

《数据结构》

实验报告

姓名： 庄李晨

学号： 10522238

成绩：

电子与计算机工程学院

School of Electronic & Computer Engineering

2023年10月

【实验名称】队列的实现与应用

【实验目的】

一、掌握链队列的基本操作，实现队列的入队、出队、返回队首元素等基本运算。

二、掌握循环队列的基本操作，实现队列的入队、出队、返回队首元素等基本运算。

【实验内容】

1. 实验要求：

1. 仔细阅读链队列实验代码，补充下面6个函数的实现过程。补充完成main（）函数的定义，调试链队列程序，给出测试用例及测试结果。

（1）始化链队列为空队列

bool InitQueue (Queue &Q) {}

（2）新元素e入队列，成功返回true，失败返回false

bool EnQueue (Queue &Q, QElemType e) {}

（3）出队列，即删除队首元素，并用e返回出元素值 ，成功返回true，失败返回false

bool DeQueue (Queue &Q, QElemType &e) {}

（4）读队首元素，用e返回队首元素，不出队 ，成功返回true，失败返回false

bool GetHead(QElemType &e){}

（5）遍历链队列，将队列数据元素依次输出

void Traverse(Queue Q ){}

（6）销毁链队列

void Destroy(Queue &Q){}

2.仔细阅读循环队列实验代码，补充下面6个函数的实现过程。补充完成main（）函数的定义，调试循环队列程序，给出测试用例及测试结果。

（1）初始化长度为MAXQSIZE的空队列

bool InitQueue (SqQueue &Q,int MAXQSIZE ){}

（2）新元素e入队列，成功返回true，失败返回false

bool EnQueue (SqQueue &Q, QElemType e){}

（3）出队列，即删除队首元素，并用e返回出元素值 ，成功返回true，失败返回false

bool DeQueue (SqQueue &Q, QElemType &e) {}

（4）读队首元素，用e返回队首元素，不出队 ，成功返回true，失败返回false

bool GetHead(QElemType &e){}

（5）遍历队列

void Traverse(SqQueue Q){}

（6）销毁循环队列

void Destroy(SqQueue Q){}

3. 选作题目（三选一）

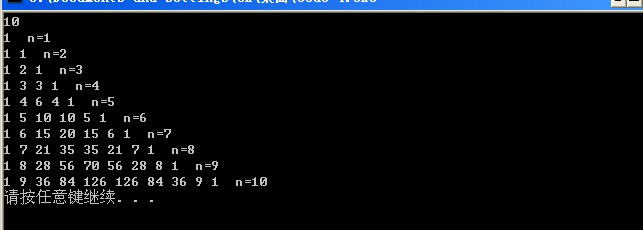
题目一、定义测试类，利用队列实现杨辉三角。

要求:（1）基于队列的操作来实现杨辉三角的不断生成过程。

输入形式：输入一个整数n ，0<=n<=20

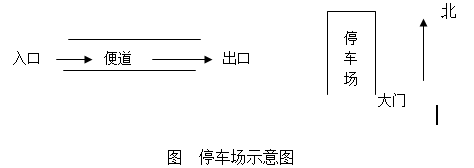
输出形式：打印出来前(n+1)行的杨辉三角数列

功能实现：输出前20层的杨辉三角序列



题目二、停车场管理

设停车场（如下图所示）内只有一个可停放几辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已经停满几量汽车，则后来的汽车只能在门外的便道上等候，一旦停车场内有车开走，则排在便道上的第一辆汽车即可开入；当停车场内某车辆要离开时，由于停车场是狭长的通道，在它之后开入车场的车辆必须先退出车场为它让路，待该车辆开出大门外后，为它让路的车辆再按原次序进入车场。在这里假设汽车不能从便道上开走。试设计一个停车场管理程序（这里只是一个假想的停车场管理，并不代表实际的停车场管理）。



分析：汽车在停车场内进出是按照栈的运算方式来实现的，先到的先进停车场；停车场的汽车离开停车场时，汽车场内其它汽车为该辆汽车让路，也是按栈的方式进行；汽车在便道上等候是按队列的方式进行的。因此，将停车场设计成一个栈，汽车让路也需要另一个栈来协助完成，汽车进出便道用队列来实现。

