序。

测试数据

设*n*＝2，输入数据为(A, 1, 5)、(A, 2, 10)、(D, 1, 15)、(A, 3, 20)、(A, 4, 25)、(A, 5, 30)、(D, 2, 35)、

(D, 4, 40)和(E, 0, 0)。每一组输入数据包括3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，其中，“A”表示到达，“D”表示离去，“E”表示输入结束。

基本要求

以顺序栈模拟停车场，以链队列模拟便道，按照从终端读入的输入数据序列进行模拟管理。每一

组输入数据包括3项：是“到达”还是“离去”、汽车牌照号码及“到达”或“离去”的时刻。对每一

组输入数据进行操作后的输出数据为：如果是到达的车辆，则输出其在停车场内或便道上的停车位置；

如果是离去的车辆，则输出其在停车场内的停留时间和应交的费用（在便道上停留的时间不收费）。

栈以顺序结构实现，队列以链表实现。

实现提示

需另设一个栈，临时停放为给要离去的汽车让路而从停车场退出来的汽车，也用顺序存储结构实

现。输入数据按到达或离去的时刻有序。栈中每个元素表示一辆汽车，包含两个数据项：汽车的牌照

号码和进入停车场的时刻。

选作内容

(1) 两个栈共享空间，思考应开辟数组的空间是多少？

(2) 汽车可有不同种类，则它们的占地面积不同，收费标准也不同，如1辆客车和1.5辆小汽车的

占地面积相同，1辆十轮卡车占地面积相当于3辆小汽车的占地面积。

(3) 汽车可以直接从便道上开走，此时排在它前面的汽车要先开走让路，然后再依次排到队尾。

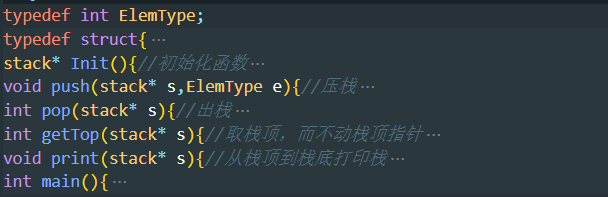
(4) 停放在便道上的汽车也收费，收费标准比停放在停车场的车低，请思考如何修改结构以满足

