**重庆交通大学信息科学与工程学院**

**课程设计报告**

|  |  |
| --- | --- |
| 题 目 | 智能餐厅点餐结算系统设计报告 |
| 课程名称 | 程序设计综合实践 |
| 专业班级 | 电子信息专业（类）24级 4班 |
| 姓 名 | 谢东东、许肖、周新城 |
| 指导教师 | 何立兰 |

2024年12月

**学生基本信息管理系统**

**设**

**计**

**说**

**明**

**书**

**2023年12月**

目录

[第一章 系统分析 1](#_Toc14463)

[1.1 系统概述 1](#_Toc18299)

[1.2 系统使用角色及用户分析 1](#_Toc419)

[1.3 系统业务流程分析 2](#_Toc6215)

[1.4 项目团队成员分工 3](#_Toc21940)

[第二章、系统设计 5](#_Toc26446)

[2.1 系统功能设计 5](#_Toc11024)

[2.2 程序结构与数据结构设计 8](#_Toc13193)

[第三章 系统实现及测试 9](#_Toc9875)

[3.1系统实现 9](#_Toc20741)

[3.2测试 9](#_Toc3119)

[第四章 总结 10](#_Toc7070)

[4.1制作流程与挑战 10](#_Toc6461)

[4.2反思 12](#_Toc4236)

[4.3、不足与提高 14](#_Toc15948)

[第五章 致谢 15](#_Toc3443)

[附录 代码 16](#_Toc7293)

[main.cpp 16](#_Toc10448)

[head.h 16](#_Toc2735)

[关于本系统.h 22](#_Toc30462)

[厨师.h 23](#_Toc18687)

[厨师.main 28](#_Toc26659)

[厨师结构体.h 29](#_Toc32112)

[服务员main.h 30](#_Toc3783)

[服务员上菜.h 31](#_Toc10940)

[服务员添加菜品.h 37](#_Toc23522)

[顾客.h 45](#_Toc6797)

[顾客main.h 46](#_Toc1191)

[结构体.h 51](#_Toc16057)

[特殊需求.h 53](#_Toc10069)

[投诉与建议.h 53](#_Toc26938)

[小票.h 55](#_Toc12239)

[选位子.h 59](#_Toc30961)

[餐桌管理.h 62](#_Toc16854)

[查看统计数据.h 68](#_Toc18808)

[获取修改菜单.h 73](#_Toc24077)

[调试功能模块.h 82](#_Toc26255)

[管理员.h 83](#_Toc24630)

[账号管理.h 85](#_Toc18409)

[老板main.h 86](#_Toc10315)

[收营员main.h 87](#_Toc16481)

[收营员结算小票.h 89](#_Toc3420)

[directory.h 99](#_Toc25571)

[读取文件.h 100](#_Toc28761)

[密码.h 102](#_Toc394)

[排序,h 104](#_Toc13871)

[输入判断.h 105](#_Toc31683)

[展示全部文件.h 106](#_Toc27628)

# 第一章 系统分析

## 1.1 系统概述

在餐饮行业数字化转型的背景下，本餐厅点餐结算系统旨在提升餐厅运营效率和顾客体验。通过整合点餐、结算、菜品管理、人员协作等功能，实现餐厅业务流程的信息化管理，降低人工成本，提高服务质量，增强餐厅竞争力。系统具备以下特性：

1. 高效性：优化点餐和结算流程，减少顾客等待时间，提高餐厅翻台率。

2. 准确性：精确计算菜品价格、库存管理和财务统计，避免人为错误。

3. 实时性：实时更新菜品状态、订单信息和统计数据，支持餐厅老板和管理人员及时决策。

4. 易用性：简洁直观的用户界面，方便各类用户操作，降低培训成本。

## 1.2 系统使用角色及用户分析

1. 消费者：追求便捷的点餐体验，希望能够快速浏览菜单、选择心仪菜品、预订餐位，并获取准确的消费信息。他们注重系统的操作简便性和界面友好性。2. 收银员：需要高效准确地处理结算业务，包括计算消费金额、处理多种支付方式、打印结算凭证，并及时统计当天的结算数据。他们依赖系统的稳定性和计算准确性。

3. 厨师：关注菜品制作流程的有序性，能清晰查看待做菜品并及时更新状态，同时希望方便查询已完成菜品信息，以合理安排工作节奏。

4. 服务员：要求系统能辅助记录服务过程，快速响应顾客加菜等需求并准确更新相关信息，确保服务的及时性和准确性。

5. 餐厅老板：需要全面掌握餐厅运营状况，通过系统获取各类数据报表，进行深入分析，以制定经营策略和管理决策。他们重视数据的完整性和分析功能的强大性。

6. 系统管理员：负责系统的基础设置和维护，确保系统正常运行，包括管理用户权限、更新菜品和餐位信息等，需要系统具备良好的管理界面和操作便利性。

**1.3 系统业务流程分析**

1. 消费者点餐流程

- 消费者进入餐厅，可选择现场点餐或预订餐位。若现场点餐，系统自动分配餐位（预订则按指定时间和餐位安排），消费者浏览菜单后选择菜品及数量，系统生成包含点餐详情、总价、餐位号等信息的小票。

