华东师范大学计算机科学与技术学院上机实践报告

课程名称：数据结构	**年级**：2022 级	**上机实践成绩**：
指导教师：金健	**姓名**：岳锦鹏	**上机实践时间**：2 学时
上机实践名称：第三章作业	**学号**：10213903403	**上机实践日期**：2023/10/20

一、 实验目的

1. 使用第三章知识解决链表队列问题。

二、 实验内容

1. 使用链队列模拟银行窗口业务。假设每位客户有 2 项信息：序号、金额。早上开张后，按随机时间有客户上门办理业务，假设携带的现金是 1-20 万之间的随机数。来一位客户，就将其加入队列，等待办理业务。假设办理业务所需要消耗的时间和金额成正比（可设置一个系数），位于队头的客户办理完业务后，就从队列中移走。请编写程序动态模拟这个过程。
2. （选做）在第一题的基础上，如果银行有多个窗口。
   (1) 客户过来时挑队列最短的排队，后面不会换队列。
   (2) 如果排队过程中，出现相邻队列比自己所排队列短，则可以重新选择队列。

三、 实验原理

1. 程序设计原理。

四、 实验步骤

1. 问题抽象
2. 编写程序
3. 调试程序
4. 完善总结

五、 调试过程、结果和分析

1. C++ 采用面向对象的方式设计复杂程序时，要处理非常多的类型，因此可能会比较复杂；
2. 传参数有引用传递、指针传递、值传递；
3. 类的模板是静态多态，类的继承是动态多态；
4. 子类对父类函数的重写、重载、覆盖；
5. 私有、保护、公共、友元的成员访问权限；
6. 比起 Python 和 JavaScript，C++ 的各种类型、各种限制、各种访问权限非常复杂，由于初次调试这种复杂的程序，出现各种错误时也需要查资料解决，导致花费大约三天时间才完成这次作业；

7. 但也许这就是 C++ 这种强类型语言的特点，编写代码时需要对类型进行非常确定的定义（模板可能是例外），但这也保证了运行效率和安全性；
8. 此次作业没有尝试使用异步、多线程、多进程等并发执行方式来提升执行效率并改善图形（命令行）界面体验，之后可以尝试加入；
9. 也是初次使用 cmake 进行程序的构建、测试、打包，cmake 用来管理这种多文件的复杂程序时确实比较方便。

六、 总结

1. 对于这种较为复杂的程序，首先考虑它的架构，由于此程序需要使用到链队列的模型，而且还需要进行图形化或命令行的展示，因此考虑采用 MVC 架构，但又由于程序功能较少，因此不想使用多线程或异步的事件循环，因此尝试是否能在单线程同步的架构下实现 MVC 架构；
2. 事实证明是可以在单线程同步的架构下实现 MVC 架构的，但是可能一些细节会有所更改，例如：
   (1) MVC 中的 Controller 接受用户事件并通知 Model，这种行为在没有事件循环的情况下采用了 View 直接调用 Model 来实现；
   (2) Model 通知 View 进行修改的行为，改为 View 直接依赖 Model，之后更改 View 的时候只需调用 View::refresh 的方法，View 就会从自己已经绑定的 Model 中获取数据并进行刷新。

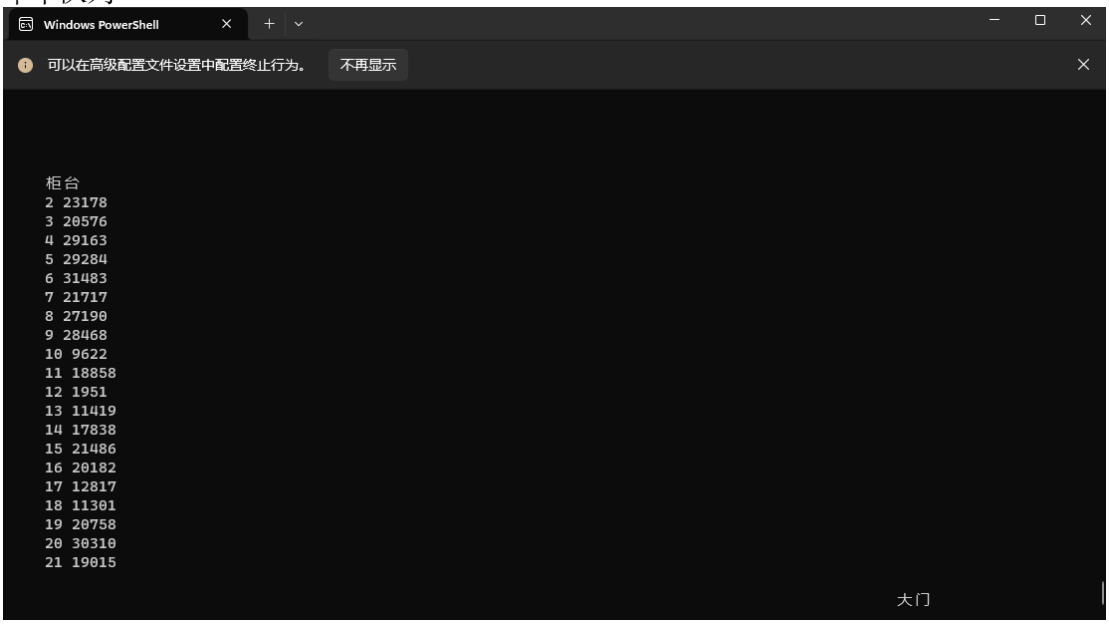
   但是这样造成了 View 依赖 Model，Controller 依赖 View 和 Model，也许可能不太符合 MVC。
3. 之后就需要考虑如何拆分各个模块，这里采用了面向对象的思想：
   (1) Model: 链队列的实现、客户对象；
   (2) View: 背景界面、排队的显示、客户的显示、走路过程中的客户的显示；
   (3) Controller: 实现主循环，单个 Controller 绑定 Model 和 View，调度 Model 和 View 的更新；
   (4) MVC: 调用主循环，综合多个 Controller，统一调度所有 Controller 的更新，之后所有 View 一起更新；
   (5) main: 实现根据用户输入进行调整调用参数，是入口文件。