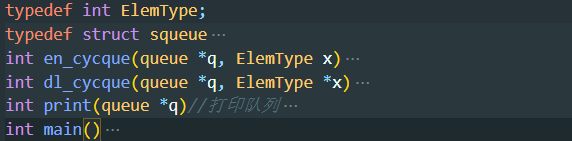
这种要求。

1. 概要设计

* 验证性实验

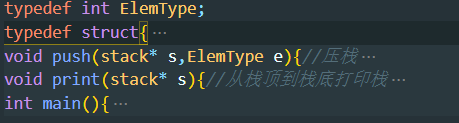
函数：





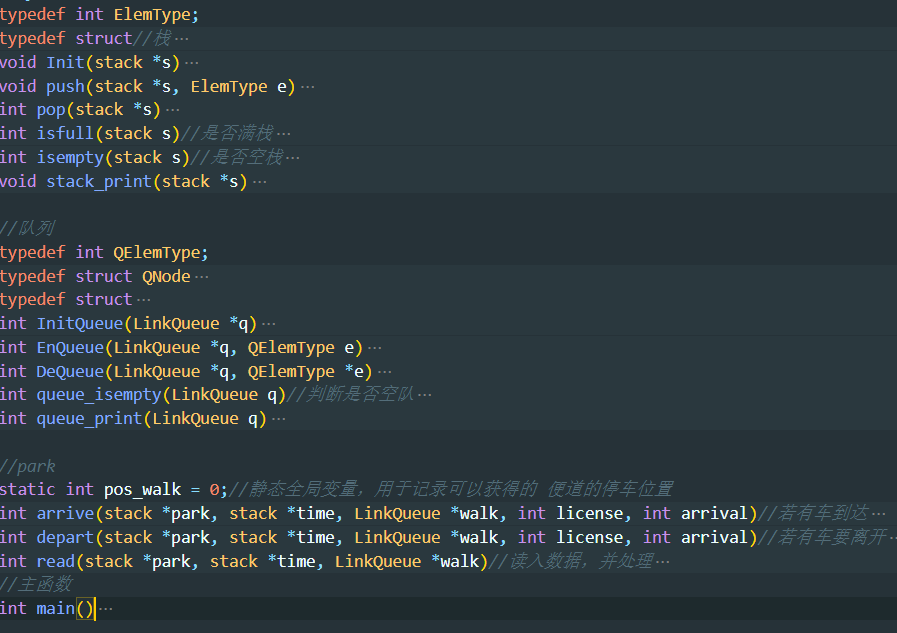
* 设计性实验

函数：



* 综合性实验

函数：



1. 详细设计

* 验证性实验

|  |
| --- |
| *#include*<stdio.h>  *#include*<stdlib.h>  *#define* maxsize 10  typedef int ElemType;  typedef struct{      ElemType\* base;      ElemType top;*//下标*  }stack;  stack\* Init(){*//初始化函数*      stack\* s;      s->base=(ElemType\*)malloc(maxsize\*sizeof(ElemType));*//申请空间*  *if*(s==NULL){          printf("malloc fail in Init");  *return* NULL;      }      s->top=0;  *return* s;  }  void push(stack\* s,ElemType e){*//压栈*  *if*(s->top==maxsize){*//溢出*          printf("stack is overflow");  *return*;      }      \*(s->base+s->top)=e;*//压入*      s->top++;*//栈顶上移*  }  int pop(stack\* s){*//出栈*  *if*(s->top==0){*//空栈*          printf("stack is underflow");  *return* 0;      }      s->top--;*//栈顶下移*  *return* \*(s->base+s->top);*//返回移出的值*  }  int getTop(stack\* s){*//取栈顶，而不动栈顶指针*  *if*(s->top==0){*//空栈*          printf("stack is underflow");  *return* 0;      }  *return* \*(s->base+s->top-1);  }  void print(stack\* s){*//从栈顶到栈底打印栈*      printf("stack:\n");  *for*(int i=s->top-1;i>=0;i--){          printf("%d\n",\*(s->base+i));      }  }  int main(){      stack\* s=(stack\*)malloc(sizeof(stack));*//产生一个栈*      s->base=(ElemType\*)malloc(maxsize\*sizeof(ElemType));  *if*(s==NULL){          printf("malloc fail in main");  *return* 0;      }      s->top=0;*//栈顶指针初始为0，该指针仅代表数组的下标*      push(s,1);      push(s,2);      push(s,3);      print(s);      printf("pop:%d\n",pop(s));      print(s);      printf("getTop:%d\n",getTop(s));  *return* 0;  } |

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#define* queuesize 100  *#define* overflow 1  *#define* underflow -1  *#define* ok 0  typedef int ElemType;  typedef struct squeue  {      ElemType sq[queuesize];      ElemType front;      ElemType rear;  } queue;  int en\_cycque(queue \*q, ElemType x)  {                                      *//入队*      int i = (q->rear + 1) % queuesize; *//循环队列*  *if* ((i == q->front))      {                         *//空出一个不用*          printf("overflow\n"); *//溢出*  *return* overflow;      }      q->sq[q->rear] = x; *//写入,rear指向队列最后一个元素后面的位置*      q->rear = i;        *//尾指针移动*  *return* ok;  }  int dl\_cycque(queue \*q, ElemType \*x)  { *//出队*  *if* (q->front == q->rear)      { *//空队*          printf("underflow\n");  *return* underflow;      }      \*x = q->sq[q->front];                  *//用x带出*      q->front = (q->front + 1) % queuesize; *//头指针后移，从队头删除*  *return* ok;  }  int print(queue \*q)*//打印队列*  {  *if* (q->front == q->rear)*//溢出*      {          printf("underflow\n");  *return* underflow;      }  *else* *if* ((q->rear + 1) % queuesize == q->front)*//空队*      {          printf("overflow");  *return* overflow;      }  *else* *if* (q->rear > q->front)*//若队尾指针在队头指针后面*      {  *for* (int i = q->front; i < q->rear; i++)*//遍历打印*          {              printf("%d\n", q->sq[i]);          }      }  *else* *if* (q->rear < q->front)*//若队尾指针在队头指针前面*      {  *for* (int i = q->front; i <= queuesize; i++)          {              printf("%d\n", q->sq[i]);          }  *for* (int i = 0; i < q->rear; i++)          {              printf("%d\n", q->sq[i]);          }      }  }  int main()  {      queue q;*//生成循环队列*      q.front=0;*//使队头队尾皆为0*      q.rear=0;      en\_cycque(&q, 1);      en\_cycque(&q, 2);      en\_cycque(&q, 3);      print(&q);      int e;*//带出删掉的元素*      dl\_cycque(&q,&e);  *// printf("%d\n",e);//查看删掉的元素*      print(&q);  *return* 0;  } |