- 用餐过程中，若顾客需增加菜肴或酒水，服务员接到需求后在系统中为该顾客的小票添加相应内容。

2. 结算流程

- 消费者用餐完毕前往收银台结账。收银员扫描小票，系统自动计算消费金额（仅计算“已出菜”菜品），若消费不足500元，自动添加餐位费。

- 收银员确认金额后，消费者选择支付方式完成付款，系统记录支付信息，打印结算凭证，同时更新当天结算统计数据。

3. 厨师做菜流程

- 厨师登录系统，按顺序查看未完成的小票，根据小票信息制作菜品。

- 菜品制作完成后，厨师在系统中将其状态更新为“已做”，以便服务员上菜和后续结算。

4. 服务员服务流程

- 服务员在服务过程中，可随时查看顾客小票，记录服务事项，如上菜时间、顾客特殊需求等。

- 当菜品端上桌后，服务员及时在系统中将菜品状态改为“已出菜”。

5. 餐厅老板查询流程

- 餐厅老板登录系统，可查看所有用户的操作记录，包括消费者点餐详情、收银员结算记录、厨师做菜进度、服务员服务记录等。

- 老板可进行各类统计查询，如按时间段统计菜品销售情况、分析热门菜品、查询不同餐位使用频率等，为经营决策提供数据支持。

6. 系统管理员管理流程

- 系统管理员负责维护系统基础数据，包括添加、修改、删除餐位信息，更新菜肴和酒水种类、价格、库存等。

- 管理员可管理用户信息，如添加新用户、修改用户权限、重置用户密码等，确保系统安全和正常运行。

1.4 项目团队成员分工

1. 谢东东：负责系统整体架构设计、用户管理模块和系统管理模块的开发。包括设计数据库结构、搭建系统框架、实现用户注册登录及信息管理功能，以及管理餐位、菜肴、酒水等基础数据和用户权限。同时，参与数据结构设计和算法设计，确保系统的稳定性和安全性。要负责整个小组的统筹协调工作，同时要负责代码部分的协调与领导工作。

2. 许肖：承担点餐管理模块和结算管理模块的开发工作。实现菜品展示、点餐操作、餐位分配、结算功能以及相关统计功能。负责与数据库的交互操作，确保数据的准确存储和读取。参与系统测试，对功能进行优化和完善。同时要负责项目报告书的协调和领导工作。

3. 周新城：专注于厨师操作模块、服务员服务模块和餐厅老板查询模块的开发。实现厨师做菜流程、菜品状态更新、服务员服务记录与菜品添加、餐厅老板各类查询功能的开发。协助进行界面设计，提高系统的用户体验。要负责ppt的优化与制作ppt的协调工作。

**1.5 系统业务功能及数据分析**

1. 消费者功能需求

- 点餐功能：消费者能够浏览详细的菜单，包括菜品名称、价格、图片、简介、分类等信息，方便选择心仪菜品。可自由选择菜品数量，支持多种点餐方式，如按菜品分类浏览点餐、搜索菜品名称点餐、查看推荐菜品点餐等。点餐过程中能实时查看已选菜品列表及总价，方便调整点餐内容。

- 预订功能：消费者可提前预订餐位，选择预订日期、时间、用餐人数，系统自动分配可用餐位并提供预订确认信息，同时可在预订时间前修改或取消预订。

- 信息查看功能：消费者能随时查看自己的点餐历史记录、消费明细，方便了解个人消费情况。

2. 收银员功能需求

- 结算功能：准确计算消费者消费金额，自动识别“已出菜”菜品进行结算，对消费不足500元的订单自动添加餐位费，支持多种支付方式，如现金、银行卡、移动支付等，并能快速打印结算凭证。

- 统计功能：实时统计当天的结算信息，包括总营业额、订单数量、不同支付方式的金额分布、各时段营业额等，为财务管理提供数据支持。同时统计当天“待做”和“已做”菜品数量及金额，协助餐厅掌握菜品制作进度和销售情况。

3. 厨师功能需求

- 做菜流程管理功能：按照系统提示的小票顺序制作菜品，确保先点先做。完成菜品制作后，能方便快捷地在系统中将菜品状态更新为“已做”，避免误操作。

- 菜品查询功能：可查看当天已完成的所有菜品记录，包括菜品名称、制作时间、所属订单等信息，便于回顾工作和统计个人工作量。

4. 服务员功能需求

- 服务记录功能：在为顾客服务过程中，记录服务时间、服务内容、顾客特殊需求等信息，便于后续服务质量跟踪和管理。

- 菜品添加功能：根据顾客要求，准确将增加的菜肴或酒水信息录入系统，并更新相应订单的总价和菜品列表，确保信息同步。

- 菜品状态更新功能：及时将已端上桌的菜品状态更新为“已出菜”，确保结算和厨师操作的准确性。

5. 餐厅老板功能需求

- 信息查看功能：全面查看餐厅所有运营信息，包括消费者点餐详情、收银员结算记录、厨师做菜进度、服务员服务记录等，实时掌握餐厅动态。

- 统计分析功能：按不同维度进行统计分析，如按时间段（日、周、月、季、年）统计菜品销售数量和金额，分析菜品销售趋势；查询不同类别菜品（热菜、凉菜、主食、饮料等）的销售占比，了解菜品受欢迎程度；统计不同餐位的使用频率，为餐厅布局和资源配置提供参考；分析消费者消费习惯（人均消费、消费时间段、消费菜品偏好等），以便制定营销策略。