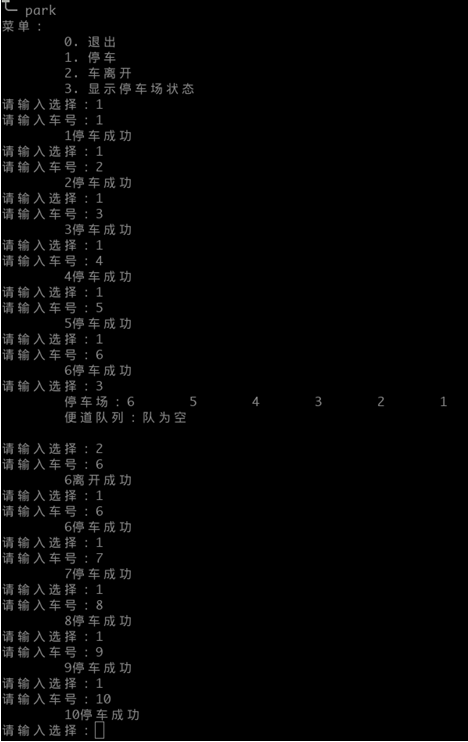
本设计，栈采用顺序栈结构，队列用链式存储结构。

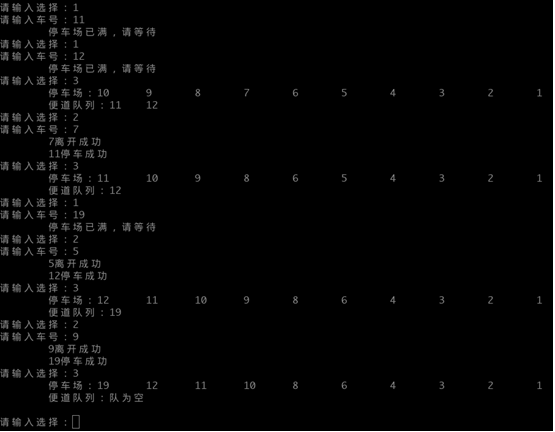
停车场管理的算法描述如下：

1）接受命令和车号，若是汽车要进停车场，先判断停车场栈是否满，若不满，则汽车入栈，否则汽车进入便道队列等候。

2）若是汽车要离开停车场，为给汽车让路，将停车场栈上若干辆汽车入临时栈，等这辆车出停车场后，临时栈中的汽车出栈，在回到停车场栈，然后看便道队列是否为空，若不空则说明有汽车等候，从队头取出汽车号，让该车进入停车场栈。

3）重复1），2）直到为退出命令（车号为0或负数）。





题目三、银行业务模拟与离散事件模拟

假设某银行有4个窗口对外接待客户，从早晨银行开门（开门9：00am，关门5：00pm）起不断有客户进入银行。由于每个窗口在某个时刻只能接待一个客户，因此在客户人数众多时需要在每个窗口前顺次排队，对于刚进入银行的客户（建议：客户进入时间使用随机函数产生），如果某个窗口的业务员正空闲，则可上前办理业务；反之，若4个窗口均有窗户所占，他便会排在人数最少的队伍后面。

1)编制一个程序以模拟银行的这种业务活动并计算一天中客户在银行逗留的平均时间。

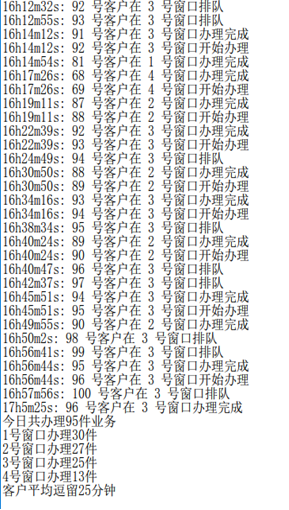
2) 建议有如下设置：

a) 客户到达时间随机产生，一天客户的人数设定为100人。

b) 银行业务员处理时间随机产生，平均处理时间10分钟。

3) 将一天的数据（包括业务员和客户）以文件方式输出。

测试数据由随机数产生器生成



1. 程序清单

**链队列**

#include <iostream>

#include <stdlib.h>

using namespace std;

typedef int QElemType;

// 结点定义

typedef struct QNode {

QElemType data; // 数据域

struct QNode\* next; // 指针域

}\*QueuePtr;

// 链队列定义

typedef struct {

QueuePtr front; // 队列的头指针

QueuePtr rear; // 队列的尾指针

int length;

}Queue; // 链队列

//初始化链队列为空队列

bool InitQueue(Queue& Q)

