哈尔滨工业大学计算机科学与技术学院  
实验报告

课程名称：数据结构与算法  
课程类型：必修  
实验项目名称：实验一   
实验题目：食堂售饭问题

实验时间： 2021.10.11

班级： 计算机2003009班

姓名： 孙莹

1. **实验目的**

假设某食堂有四个窗口对外售饭，从上午11：00开始到下午13：00结束。由于某窗口在某个时间只能接待一个同学，因此在学生多的时候需要在窗口前排队，刚来的同学如果发现有空闲窗口即可上前买饭，反之若均有同学则要排到人数最少的窗口的后面。现要设计一个算法模拟这种业务并计算一天中午饭同学在食堂的平均时间。  
**二、实验要求及实验环境**

实现上述问题中的模拟算法。

使用语言为C++,编译环境为visual Stdio 2019。

1. 设计思想（用到的主要函数、数据类型的定义，主要功能的流程图（1-2个）及各程序模块之间的调用关系，自己扩展内容的等）

**主要函数：**

//事件相关的处理

void LunchSimulation(); //模拟食堂

void OpenLunch(); //食堂开门

void CloseLunch(); //食堂关门

int GetMin(LinkSueue q[]); //找到排队人最少的队列

void StudentArriving(); //预计同学到达

void StudentDeparture(); //同学离开

//链表处理

void InitList(LinkList\* pL); //初始化链表

int ListEmpty(LinkList L); //判断链表是否为空

void OrderInsert(LinkList\* pL, LElemType en); //插入事件结点

void DelFirst(LinkList\* pL, LElemType\* e); //删除事件结点

//队列处理

void InitSueue(LinkSueue\* Q); //初始化队列

int DelSueue(LinkSueue\* pQ, SElemType\* e); //删除同学结点

int EnSueue(LinkSueue\* pQ, SElemType e); //插入同学结点

int SueueLength(LinkSueue Q); //查看队列长度

**数据类型的定义：**

本程序主要涉及到的需要自定义的数据有：午饭事件（发生时间、事件类型），事件链表（每个结点都为一个午饭时间），同学（到达食堂时间，打饭事件，吃饭时间，同学编号），窗口队列（队头：下一个打饭的同学，队尾：下一个排队同学入队）

//事件(包括到达，排队，打饭，吃饭)

//事件列表：有头结点的单链表

typedef struct {

long occur\_time; //事件发生时间点

//为方便计算本程序中时间都为距食堂开门的时间。输出时另行处理为时刻。

int type; //0:预计下一名同学；1、2、3、4分别为在1、2、3、4窗口打饭；

}LElemType;

typedef LElemType Event; //定义事件

typedef struct LNode {

Event data;

struct LNode\* next;

}LNode, \* LinkList;

typedef LinkList EventList; //定义事件表：有序链表

//同学

typedef struct {

long arrival\_time; //到达食堂的时间

long get\_time; //打饭的时间

long eat\_time; //吃饭的时间

int id; //第几名同学

}SElemType;

typedef SElemType Student; //定义同学

//窗口队列

typedef struct SNode {

SElemType data;

struct SNode\* next;

}SNode, \* SueuePtr;