6. 系统管理员功能需求

- 基础数据管理功能：维护餐位信息，包括添加、删除、修改餐位编号、座位数量、位置等属性；管理菜肴和酒水信息，如添加新菜品、修改菜品价格、调整库存、上传菜品图片和简介等；设置系统参数，如税率、折扣规则、营业时间段等。

- 用户管理功能：添加、删除、修改用户账号信息，设置用户角色和权限，确保不同用户只能访问和操作其权限范围内的功能。

- 数据备份与恢复功能：定期自动备份系统数据，防止数据丢失。在系统出现故障或数据错误时，能快速恢复数据至最近备份状态，保障系统正常运行。

第二章、系统设计

2.1 系统功能设计

1. 用户管理子系统

- 注册与登录模块：实现不同用户角色的注册和登录功能，对用户输入的信息进行验证，确保信息准确有效。登录后根据用户角色显示相应的操作界面，限制用户访问未授权功能。

- 信息管理模块：消费者可修改个人信息，如联系方式、地址等；系统管理员能够管理所有用户信息，包括添加新用户、删除用户、修改用户权限等操作，保障系统用户数据的安全性和准确性。

2. 点餐管理子系统

- 菜单展示模块：以清晰、美观的界面展示菜品和酒水信息，包括名称、价格、图片、简介、分类等，支持多种展示方式，如列表展示、图片展示、分类展示等，方便消费者浏览和选择。

- 点餐操作模块：实现消费者点餐的核心功能，包括选择菜品、确定数量、添加到购物车、选择堂食或外卖、确认订单等操作。点餐过程中实时计算总价，提供订单预览功能，让消费者确认点餐内容。

- 餐位分配模块：对于堂食订单，根据餐厅当前餐位使用情况自动分配空闲餐位，可采用随机分配或按照一定规则分配（如优先分配靠窗、安静区域等）。支持餐位预订功能，消费者可提前选择预订时间和餐位，系统自动记录并在预订时间到达时为其保留餐位。

3. 结算管理子系统

- 结算计算模块：收银员扫描小票后，系统自动计算消费金额，仅计算“已出菜”菜品价格。若消费金额不足500元，按照设定规则添加餐位费，确保结算金额准确无误。

- 支付处理模块：支持多种支付方式，如现金、银行卡、微信支付、支付宝支付等，与相应支付接口对接，实现安全、快捷的支付功能。支付成功后，系统自动更新订单状态为“已支付”，并记录支付信息。

- 统计报表模块：实时生成并展示当天的结算统计报表，包括总营业额、订单数量、不同支付方式的金额分布、各时段营业额等图表和数据。同时统计当天“待做”和“已做”菜品数量及金额，为餐厅管理人员提供详细的财务和运营数据。

4. 厨师操作子系统

- 做菜管理模块：厨师登录后，系统按顺序展示未完成的小票信息，包括菜品名称、数量、制作要求等。厨师完成菜品制作后，点击相应按钮即可将菜品状态更新为“已做”，系统自动通知服务员上菜。

- 已完成菜品查询模块：厨师可查看当天已完成的所有菜品记录，包括菜品名称、制作时间、所属订单等信息，便于厨师了解自己的工作进度和工作量，同时也方便餐厅管理人员对厨师工作进行监督和评估。

5. 服务员服务子系统

- 服务记录模块：服务员在服务过程中，通过系统记录服务事项，如上菜时间、顾客特殊需求、服务满意度等信息。这些记录有助于餐厅提高服务质量，对服务员工作进行评估和管理。

- 菜品添加模块：当顾客要求增加菜肴或酒水时，服务员使用此模块将新增菜品信息录入系统，选择对应的订单号，输入菜品名称、数量等信息，系统自动计算并更新订单总价和菜品列表，确保信息及时准确传递给厨房和收银台。