* 设计性实验

|  |
| --- |
| *#include*<stdio.h>  *#include*<stdlib.h>  *#define* maxsize 10  typedef int ElemType;  typedef struct{      ElemType\* base;      ElemType top;*//下标*  }stack;  void push(stack\* s,ElemType e){*//压栈*  *if*(s->top==maxsize){*//溢出*          printf("stack is overflow");  *return*;      }      \*(s->base+s->top)=e;*//压入*      s->top++;*//栈顶上移*  }  void print(stack\* s){*//从栈顶到栈底打印栈*  *for*(int i=s->top-1;i>=0;i--){          printf("%d",\*(s->base+i));      }  }  int main(){  *//产生一个栈*      stack\* s=(stack\*)malloc(sizeof(stack));*//产生一个栈*      s->base=(ElemType\*)malloc(maxsize\*sizeof(ElemType));  *if*(s==NULL){          printf("malloc fail in main");  *return* 0;      }      s->top=0;*//栈顶指针初始为0，该指针仅代表数组的下标*  *//转换数制*      int d=8;*//要转化为的数制*      int dest=1348;*//目标数*      int cout=dest;*//记录，用于输出*      int remainder=0;*//余数*  *while*(1){          remainder=dest%d;*//取余，压到栈中*          push(s,remainder);          dest=dest/d;  *if*(dest==0){  *break*;          }      }      printf("十进制数%d转化为八进制数后为：",cout);      print(s);  } |