   每个文件中也有类之间的继承关系：
   (1) SingleQueueModel 继承了 Model；
   (2) SimpleQueueView 继承了 QueueView，QueueView 继承了 View；
   (3) SingleQueueController 继承了 Controller，DriftingController 也继承了 Controller。
4. 虽然采用面向对象，总体是成功实现了，但各个模块之间的职责划分仍不是特别清晰，并且产生了很多依赖关系，因此仍需改进。
5. 由于时间有限，暂未实现相邻队列换位的模拟。

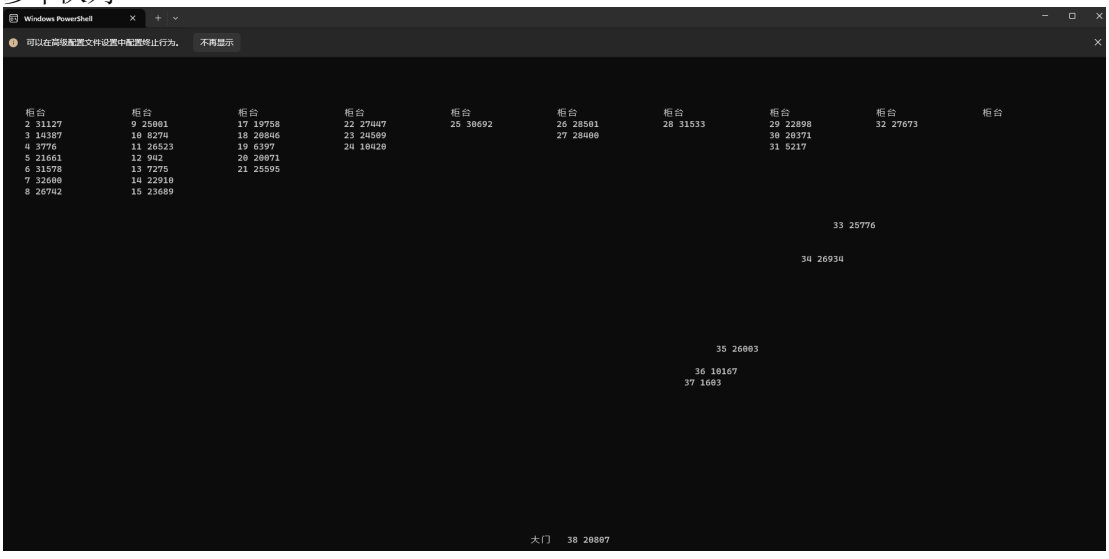
七、 附件

运行时截图：

1. 单个队列

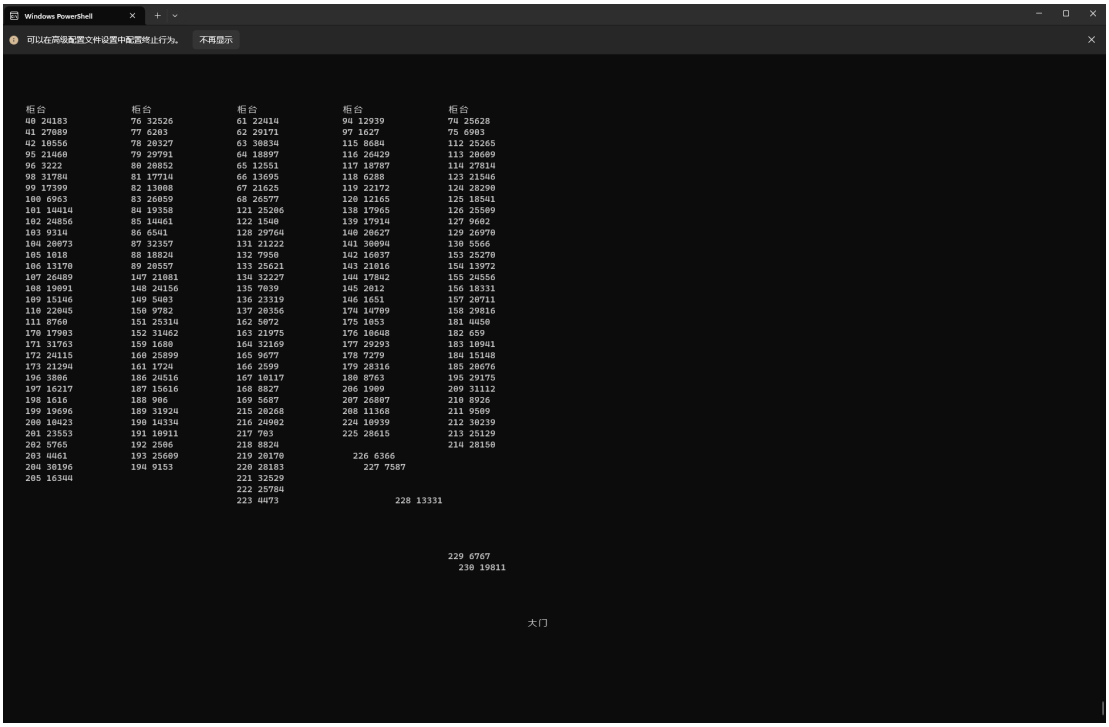


2. 多个队列



3. 自定义

```
请输入模式，1为单个队列，2为多个队列，3为自定义，其他整数表示退出（默认值2）：  3
请输入每个时刻有1/{输入值}的概率来人（默认值10）：  5
请输入每个窗口办理的速度（默认值100）：  200
请输入总时刻数（默认值10000）：  10000
请输入最大携带金额（默认值200000）：  200000
请输入柜台数量（默认值10）：  5
请输入人的走路速度（浮点数）（默认值2）：  1.5
请输入每次刷新间隔多少毫秒（默认值10）：  1
```



代码：

(1) constants.h

```
1  // @Time    :   2023-10-23 21:42:40
2  // @FileName:   constants.h
3  // @Author  :   423A35C7
4  // @Software:   VSCode
5
6  #ifndef _CONSTANTS
7  #define _CONSTANTS
8  #include <bits/stdc++.h>
9
10 typedef int Status;
11 typedef int int_;       // 用来扩展，可以更改来实现 long long 长度的，
   ↪ 但好像违反了对修改封闭的原则
12 #define int_MAX INT_MAX // 配合上面的扩展用
13 using coordinate = std::pair<int, int>;
14
15
```

```
16 #define DEFAULT_PROBABILITY_NUM 10 // 默认每个时刻有 1/DEFAULT_PROBABILITY_NUM 的概率来人