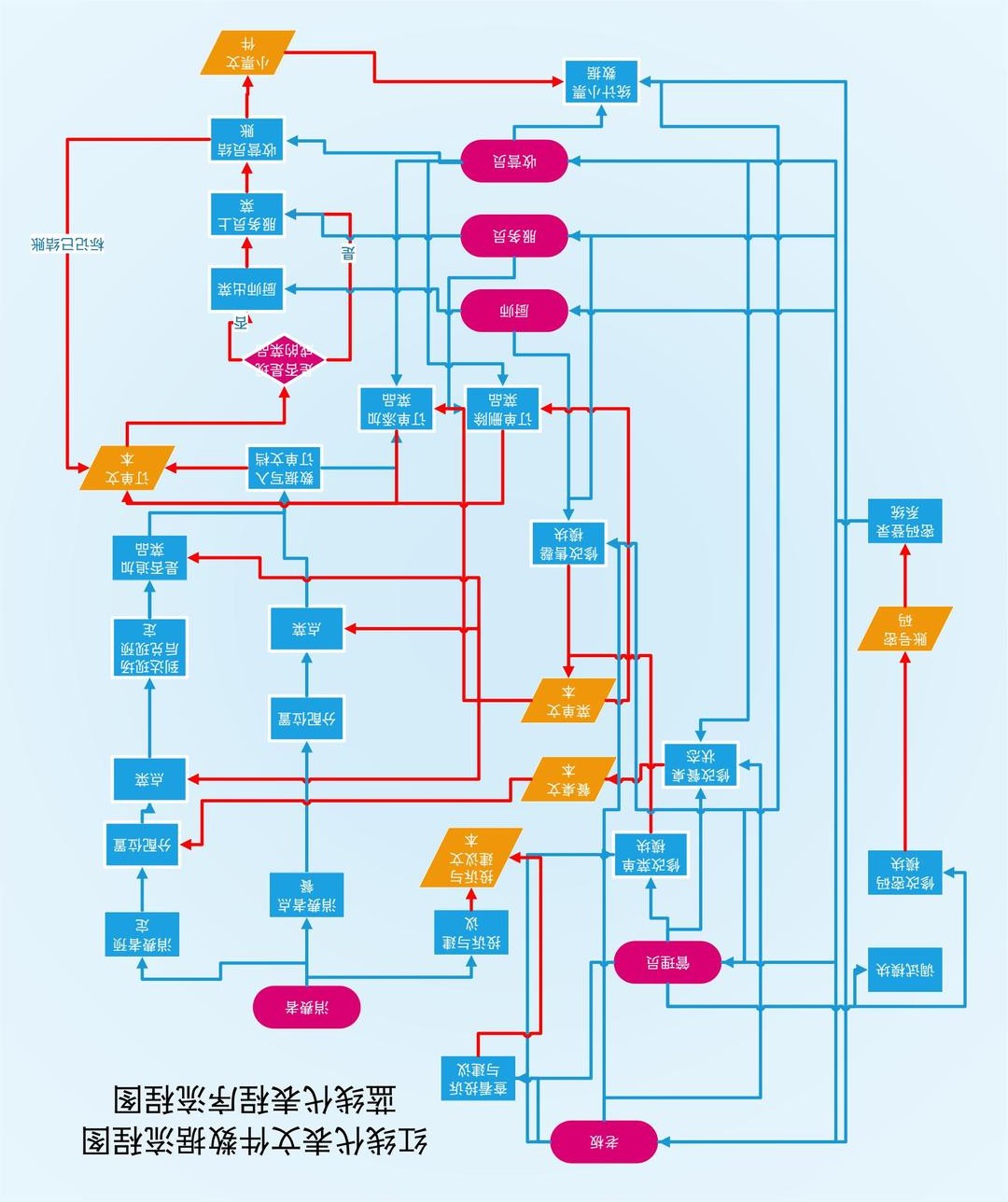
- 菜品状态更新模块：服务员将菜品端上桌后，及时在系统中将菜品状态更新为“已出菜”，触发相关结算和统计功能，保证餐厅各环节数据的一致性和准确性。

6. 餐厅老板查询子系统

- 信息查看模块：餐厅老板可以查看系统中所有用户的操作记录，包括消费者点餐详情、收银员结算记录、厨师做菜进度、服务员服务记录等，全面了解餐厅运营状况，及时发现问题并做出决策。

- 统计分析模块：提供丰富的统计分析功能，如按时间段统计菜品销售数量和金额。

## 2.2 程序结构与数据结构设计



程序的运行流程从消费者的操作开始，消费者可以通过系统进行预定。预定完成后，消费者到达现场，兑现预定并开始点菜。点菜过程中，消费者可以选择直接选定菜品，也可以在后续追加菜品。点菜信息会实时传递到系统，并分配给厨师进行出菜。服务员根据厨师完成的菜品准备情况，上菜至消费者的餐桌。

在消费者就餐结束后，收营员负责结账操作，标记订单为已完成，并打印小票作为消费凭证。小票信息同时被存储于系统文件中，以供后续统计和查询。

管理员负责系统的整体管理与维护。管理员可以修改菜单内容，标记售罄菜品，以及处理用户通过系统提交的投诉与建议。管理员还可以查看小票统计数据，以分析营业情况。此外，餐桌状态的管理（如预定、占用或空闲）也由管理员负责更新。

文件操作的流程与上述程序流程紧密相关。系统以文件的形式管理多种数据，包括菜单文本、消费者订单文本、小票文件、投诉与建议文本，以及餐桌状态文本。菜单文件用于提供实时菜单信息，消费者订单文件记录预定和点菜信息，小票文件保存每笔交易的详细记录，投诉与建议文件用于存储反馈内容，而餐桌文件则用于动态管理餐桌状态。这些文件间的关联与操作流程相辅相成，确保系统功能的完整性和数据的实时更新。

# 第三章 系统实现及测试

3.1系统实现

以消费者点餐模块为例，消费者进入点餐界面，系统展示菜品分类和推荐菜品，消费者可点击菜品查看详情，选择菜品及数量后加入购物车，在购物车中可修改菜品信息或删除菜品，确认无误后提交订单，系统自动分配餐位，生成订单记录，关联消费者信息、菜品信息和餐位信息，推送订单至厨房制作，并实时更新订单状态。消费者可在订单列表中查看订单状态，用餐结束后进行评价。

3.2测试

1. 单元测试：对各个功能模块进行单独测试，如消费者点餐功能测试，包括菜品选择、数量修改、购物车操作、订单提交、餐位分配等功能点；订单管理功能测试，验证订单状态更新、查询订单详情、评价订单等操作的正确性；菜品管理功能测试，检查菜品添加、修改、删除、上下架等操作的有效性。针对每个功能点设计详细测试用例，确保模块功能的准确性。

