**程序设计与算法综合训练**

**实验报告**

|  |  |
| --- | --- |
| **实验名称**： | 银行业务模拟系统的设计与实现 |
| **专业**： | 机器人工程 |
| **班级**： | 机器人工程三班 |
| **学号**： | WA2224013 |
| **姓名**： | 郭义月 |

**目录**

[一、 实验内容及要求 4](#_Toc182396519)

[1.1 实验目的 4](#_Toc182396520)

[1.2 实验内容 4](#_Toc182396521)

[1.3 实验要求 4](#_Toc182396522)

[1.4 实验任务 5](#_Toc182396523)

[二、 任务1：请用自己的语言描述解决该问题的主要思路 5](#_Toc182396524)

[2.1 问题描述： 6](#_Toc182396525)

[2.2 基础内容 7](#_Toc182396526)

[2.2.1 队列 7](#_Toc182396527)

[2.2.2 随机数 9](#_Toc182396528)

[三、 任务2：.完成示例代码并解决其中的bug 10](#_Toc182396529)

[3.1 定义相关结构体 10](#_Toc182396530)

[3.2 重要代码展示与分析 11](#_Toc182396531)

[3.3 结果展示 12](#_Toc182396532)

[四、 任务3：输入和输出时间使用绝对时间 12](#_Toc182396533)

[4.1 定义times结构体 12](#_Toc182396534)

[4.2 结果截图展示 13](#_Toc182396535)

[五、 任务4计算某银行一周中的平均客户逗留时间 13](#_Toc182396536)

[5.1 一天内上午8:30上班，下午5:00下班，中午不休息 13](#_Toc182396537)

[5.1.1 问题分析 13](#_Toc182396538)

[5.1.2 结果展示 14](#_Toc182396539)

[5.2 一周内上午8:30上班，下午5:00下班，周日不上班，中午不休息 15](#_Toc182396540)

[5.2.1 问题分析 15](#_Toc182396541)

[5.2.2 结果展示 16](#_Toc182396542)

[5.3 中午休息（12:00至14:00）无值班窗口 16](#_Toc182396543)

[5.3.1 问题分析 16](#_Toc182396544)

[5.3.2 结果展示 18](#_Toc182396545)

[5.4 中午休息（12:00至14:00）有一个值班窗口 19](#_Toc182396546)

[5.4.1 问题分析 19](#_Toc182396547)

[5.4.2 结果展示 19](#_Toc182396548)

[六、 实验总结与常见问题汇总 20](#_Toc182396549)

[6.1 实验总结 20](#_Toc182396550)

[6.2 常见问题汇总 21](#_Toc182396551)

# 实验内容及要求

## 实验目的

加深对队列数据结构的理解，掌握离散事件问题的模拟方法，强化学生的逻辑思维能力和动手能力，巩固良好的编程习惯，掌握工程软件设计的基本方法，为后续课程的学习打下坚实基础。

## 实验内容

问题描述：

假设某银行有四个窗口对外接待客户，从早晨银行开门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在每个窗口前顺次排队，对于刚进入银行的客户，如果某个窗口的业务员正空闲，则可上前办理业务，反之，若四个窗口均有客户所占，他便会排在人数最少的队伍后面。现在需要编制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。

## 实验要求

基本要求：

（1）初始化（OpenForDay），模拟银行开门时各数据结构—结构体的状态。

（2）事件驱动（EventDrived）, 对客户到达和离开事件做相应处理，即到达、排队、业务办理等。

（3）下班处理（CloseForDay），模拟银行关门时的动作，统计客户平均逗留时间。

问题：

1）编程模拟银行的业务活动；

2）一天结束，计算某银行一天中的平均客户逗留时间；

核心：计算某银行一天服务了多少个客户，以及每个客户逗留时间（排队时长+办理业务时长（随机数））

## 实验任务

1. 请用自己的语言描述解决该问题的主要思路

2. 完成示例代码并解决其中的bug

3. 输入和输出时间使用绝对时间（即日常生活使用的时间，比如对于8:30，输入可以分开处理“时”和“分”，比如8 30，也可以输入字符串”8:30”来处理，输出使用类似“8:30”的格式），时间流逝单位为“分”

4. 计算某银行一周中的平均客户逗留时间

1) 上午8:30上班，下午5:00下班

2) 周日不上班

3) 中午不休息

4) 中午休息（12:00至14:00）

a. 一个值班窗口

b. 无值班窗口

# 任务1：请用自己的语言描述解决该问题的主要思路

为解决这个银行窗口排队模拟问题，我们可以按以下思路设计三个函数：

1.初始化（OpenForDay）

1. 定义四个队列，每个队列表示一个窗口的排队队列，队列中存储客户结构体（包括到达时间、开始服务时间、服务时长等）。
2. 初始化变量记录客户总数和逗留时间，用于后续计算平均逗留时间。
3. 设置银行开门时间，准备接收客户。