17 #define DEFAULT_SPEED 1e2           // 默认每个窗口办理的速度
18 #define DEFAULT_TOTAL_TIME 1e4      // 默认的总时刻数
19 #define MAX_MONEY 200000            // 最大携带金额
20 #define DEFAULT_WINDOW_NUM 10       // 默认柜台数量
21 #define DEFAULT_BASE_X 5            // 默认起始位置距离终端上边几个字符的距离
22 #define DEFAULT_BASE_Y 5            // 默认起始位置距离终端左边几个字符的距离
23 #define DEFAULT_SEP 20              // 默认每个窗口间隔多少距离
24 #define DEFAULT_WALK_SPEED 2.0      // 默认的人的走路速度
25 #define DEFAULT_GATE_X 50           // 默认的大门的位置终端上边几个字符的距离
26 #define DEFAULT_GATE_Y 100          // 默认的大门的位置终端左边几个字符的距离
27 #define DEFAULT_SLEEP_TIME 10       // 默认每次刷新间隔多少毫秒
28
29 #define DEFAULT_RANDOM_SEED (unsigned)std::time(NULL) // 默认的随机数种子
30
31 const Status OK = 0;
32 const Status ERROR = -1;
33
34 // const unsigned seed = DEFAULT_RANDOM_SEED;
35
36 // https://www.cnblogs.com/shirishiqi/p/5431627.html
37 // 此宏展开后，类似于 printf("123"),printf("456");
38 #define TRACE_CMH_1 (printf("%s(%d)-<%s>: ", __FILE__, __LINE__, __FUNCTION__), printf)
39
40 // 此宏展开后，类似于 printf("%d""%d", 1, 2);
41 #define TRACE_CMH_2(fmt, ...) \
42     printf("%s(%d)-<%s>: "##fmt, __FILE__, __LINE__, __FUNCTION__, ##__VA_ARGS__)
43
44 #endif
```