2. 集成测试：测试各模块之间的接口和交互，重点关注订单流程中消费者点餐、服务员处理额外需求、厨房制作菜品、收银员结算等环节的数据传递和状态更新是否正确。例如，测试消费者下单后，订单信息是否准确无误地传递到厨房和收银员端，厨房制作完成菜品后，订单状态是否及时更新并反馈给消费者和服务员，服务员处理额外需求后，相关信息是否正确更新到订单和库存管理模块。

3. 系统测试：进行全面的系统测试，包括功能测试、性能测试、兼容性测试、安全测试等。功能测试覆盖系统所有功能模块，确保系统在各种正常和异常情况下的功能正确性；性能测试模拟高并发场景，测试系统在大量消费者同时点餐、查询订单等操作时的响应时间、吞吐量等性能指标，确保系统能够稳定运行；兼容性测试检查系统在不同操作系统、浏览器、移动设备上的显示和操作是否正常；安全测试检查系统在用户认证、数据传输、存储等方面的安全性，防止数据泄露和非法操作。

测试过程中，针对每个测试用例详细记录测试目的、预置条件、操作步骤、输入数据、期望结果、执行结果。对发现的缺陷及时提交给开发团队进行修复，修复后进行回归测试，确保问题得到彻底解决，保证系统质量和稳定性。

# 第四章 总结

4.1制作流程与挑战

在团队协作过程中，我们遇到了许多困难与挑战，这些问题促使我们不断学习和改进。从一开始，我们小组计划使用TXT文件进行数据读取，但由于编码问题导致读取文件时出现乱码。这期间，肖皓轩指出了问题的根源是编码问题，而涂智强帮助我们调整了编译器的编码格式，尽管这一方案最终没有被采用。小组尝试了多种解决方案，包括涂智强建议的更改项目编码、许肖提出的直接在代码区设置编码格式以及周新城提议的更改文件编码。经过讨论，我们最终决定统一采用ANSI编码格式，在这种编码标准下中文能够以GB2312正常输出，同时也是编译器默认支持的编码方式。

在项目的早期阶段，我们在代码框架上也遇到了问题。一开始按照多文件的思路去开发，但很快发现这种方法在实际操作中存在局限性。幸运的是，我们及时发现了问题并重构了前期代码。在这一过程中，王家伟老师、涂智强、葛昌威、陈龙、赖子豪、胡明哲、郭叶鹏和简豪在群内的讨论中为我们指出了问题所在，避免了小组陷入更深的困境。我们还参考了周敬人的代码，重新设计了项目架构。重构的过程对组长谢东东的心理造成了一定影响，但在与智慧城市学院的王净萍副书记交流后，她获得了极大的心理支持和鼓励。

在代码逻辑处理上，我们也遇到了一些难题。例如，谢东东提出要处理输入非法字符的问题。如果不加以控制，非法字符输入可能导致程序陷入死循环。为了解决这一问题，我们设计了循环处理逻辑，非法输入会提示用户重新输入。然而，这种方法最初因为输入流的异常而失败，是涂智强指出了输入流损坏的原因。随后，周新城通过学习输入流的清理和重置，编写了一个名为int\_input的函数，成功解决了这个问题。

在项目开发中，我们遇到部分C++函数被限制使用的问题，而小组又希望实现类似C++ sort函数的功能。在何立兰老师的建议下，谢东东通过指针实现了这一功能，并通过学习模板函数使代码能够灵活接收不同类型的参数，从而实现了类似sort的功能。

为了提高效率，我们尝试使用AI生成代码。许肖尝试了豆包，谢东东使用了通义千问和GPT-4，但生成的代码都不太令人满意。小组总结认为，AI生成短小代码或修正错误时表现较好，但在处理大型项目时，生成代码往往存在问题。此外，AI生成的代码长度有限，分段生成的代码质量也较差，且易出现命名冲突。最终，我们决定仅将AI用于代码规范、命名以及编写重复性代码，而不再依赖其生成大段逻辑。

文件管理方面，我们决定将订单和小票数据分别存储在两个文件中。文件的遍历曾是一个难点，小组提出了多种方案，例如使用第三方库或将文件名存入TXT文件，但最终采用了数字命名的方式，通过遍历数字命名的文件来实现目标。我们还从最初使用Dev-C++转为VS 2022开发环境，通过解决方案资源管理器管理文件夹，提升了代码可读性。尽管谢东东起初反对使用中文命名文件，但鉴于大家的编译器均支持中文，他最终同意了这一方案。

在文件管理中，我们发现VS 2022生成的项目文件过于庞大，代码分享时发现项目体积高达0.8GB。经葛昌威和涂智强建议，我们清理了.vs和X64文件，成功将文件体积降至正常水平。

由于小组成员各自负责两个模块，初期出现了命名冲突的问题。为此，我们决定使用命名空间解决这一问题。尽管是第一次使用非全局命名空间，我们也因此犯了一些错误，例如在定义命名空间时错误地加上了分号。葛昌威及时指出了这些问题，使我们得以及时改正。

通过项目开发，我们深刻体会到实际开发与算法竞赛及学习之间的差异。在这个过程中，我们不仅需要妥善调配组员任务，还要考虑非预期情况的发生。尽管遇到了种种困难，我们积极寻求帮助并不断克服问题，最终完成了整个项目代码的开发。