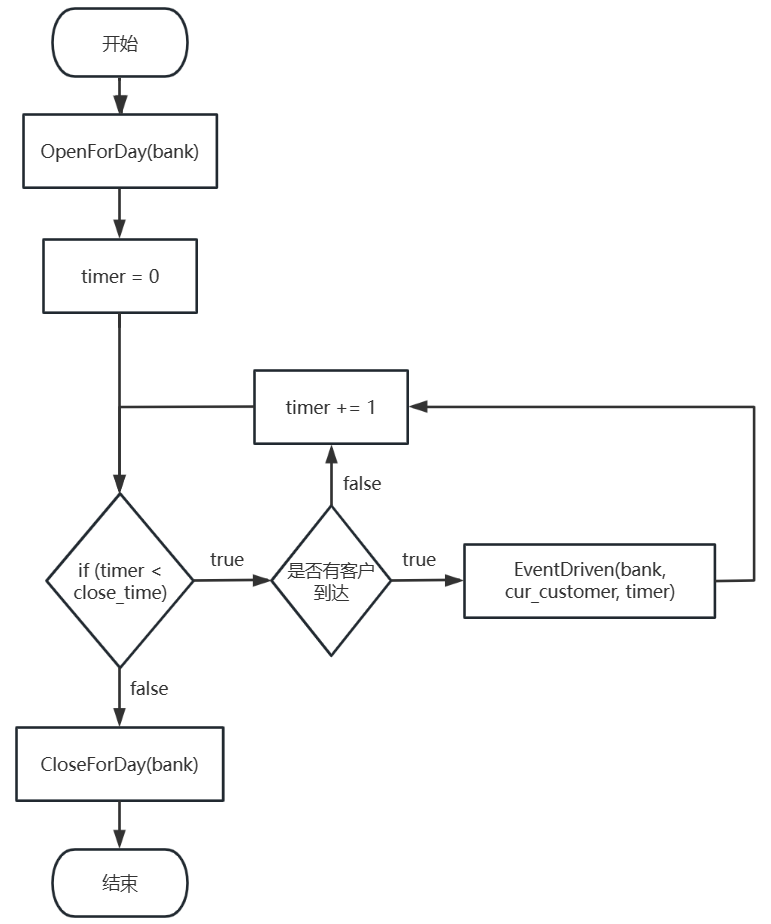
2.事件驱动（EventDriven）

1. 模拟客户的到达和离开事件：
2. 到达事件：每当有客户到达时，检查窗口情况。如果某个窗口空闲，则客户直接办理业务，记录客户的开始服务时间；否则，客户选择队伍最短的窗口排队。
3. 离开事件：对每个窗口检查当前客户的服务是否完成。如果完成，将客户出队并处理下一个客户，更新服务开始时间。
4. 更新时间，每次事件触发后重新检查客户排队和服务状态。

3.下班处理（CloseForDay）

1. 结束当天的接待，确保所有窗口的客户服务完成。
2. 计算客户的总逗留时间和总人数，计算并输出平均逗留时间

## 问题描述：



目标：模拟银行的业务活动，记录和计算一天中所有客户在银行的逗留时间，以求得平均的等待时间和办理业务的时间。

首先，银行开始一天的运营，执行 OpenForDay 函数，初始化服务环境，为客户提供接待服务的准备工作完成。

然后，初始化计时器，将 timer 设置为0，标记一天开始的时间，并且用于追踪当天的时间流逝。

接着，进入一个循环，每次循环代表一单位的时间（可能是一分钟）。在循环中，首先检查当前时间（timer）是否小于设定的关门时间（close\_time）。如果当前时间已经超过或等于关门时间，则流程进入 CloseForDay 函数，关闭银行，结束服务。

然后，在仍然营业的情况下，系统检查是否有新客户进入银行。如果没有客户进入，则直接进入 timer += 1 的步骤，计时器加一，然后回到循环的起始点，继续下一个时间单位的检查。

如果有客户进入，则执行 EventDriven 函数，处理该客户的服务流程。在这个过程中，系统检查四个服务窗口的状态，如果有空闲的窗口，客户可以直接上前办理业务。如果四个窗口都在服务中，客户则会选择排在人数最少的队伍后面等待。

之后，计时器继续加一，并返回循环开始，重复检查当前时间和客户情况，直至达到关门时间。

最后，当时间达到或超过 close\_time 时，银行执行 CloseForDay 函数，结束一天的服务流程，标志一天营业结束。

## 基础内容

### 队列

队列是一种线性数据结构，它具有先进先出（FIFO）的特性。

队列通常有两个基本操作：入队（enqueue）和出队（dequeue）。

入队操作将元素添加到队列的末尾，出队操作则从队列的前端移除元素。

队列可以用数组（顺序队列）或链表（链式队列）来实现。

链式队列代码示例：

1. //队列结点
2. **typedef** **struct** Node
3. {
4. **int** data;
5. **struct** Node\* next;
6. } Node;