typedef struct {

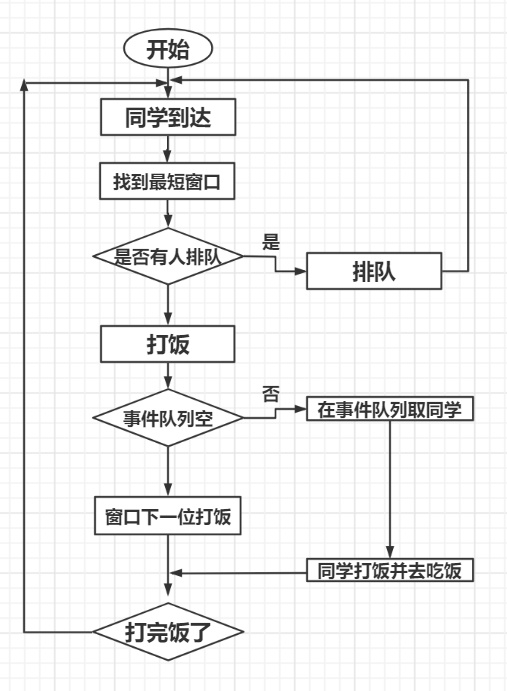
SueuePtr front; //排队的队头

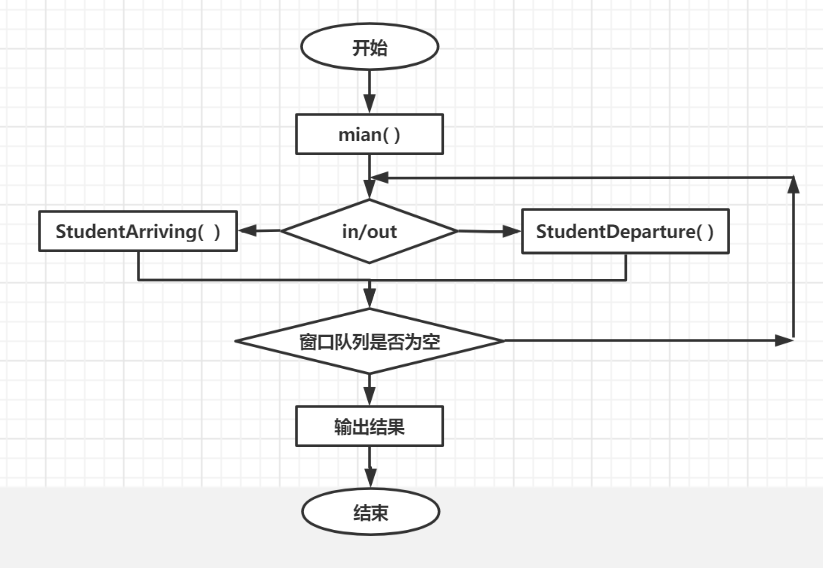
SueuePtr rear; //排队的队尾

}LinkSueue;