* 综合性实验

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *//栈*  *#define* maxsize 2*//栈的空间*  *#define* full 1  *#define* notfull -1  *#define* empty 0  *#define* notempty 1  *#define* ok 0  *#define* fail 1  *#define* underflow -1  typedef int ElemType;  typedef struct*//栈*  {      ElemType \*base;*//基指针*      ElemType top; *//下标*  } stack;  void Init(stack \*s)  {                                                             *//初始化函数*      s->base = (ElemType \*)malloc(maxsize \* sizeof(ElemType)); *//申请空间*  *if* (s == NULL)      {          printf("malloc fail in Init");  *return*;      }      s->top = 0;*//初始化时下标为0*  }  void push(stack \*s, ElemType e)  { *//压栈*  *if* (s->top == maxsize)      { *//溢出*          printf("stack is overflow");  *return*;      }      \*(s->base + s->top) = e; *//压入*      s->top++;                *//栈顶上移*  }  int pop(stack \*s)  { *//出栈*  *if* (s->top == 0)      { *//空栈*          printf("stack is underflow");  *return* 0;      }      s->top--;                   *//栈顶下移*  *return* \*(s->base + s->top); *//返回移出的值*  }  int isfull(stack s)*//是否满栈*  {  *if* (s.top == maxsize)      {  *return* full;      }  *else*      {  *return* notfull;      }  }  int isempty(stack s)*//是否空栈*  {  *if* (s.top == 0)      {  *return* empty;      }  *else*      {  *return* notempty;      }  }  void stack\_print(stack \*s)  { *//从栈顶到栈底打印栈*      printf("stack:\n");  *for* (int i = s->top - 1; i >= 0; i--)      {          printf("%d\n", \*(s->base + i));      }  }  *//队列*  typedef int QElemType;  typedef struct QNode  { *//节点*      QElemType data;      struct QNode \*next; *//链队列*  } QNode;  typedef struct  {      QNode \*front; *//头指针，带空头节点*      QNode \*rear;  *//尾指针*  } LinkQueue;  int InitQueue(LinkQueue \*q)  {                                                        *//初始化*      q->front = q->rear = (QNode \*)malloc(sizeof(QNode)); *//申请空间，front指向头节点（空的）*  *if* (q->front == NULL)      { *//若是申请失败*          printf("malloc fail in InitQueue");  *return* fail;      }      q->front->next = NULL; *//空队的头节点下一个自然为空*  *return* ok;  }  int EnQueue(LinkQueue \*q, QElemType e)  {                                              *//插入*      QNode \*p = (QNode \*)malloc(sizeof(QNode)); *//将要插入的节点*  *if* (p == NULL)      {          printf("malloc fail in EnQueue");  *return* fail;      }      p->data = e;       *//p的值为e*      p->next = NULL;    *//在尾部插入，其下一个为空*      q->rear->next = p; *//接到尾节点的下一个*      q->rear = p;       *//p为新的尾节点*  *return* ok;  }  int DeQueue(LinkQueue \*q, QElemType \*e)  { *//删除*  *if* (q->front == q->rear)      { *//空队无法删除*          printf("underflow\n");  *return* underflow;      }      QNode \*p = q->front->next; *//需要删除的节点*      \*e = p->data;              *//用e带出删掉的值*      q->front->next = p->next;  *//将要删除的节点从链中断开*  *if* (q->rear = p)      {                       *//加入原来队中只有一个节点*          q->rear = q->front; *//删完后为空队，头节点与尾节点指到一起*      }      free(p);  *return* ok;  }  int queue\_isempty(LinkQueue q)*//判断是否空队*  {  *if* (q.front == q.rear)      {  *return* empty;      }  *else*      {  *return* notempty;      }  }  int queue\_print(LinkQueue q)  {                       *//打印队列，从队头到队尾*      QNode \*p = q.front; *//头节点为空， 不需要打印*  *while* (1)      {          p = p->next;             *//下一个节点*          printf("%d\n", p->data); *//打印值*  *if* (p == q.rear)          { *//到队尾退出*  *break*;          }      }  }  *//park*  static int pos\_walk = 0;*//静态全局变量，用于记录可以获得的 便道的停车位置*  int arrive(stack \*park, stack \*time, LinkQueue \*walk, int license, int arrival)*//若有车到达*  {  *if* (isfull(\*park) == notfull)*//若停车场还有位置*      {          push(park, license);*//将车牌号和到达时间压入栈*          push(time, arrival);          printf("got NO.%d position in park\n", park->top);*//输出在停车场中的位置*      }  *else* *if* (isfull(\*park) == full)*//若停车场已满，新来的车停入便道*      {          pos\_walk++;*//便道进入一辆车*          EnQueue(walk, license);*//将车牌号压入便道栈*          printf("got NO.%d position in sidewalk\n", pos\_walk);*//输出在便道中的位置*      }  *return* ok;  }  int depart(stack \*park, stack \*time, LinkQueue \*walk, int license, int arrival)*//若有车要离开*  {      stack temp\_license; *//用来存暂时退开的车*      Init(&temp\_license);      stack temp\_time;      Init(&temp\_time);  *while* (1)*//1号车离开，先将2号车存入另外一个栈，pop(1),push(2)*      {          int med\_license = pop(park); *//取出的车牌号*          int med\_time = pop(time);*//被取出车的进入时间*  *if* (med\_license != license)          { *//不是要离开的车，就存入另一个栈*              push(&temp\_license, med\_license);              push(&temp\_time, med\_time);  *continue*;          }  *else* *if* (med\_license == license)*//是要离开的车，输出停车时间和停车费，并将另外一个栈的车还回来*          {              int park\_time = arrival - med\_time; *//停车时间*              int moneyEvery = 10;                *//每单位时间停车费*              int money = park\_time \* moneyEvery; *//停车费*              printf("the car of %d have parked for %d time\n", license,park\_time);*//输出停车时间*              printf("fees is %d\n", money);*//输出停车费*  *while* (isempty(temp\_license) == notempty)*//若temp未空，就一直出栈*              { *//从temp还回暂时退开的车*                  push(park, pop(&temp\_license));                  push(time, pop(&temp\_time));              }  *break*; *//还完车就算这辆车成功离开*          }      }  *//离开一辆车后，若便道上有车等待，则车进入停车场*  *if* (queue\_isempty(\*walk) == notempty)*//若便道上有车等待*      {          QElemType e;          DeQueue(walk, &e);*//从便道驶出一辆车*          pos\_walk--;*//便道停车位置减一*          push(park, e);       *//走一辆车就进一辆车*          push(time, arrival); *//其进入停车场时间就是上一辆车离开的时间*      }  *return* ok;  }  int read(stack \*park, stack \*time, LinkQueue \*walk)*//读入数据，并处理*  {  *while* (1)      {          char order;*//A或D或E*          int license;*//车牌号*          int arrival;*//到达时间*          scanf("%c,%d,%d", &order, &license, &arrival);  *if* (order == 'A')*//若有车到达*          {              arrive(park, time, walk, license, arrival);          }  *else* *if* (order == 'D')*//若有车离开*          {              depart(park, time, walk, license, arrival);          }  *else* *if* (order == 'E')*//模拟结束*          {  *break*;          }      }  *return* ok;  }  *//主函数*  int main()  {      stack park;*//停车场*      Init(&park);      stack time;*//进入停车场的时间*      Init(&time);      LinkQueue walk;*//便道*      InitQueue(&walk);      read(&park, &time, &walk);*//读入处理数据*  *return* 0;  } |