顺序队列代码示例：

1. //队列结构
2. **typedef** **struct** {
3. **int** data[MAX\_SIZE];
4. **int** front, back;
5. } Queue;
6. //入队
7. **void** enqueue(Queue\* q, **int** x) {
8. **if** (isFull(q)) {
9. printf("Error: Queue is full.\n");
10. **return**;
11. }
12. q->data[q-> back ++] = x;
13. }
14. //出队
15. **int** dequeue(Queue\* q) {
16. **if** (isEmpty(q)) {
17. printf("Error: Queue is empty.\n");
18. **return** -1;
19. }
20. **return** q->data[q->front++];
21. }
22. //队首队尾指针
23. **typedef** **struct** Queue
24. {
25. Node\* head; //队头
26. Node\* tail; //队尾
27. } Queue;

链式队列代码示例：

1. //队列结点
2. **void** enqueue(Queue\* q, **int** data)
3. {
4. Node\* new\_node = **new** Node;
5. new\_node->data = data;
6. new\_node->next = NULL;
7. **if** (q->head == NULL) //队列为空
8. q->head = q->tail = new\_node;
9. **else** //队列不为空
10. {
11. q->tail->next = new\_node;
12. q->tail = new\_node;
13. }
14. }
15. //队首队尾指针
16. **int** dequeue(Queue\* q)
17. {
18. **if** (q->head == NULL) //队列为空
19. {
20. cout << "队列为空" << endl;
21. **return** -1;
22. }
23. **int** data = q->head->data;
24. Node\* temp = q->head;
25. q->head = q->head->next;
26. **if** (q->head == NULL) //队列已经为空
27. q->tail = NULL;
28. **delete** temp;
29. **return** data;
30. }

### 随机数

随机数

模拟客户到达时间和业务办理时长，需要用到随机数

1. 生成一定范围的随机数，例如客户业务办理时长（至少3分钟-20分钟）

问题：已有库函数rand()是取（0 - RAND\_MAX）之间的任意随机整数，不能指定随机数生成范围。

解决：可采用三种

1) 取得 [a,b) 的随机整数，使用 (rand() % (b-a))+ a;

2) 取得 [a,b] 的随机整数，使用 (rand() % (b-a+1))+ a;

3) 取得 (a,b] 的随机整数，使用 (rand() % (b-a))+ a + 1;

随机种子

伪随机问题

C语言库函数rand()产生的是伪随机数字：在随机种子相同的情况下，同一个程序在多次执行时得到的随机数是相同的。

函数 srand()用于设置 rand()的随机种子。通常使用时间作为srand()函数的输入使随机种子随时间变动，达到随机效果。

为了便于调试可以利用随机种子的原理设置固定的随机种子。

随机种子代码实例：

1. #include <cstdlib>
2. #include <ctime>
3. #include <iostream>
4. **using** **namespace** std;
6. **int** genRand(**int** min, **int** max) { // 生成[min, max]的随机整数
7. **return** (rand() % (max - min + 1)) + min;
8. }
10. **int** main() {
11. srand((unsigned)time(NULL)); // 使用当前时间作为种子
12. cout << genRand(0, 60) << endl;
13. **return** 0;
14. }

# 任务2：.完成示例代码并解决其中的bug

## 定义相关结构体

顾客结构体

typedef struct Customer {

int id; // 顾客序号

int arrival\_time; // 到达时刻

int service\_time; // 需要服务的时长

} Customer, ElemType;

服务窗口结构体

typedef struct Window {

int cur\_remaining\_service\_time; // 当前正在办理业务的客户的剩余服务时长

int remaining\_service\_time; // 总剩余服务时长?

int total\_wait\_time; // 统计该服务窗口所有顾客的等待时长

Queue queue; // 服务窗口队列

} Window;

银行结构体

typedef struct Bank {

    int open\_time, close\_time; // 银行上下班时长?

    Window windows[NUM\_WINDOWS]; // 几个服务窗口

    Queue customer\_queue; // 已生成的顾客队列

    int num\_to\_service; // 需要服务的总人数?

} Bank;

## 重要代码展示与分析

生成顾客：随机数生成顾客到达时间时，应注意：第二个顾客的到达时间应大于第一个顾客的到达时间，之后顾客的到达时间均应大于之前顾客。

定义函数genCustomers(Bank\* bank)生成顾客，以下程序的思想在于，定义变量precus\_sum记录上一个顾客到达的时间，之后生成随机数时，就从precus\_sum与MAX\_ARRIVAL\_INTERVAL之间生成。

void genCustomers(Bank\* bank)

{

    int service\_duration = bank->close\_time - bank->open\_time; // 银行开门总时长?

    int precus\_sum = 0;

    for(int i = 0; i<=10 && precus\_sum < MAX\_ARRIVAL\_INTERVAL; i++){

        Customer customer = {

            i,

            genRand(precus\_sum, MAX\_ARRIVAL\_INTERVAL), // 到达时刻

            genRand(MIN\_SERVICE\_TIME, MAX\_SERVICE\_TIME) // 服务时间

        }; // 生成新顾客?