4.2反思

在完成餐厅管理系统的开发过程中，我们深刻感受到实际开发项目与学习代码、竞赛代码之间的巨大差异。这种差异不仅体现在解决问题的方式上，也体现在思维的转变和团队协作的重要性上。通过这次项目开发，我们积累了许多经验，同时也认识到理论学习与实际开发相结合的重要性。

最初的计划往往难以完全落实。在学习中，代码逻辑通常是清晰的，任务是单一的，工具和环境基本上不会成为障碍。然而，在实际开发中，我们从项目启动阶段就遇到了诸多问题，比如编码格式导致的文件读取乱码、框架设计不合理导致的代码重构、输入非法字符引发的程序异常等。这些问题让我们意识到，实际开发中需要不断调整方案，并随时应对意料之外的情况。

这次项目让我们学会了在实践中灵活应用知识。比如，虽然在课程中我们学过指针和模板函数，但真正将这些技术用于实现类似sort的功能时，依然经历了大量尝试和失败。同样，输入流的清理和重置功能虽然简单，但需要深入理解输入流的工作机制才能实现。当面对陌生问题时，我们通过上网查阅资料、请教老师和同学，以及小组内部反复讨论，最终成功解决了许多困难。这些过程让我们明白，学习不仅限于课堂，而是要在实际项目中加以验证和完善。

此外，这次开发也改变了我们对工具和平台的认知。早期我们使用Dev-C++，后来发现随着项目规模增大，它难以胜任需求。转而使用VS 2022后，我们体会到大型开发工具的优势，同时也要学会管理冗余文件、优化项目结构。这种选择工具的经验是课本和竞赛中难以学到的，却是实际开发中的必备能力。

团队协作的重要性也在这次项目中得到了充分体现。代码开发中，命名冲突、功能模块分配不均等问题频繁出现，我们最终通过使用命名空间和明确职责分工得以解决。每位成员都发挥了独特的作用，不论是技术上的帮助，还是心理上的支持，都对项目的顺利完成起到了关键作用。这也让我们认识到，项目开发不仅是技术上的比拼，更是团队协作和沟通能力的综合体现。

最重要的是，这次项目让我们认识到理论学习和实际开发的结合才是编程能力提升的关键。学习中的代码往往结构清晰、目标单一，而竞赛代码则强调效率和算法优化。但在实际开发中，我们需要兼顾可维护性、可扩展性以及用户体验，同时还要应对各种可能的异常情况。这种从学习到实践的转变，让我们更加理解编程的本质：通过代码解决实际问题。

通过这次项目，我们不仅完成了功能完善的餐厅管理系统，更在实践中锻炼了自己，也收获了许多宝贵的经验。未来，我们将继续在实践中学习，将理论与实际紧密结合，不断提高自己的能力，为更复杂的开发任务做好准备。

通过本次餐厅管理系统的开发，我们对AI的使用有了更深入的认识。AI在处理重复性任务和文字性工作方面表现出色，例如规范函数命名、编写注释以及生成简单代码片段。然而，当涉及到项目的核心逻辑和复杂功能时，AI生成的代码往往难以满足需求。特别是在大型项目中，AI生成的代码可能会缺乏上下文理解，导致逻辑冲突或代码质量下降。因此，我们认识到，AI更适合作为辅助工具，而非完全依赖的解决方案。代码的核心部分仍需要我们亲自编写，以确保逻辑的严密性和项目的整体质量。这次经验也提醒我们，在开发过程中要学会扬长避短，将AI的优势与自身的能力结合，才能更高效地完成项目。

4.3、不足与提高

在完成餐厅管理系统的开发后，尽管项目功能基本实现，但我们也意识到其中存在诸多不足。这些不足不仅是技术层面的限制，更反映了我们对实际应用需求的理解还有待深入。这次反思旨在梳理问题并为未来的开发指明改进方向。

首先，项目没有采用GUI库，导致用户界面仅限于终端输出。虽然终端操作简单直观，但从用户体验的角度来看，这种方式过于单调且不够美观，不利于实际应用场景中的操作体验。在真实的餐厅管理中，图形化界面能够更好地展示信息，提供快捷的交互方式，同时也更符合用户的直觉习惯。未来，如果有机会优化项目，我们计划学习并引入GUI库（如Qt或wxWidgets），设计一个更友好的用户界面，让系统在操作和视觉上更贴近实际需求。

其次，系统数据存储采用的是TXT文件操作，这种方式虽然简单，但在实际应用中存在很大局限性。一方面，文件操作效率低下，特别是在数据量较大的情况下会影响性能；另一方面，数据的维护和更新较为麻烦，也缺乏灵活的查询和统计功能。相比之下，使用数据库管理系统（如MySQL或SQLite）能更好地解决这些问题。数据库可以支持高效的数据存储和检索，支持复杂的操作和并发管理，同时也便于扩展。因此，未来的版本中，我们应当学习并引入数据库技术，将数据管理提升到更高的水平。

此外，目前的系统只支持单程序运行，缺乏对多用户并发操作的支持。这显然与实际餐厅的需求不符。现实中的餐厅管理系统需要同时处理多个终端的请求，例如前台点餐、后厨备餐和后台统计等多个功能模块并行运行。我们在设计时忽略了这一点，导致系统在功能上有较大的局限性。未来需要学习多线程编程或分布式系统的相关技术，提升系统对多用户环境的支持能力。