typedef LinkSueue StudentSueue;//定义排队队列

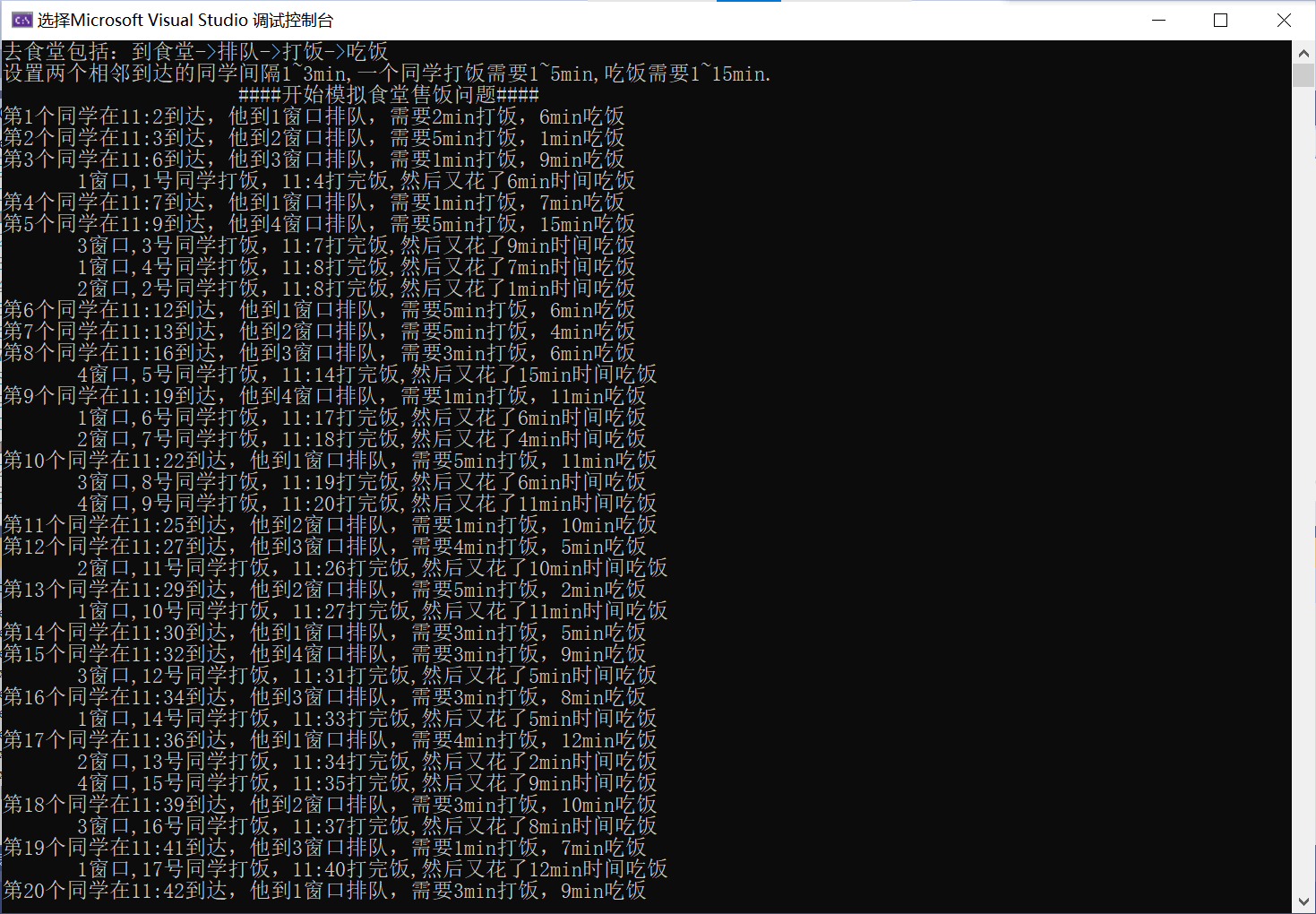
**主要功能流程图：**

