数据结构与算法实验报告-模拟叫号系统

18324034 林天皓 linth5@mail2.sysu.du.cn

摘要

本次数据结构与算法实验根据题目需求完成了一个模拟叫号系统。

引言

实验目的：设计一个模拟叫号系统，熟悉标准模板类 Stack 和 Queue 的使用，熟悉栈和队列的实现。

解决方法

在本次程序中自己定义的数据类型说明：

```
typedef int TIME;                                          //
typedef pair<int, TIME> CONSUMER;                          //id,processing_time
typedef pair<TIME, CONSUMER> WAITING_COMSUMER;             //in_time,CONSUMER
typedef tuple<TIME, TIME, WAITING_COMSUMER> PROCESSING_COMSUMER; //inpro_time,end_time,CONSUMER
typedef queue<WAITING_COMSUMER> WAITTING_LIST;
```

TIME 为 int，表示本次实验中所有的和时间有关的变量，以秒为单位。

CONSUMER 包含顾客本身的编号的和随机生成的处理该顾客需要的时间。

WAITTING_CONSUMER 包含了顾客进入排队队列的时间和 CONSUMER 的信息

PROCESSING_CONSUMER 包含了顾客开始被处理的时间和预计结束的时间，同时储存了 WAITTING_CONSUMER 的所有信息。

WAITTING_LIST 包含了目前正在等待的顾客的队列。

本程序通过一种面向对象的过程来完成整个服务队列的模拟。

模拟队列的对象定义如下

```
class order
{
private:
    static const PROCESSING_COMSUMER NULL_PROCESSING_COMSUMER;
    static Random r;
    int PROCESSING_TIME_PRE_CONSUMER;        //平均顾客处理时间
    int AVERAGE_CONSUMER_PER_HOUR;           //每小时平均产出的顾客
    int SERVER_PORT;                         //开启的服务窗口数量
    int START_HOUR;                          //开始营业的时间
    int END_HOUR;                            //设置营业结束时间
    int total_consumer_number;               //储存总共生成的顾客数目
    int now_time;                            //当前时间
    int servered_count;                      //总共处理的顾客数目
    int total_waiting_time;                  //全部顾客的等待时间
    bool stastics_ready = false;             //是否已经执行过 daytime，是否可以展
示数据
    WAITTING_LIST waitting_list;             //正在等待的顾客队列
    PROCESSING_COMSUMER *processing_port;    //每个处理窗口
    vector<PROCESSING_COMSUMER> result_list; //经过处理的顾客
    //随机生成前来排队的顾客，服从泊松分布
    void consumerGenerater(WAITTING_LIST &waitting_list, TIME now_time, int &to
tal_consumer_number);
    //将排队的顾客分配到服务窗口上
    void lineUPConmsumer(WAITTING_LIST &waitting_list, PROCESSING_COMSUMER proc
essing_port[], TIME now_time);
    //将处理结束的顾客离开处理窗口
    void realeaseConsumer(PROCESSING_COMSUMER processing_port[], TIME now_time)
;
    //输出进入排队的顾客信息
    void log(WAITING_COMSUMER w_con, WAITTING_LIST &waitting_list);
    //输出当前进入窗口处理阶段的顾客
    void log(PROCESSING_COMSUMER p_con, int port);
```

```
    //输出结束处理的顾客
    void log(TIME now_time, PROCESSING_COMSUMER p_con, int port);
    //初始化窗口
    void processingPortInit(PROCESSING_COMSUMER processing_port[], int SERVER_P
ORT);
    //输出根据 60 进制输出时间
    string outTime(TIME time);

public:
    order();
    ~order();
    void daytime();
    void showStastics();
    void setProcessingTimePreConsumer(int ProcessingTimePreConsumer);
    void setAvargeConsumerPerHour(int AvargeConsumerPerHour);
    void setServerPort(int ServerPort);
    void setStartHour(TIME StartHour);
    void setEndHour(TIME EndHour);
};
```