{

Q.front = Q.rear = (QueuePtr)(malloc(sizeof(QNode)));

if (!Q.front) exit(OVERFLOW);

Q.front->next = NULL;

cout << "初始化已成功" << endl;

return true;

}

//新元素 e 入队列，成功返回 true，失败返回 false

bool EnQueue(Queue& Q, QElemType e)

{

QueuePtr p; //新建一个链队节点

p = (QueuePtr)malloc(sizeof(QNode)); //初始化这个链队节点

if (!p) exit(OVERFLOW);

p->data = e; p->next = NULL; //新节点初始化

Q.rear->next = p; //将新节点纳入链队列

Q.rear = p;

cout << "元素" << e << "已入队列" << endl;

return true;

}

//出队列，即删除队首元素，并用 e 返回出元素值 ，成功返回 true，失败返回 false

bool DeQueue(Queue& Q, QElemType& e)

{

QueuePtr p; //新建一个链队节点

p = (QueuePtr)malloc(sizeof(QNode)); //初始化这个链队节点

if (Q.front == Q.rear) return false;

p = Q.front->next;

e = p->data;

cout << "元素" << e << "已出队列" << endl;

Q.front->next = p->next;

if (Q.rear == p) Q.rear = Q.front; //Q.rear指向着最后一个元素，这一步避免把Q.rear也删了

delete p;

return true;

}

//读队首元素，用 e 返回队首元素，不出队 ，成功返回 true，失败返回 false

bool GetHead(Queue& Q, QElemType& e)

{

if (Q.front == Q.rear) {

cout << "队列为空" << endl;

return false;

}

e = Q.front->next->data;

cout <<"队列首元素为" << e << endl;

return true;

}

//遍历链队列，将队列数据元素依次输出

void Traverse(Queue Q)

{

QNode\* p = Q.front->next;

cout << "该队列元素为:" << endl;

while (p/\* != Q.rear->next\*/)

{

cout << p->data<<" ";

p = p->next;

}

}

//销毁链队列

void Destroy(Queue& Q)

{

while (Q.front) {

Q.rear = Q.front->next; //一个接着一个的销毁节点

free(Q.front);

Q.front = Q.rear;

}

cout << "该链已销毁" << endl;

}

int main(void)

{

Queue lq;

int c = 0;

QElemType e;

while (c != 7)

{

cout << endl << "1. 初始化链队";

cout << endl << "2. 入队";

cout << endl << "3. 出队";

cout << endl << "4. 读队首";

cout << endl << "5. 遍历队列";

cout << endl << "6. 销毁队列";

cout << endl << "7. 退出";

cout << endl << "选择功能(1~7):";

cin >> c;

switch (c)

{

case 1:

{

InitQueue(lq);

break;

}

case 2:

{

cout << "请输入需要加入队列的值" << endl;

cin >> e;

EnQueue(lq, e);

break;

}

case 3:

{

DeQueue(lq,e);

break;

}

case 4:

{

GetHead(lq, e);

break;

}

case 5:

{

Traverse(lq);

break;

}

case 6:

{

Destroy(lq);

break;

}

case 7:

{

system("pause");

break;

}

}

}

}

**循环队列**

#include <iostream>

#include <stdlib.h>

using namespace std;

typedef int QElemType;

//循环队列定义

typedef struct {

QElemType\* base; // 存储空间基址

int rear; // 队尾指针

int front; // 队头指针

int queuesize; // 允许的最大存储空间

} SqQueue;

//初始化长度为 MAXQSIZE 的空队列

bool InitQueue(SqQueue& Q, int MAXQSIZE)

{

Q.queuesize = MAXQSIZE;

Q.base = (int\*)malloc(sizeof(int) \* (MAXQSIZE + 1));

Q.front = 0; //初始化的不同，会导致后面判断队列是否为空以及是否已满的不同

Q.rear = 0;

cout << "初始化完成" << endl;

return true;

}

//判断队列是否已满

bool isFull(SqQueue& Q) {

if ((Q.rear + 1) % (Q.queuesize + 1) == Q.front) return true;

else return false;

}

//判断是否为空

bool isEmpty(SqQueue& Q) {

return Q.front == Q.rear;

}

//新元素 e 入队列，成功返回 true，失败返回 false

bool EnQueue(SqQueue& Q, QElemType e)