        enqueue(&bank->customer\_queue, customer); // 将生成的顾客入队

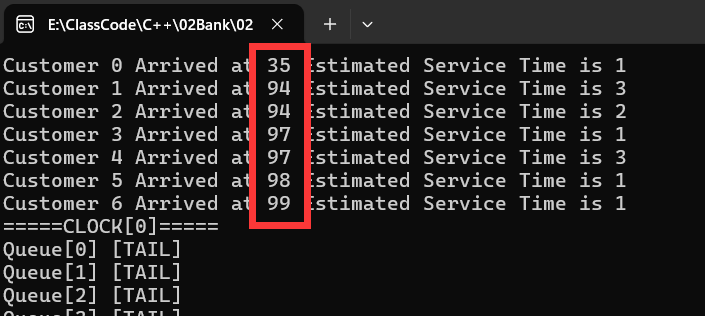
        cout << "Customer " << customer.id << " Arrived at " << customer.arrival\_time << " Estimated Service Time is " << customer.service\_time << endl;

        precus\_sum = customer.arrival\_time;

    }

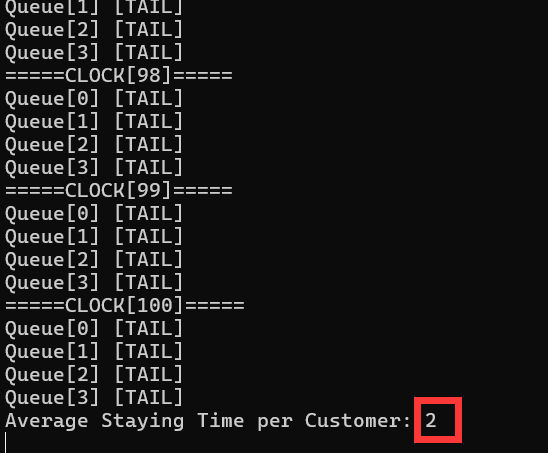
}

这种方法虽然可以生成顺序到达的顾客，但容易出现如下图所示后期生成的顾客到达时间非常集中的情况，这个问题在之后的任务中通过改进程序可以得到解决。



## 结果展示

从操作截图可以看出，可以实现计算客户平均逗留时间的目的



# 任务3：输入和输出时间使用绝对时间

## 定义times结构体

任务二是通过now来模拟时间，想生成以分为流逝单位的绝对时间，只需要通过除以60向下取整加8得到小时，对60取余得到分钟，结构体如下：

times calculate\_time(int now){

    times arr\_time;

    arr\_time.hour = 8 + now/60;

    arr\_time.min = now % 60;

    return arr\_time;

}

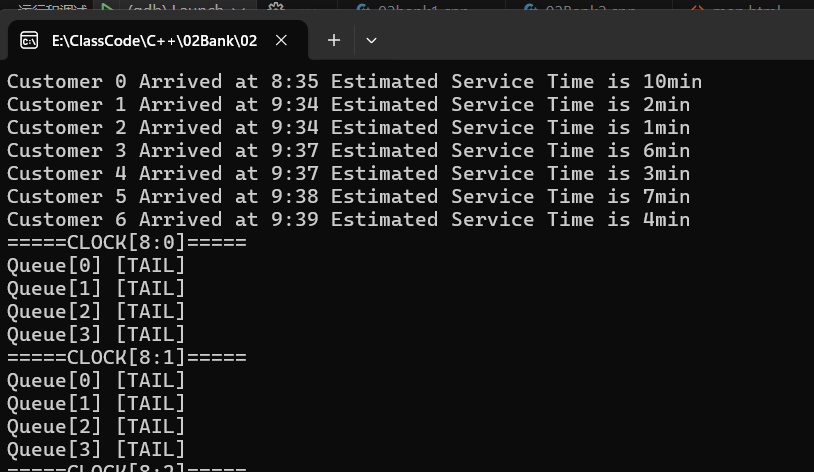
在生成顾客的到达时间是，只需要修改以下部分，即可用绝对时间表示

times time\_arr = calculate\_time(customer.arrival\_time);

cout << "Customer " << customer.id << " Arrived at " << time\_arr.hour << ":" << time\_arr.min << " Estimated Service Time is " << customer.service\_time << "min" <<endl;

## 结果截图展示

结果如下图，可以看到，已经实现了用绝对时间显示，流逝时间为分



# 任务4计算某银行一周中的平均客户逗留时间

## 一天内上午8:30上班，下午5:00下班，中午不休息

### 问题分析

想要实现此功能，只需要设置now的取值范围即可，由于流逝单位为分，可以计算从8:30到17:00的分钟为510，修改总服务时长即可，代码如下：

#define SIMULATION\_DURATION 510

除此之外，由于是8:30开始上班，所以绝对时间的表示的逻辑也要更改，最简单的想法是，当now小于30时，hour=8，min=30+now,当now大于30时，hour从9点开始，定义calculate\_time(int now)函数，返回times类型的结构体。

times calculate\_time(int now){

    times arr\_time;

    if(now < 30){

        arr\_time.hour = 8;

        arr\_time.min = 30 + now;

    }

    else{

        arr\_time.hour = 9 + (now - 30)/60;

        arr\_time.min = (now - 30) % 60;