每个对象的作用均在注释中已经说明，此处不再赘述。

每个成员函数的内部算法描述如下

由于 set 函数较为简单，此处不再描述

```
    void consumerGenerater(WAITTING_LIST &waitting_list, TIME now_time, int &to
tal_consumer_number);
```

这个函数通过一个随机数，来决定这个时刻（以一秒为单位）是否有顾客被生成，如果有生成的顾客则创建一个有顾客 ID 的一个随机的顾客需要处理的时间的值，并且将这个顾客加入正在等待的顾客队列中。

```
    void lineUPConmsumer(WAITTING_LIST &waitting_list, PROCESSING_COMSUMER proc
essing_port[], TIME now_time);
```

这个函数通过轮询当前的服务窗口时候已经空位，如果有空位，则将服务队列的最前端的顾客添加开始处理的时间并且安排到这个服务窗口处理。

```
void realeaseConsumer(PROCESSING_COMSUMER processing_port[], TIME now_time);
```

该函数通过轮询所有窗口的顾客的预估结束时间是否等于当前的时间，如果等于则代表该顾客可以结束工作离开，则重新将该窗口设置为空闲状态，将已经完成处理的顾客的数量加一，同时将这个顾客添加一下信息，并纳入完成处理的顾客列表中，方便后续的统计工作。

```
//输出进入排队的顾客信息
void log(WAITING_COMSUMER w_con, WAITTING_LIST &waitting_list);
//输出当前进入窗口处理阶段的顾客
void log(PROCESSING_COMSUMER p_con, int port);
//输出结束处理的顾客
void log(TIME now_time, PROCESSING_COMSUMER p_con, int port);
```

该三个 log 函数在 daytime 的运行过程中输出运行情况，包括顾客的生成，开始处理顾客和顾客离开的信息。

```
void processingPortInit(PROCESSING_COMSUMER processing_port[], int SERVER_PORT);
```

该函数在每次模拟的开始情况下执行，将所有窗口的状态设置为空闲。

```
void daytime();
```

该函数的负责执行整个的模拟过程，在每个时刻按照顺序分别调用顾客的生成，开始处理顾客和顾客离开的信息，同时将时间递增，直到营业时间结束。

```
void showStastics();
```

该函数将题目所需要的等待顾客的人数，每个顾客得到服务之前等到的时间，所有顾客离开之后，顾客的平均等待时间的信息。而题目中另外需要的每个窗口正在办理业务的顾客顺序号，目前等待的顾客人数，每个顾客得到服务之前的等待时间单位数则在三个 log 函数中输出与展示。

输入与输出的形式：

Main 函数如下

```
int main()
{
    order test1;
    test1.setAvargeConsumerPerHour(100);          //设置平均每个小时的顾客数量
    test1.setStartHour(9);                      //设置开始营业的时间
    test1.setEndHour(10);                       //设置停止营业的时间
    test1.setServerPort(5);                     //设置服务的窗口数量
    test1.setProcessingTimePreConsumer(180); //设置平均处理每个顾客需求的时间，服从泊松分布
    test1.daytime();                            //开始执行队列模拟
    test1.showStastics();                       //在队列模拟之后展示每个执行顾客的编号，到来时间，排队时间， 结束时间。并最终输出顾客总数和 平均等待时间
}
```

输入输出说明：本程序没有输入，通过提前在代码中调用成员的 set 函数设定需要模拟的变量，输出通过三种 log 函数在运行中输出顾客的生成，进入窗口处理问题与离开窗口的过程。在 main 函数中，通过先创建一个对象，然后先设定开始营业，停止营业的时间，开通服务窗口的数量，设定平均处理每个顾客的处理时间和平均每个消失顾客来临的数量，再运行 daytime 成员函数进行整个服务队列的模拟，会将所有顾客的模拟信息储存在 result_list 中，方便后续统计顾客的行为等等信息。最后调用 ShowStatic 函数，进行最终每个 result_list 的信息的展示和统计总共服务顾客的数量和每个顾客的平均等待的时间。