4.调试分析

1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。

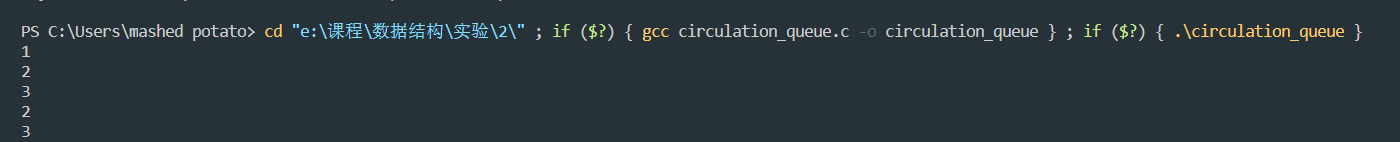
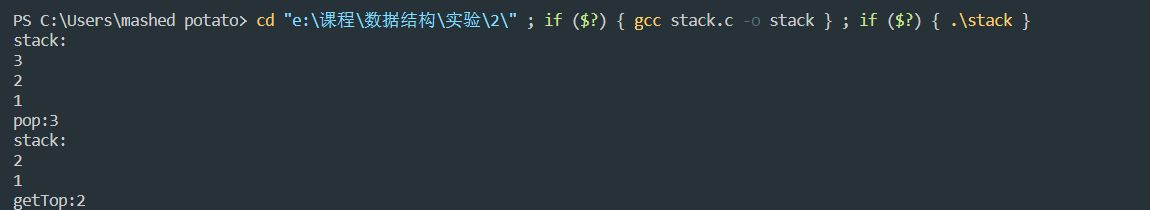
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

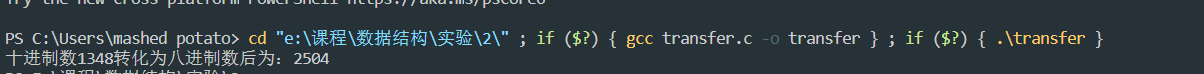
按照屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

6.测试程序运行结果

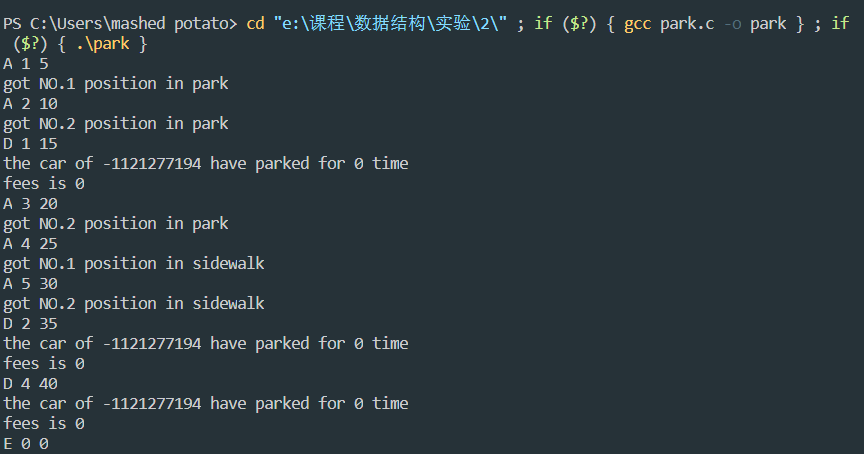
* 验证性实验



* 设计性实验



* 综合性实验



1. **心得体会**

通过本次实验，使我对数据结构有了更深的理解，对指针的运用更加熟练，熟悉了对函数的定义和操作。