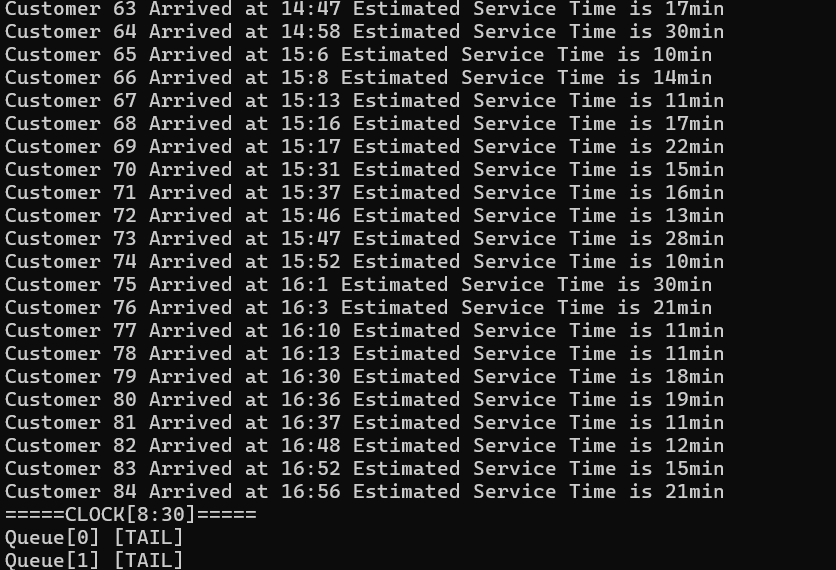
    }

    return arr\_time;

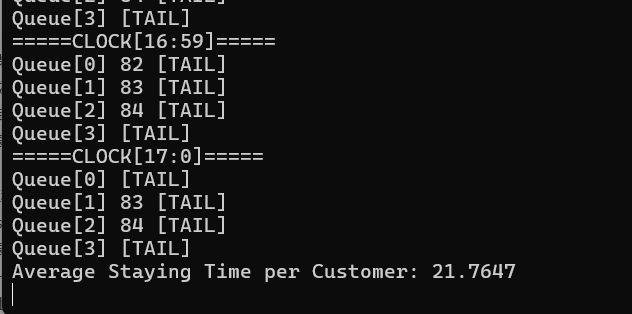
}

### 结果展示

共有85个顾客，生成的每位顾客都是8:30-17:00到达。



可以计算顾客的平均逗留时间



## 一周内上午8:30上班，下午5:00下班，周日不上班，中午不休息

### 问题分析

一周内周日不上班，只需要把主函数循环六次即可。计算平均逗留时间时，最简单的方法就是定义一个数组，分别存储每天的顾客人数，每天顾客的平均逗留时长，每天顾客所有等待的总时长。

int number\_customers\_perday[8];// 每天的顾客人数

int Aver\_Staytime\_perday[8]; //每天顾客的平均逗留时长

int Sum\_time\_perday[8]; //每天所有顾客等待的总时长

由于之前的程序已经可以算出一天的顾客平均逗留时长，所以可以通过平均逗留时长\*每天顾客的人数来计算所有顾客总的逗留时长，结果存储在Sum\_time\_perday数组中，代码如下：

for(int week = 1;week < 7; week++){

    OpenForDay(&bank); // 开门?

    number\_customers\_perday[week] = bank.customer\_queue.size;

    Clock(&bank); // 时间流逝-> 事件驱动

    Aver\_Staytime\_perday[week] = CloseForDay(&bank);

    Sum\_time\_perday[week] = number\_customers\_perday[week]\*Aver\_Staytime\_perday[week];

    }

For循环6次之后，把总时长相加除以每天顾客人数相加即可。

int WeekTime = 0;

int WeekPer = 0;

for (int i = 1;i < 7;i++){

WeekTime += Sum\_time\_perday[i];

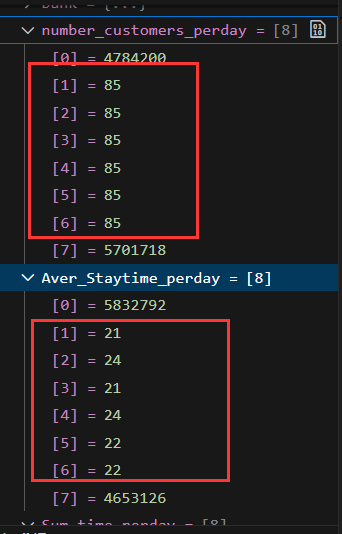
WeekPer += number\_customers\_perday[i];

}

cout << "Average Staying Time per Customer: " << WeekTime/WeekPer<< endl; // 关门

### 结果展示

程序顺利循环六次，由于随机种子的设定，每天的顾客人数保持不变，但是平均逗留时间可以变化，各自的值如图所示



一周内的平均逗留时间可通过WeekTime/WeekPer求得，结果如图：

|  |  |
| --- | --- |
|  |  |

## 中午休息（12:00至14:00）无值班窗口

### 问题分析

想实现此功能，只需要在生成顾客的时候更改就可以，具体逻辑为：如果生曾的顾客到达时间是2:00至14:00（8:30上班，即now大于210且小于330时间段）之间，就不让顾客进行入队操作，即在此时间段虽然正常生成顾客，但是不进行任何操作。