程序使用和测试说明

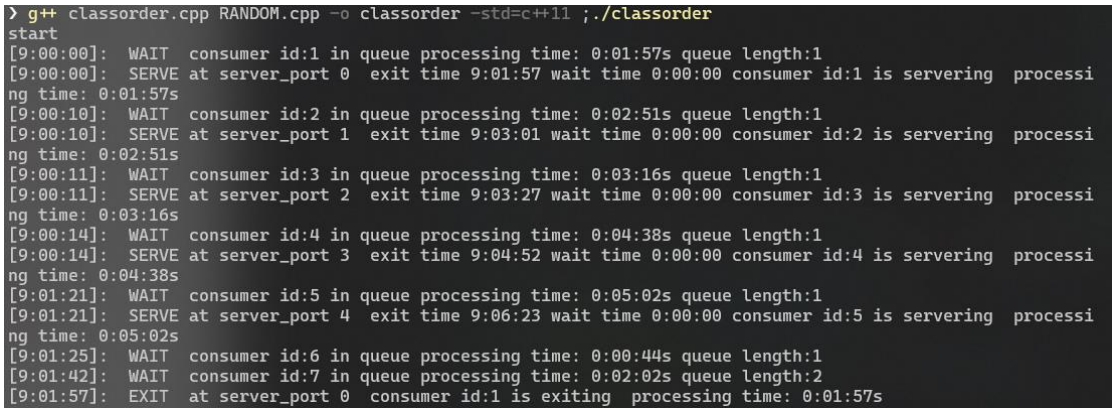
按照以上的 main 函数进行运行，

（注：本次函数考虑到输出太多，故在演示时仅将营业时间设置为一

个小时）

运行输出结果分为两部分，这里分别节选每个部分，第一部分为在 daytime 函数中所实时输出的 log 函数输出的结果，第二部分在 daytime 函数运行结束之后运行 showStastic 函数说输出的所有顾客的统计信息和统计的平均等待时间。

第一部分：

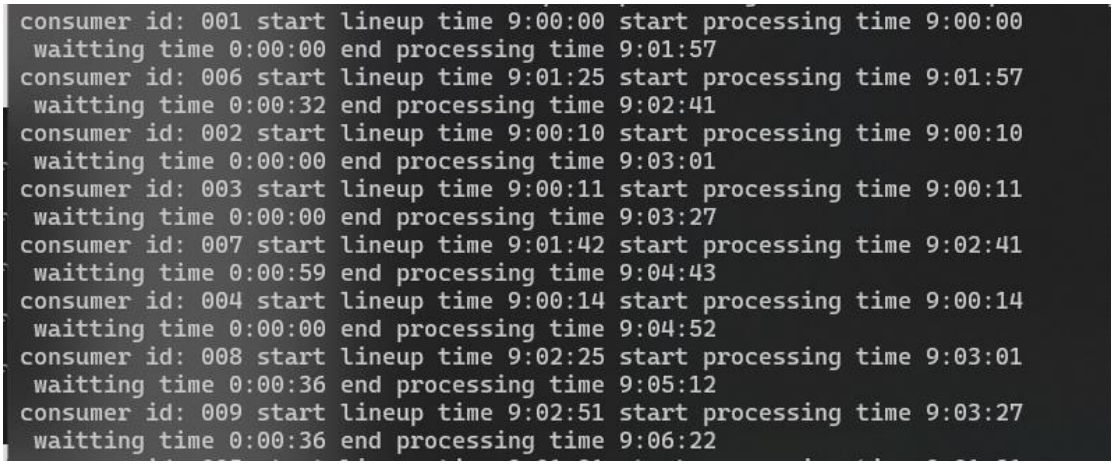


最前方为当前语句的输出时间。第一行中的 WAIT 代表有顾客被生成并且加入的等待队列中，之后分别给出了顾客的顺序号，所需要的处理时间，当前排队队列中的人数。

第二行的语句中 SERVE 代表有窗口开始处理的新的顾客业务，后面分别给出了窗口号，预估顾客的离开时间，顾客在队列中排队等待的时间。和顾客需要在窗口等待的时间。

上图的最后一行中，EXIT 代表该行有顾客离开窗口，紧随其后的输出是顾客从哪一个窗口离开，顾客的顺序号，顾客处理业务员所进行的时间。

第二部分：



第二部分的开头如下，按照顾客被处理结束的时间排序，分别为顾客顺序号，顾客开始排队的时间，顾客在窗口处理业务的时间，顾客等到的时间，顾客离开窗口的时间。

总结与讨论:

本次实验还通过 typedef 自定义了很多数据类型，为后来填写函数的参数和声明对象的代码可读性提高了，同时灵活运用 c++ 11 的迭代器方式，将冗长的 for 循环改为更加简短的遍历方式。

本次实验一开始采取函数式的运行方式，后来经过改为面向对象的方式。通过面向对象的方式，更容易地设置了模拟运行其中的参数，为多次重复，更容易的进行了多次实验，通过多次对比发现模拟队列的运行结果，也为更容易通过对象继承的方式将该服务窗口模拟更改为更多种和更多窗口的模拟。

参考文献

无

完整代码如下 ，请使用支持 c++11 以上的编译器编译运行,同时必须将老师给出了 RANDOM.H 文件和 RANDOM.CPP 文件一起编译

```
#include "RANDOM.H"
#include <iostream>
#include <list>
#include <utility>
#include <queue>
#include <tuple>
#include <iomanip>
using namespace std;
typedef int TIME;                                                    //
typedef pair<int, TIME> CONSUMER;                                    //id,p
rocessing_time
typedef pair<TIME, CONSUMER> WAITING_COMSUMER;                       //in_t
ime,CONSUMER
typedef tuple<TIME, TIME, WAITING_COMSUMER> PROCESSING_COMSUMER; //inpr
o_time,end_time,CONSUMER
typedef queue<WAITING_COMSUMER> WAITTING_LIST;
class order
{
private:
    static const PROCESSING_COMSUMER NULL_PROCESSING_COMSUMER;
    static Random r;
    int PROCESSING_TIME_PRE_CONSUMER;        //平均顾客处理时间
    int AVERAGE_CONSUMER_PER_HOUR;           //每小时平均产出的顾客
    int SERVER_PORT;                         //开启的服务窗口数量
    int START_HOUR;                          //开始营业的时间
    int END_HOUR;                            //设置营业结束时间
    int total_consumer_number;               //储存总共生成的顾客数目
    int now_time;                            //当前时间
    int servered_count;                      //总共处理的顾客数目
    int total_waiting_time;                  //全部顾客的等待时间
    bool stastics_ready = false;             //是否已经执行过 daytime，是
否可以展示数据
    WAITTING_LIST waitting_list;             //正在等待的顾客队列
    PROCESSING_COMSUMER *processing_port;    //每个处理窗口
    vector<PROCESSING_COMSUMER> result_list; //经过处理的顾客
    //随机生成前来排队的顾客，服从泊松分布
    void consumerGenerater(WAITTING_LIST &waitting_list, TIME now_time,
 int &total_consumer_number);
```

```
    //将排队的顾客分配到服务窗口上
    void lineUPConmsumer(WAITTING_LIST &waitting_list, PROCESSING_COMSU
MER processing_port[], TIME now_time);
    //将处理结束的顾客离开处理窗口
    void realeaseConsumer(PROCESSING_COMSUMER processing_port[], TIME n
ow_time);
    //输出进入排队的顾客信息
    void log(WAITING_COMSUMER w_con, WAITTING_LIST &waitting_list);
    //输出当前进入窗口处理阶段的顾客
    void log(PROCESSING_COMSUMER p_con, int port);
    //输出结束处理的顾客
    void log(TIME now_time, PROCESSING_COMSUMER p_con, int port);
    //初始化窗口
    void processingPortInit(PROCESSING_COMSUMER processing_port[], int
SERVER_PORT);
    //输出根据 60 进制输出时间
    string outTime(TIME time);

public:
    order();
    ~order();
    void daytime();
    void showStastics();
    void setProcessingTimePreConsumer(int ProcessingTimePreConsumer);
    void setAvargeConsumerPerHour(int AvargeConsumerPerHour);
    void setServerPort(int ServerPort);
    void setStartHour(TIME StartHour);
    void setEndHour(TIME EndHour);
};
Random order::r = Random(false);
const PROCESSING_COMSUMER order::NULL_PROCESSING_COMSUMER = make_tuple(
-1, -1, make_pair(-1, make_pair(-1, -1)));
void order::processingPortInit(PROCESSING_COMSUMER processing_port[], i
nt SERVER_PORT)
{
    for (int i = 0; i < SERVER_PORT; i++)
    {
        processing_port[i] = NULL_PROCESSING_COMSUMER;
    }