最后，账户管理部分设计得过于简单，仅满足了基本的登录和权限管理需求，但没有实现更加精细化的功能。在实际应用中，账户管理不仅需要支持多级权限控制，还可能涉及日志记录、安全认证等功能，以确保系统的安全性和可追溯性。我们在这一部分的功能设计上需要更加细致和全面。

**第五章 致谢**

在完成这次餐厅管理系统的开发过程中，我们经历了许多困难与挑战，但也收获了宝贵的经验和成长。在此，我们谨向所有帮助过我们的人致以诚挚的感谢。首先，感谢我们的指导老师何立兰老师，在我们遇到功能实现上的难题时，她耐心地给予了指导，并提出了很多宝贵的建议，帮助我们掌握了如指针与模板函数等高级技术的使用，为项目开发奠定了坚实的基础。

其次，感谢在项目开发过程中给予我们帮助的涂智强学长、葛昌威学长、陈龙学长、赖子豪同学、胡明哲学长、郭叶鹏学长、简豪学长、肖皓轩同学、王家伟老师、张微女士、王净萍副书记。它们在它们在问题定位、编码解决方案、框架调整、工具优化、心理帮助等方面提出的意见和建议或者帮助，让我们能够及时纠正方向，克服困难，避免项目陷入更大的问题。

特别感谢涂智强学长对我们小组提供的的帮助，帮助我们小组改了编译器格式，解答命名空间分配相关问题，优化解决方案资源管理器，清理输入流，以及解答我们小组提出的各种问题。

同时，我们要感谢每一位组员的努力与付出。

这次项目开发虽然充满挑战，但在老师、同学以及团队成员的共同努力下，我们克服了一个个难题，最终完成了系统开发。再次向所有帮助过我们的人致以衷心的感谢！

附录 代码

main.cpp

#include"head.h"//需要C++17以及以上

using namespace std;

int main()

{

directory();

return 0;

}

//终于写完了，太好了！！！！

head.h

#pragma once

#pragma comment(linker, "/STACK:52428800,4096")

#include <fstream>

#include <filesystem>

#include <iomanip>

#include <iostream>

#include<algorithm>

#include <sstream>

#include <string>

#include <conio.h>

#include<windows.h>

const int MAX\_MENU\_ITEMS = 150;

const int MAX\_NUMBER = 50;

#include"顾客/结构体.h"

#include"厨师/厨师结构体.h"

#include"头文件/展示全部文件.h"

#define 暂停 system("pause")

#define 清屏 system("cls")

namespace boss

{

void boss();

}

namespace cook {

int readOrderFile(const std::string& filepath, const std::string& order\_number, cook::Dish dishes[], int start\_index);

void updateOrderStatus(const std::string& filepath, const cook::Dish& dish);

int cooker\_cook();

void cook();

}

namespace server {

int readWaiterFile(const std::string& filepath, cook::Dish dishes[], int start\_index);

int readWaiterFile(const std::string& filepath, cook::Dish dishes[], int start\_index, const std::string& order\_number);

void markDishAsCompleted(const std::string& filepath, const cook::Dish& dish);

int waiter\_serve();

void server();

void displayAvailableOrders(const std::string& order\_directory = "数据存储/订单数据");

void serverAddDishesToOrder(const std::string& order\_file, const MenuItem menu[], int menu\_size);

void serveradd();

void serverDeleteDishFromOrder(const std::string& order\_file);

void serverDelete();

}

namespace costomer {

int updateSerialNumber(const std::string& filename);

int customer();

int getValidQuantity();

int selectMenuItems(MenuItem menu[], int itemCount, Order& order);

string special\_requests\_hot(int n);

string getSpecialRequests(int type);

void saveFeedback();

void calculateTotal(Order& order);

string class\_dishes(int n);

void getUserDetails(Order& order);

void printOrderSummary(const Order& order);

void addMenuItem(Order& order, const MenuItem menu[], int itemCount);

void saveOrderToFile(const Order& order, int order\_number);

bool cmd1(const Table& a, const Table& b);

int loadTablesFromFile(const string& filename, Table tables[], int maxTables);

void saveTablesToFile(const string& filename, Table tables[], int tableCount);

int assignTable(int customerCount, Table tables[], int tableCount, Order& order);

void printAssignedTables(const Order& order);

int select\_table();

int book\_customer();

void book\_customer\_arrival();

}

int loadMenuFromFile(const string& filename, MenuItem menu[]);

string showmenutype(int n);

void showMenu(MenuItem menu[], int itemCount);

void changemunu(const string& filename);

template<typename T, typename Compare>

void bubbleSort(T\* head, T\* tail, Compare comp);

namespace sys

{

struct DishStats {

string name;

int completed\_count;

int pending\_count;

};

using namespace std;

void loadMenu(const string& filename);

void saveMenu(const string& filename);

void displayMenu();

void addMenuItem();

void deleteMenuItem();

int change\_menu();

string removeSpaces(const string& str);

void processFile(const string& filename, DishStats dishStatistics[], int& dishCount, double& totalRevenue);

void analyzeReceipts(const string& folderPath);

int loadMenuFromFile(const string& filename, MenuItem menu[]);

string showmenutype(int n);

void showMenu(MenuItem menu[], int itemCount);

void changemunu(const string& filename);

void loadMenu(const string& filename);

void saveMenu(const string& filename);

void displayMenu();

void addMenuItem();

void deleteMenuItem();

int change\_menu();

void password(string name);

bool checkPassword(const string& username, const string& inputPassword);