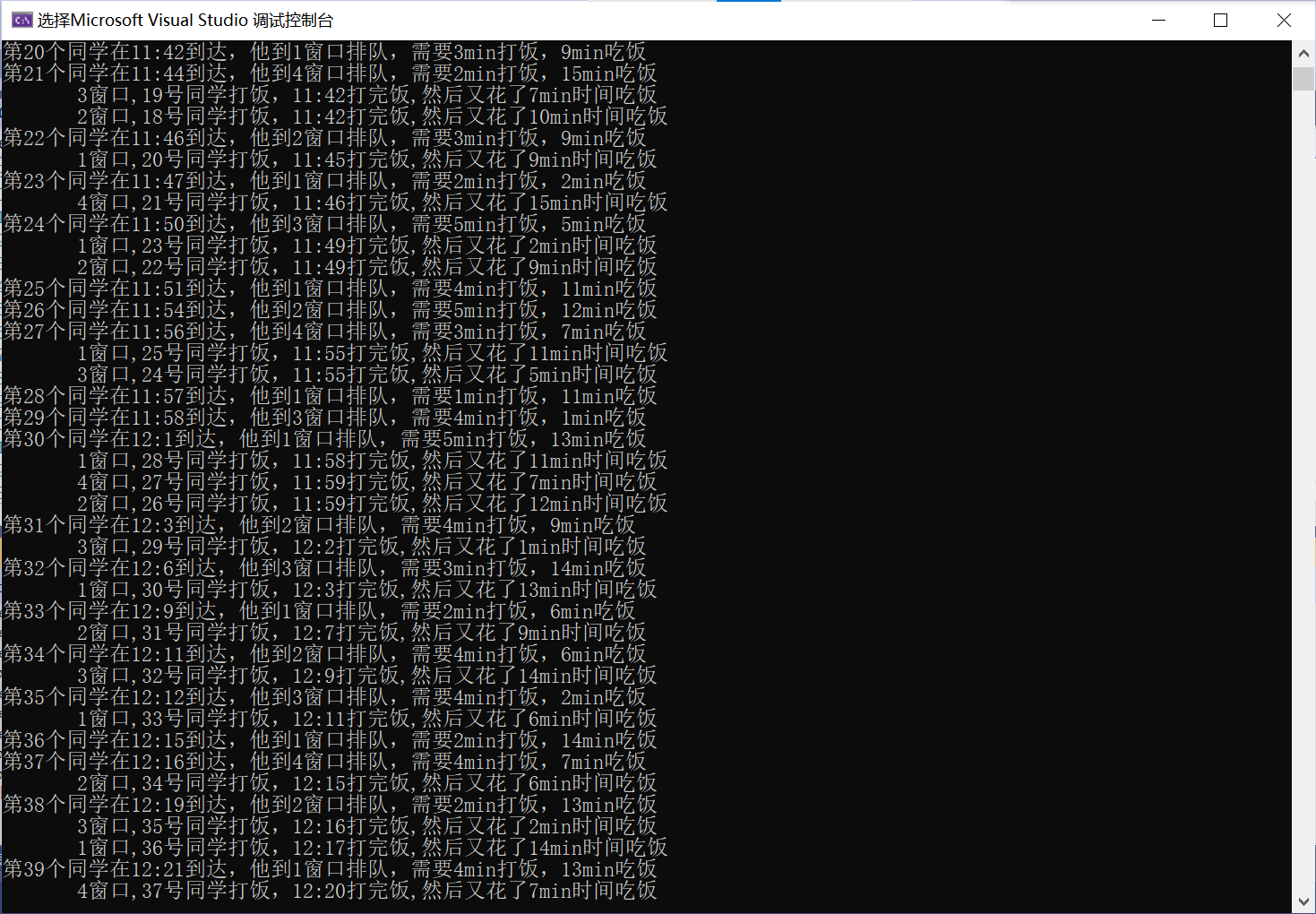


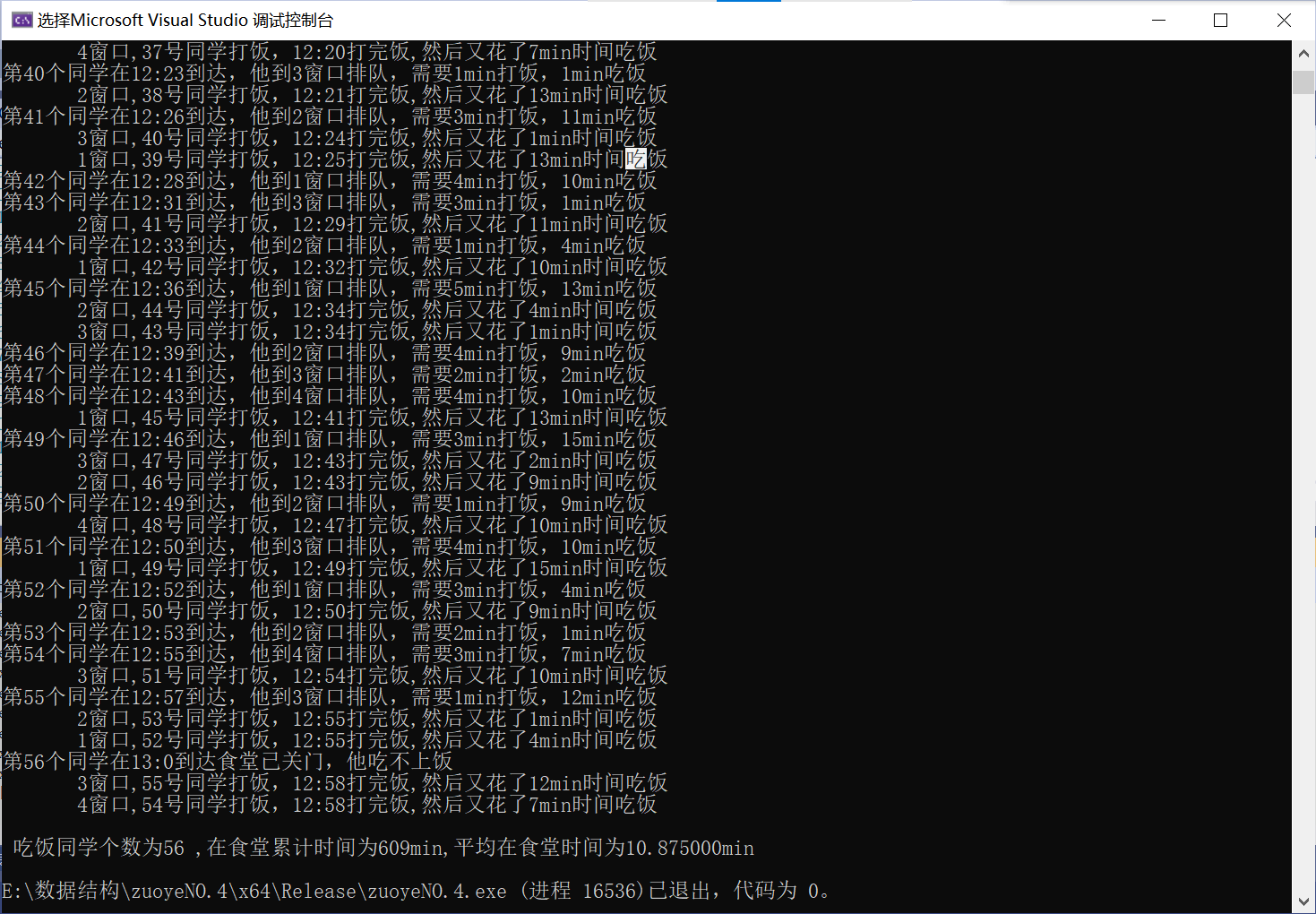


**四、测试结果**

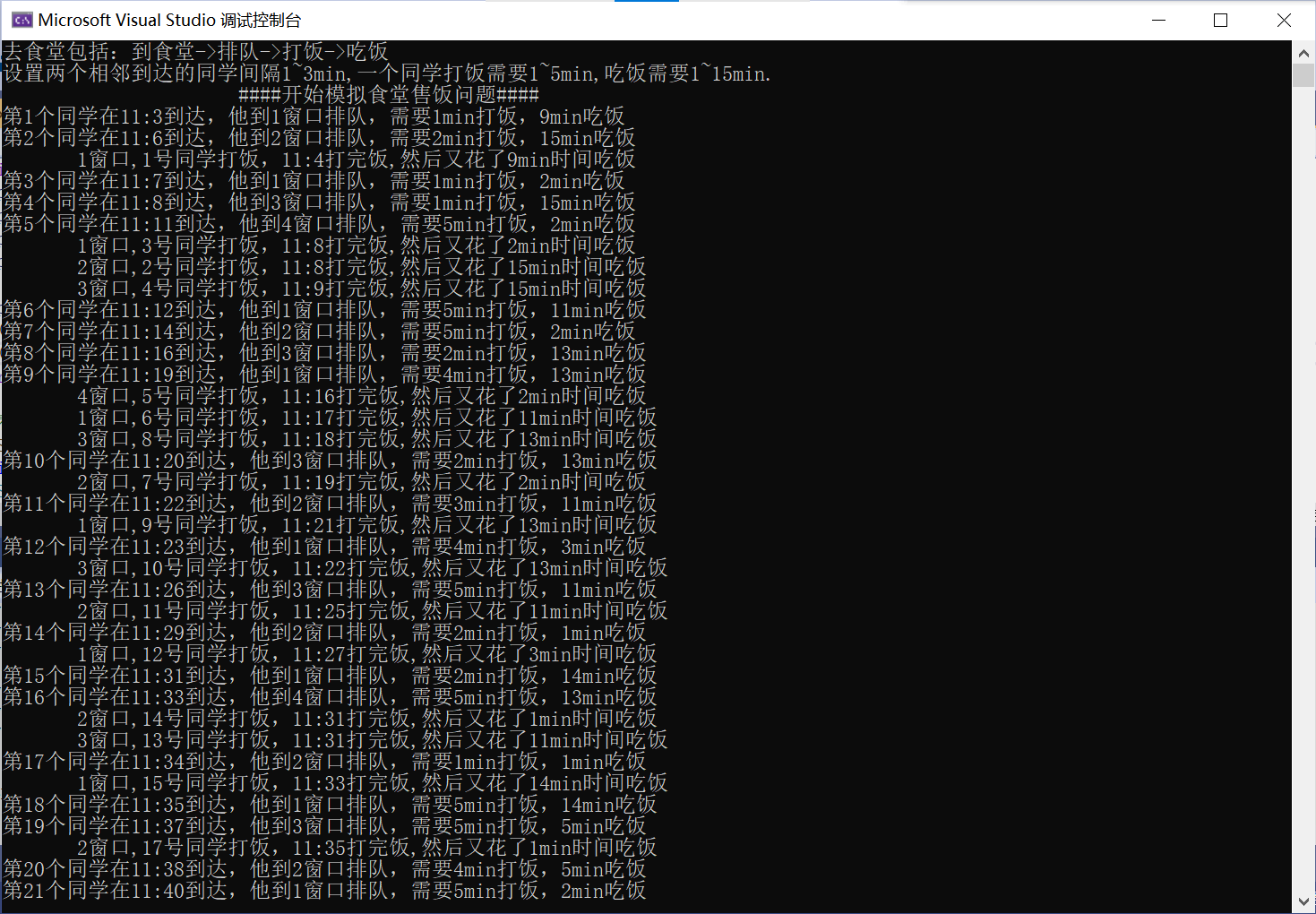