}

void order::daytime()
{
    cout << "start" << endl;
```

```
    total_consumer_number = 0;
    total_waiting_time = 0;
    servered_count = 0;
    result_list.clear();
    processing_port = new PROCESSING_COMSUMER[SERVER_PORT];
    processingPortInit(processing_port, SERVER_PORT);
    TIME start_time = START_HOUR * 60 * 60;
    TIME now_time = start_time;
    TIME end_time = END_HOUR * 60 * 60;
    while (now_time <= end_time)
    {
        consumerGenerater(waitting_list, now_time, total_consumer_numbe
r);
        realeaseConsumer(processing_port, now_time);
        lineUPConmsumer(waitting_list, processing_port, now_time);
        now_time++;
    }
    stastics_ready = true;
    delete[] processing_port;
}
void order::setProcessingTimePreConsumer(int ProcessingTimePreConsumer)
{
    if (ProcessingTimePreConsumer < 0)
        return;
    PROCESSING_TIME_PRE_CONSUMER = ProcessingTimePreConsumer;
}
void order::setAvargeConsumerPerHour(int AvargeConsumerPerHour)
{
    if (AvargeConsumerPerHour < 0)
        return;
    AVERAGE_CONSUMER_PER_HOUR = AvargeConsumerPerHour;
}
void order::setServerPort(int ServerPort)
{
    if (ServerPort < 0)
        return;
    SERVER_PORT = ServerPort;
}
void order::setStartHour(TIME StartHour)
{
    if (StartHour < 0 || StartHour > 24)
        return;
    START_HOUR = StartHour;
}
```

```
void order::setEndHour(TIME EndHour)
{
    if (EndHour < 0 || EndHour > 24)
        return;
    END_HOUR = EndHour;
}
string order::outTime(TIME time)
{
    cout << time / 3600 << ":" << setw(2) << setfill('0') << time % 360
0 / 60 << ":" << setw(2) << setfill('0') << time % 60;
    return "";
}
void order::consumerGenerater(WAITTING_LIST &waitting_list, TIME now_ti
me, int &total_consumer_number)
{ //随机生成顾客前来排队，服从泊松分布
    double probability_per_second = (double)AVERAGE_CONSUMER_PER_HOUR /
 3600;
    //cout<<probability_per_second<<endl;
    double k = r.random_real();
    //cout<<"possion "<<r.poisson(PROCESSING_TIME_PRE_CONSUMER)<<endl;
    int processing_time = r.random_integer(AVERAGE_CONSUMER_PER_HOUR /
3, AVERAGE_CONSUMER_PER_HOUR * 3);
    //cout<<processing_time<<endl;

    if (k < probability_per_second)
    {
        CONSUMER temp = make_pair(total_consumer_number + 1, processing
_time);
        total_consumer_number++;
        WAITING_COMSUMER temp_w_con = make_pair(now_time, temp);
        waitting_list.push(temp_w_con);
        log(temp_w_con, waitting_list);
    }
}
void order::lineUPConmsumer(WAITTING_LIST &waitting_list, PROCESSING_CO
MSUMER processing_port[], TIME now_time)
{ //将排队的顾客分配到窗口
    for (int i = 0; i < SERVER_PORT; i++)
    {
        if (get<0>(processing_port[i]) == -1 && !waitting_list.empty())
        {
            WAITING_COMSUMER top_consumer = waitting_list.front();
            waitting_list.pop();
```

```
            PROCESSING_COMSUMER temp_p_con = make_tuple(now_time, now_t
ime + get<1>(top_consumer.second), top_consumer);
            processing_port[i] = temp_p_con;
            total_waiting_time += now_time - top_consumer.first;
            servered_count++;
            log(temp_p_con, i);
        }
    }
}