(2) model.hpp

```
1 // @Time    :   2023-10-23 20:09:23
```

```
2  // @FileName:   model.h
3  // @Author  :   423A35C7
4  // @Software:   VSCode
5
6  #ifndef _MODEL
7  #define _MODEL
8
9  #include "constants.h"
10 #include <bits/stdc++.h>
11
12 // using namespace std; // 会导致 byte 冲突
13
14 class Model {
15     // virtual void init(...) = 0;
16 };
17
18 template <class T, class Ref, class Ptr>
19 struct __SingleQueueModel_iterator;
20
21 template <typename T>
22 class SingleQueueModel : Model {
23 public: // 这里用保护属性会在 view 的 resfresh 里报错
24     class Node;
25
26 private:
27     // unique_ptr<Node> head(new Node()); // 这样会报错 error:
      ↪  expected identifier before 'new'
28      // vector 内对一个对象可能有多个引用，所以可能使用了 unique_ptr 就
       ↪  无法放置在 vector 里
29     std::shared_ptr<Node> head = std::make_shared<Node>(); // 好像用
      ↪  unique_ptr 直接初始化会报错
30     Node *tail = head.get();
31     typedef __SingleQueueModel_iterator<T, const T &, const T *>
      ↪  const_iterator;
32
33 protected:
34     int_ length = 0;
35
36 public:
37     void init(){};
38
39     void push(const T &data) {
```

```
40          this->tail->next = new Node(data); // head 指向的头结点无值
41          this->tail = this->tail->next; // tail 指向的始终有值
42          this->length ++;
43      }
44      // Status pop(T&) = 0;
45      T shift() {
46          if (this->head->next == this->tail) {
47              this->tail = this->head.get(); // 只剩下一个元素的情况下删除后尾指针应指向头结点
48          }
49          Node node = *this->head->next; // 这个临时的结点在退出此函数时也会调用析构函数
50          delete this->head->next;        // 这里会调用析构函数
51          this->head->next = node.next;
52          this->length --;
53          return node.data;
54      }
55
56      T &top() {
57          return this->head->next->data;
58      }
59
60      int_ get_length() {
61          return this->length;
62      }
63
64      // https://blog.csdn.net/qq_54851255/article/details/123939684
65      const_iterator begin() const {
66          return const_iterator(this->head->next);
67      }
68
69      const_iterator end() const {
70          return const_iterator(this->tail->next);
71      }
72  };
73
74  template <class T, class Ref, class Ptr>
75  struct __SingleQueueModel_iterator {
76      typedef class SingleQueueModel<T>::Node Node;
77      typedef __SingleQueueModel_iterator<T, Ref, Ptr> self;
78      Node *_node;
79
```

```
80      __SingleQueueModel_iterator(Node *x)
81          : _node(x) {}
82      Ref operator*() {
83          return _node->data;
84      }
85
86      Ptr operator->() {
87          return &_node->data;
88      }
89
90      self &operator++() {
91          _node = _node->next;
92          return *this;
93      }
94
95      self operator++(int) {
96          self tmp(*this);
97          _node = _node->next;
98          return tmp;
99      }
100
101     bool operator!=(const self& it) const
102     {
103         return _node != it._node;
104     }
105 };
106
107 // template <typename T>
108 // class SimpleQueueModel : public SingleQueueModel<T> {
109 //     class Node; // 这一行必须得加上，表示子类需要重写父类的成员，否则
    ↪ 在外部定义时会报错 invalid class name in declaration of 'class
    ↪ SimpleQueueModel<T>::Node'
110 // };
111
112 template <typename T>
113 class SingleQueueModel<T>::Node {
114 public:
115     T data;
116     Node *next = NULL;
117     Node() {}
118     // Node(Node *) {}
119     Node(const T &data) {
```

```
120         this->data = data;
121     }
122 #if _DEBUG
123     ~Node() {
124         TRACE_CMH_1;
125         std::cout << "Node " << data << " at " << this << "
        ↪  destructed" << std::endl;
126     }
127 #endif
128 };
129
130 template <typename T>
131 class ComplexSingleQueueModel : SingleQueueModel<T> {
132 public:
133     Status push(T) = 0;
134     Status pop(T &) = 0;
135     Status shift(T &) = 0;
136 };
137
138 template <typename SingleQueueModel>
139 class MultiQueueModel : Model {
140 public:
141     Status init(int_ queue_num) = 0;
142     Status get(int_ index, SingleQueueModel &) = 0;
143     Status move(int_ start, int_ destination) = 0;
144 };
145
146 struct Customer {
147     int_ number;
148     long cash;
149 };
150
151 std::ostream &operator<<(std::ostream &out, const Customer &customer)
    ↪  {
152     out << customer.number << " " << customer.cash;
153     return out;
154 };
155
156 #endif
```

(3) view.hpp

```cpp
// @Time    :   2023-10-23 21:37:53
// @FileName:   view.h
// @Author  :   423A35C7
// @Software:   VSCode

#ifndef _VIEW
#define _VIEW

#include "constants.h"
#include "model.hpp"
#include <bits/stdc++.h>

#define move_and_output(x, y, str) printf("\033[s\033[%d;%dH%s\033u", x, y, str)
#define moveto(x, y) printf("\033[s\033[%d;%dH", x, y)
// 下移光标
#define movedown(x) printf("\033[%dB", (x))
#define save_cursor() printf("\033[s")
#define restore_cursor() printf("\033[u")
#define clear_char(num)           \
    for (int i = 0; i < num; i++) \
    printf(" ")


class View {
public:
    virtual void refresh() = 0;
};

class BackgroundView : View {
private:
    int gate_x;
    int gate_y;

public:

    void refresh() {
        printf("\033[2J\033[?25l");
        move_and_output(this->gate_x + 1, this->gate_y, " 大门");
    }