测试用例一：

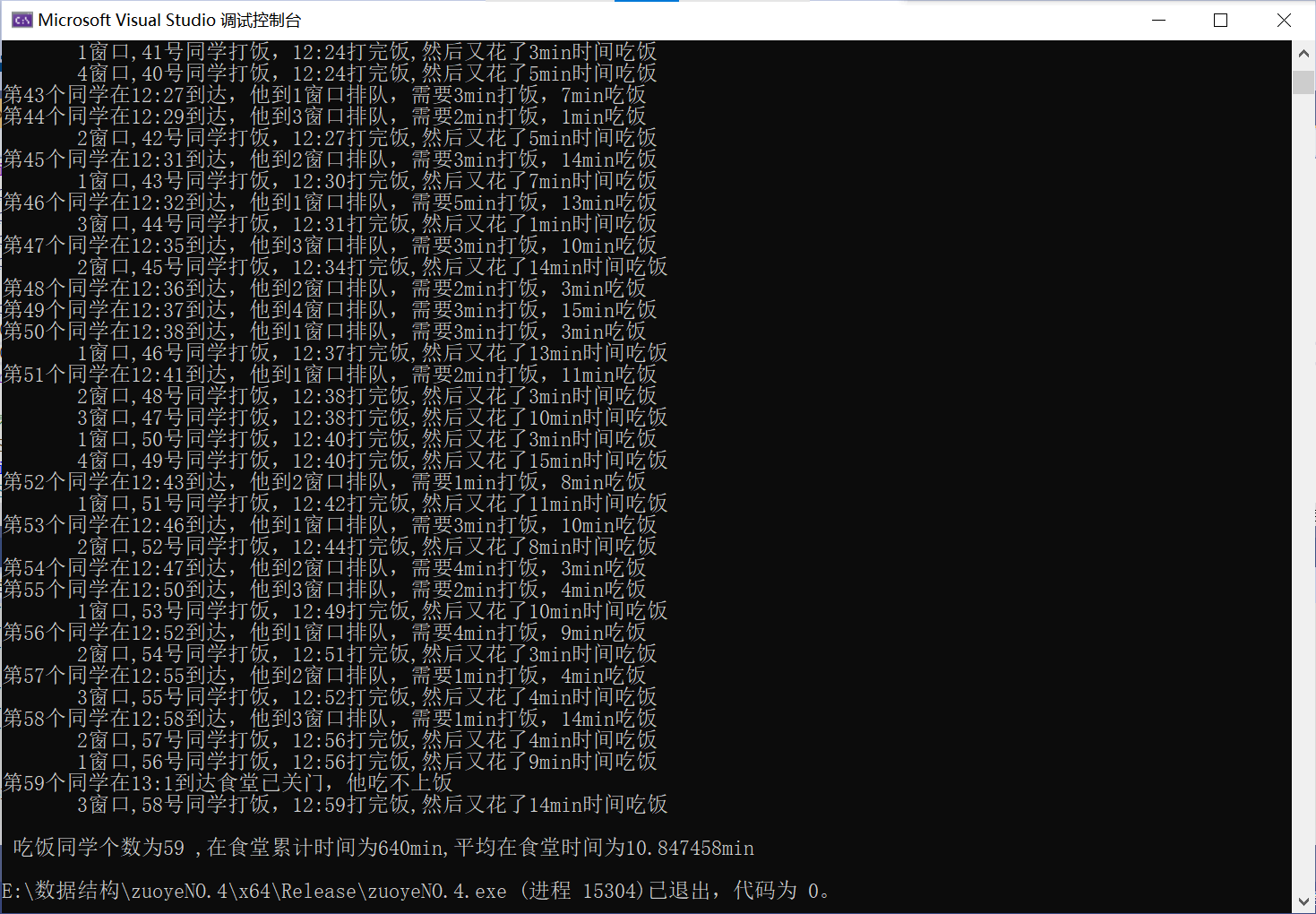
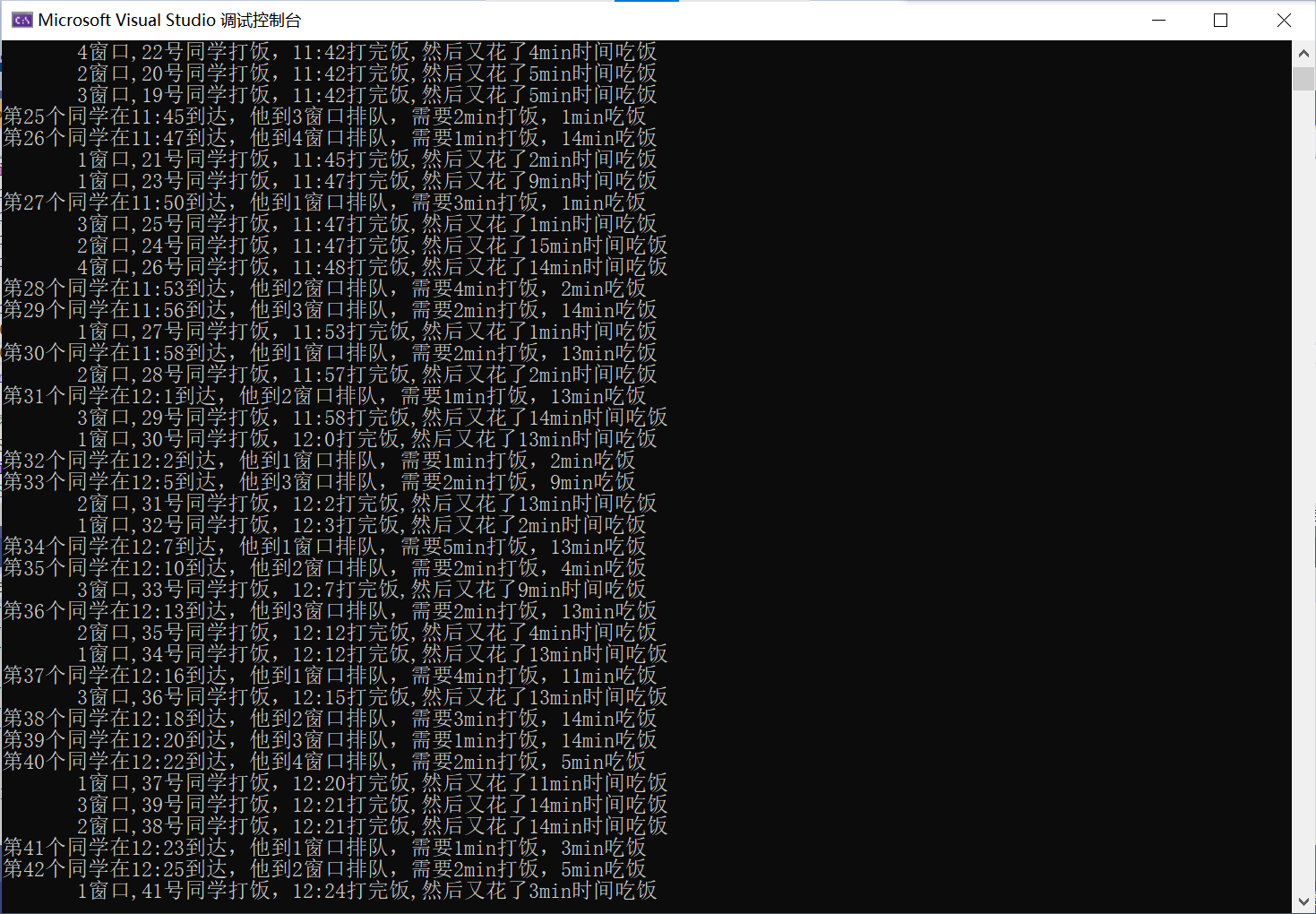