{

if (isFull(Q)) {

cout << "该队列已满，入队失败" << endl;

return false;

}

else {

Q.base[Q.rear] = e;

Q.rear = (Q.rear + 1) % (Q.queuesize + 1);

cout << "元素" << e << "已成功入队" << endl;

return true; //成功返回1，失败返回0

}

}

//出队列，即删除队首元素，并用 e 返回出元素值 ，成功返回 true，失败返回 false

bool DeQueue(SqQueue& Q, QElemType& e)

{

if (isEmpty(Q)) {

cout << "队列为空" << endl;

return false;

}

else {

e = Q.base[Q.front];

Q.front = (Q.front + 1) % (Q.queuesize + 1);

cout << "元素" << e << "已出队列" << endl;

return true; //成功返回1，失败返回0

}

}

//读队首元素，用 e 返回队首元素，不出队 ，成功返回 true，失败返回 false

bool GetHead(SqQueue& Q, QElemType& e)

{

if (isEmpty(Q)) {

cout << "该队列为空" << endl;

return false;

}

e = Q.base[Q.front];

cout << "队首元素为" << e << endl;

return true;

}

//遍历队列

void Traverse(SqQueue Q)

{

if (isEmpty(Q)) {

cout << "该队列为空" << endl;

return;

}

for (int i = Q.front; i != Q.rear;i++)

{

cout << Q.base[i] << " ";

}

}

//销毁循环队列

void Destroy(SqQueue Q)

{

if (isEmpty(Q)) {

cout << "该队列为空" << endl;

return;

}

free(Q.base);

Q.base = NULL;

Q.queuesize = Q.front = Q.rear = 0;

cout << "该队列已销毁" << endl;

}

int main(void)

{

SqQueue sq;

int c = 0;

QElemType e;

QElemType max;

while (c != 7)

{

cout << endl << "1. 初始化循环队列";

cout << endl << "2. 入队";

cout << endl << "3. 出队";

cout << endl << "4. 读队首";

cout << endl << "5. 遍历队列";

cout << endl << "6. 销毁循环队列";

cout << endl << "7. 退出";

cout << endl << "选择功能(1~7):";

cin >> c;

switch (c)

{

case 1:

{

cout << "输入队列最大元素个数" << endl;

cin >> max;

InitQueue(sq,max);

break;

}

case 2:

{

cout << "输入需要放入队列的元素" << endl;

cin >> e;

EnQueue(sq, e);

break;

}

case 3:

{

DeQueue(sq, e);

break;

break;

}

case 4:

{

GetHead(sq, e);

break;

}

case 5:

{

Traverse(sq);

break;

}

case 6:

{

Destroy(sq);

break;

}

case 7:

{

system("pause");

break;

}

}

}

}

**杨辉三角**

#include<iostream>

using namespace std;

typedef int ElemType;

typedef struct LinkNode {

ElemType data;

struct LinkNode\* next;

}LinkNode;

typedef struct {

LinkNode\* front, \* rear;

}LinkQueue;

// 初始化队列

bool InitQueue(LinkQueue& Q) {

Q.front = Q.rear = (LinkNode\*)malloc(sizeof(LinkNode));

return true;

}

// 入队

void EnQueue(LinkQueue& Q, ElemType x) {

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

if (!s)cout << "输入格式错误!" << endl; // 判空

else s->data = x;

Q.rear->next = s;

Q.rear = s;

}

// 出队

bool DeQueue(LinkQueue& Q, ElemType& x) {

if (Q.front == Q.rear) return false;

LinkNode\* p = Q.front->next;

x = p->data;

Q.front->next = p->next;

if (Q.rear == p)Q.rear = Q.front;

free(p);

return true;

}

int main() {

LinkQueue Q;

InitQueue(Q);

int n, x;

cout << "请输入一个整数n ，0<=n<=20" << endl;

cin >> n;

EnQueue(Q, 1);

for (int i = 0; i < n; i++) {

for (int j = 0; j < i; j++) {

int left = Q.front->next->data;

DeQueue(Q, x);

cout << x << " ";

int right = Q.front->next->data;

EnQueue(Q, left + right);

cout << " ";

}

cout << 1 << endl;

EnQueue(Q, 1);

}

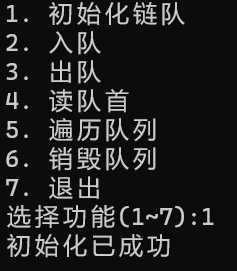
system("pause");