```

```
41      BackgroundView(int gate_x, int gate_y) : gate_x(gate_x),
        ↪ gate_y(gate_y) {
42          this->refresh();
43      }
44      ~BackgroundView() {
45          printf("\033[2J\033[0m\033[?25h");
46      }
47  };
48
49  template <typename T, typename QueueModel>
50  class QueueView : View {
51  protected:
52      virtual void init() {}
53      QueueModel &queue_model;
54  public:
55      QueueView() {}
56      QueueView(QueueModel &_queue_model) : queue_model(_queue_model)
        ↪ {}
57      // 这里如果不使用引用会在调用时报错:
58      // 无法引用 函数 "SingleQueueModel<T>::SingleQueueModel(const
        ↪ SingleQueueModel<Customer> &) [其中 T=Customer]" (已隐式声明)
        ↪ -- 它是已删除的函数 C/C++(1776)
59      virtual void refresh() {}
60
61      // 此方法不一定合理，因为 View 直接更改了 Model
62      void push_to_model(T &data) {
63          this->queue_model.push(data);
64      }
65      virtual std::pair<int, int> get_last_pos() {
66          return std::make_pair(-1, -1);
67      }
68  };
69
70  // MVC 中的 View 不应该依赖 Model，这里的模板中只是使用了 Model 的名称,
    ↪ 实际上并不依赖 model 实现的代码，只要这个模板类型有迭代器的实现就行
71  template <typename T, typename QueueModel>
72  class SimpleQueueView : public QueueView<T, QueueModel> {
73  friend View;
74  private:
75      int_ base_x; // x 是向下增加
76      int_ base_y; // y 是向右增加
77      int_ tail_y;
```

```
78
79  protected:
80      QueueModel &queue_model;
81
82  public:
83      SimpleQueueView(QueueModel &_queue_model) :
        ↪  queue_model(_queue_model), QueueView<T,
        ↪  QueueModel>(_queue_model) {
84          this->base_x = 0;
85          this->base_y = 0;
86      }
87      SimpleQueueView(QueueModel &_queue_model, int_ base_x, int_
        ↪  base_y) : queue_model(_queue_model), QueueView<T,
        ↪  QueueModel>(_queue_model) {
88          this->base_x = base_x;
89          this->base_y = base_y;
90      }
91      std::pair<int, int> get_last_pos() {
92          return std::make_pair(this->base_x + 1 +
            ↪  this->queue_model.get_length(), this->base_y);
93      }
94
95      void refresh() {
96          move_and_output(base_x, base_y, " 柜台");
97          // this->base_x++;
98          moveto(base_x + 1, base_y);
99          for (auto node = this->queue_model.begin(); node !=
            ↪  this->queue_model.end(); node++) {
100             save_cursor();
101             std::cout << *node;
102             restore_cursor();
103             movedown(1);
104         }
105         restore_cursor();
106     }
107 };
108
109 class MultiQueueView : View {
110     virtual Status init(int_ queue_num) = 0;
111     virtual Status move(int_ start, int_ destination) = 0;
112 };
113
```

```
114
115 // 还没法转成动态多态，不然还是需要一个 T 的模板，所以还不如直接用 Model 作为模板
116 // 这个类可能只能弄成有状态的，也就是说每次刷新的行为不仅与当前时刻的 model 有关，还与之前时刻的 model 有关
117 // 还是直接用 T 作为模板吧，虽然增加了模块之间互相依赖，即 controller 需要直接向 view 传节点，但减少了代码量
118 template <typename T>
119 class DriftingView : View {
120     public:
121     struct Node {
122         T data;
123         // ../include/view.hpp:152:58: error: passing 'const QueueView<SingleQueueModel<Customer> >' as 'this' argument discards qualifiers [-fpermissive]
124 //   152 |             coordinate target = node.target->get_last_pos();
125     //   |                                 ~~~~~~~~~~~~~~~~~~~~~~~~~^~
126         QueueView<T, SingleQueueModel<T>> & target;
127         int_ remained_time;
128         coordinate current;
129     };
130     private:
131     std::list<Node> state;
132     public:
133     void push(Node node) {
134         this->state.push_back(node);
135     }
136     Status main() {
137         // 如果用 auto 遍历，node 就应该不是迭代器类型
138         for (auto node = this->state.begin(); node != this->state.end();) {
139             coordinate target = node->target.get_last_pos();
140             node->current.first += (target.first - node->current.first) / node->remained_time;
141             node->current.second += (target.second - node->current.second) / node->remained_time;
142             node->remained_time--;
143             if (node->remained_time <= 0) {
144                 node->target.push_to_model(node->data); // View 应该直接调用 Model 吗？
```

```
145                 this->state.erase(node++); // 遍历时需要先自增再删除
146             } else {
147                 node++;
148             }
149         }
150         return OK;
151     }
152     void refresh() {
153         for (auto &node : this->state) {
154             moveto(node.current.first, node.current.second);
155             std::cout << node.data;
156             restore_cursor();
157         }
158     }
159 };
160
161 #endif
```