测试用例二：





**五、系统不足与经验体会**

1.系统不足：

输出界面不够简洁，本程序输出主语句为①“第X个同学在xx:xx到达，他到x窗口排队，需要x min打饭，x min吃饭”②“x窗口，x号同学打饭，xx:xx打完饭，然后又花了x min吃饭”。能够较详细地显示算法的模拟过程，但是显示界面有些繁复，数据显示不够简洁，或许可以有更形象、更直观的输出设计，比如按窗口输出。

1. 系统不足：

输入可以更灵活，本程序预先设定了同学到达食堂间隔、打饭时间区间、吃饭时间区间，还可以更灵活，比如每个窗口因为菜的样式不同，所以一个同学排在不同窗口打饭、吃饭的时间区间都不同。

1. 经验体会

借鉴了网上的类似示例后，我学习到食堂售饭本质是离散事件模拟。将食堂售饭问题转化为离散事件模拟，随机生成同学们到食堂等待、打饭、吃饭的事件列表，然后按时间逐个处理事件。

其中，我们根据相应的数据结构特点，让窗口排队抽象为“先进先出”的队列，事件表抽象为方便进行插入和删除操作的链表。

1. **带注释的源代码**

//设置两个相邻到达的同学间隔1~3min,一个同学打饭需要1~5min,吃饭需要1~15min.