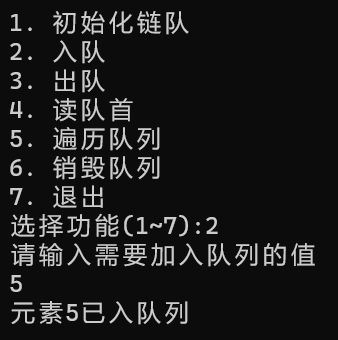
return 0;

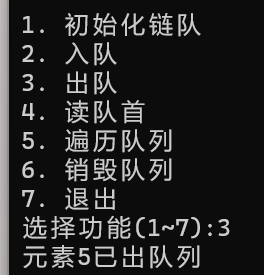
}

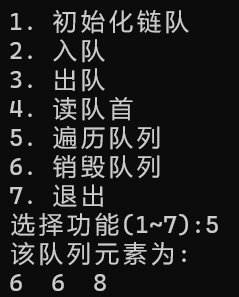
1. 结果截图

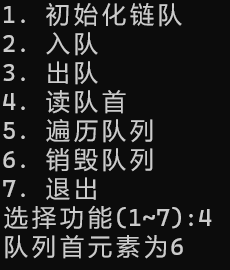
**链队列**

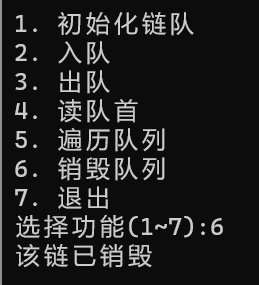




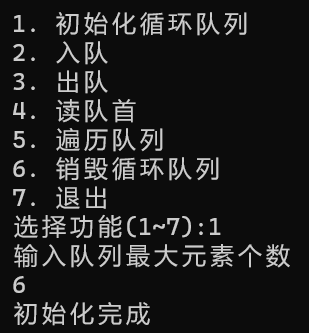


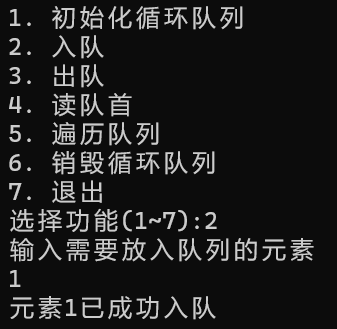


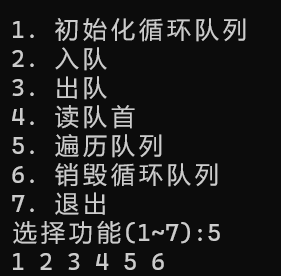


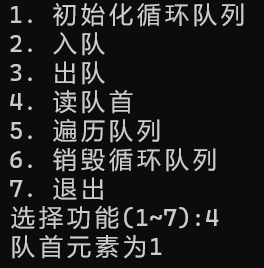


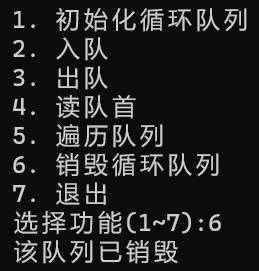
**循环队列**

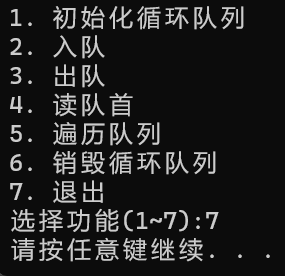




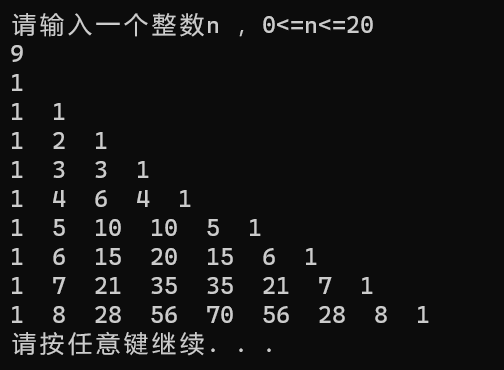








**杨辉三角**



【实验体会】

能够熟悉并掌握链队列的基本操作，实现队列的入队、出队、返回队首元素等基本运算。能够熟悉并掌握循环队列的基本操作，实现队列的入队、出队、返回队首元素等基本运算。