但是要注意，之前的操作都是输出customer.id，但是在这个问题中，由于有部分顾客没有入队，所以不能用customer.id来代替第几位顾客，可以改成bank->customer\_queue.size，即用顾客实际入队的队列长度来表示第几位顾客。

if(!(customer.arrival\_time>210 && customer.arrival\_time< 330)) {

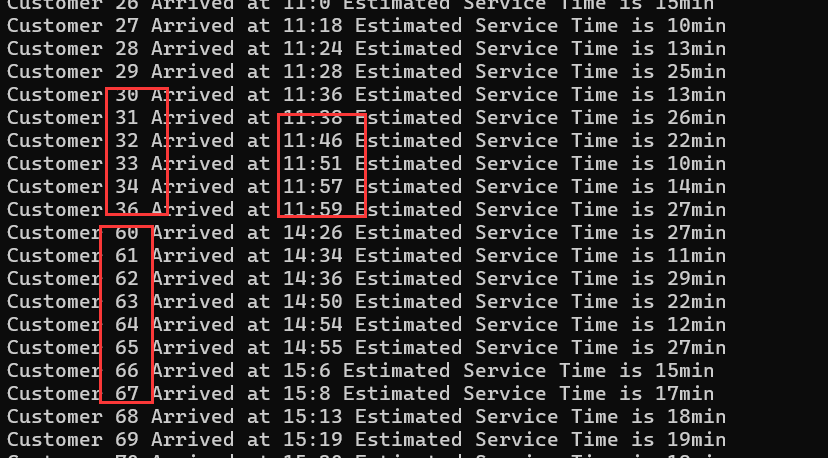
enqueue(&bank->customer\_queue, customer);// 将生成的顾客入队

times time\_arr = calculate\_time(customer.arrival\_time);

cout << "Customer " << bank->customer\_queue.size << " Arrived at " << time\_arr.hour << ":" << time\_arr.min << " Estimated Service Time is " << customer.service\_time << "min" <<endl;