void order::realeaseConsumer(PROCESSING_COMSUMER processing_port[], TIM
E now_time)
{ //查询是否有顾客处理结束
    for (int i = 0; i < SERVER_PORT; i++)
    {
        if (get<1>(processing_port[i]) == now_time)
        {
            log(now_time, processing_port[i], i);
            result_list.push_back(processing_port[i]);
            processing_port[i] = NULL_PROCESSING_COMSUMER;
        }
    }
}
void order::showStastics()
{
    /*for(int i=0;i<SERVER_PORT;i++){
        cout<<" port "<<i+1<<" "
        <<"serveing consumer id "<<get<\0>(get<2>(processing_port[i]).s
econd)<<endl;
    }*/
    if (!stastics_ready)
    {
        cout << "please run daytime() first" << endl;
        return;
    }
    for (auto &i : result_list)
    {
        WAITING_COMSUMER &w_con = get<2>(i);
        CONSUMER &con = w_con.second;
        cout << "consumer id: " << setw(3) << con.first
             << " start lineup time " << outTime(w_con.first)
             << " start processing time " << outTime(get<0>(i)) << endl
             << " waitting time " << outTime(get<0>(i) - w_con.first)
             << " end processing time " << outTime(get<1>(i)) << endl;
    }
```

```
    cout << "total served consumer " << servered_count << " "
         << "avarge waitting time " << setprecision(6) << (double)total
_waiting_time / servered_count
         << " ";
}
//consumerGeneraterlog
void order::log(WAITING_COMSUMER w_con, WAITTING_LIST &waitting_list)
{
    cout << "[" << outTime(w_con.first) << "]: "
         << " WAIT  "
         << "consumer id:" << get<0>(w_con.second) << " in queue"
         << " processing time: " << outTime(get<1>(w_con.second)) << "s
"
         << " queue length:" << waitting_list.size() << endl;
}
//lineUPConmsumerlog
void order::log(PROCESSING_COMSUMER p_con, int port)
{
    cout << "[" << outTime(get<0>(p_con)) << "]: "
         << " SERVE "
         << "at server_port " << port << " "
         << " exit time " << outTime(get<1>(p_con))
         << " wait time " << outTime(get<0>(p_con) - get<2>(p_con).firs
t);
    WAITING_COMSUMER w_con = get<2>(p_con);
    cout << " consumer id:" << get<0>(w_con.second) << " is servering "
         << " processing time: " << outTime(get<1>(w_con.second)) << "s
" << endl;
}
//realeaseConsumerlog
void order::log(TIME now_time, PROCESSING_COMSUMER p_con, int port)
{
    cout << "[" << outTime(now_time) << "]: "
         << " EXIT  "
         << "at server_port " << port << " ";
    WAITING_COMSUMER w_con = get<2>(p_con);
    cout << " consumer id:" << get<0>(w_con.second) << " is exiting "
         << " processing time: " << outTime(get<1>(w_con.second)) << "s
" << endl;
}
order::order(/* args */) {}

order::~order() {}
```

```
int main()
{
    order test1;
    test1.setAvargeConsumerPerHour(110);
    test1.setStartHour(9);                      //设置开始营业的时间
    test1.setEndHour(10);                       //设置停止营业的时间
    test1.setServerPort(5);                     //设置服务的窗口数量
    test1.setProcessingTimePreConsumer(300); //设置平均处理每个顾客需求的
时间，服从泊松分布
    test1.daytime();                            //开始执行队列模拟
    test1.showStastics();                       //在队列模拟之后展示每个执行
顾客的编号，到来时间，排队时间， 结束时间。并最终输出顾客总数和 平均等待时间
}
```