(4) controller.hpp

```
1  // @Time    :   2023-10-24 11:17:40
2  // @FileName:   controller.h
3  // @Author  :   423A35C7
4  // @Software:   VSCode
5
6  #ifndef _CONTROLLER
7  #define _CONTROLLER
8
9  #include "constants.h"
10 // #include "model.hpp" // view 里已经依赖 model 了
11 #include "view.hpp"
12
13 class Controller {
14     // virtual Status main() = 0;
15 };
16
17 template <typename T>
18 class DriftingController;
19
20 template <typename T>
21 class SingleQueueController : Controller {
```

```
22  friend class DriftingController<T>;
23  private:
24      int_ consumed_time = 0;
25      int speed = DEFAULT_SPEED;
26
27      SingleQueueModel<T> &model;
28      // BackgroundView background_view;
29      QueueView<T, SingleQueueModel<T>> &view;
30      // void (View:: a)() = background_view.init;
31      // 捕获了 this 的 lambda 函数好像不能转化为 void 函数指针
32      // initializer_list<void ()> temp {
33      //     [&](this){background_view.init();},
34      //     [&](this){view.refresh(model);}
35      // };
36  public:
37      // refresh 可以认为是 controller 内部自己进行更新
38      void refresh() {
39          if (this->model.get_length() <= 0) {
40              this->consumed_time = 0;
41          } else {
42              this->consumed_time++;
43              if (this->consumed_time * this->speed >=
                ↪  this->model.top().cash) {
44                  this->model.shift();
45                  this->consumed_time = 0;
46              }
47          }
48      }
49      SingleQueueController(SingleQueueModel<T> &model, QueueView<T,
        ↪  SingleQueueModel<T>> &view, int speed = DEFAULT_SPEED) :
        ↪  model(model), view(view), speed(speed) {
50          this->refresh();
51      }
52  };
53
54  template <typename T>
55  class DriftingController : Controller {
56  private:
57      std::vector<std::shared_ptr<SingleQueueController<T>>>
        ↪  &single_queue_controllers;
58      DriftingView<T> &drifting_view;
59      double walk_speed = DEFAULT_WALK_SPEED;
```

```
60  public:
61      void push(T data, int target, coordinate current_pos =
        ↪  std::make_pair(DEFAULT_GATE_X, DEFAULT_GATE_Y), double
        ↪  walk_speed = DEFAULT_WALK_SPEED) {
62          auto &view = this->single_queue_controllers[target]->view;
63          coordinate target_pos = view.get_last_pos();
64          int_ remained_time = (int_)(sqrt(pow(target_pos.first -
            ↪  current_pos.first, 2) + pow(target_pos.second -
            ↪  current_pos.second, 2)) / walk_speed);
65          // DriftingView<Customer>::Node temp ;
66          // 如果形参是引用的话会出现如下问题:
67          // error: cannot bind non-const lvalue reference of type
            ↪  'DriftingView<Customer>::Node&' to an rvalue of type
            ↪  'DriftingView<Customer>::Node'
68          this->drifting_view.push({
69              data,
70              view,
71              remained_time,
72              current_pos
73          });
74      }
75      DriftingController(
76          std::vector<std::shared_ptr<SingleQueueController<T>>>
            ↪  &single_queue_controllers,
77          DriftingView<T> &drifting_view
78      ) :
79          single_queue_controllers(single_queue_controllers),
80          drifting_view(drifting_view) {}
81  };
82
83  Status mainloop(std::function<Status()> main, std::function<void()>
    ↪  refresh, int_ total_time = DEFAULT_TOTAL_TIME) {
84      Status temp;
85      for (int i = 0; i < total_time; i++) {
86          if ((temp = main()) != OK)
87              return temp;
88          refresh();
89      }
90      return OK;
91  }
92
93  #endif
```

(5) MVC.h

```h
// @Time    :   2023-10-28 18:25:30
// @FileName:   MVC.h
// @Author  :   423A35C7
// @Software:   VSCode

#ifndef _MVC
#define _MVC

#include "constants.h"


// 在 Simple 中都是通过全局变量设置的

// 在 Multi 中,
// 以下的是通过向类传递参数设置的
extern int probability_num; // 每个时刻有 1/probability_num 的概率来人
extern int speed;           // 每个窗口办理的速度
extern int_ total_time;     // 总时刻数
extern int max_money;       // 最大携带金额
extern int window_num;      // 柜台数量
extern double walk_speed;   // 人的走路速度

// 以下的是通过全局变量设置的
extern int sleep_time; // 每次刷新间隔多少毫秒
extern int base_x;     // 起始位置距离终端上边几个字符的距离
extern int base_y;     // 起始位置距离终端左边几个字符的距离
extern int sep;        // 每个窗口间隔多少距离
extern int gate_x;     // 大门的位置终端上边几个字符的距离
extern int gate_y;     // 大门的位置终端左边几个字符的距离

class Simple;
class Multi;
Status main_simple();
Status main_multi();

#endif
```