#include<stdio.h>

#include<stdlib.h>

#include <iostream>

#include<time.h>

using namespace std;

#define MAX 10000

#define WINDOWS\_NUM 4 //食堂有4个窗口

//事件(包括到达，排队，打饭，吃饭)

//事件列表：有头结点的单链表

typedef struct {

long occur\_time; //事件发生时间点

//为方便计算本程序中时间都为距食堂开门的时间。输出时另行处理为时刻。

int type; //0:预计下一名同学；1、2、3、4分别为在1、2、3、4窗口打饭；

}LElemType;

typedef LElemType Event; //定义事件

typedef struct LNode {

Event data;

struct LNode\* next;

}LNode, \* LinkList;

typedef LinkList EventList; //定义事件表：有序链表

//同学

typedef struct {

long arrival\_time; //到达食堂的时间

long get\_time; //打饭的时间

long eat\_time; //吃饭的时间

int id; //第几名同学

}SElemType;

typedef SElemType Student; //定义同学

//窗口队列

typedef struct SNode {

SElemType data;

struct SNode\* next;

}SNode, \* SueuePtr;

typedef struct {

SueuePtr front; //排队的队头

SueuePtr rear; //排队的队尾

}LinkSueue;

typedef LinkSueue StudentSueue;//定义排队队列

//事件相关的处理

void LunchSimulation(); //模拟食堂

void OpenLunch(); //食堂开门

void CloseLunch(); //食堂关门

int GetMin(LinkSueue q[]); //找到排队人最少的队列

void StudentArriving(); //预计同学到达

void StudentDeparture(); //同学离开

//链表处理

void InitList(LinkList\* pL); //初始化链表

int ListEmpty(LinkList L); //判断链表是否为空

void OrderInsert(LinkList\* pL, LElemType en); //插入事件结点

void DelFirst(LinkList\* pL, LElemType\* e); //删除事件结点

//队列处理

void InitSueue(LinkSueue\* Q); //初始化队列

int DelSueue(LinkSueue\* pQ, SElemType\* e); //删除同学结点

int EnSueue(LinkSueue\* pQ, SElemType e); //插入同学结点

int SueueLength(LinkSueue Q); //查看队列长度

//全局变量

int total\_time = 0, student\_num = 0; //累计同学在食堂时间，同学数目

EventList eventList; //事件表

Event event; //事件

StudentSueue windows[WINDOWS\_NUM + 1]; //四个食堂窗口的队列，1、2、3、4

Student student; //学生

int close\_time = 120; //食堂开门两个小时，120分钟

//主函数

int main()

{

srand(time(NULL)); //随机种子，否则每次得到的随机数都相同

cout << "去食堂包括：到食堂->排队->打饭->吃饭\n设置两个相邻到达的同学间隔1~3min,一个同学打饭需要1~5min,吃饭需要1~15min.\n";

LunchSimulation(); //模拟食堂

return 0;

}

//模拟食堂

void LunchSimulation() {

OpenLunch(); //食堂开门

while (!ListEmpty(eventList)) { //当还有事件没有处理完

DelFirst(&eventList, &event); //因为链表是时间有序，删除第一个事件结点

if (event.type == 0)

{

StudentArriving(); //模拟下一个同学到达

}

else

{

StudentDeparture(); //有人打完饭了，离开窗口

}

}

CloseLunch(); //食堂关门

}

//食堂开门

void OpenLunch() {

int i = 0;

total\_time = 0; //开始计时

student\_num = 0; //开始计学生数

InitList(&eventList); //初始化事件链表

event.occur\_time = 0;

event.type = 0; //初始化事件信息

OrderInsert(&eventList, event); //插入事件信息，要按时间顺序

InitSueue(&windows[1]);

InitSueue(&windows[2]);

InitSueue(&windows[3]);

InitSueue(&windows[4]); //初始化食堂窗口

printf(" ####开始模拟食堂售饭问题####\n");

}

//食堂关门

void CloseLunch() {

float mean\_time = float(total\_time) / student\_num;

printf("\n 吃饭同学个数为%d ,在食堂累计时间为%dmin,平均在食堂时间为%1fmin\n", student\_num, total\_time, mean\_time);

}

//预计同学到达

void StudentArriving() {

long gettime, intertime, eattime;

int minWindow;

student\_num++; //学生数目加一

//开始创建到达的同学信息

intertime = rand() % 3 + 1; //同学到达间隔时间1~3min

gettime = rand() % 5 + 1; //打饭所需时间1~5min

eattime = rand() % 15 + 1; //吃饭所需时间1~15min

student.id = student\_num; //第几个到达

student.arrival\_time = event.occur\_time + intertime;

//到达时间点

student.get\_time = gettime; //打饭时间

student.eat\_time = eattime; //吃饭时间

if (student.arrival\_time >= close\_time) {

printf("第%d个同学在%d:%d到达食堂已关门，他吃不上饭\n", student.id, 11 + student.arrival\_time / 60, student.arrival\_time % 60);