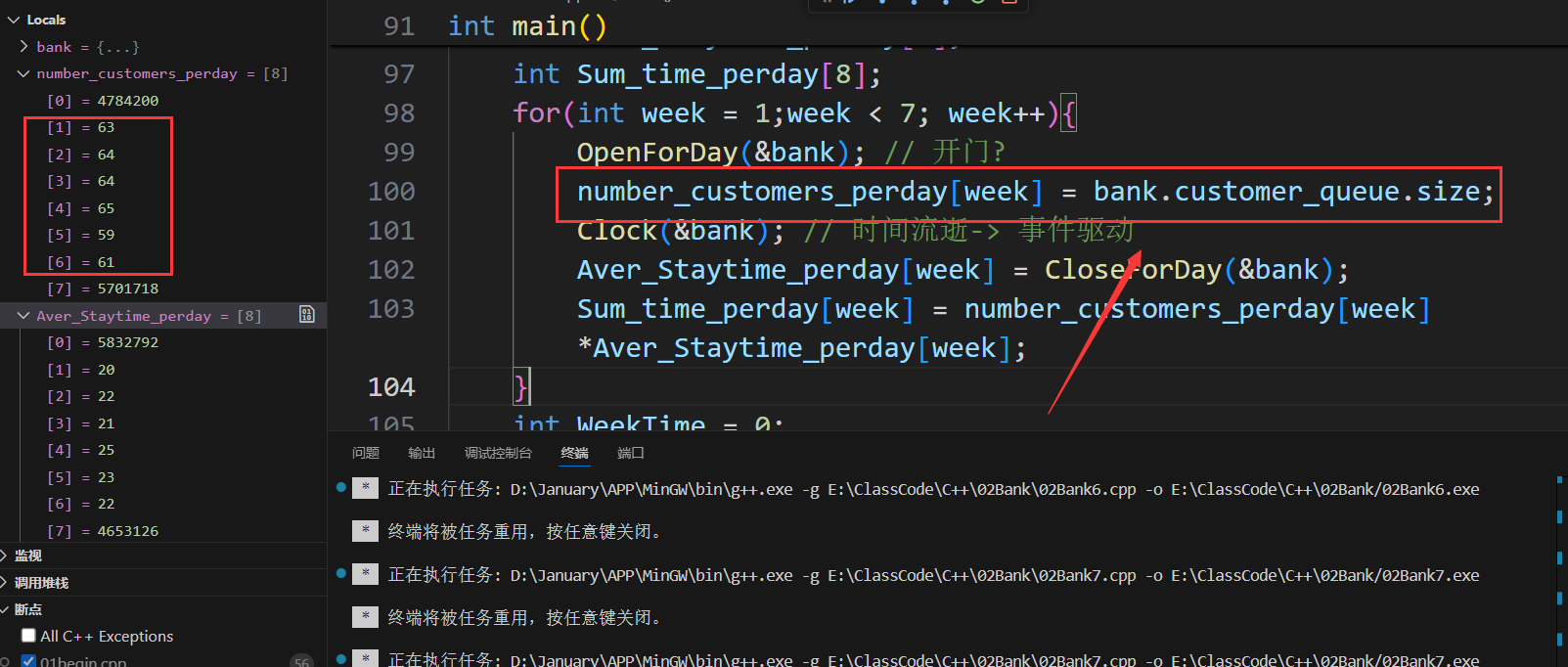
}

如果输出customer.id来表示顾客序号，虽然能实现中午休息的操作，但是customer 的id会发生跳变，这是因为在休息时间段虽然正常生成顾客，但是没有让其入队，customer 的id依然在增长，不符合实际情况，用bank->customer\_queue.size来表示第几位顾客是符合实际的



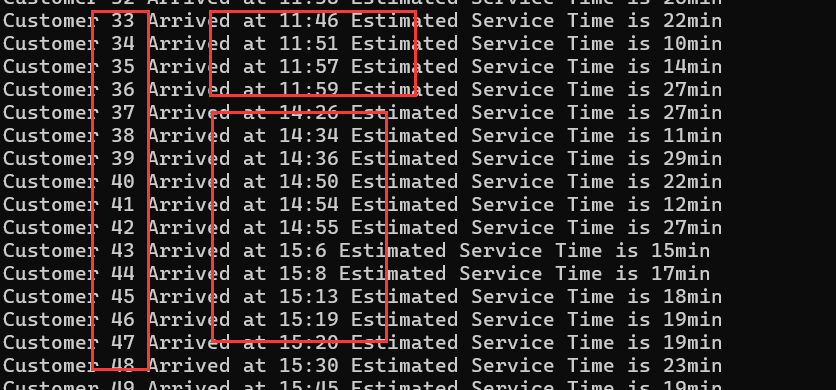
同时还应该注意，计算平均逗留时长时，还需要改每天的顾客人数number\_customers\_perday不是用顾客id表示，而是用每次队列的长度表示，这样才可以计算实际的平均逗留时间（实际生成85位顾客，但是实际入队的只有65位左右），查看number\_customers\_perday可以检验结果的正确性。

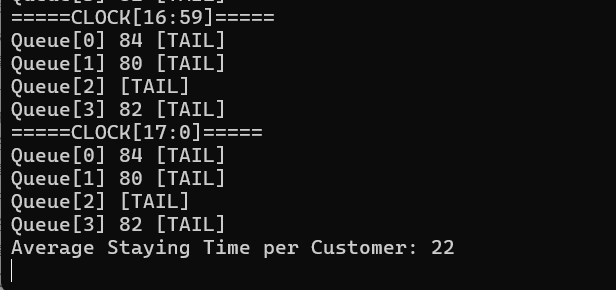
number\_customers\_perday[week] = bank.customer\_queue.size;



### 结果展示

可以实现中午休息，且顾客序号正常，能正常计算平均逗留时长





## 中午休息（12:00至14:00）有一个值班窗口

### 问题分析

实现此题的思路在于，如果顾客是在午休时间来的，与正常时间段的顾客不同之处在于，正常时间段的顾客要选择最短的窗口入队，如果判断顾客是在午休时间来的，就直接让其去0号窗口排队，具体在EventDriven(Bank\* bank, Customer\* cur\_customer, int now)函数中的修改如下：

int min = 0;

if(cur\_customer->arrival\_time >= 210 && cur\_customer->arrival\_time <= 330){

    min = 0;

}

else{

    for (int i = 0; i < NUM\_WINDOWS; ++i){

    if (bank->windows[i].queue.size < bank->windows[min].queue.size){

        min = i;