(6) MVC.cpp

```cpp
// @Time    :   2023-10-28 16:11:17
// @FileName:   MVC.cpp
// @Author  :   423A35C7
// @Software:   VSCode

#include "controller.hpp"
#include "MVC.h"
// #ifdef __WIN32__
// 添加"-D_HAS_STD_BYTE=0", 的方法不知道为什么没用
// #include <windows.h>
// #else
// #include <unistd.h>
// #define Sleep(a) usleep(a * 1000) // 需要小于一秒
// #endif
// 用 _sleep 了，虽然会提示不建议使用了，但是用 window.h 编译出来的会被认为是病毒


class Simple {
private:
    SingleQueueModel<Customer> model;
    SimpleQueueView<Customer, SingleQueueModel<Customer>> view{model, 5, 5};
    // SimpleQueueView<SingleQueueModel<Customer>> view = SimpleQueueView<SingleQueueModel<Customer>>(model, 5, 5, 20);
    BackgroundView background_view{gate_x, gate_y};
    SingleQueueController<Customer> controller{model, view, speed};
    // int probability_num = DEFAULT_PROBABILITY_NUM;

public:
    // main 可以认为是外部对 Controller 的操作
    // num 是序号
    Status _main() {
        static int_ num = 0;
        if (rand() % probability_num == 0) { // 0.01 的概率
            Customer customer{num++, rand() % (max_money) + 1};
            this->model.push(customer);
        }
        return OK;
    }

    // 可以认为此操作不应出错
```

```
40      void refresh() {
41          this->controller.refresh();
42          this->background_view.refresh();
43          this->view.refresh();
44          _sleep(sleep_time);
45      }
46  };
47
48  class Multi {
49      using one_model = SingleQueueModel<Customer>;
50      using one_view = SimpleQueueView<Customer,
        ↪  SingleQueueModel<Customer>>;
51      using one_controller = SingleQueueController<Customer>;
52
53      std::vector<std::shared_ptr<one_model>> models;
54      std::vector<std::shared_ptr<one_view>> views;
55      BackgroundView background_view{gate_x, gate_y};
56      std::vector<std::shared_ptr<one_controller>> controllers;
57      DriftingView<Customer> drifting_view;
58      DriftingController<Customer> drift_controller;
59      int len = 0;
60      int probability_num = DEFAULT_PROBABILITY_NUM;
61      int speed = DEFAULT_SPEED;
62      int max_money = MAX_MONEY;
63      double walk_speed = DEFAULT_WALK_SPEED;
64
65  public:
66      Multi(int len, int probability_num = DEFAULT_PROBABILITY_NUM, int
        ↪  speed = DEFAULT_SPEED, int max_money = MAX_MONEY, double
        ↪  walk_speed = DEFAULT_WALK_SPEED) : len(len),
        ↪  probability_num(probability_num), speed(speed),
        ↪  max_money(max_money), walk_speed(walk_speed),
        ↪  drift_controller(controllers, drifting_view) {
67          if (len < 1)
68              throw std::bad_array_new_length(); // 异常这样用好像不太对
69          for (int i = 0; i < len; i++) {
70              // one_model _model {};
71              // one_view _view(_model, 5 + i * 20, 5);
```

```
72              // one_controller _controller(_model, _view); // 这里的引
                ↪  用很奇怪，可能是自己创建的类在 _model 离开作用域后继续引
                ↪  用它无法保证它不被修改，也就是说它引用的可能还是原来的变
                ↪  量名，但原来的变量名被更改了，但 vector 里可以继续正确引
                ↪  用
73              // one_view _view(this->models.back()); // 这样也无法正确
                ↪  引用，看来还是得用指针
74              // one_controller _controller(this->models.back(),
                ↪  this->views.back());
75
76              // vector 内对一个对象可能有多个引用，所以可能使用了
                ↪  unique_ptr 就无法放置在 vector 里
77              std::shared_ptr<one_model> _model(new one_model());
78              std::shared_ptr<one_view> _view(new one_view(*_model,
                ↪  base_x, base_y + i * sep));
79              std::shared_ptr<one_controller> _controller(new
                ↪  one_controller(*_model, *_view, this->speed));
80              this->models.push_back(_model);
81              this->views.push_back(_view);
82              this->controllers.push_back(_controller);
83          }
84      }
85
86      // 如果有多个最短的，最左边的优先
87      int get_shortest_queue() {
88          int min_value = int_MAX;
89          int argmin = 0;
90          for (int i = 0; i < this->len; i++) {
91              if (this->models[i]->get_length() < min_value) {
92                  min_value = this->models[i]->get_length();
93                  argmin = i;
94              }
95          }
96          return argmin;
97      }
98
99      Status _main() {
100         static int_ num = 0;
101         this->drifting_view.main();
102         if (rand() % this->probability_num == 0) { // 0.01 的概率
103             Customer customer{num++, rand() % (this->max_money) + 1};
```

```
104             //
                ↪ this->models[this->get_shortest_queue()]->push(customer);
105             this->drift_controller.push(
106                 customer,
107                 this->get_shortest_queue(),
108                 std::make_pair(gate_x, gate_y),
109                 this->walk_speed);
110         }
111         return OK;
112     }
113
114     void refresh() {
115         for (auto _controller : this->controllers) {
116             _controller->refresh();
117         }
118         this->background_view.refresh();
119         for (auto _view : this->views) {
120             _view->refresh();
121         }
122         this->drifting_view.refresh();
123         _sleep(sleep_time);
124     }
125 };
126
127 Status main_simple() {
128     Simple simple;
129     std::function<Status()> _main = std::bind(&Simple::_main,
    ↪ &simple);
130     std::function<void()> refresh = std::bind(&Simple::refresh,
    ↪ &simple);
131     return mainloop(_main, refresh, total_time);
132 }
133
134 Status main_multi() {
135     Multi multi(window_num, probability_num, speed, max_money);
136     std::function<Status()> _main = std::bind(&Multi::_main, &multi);
137     std::function<void()> refresh = std::bind(&Multi::refresh,
    ↪ &multi);
138     return mainloop(_main, refresh, total_time);
139 }
```