//如果到的时候已关门，则吃不上饭，不用将其加入时间链表

}

else {

//如果到的时候没关门，可以开始等待，将其加入排队队列

minWindow = GetMin(windows); //找到此时最短的队

EnSueue(&windows[minWindow], student); //去最短队排队

//需要把该同学打完饭时间加入事件链表

event.occur\_time = student.arrival\_time + gettime;//打完饭的时间

event.type = minWindow;

if (SueueLength(windows[minWindow]) == 1)

OrderInsert(&eventList, event); //只有一个人排队，他可以离开了

printf("第%d个同学在%d:%d到达，他到%d窗口排队，需要%dmin打饭，%dmin吃饭 \n", student.id, 11 + student.arrival\_time / 60, student.arrival\_time % 60, minWindow, student.get\_time, student.eat\_time);

//继续预计下一个同学到达

event.occur\_time = student.arrival\_time;

event.type = 0;

if (event.occur\_time < close\_time) //排到他还没关门，插入

OrderInsert(&eventList, event);

}

}

//打完饭了，同学离开窗口

void StudentDeparture() {

int type = event.type; //第几个窗口

SNode\* p;

SElemType qe;

DelSueue(&windows[type], &student); //打完饭的同学

printf(" %d窗口,%d号同学打饭，", type, student.id);

if (event.occur\_time > close\_time) {

printf("但是排到他的时候食堂窗口已关闭了，他没吃上饭\n");

total\_time += close\_time - student.arrival\_time; //这个同学没打饭，但是他的等待时间也要加上

}

else {

printf("%d:%d打完饭,然后又花了%dmin时间吃饭\n", 11 + event.occur\_time / 60, event.occur\_time % 60, student.eat\_time);

total\_time += event.occur\_time - student.arrival\_time + student.eat\_time; //把时间累加上

if (SueueLength(windows[type])) { //队列还有人，继续打饭

p = windows[type].front->next;

qe = p->data;

student.arrival\_time = qe.arrival\_time;

student.get\_time = qe.get\_time;

student.eat\_time = qe.eat\_time;

event.occur\_time += student.get\_time;

event.type = type;

OrderInsert(&eventList, event);

}

}

}

//找排队最短的窗口

int GetMin(LinkSueue q[]) {

int k, min;

int cnt;

SNode\* p;

min = MAX;

for (int i = 1; i <= WINDOWS\_NUM; i++) {

if (q[i].front == q[i].rear) {

cnt = 0; //该窗口没有人

}

else {

for (cnt = 1, p = q[i].front->next; p != q[i].rear; p = p->next)

{

cnt++;

}

}

if (min > cnt) {

min = cnt;

k = i;

}

}

return k;

}

//链表处理

//链表初始化

void InitList(LinkList\* pL) {

\*pL = (LNode\*)malloc(sizeof(LNode));

if (!\*pL) exit(0);

(\*pL)->next = NULL;

}

//判断链表为空

int ListEmpty(LinkList L) {

if (L->next == NULL) return 1;

else return 0;

}

//插入结点 ，需要按事件时间(occur\_time)排序

void OrderInsert(LinkList\* pL, LElemType en) {

LNode\* p, \* q, \* s;

for (p = \*pL, q = p->next; q && q->data.occur\_time < en.occur\_time; p = q, q = p->next);

//按时间找到插入位置

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

if (!s) exit(0);

s->data.type = en.type;

s->data.occur\_time = en.occur\_time;

p->next = s;

s->next = q;

}

//删除事件结点

void DelFirst(LinkList\* pL, LElemType\* e) {

LNode\* p;

p = (\*pL)->next;

(\*pL)->next = p->next;

e->occur\_time = p->data.occur\_time;

e->type = p->data.type;

free(p);

}

//队列处理

//队列初始化

void InitSueue(LinkSueue\* Q) {

SNode\* p;

p = (SNode\*)malloc(sizeof(SNode));

Q->front = p;

Q->rear = p;

Q->front->next = nullptr;

}

//出队操作，先进先出，从队首开始

int DelSueue(LinkSueue\* pQ, SElemType\* e) {

SNode\* p;

if ((\*pQ).front == (\*pQ).rear) return 0;

p = (\*pQ).front->next;

e->arrival\_time = p->data.arrival\_time;

e->get\_time = p->data.get\_time;

e->eat\_time = p->data.eat\_time;

e->id = p->data.id;

(\*pQ).front->next = p->next;

if ((\*pQ).rear == p) //如果删除后队列为空，需要处理一下，以便操作时检查

(\*pQ).rear = (\*pQ).front;

free(p);

return 1; //删除成功

}

//入队操作，在队尾入

int EnSueue(LinkSueue\* pQ, SElemType e) {

SNode\* p;

p = (SNode\*)malloc(sizeof(SNode));

if (!p) exit(0);

p->data.arrival\_time = e.arrival\_time;

p->data.get\_time = e.get\_time;

p->data.eat\_time = e.eat\_time;

p->data.id = e.id;

p->next = NULL;

(\*pQ).rear->next = p;

(\*pQ).rear = p;

return 1;

}

//查看队列长度

int SueueLength(LinkSueue Q) {

int count = 0;

SNode\* p;

if (Q.front == Q.rear) count = 0;

else {

p = Q.front;

while (p != Q.rear) {

count++;

p = p->next;

}

}

return count;

}