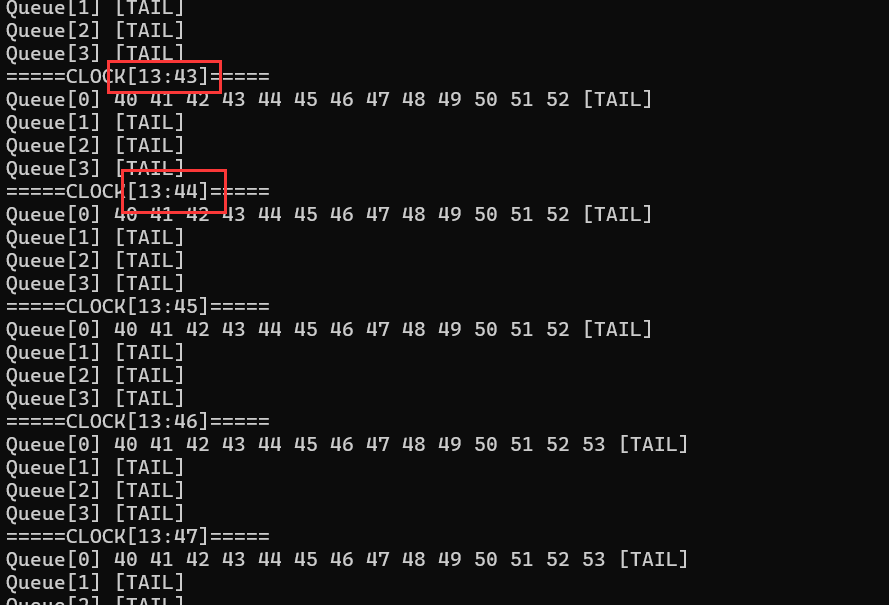
        }

    }

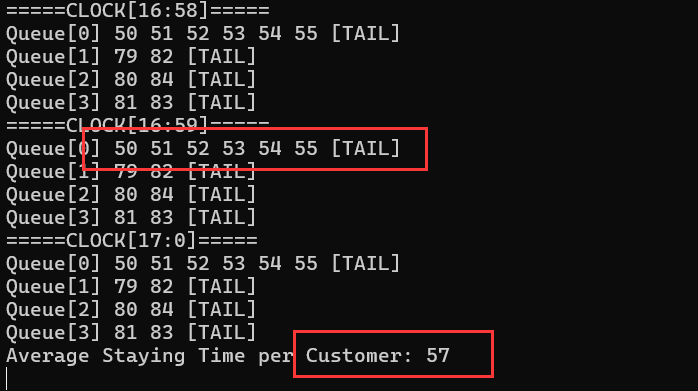
}

### 结果展示

为了展示午休时间排队的效果，延长了顾客的最大服务长与最小服务时长



通过结果可以发现一个很明显的问题，由于窗口队列先进先出的性质，在休息时间到的顾客只能在0号窗口排队，即使下午开门后，已经入队的顾客不能更换到排队更少的窗口，这是由于队列先进先出的性质决定的。



由于休息期间到来的顾客无法更换到排队人数更少的窗口，只能一直在0号窗口等待，所以即使设置顾客的最大服务时长为30，但是平均的等待时长依然在60左右。

#define MIN\_SERVICE\_TIME 10

#define MAX\_SERVICE\_TIME 30

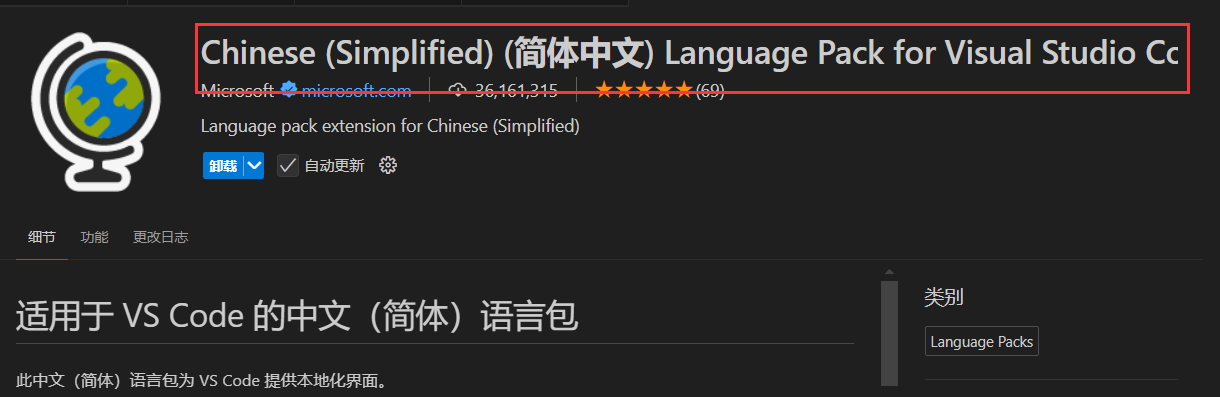
# 实验总结与常见问题汇总

## 实验总结

通过本次实验，加深了我对队列这一数据结构的理解，尤其是在模拟现实场景中如何运用队列管理多个窗口前的排队情况。同时，定义结构体的使用帮助我更清晰地表达了客户和窗口的属性及其相互关系。在函数的封装和模块化方面，我意识到将不同功能拆分成独立的函数可以让代码结构更加清晰，同时便于调试和后期维护。此外，这次实验还加深了我对编程过程中“只增加不删改”思想的理解。在完成任务3与任务4时，我尽量遵循这个原则，即在功能需求增加或修改时，尽量在现有代码基础上添加新功能，而不直接删除或修改已有的代码。这种方法不仅保持了代码的稳定性，还降低了因改动而导致的潜在错误风险。

## 常见问题汇总

中文乱码：打开老师在群里发的cpp文件时，在自己的vcsode中乱码无法正常显示，下载vscode插件可以有效解决。



但是仍然存在部分中文注释无法正常显示的问题，不影响正常阅读， 且没有找到好方法解决，输出端中文乱码，找了好多方法都没有解决，但是由于本程序的输出都是英文，并不影响正常终端输出，也可以通过在return 0 处打断点的方式强迫其在cmd输出。