(7) main.cpp

```cpp
// @Time    :   2023-10-23 19:40:03
// @FileName:   main.cpp
// @Author  :   423A35C7
// @Software:   VSCode

#include "MVC.h" // 应该是 cmake 里不应该重复 include 一遍
using namespace std;

// 以下的是通过向类传递参数设置的
int probability_num = DEFAULT_PROBABILITY_NUM; // 每个时刻有 1/probability_num 的概率来人
int speed = DEFAULT_SPEED;                     // 每个窗口办理的速度
int_ total_time = DEFAULT_TOTAL_TIME;          // 总时刻数
int max_money = MAX_MONEY;                     // 最大携带金额
int window_num = DEFAULT_WINDOW_NUM;           // 柜台数量
double walk_speed = DEFAULT_WALK_SPEED;          // 人的走路速度

// 以下的是通过全局变量设置的
int sleep_time = DEFAULT_SLEEP_TIME; // 每次刷新间隔多少毫秒
int base_x = DEFAULT_BASE_X;         // 起始位置距离终端上边几个字符的距离
int base_y = DEFAULT_BASE_Y;         // 起始位置距离终端左边几个字符的距离
int sep = DEFAULT_SEP;               // 每个窗口间隔多少距离
int gate_x = DEFAULT_GATE_X;         // 大门的位置终端上边几个字符的距离
int gate_y = DEFAULT_GATE_Y;         // 大门的位置终端左边几个字符的距离
unsigned seed = DEFAULT_RANDOM_SEED; // 随机数种子

template <typename T>
Status get_input(T &variable, string prompt) {
    cout << " 请输入" << prompt << " (默认值" << variable << "):  ";
    while (true) {
        if (cin >> variable) {
            break;
        }
        cin.clear();
        cin.ignore(2048, '\n');
        cout << " 输入错误, 请重新输入: ";
    }
```

```
37      return OK;
38  }
39
40  int main(int, char **) {
41      // std::cout << "Hello, from 第三章作业!\n";
42      srand(seed);
43      int mode = 2;
44      get_input(mode, " 模式，1 为单个队列，2 为多个队列，3 为自定义，其他
        ↪  整数表示退出");
45      switch (mode) {
46      case 1:
47          return main_simple();
48          break;
49      case 2:
50          return main_multi();
51          break;
52      case 3:
53          get_input<int>(probability_num, " 每个时刻有 1/{输入值} 的概率
            ↪  来人");
54          get_input<int>(speed, " 每个窗口办理的速度");
55          get_input<int_>(total_time, " 总时刻数");
56          get_input<int>(max_money, " 最大携带金额");
57          get_input<int>(window_num, " 柜台数量");
58          get_input<double>(walk_speed, " 人的走路速度（浮点数）");
59          get_input<int>(sleep_time, " 每次刷新间隔多少毫秒");
60          get_input<unsigned>(seed, " 随机数种子");
61          srand(seed);
62          return main_multi();
63          break;
64      }
65      return OK;
66  }
```

(8) CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.0.0)
2  project(第三章作业 VERSION 0.1.0 LANGUAGES C CXX)
3
4  include(CTest)
5  enable_testing()
6
```

```
7  aux_source_directory(./src SRC_LIST)
8  # aux_source_directory(./test TEST_LIST)
9  include_directories(./include)
10
11 add_executable(chapter3 ${SRC_LIST}) # 它好像是在这里添加了某个文件才会
   ↪  不给这个文件报错
12 if (CMAKE_BUILD_TYPE STREQUAL Debug)
13     add_executable(test_model ./test/test_model.cpp)
14     add_executable(test_view ./test/test_view.cpp)
15     add_executable(test_controller ./test/test_controller.cpp)
16
17     add_test(
18         NAME test_model
19         COMMAND $<TARGET_FILE:test_model>
20     )
21
22     add_test(
23         NAME test_view
24         COMMAND $<TARGET_FILE:test_view>
25     )
26
27     add_test(
28         NAME test_controller
29         COMMAND $<TARGET_FILE:test_controller>
30     )
31
32     add_definitions(-D_DEBUG)
33     # add_definitions(-D_HAS_STD_BYTE=0) 这个好像没用
34 endif()
35
36 if (CMAKE_BUILD_TYPE STREQUAL Release)
37     # 也可以 set(CMAKE_CXX_FLAGS_RELEASE ...)
38     set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall
       ↪  -fexec-charset=GBK")
39 endif()
40
41 set(CPACK_PROJECT_NAME ${PROJECT_NAME})
42 set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
43 SET(EXECUTABLE_OUTPUT_PATH R:/)
44 include(CPack)
45
46 # 增加了安装的配置还是可能会有找不到库或者库不对的问题,
```

```
47 # 目前需要...\mingw64\bin 里的 libstdc++-6.dll、libgcc_s_seh-1.dll、
   ↪ libwinpthread-1.dll 这三个文件，库的版本不对也可能运行出错，手动从编
   ↪ 译并运行成功的设备上复制这三个文件到目标设备上和 exe 文件同一个目录
   ↪ 中是可以运行的
48 install(CODE [[
49 file(GET_RUNTIME_DEPENDENCIES
50     RESOLVED_DEPENDENCIES_VAR RESOLVED_DEPS
51     UNRESOLVED_DEPENDENCIES_VAR UNRESOLVED_DEPS
52     # EXECUTABLES $<TARGET_FILE:chapter3> # 这样总是会出错
53     # file Failed to run dumpbin on:
54     # $<TARGET_FILE:chapter3>
55     EXECUTABLES R:/chapter3.exe # 所以只能改成这样
56     DIRECTORIES $<TARGET_FILE_DIR:chapter3>
57     PRE_INCLUDE_REGEXES $<TARGET_FILE_DIR:chapter3>
58     PRE_EXCLUDE_REGEXES "system32"
59     POST_INCLUDE_REGEXES $<TARGET_FILE_DIR:chapter3>
60     POST_EXCLUDE_REGEXES "system32"
61 )
62 foreach(DEP_LIB ${RESOLVED_DEPS})
63     file(INSTALL ${DEP_LIB} DESTINATION R:/bin)
64 endforeach()
65 ]])
```