// 修改密码

bool changePassword(const string& username, const string& newPassword);

void cpass1(const string& username);//管理员端

void cpass2(const string& username);

}

namespace administrator

{

void test();

void administrator();

void set\_feel();

}

namespace cashier

{

struct Dish {

std::string name;

std::string table\_number;

double price;

int quantity;

std::string special\_requests;

std::string type;

std::string status;

};

void cashier();

void clearFileContent(const std::string& filepath);

void previewReceipt(const Dish dishes[], int count, int customer\_count);

int readWaiterFile(const std::string& filepath, cashier::Dish dishes[], int& start\_index, int& customer\_count);

void generateReceipt(const std::string& outputPath, const Dish dishes[], int count, int customer\_count, const std::string& payment\_method, double cash = 0.0);

void cashier\_recipet();

}

void inputnotright();

#define max\_int 0x7fffffff

int int\_input(int input\_max = max\_int, int input\_min = 0);

void separator(int n);

void directory();

void readallfile(std::string path = "数据存储/订单数据/");

bool cmd1(MenuItem a, MenuItem b);

int readNumberFromFile(const string& filename);

void writeNumberToFile(const string& filename, int newNumber);

namespace about

{

void about();

}

#include"头文件/读取文件.h"

#include"头文件/密码.h"

#include"管理端/建议与投诉.h"

#include"管理端/查看统计数据.h"

#include"管理端/餐桌管理.h"

#include"管理端/查看统计数据.h"

#include"头文件/排序.h"

#include"头文件/输入判断.h"

#include"管理端/获取修改菜单.h"

#include"顾客/顾客\_main.h"

#include"厨师/厨师main.h"

#include"服务员/服务员main.h"

#include"收营员/收营员main.h"

#include"管理员/管理员\_main.h"

#include"老板/老板main.h"

#include"使用帮助/关于本系统.h"

#include"头文件/directory.h"

关于本系统.h

#pragma once

namespace about

{

void about()

{

while(true)

{

cout << "1：致谢" << endl;

cout << "2：使用帮助" << endl;

cout << "3：制作名单" << endl;

cout << "4：后记" << endl;

cout << "0：返回上一级" << endl;

int n = ::int\_input(4);

system("cls");

switch (n)

{

case 0:return;

case 1: ::readFileByLine("使用帮助/致谢.txt"); break;

case 2: ::readFileByLine("使用帮助/使用帮助.txt"); break;

case 3: ::readFileByLine("使用帮助/制作名单.txt"); break;

case 4: ::readFileByLine("使用帮助/后记.txt"); break;

}

暂停;

清屏;

}

}

}

厨师.h

#include"厨师结构体.h"

namespace cook

{

using namespace std;

// 菜品信息结构体

// 从文件读取订单数据

int readOrderFile(const string& filepath, const string& order\_number, cook::Dish dishes[], int start\_index) {

ifstream file(filepath);

if (!file.is\_open()) {

// cerr << "无法打开文件: " << filepath << endl;//有些文件打不开是正常的

return start\_index;

}

string line, table\_number;

while (getline(file, line)) {

if (line.find("桌号:") != string::npos) {

table\_number = line.substr(line.find(":") + 1); // 提取桌号

}

// 读取菜品表头之后的内容

if (line.find("菜品名") != string::npos) {

while (getline(file, line) && !line.empty()) {

stringstream ss(line);

string name, price, quantity, special\_requests, type, status;

// 按固定宽度读取各列

ss >> setw(30) >> name >> setw(10) >> price >> setw(10) >> quantity

>> setw(20) >> special\_requests >> setw(10) >> type >> setw(10) >> status;

// 如果种类是正餐，则保存菜品信息

if (type == "正餐" && start\_index < 500) {

dishes[start\_index++] = { name, table\_number, order\_number, special\_requests, type, status };

}

}

}

}

file.close();

return start\_index;

}

// 更新订单状态

void updateOrderStatus(const string& filepath, const cook::Dish& dish) {

ifstream file(filepath);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filepath << endl;

return;

}

stringstream buffer;

string line;

bool updated = false;

while (getline(file, line)) {

if (line.find(dish.name) != string::npos &&

line.find(dish.status) != string::npos &&

line.find("未完成") != string::npos) {

line.replace(line.find("未完成"), 9, "已出菜");

updated = true;

}

buffer << line << "\n";

}

file.close();

if (updated) {

ofstream output\_file(filepath, ios::trunc);

output\_file << buffer.str();

output\_file.close();

cout << "状态已更新为“已出菜”：订单编号 " << dish.order\_number << "，桌号 " << dish.table\_number

<< "，菜品名 " << dish.name << endl;

}

else {

cout << "未找到需要更新的菜品记录。" << endl;

}

}

int cooker\_cook() {

string folder\_path = "数据存储/订单数据";

string serial\_file = folder\_path + "/序号.txt";

// 读取序号文件，确定订单总数

ifstream serial\_input(serial\_file);

if (!serial\_input.is\_open()) {

cerr << "无法读取序号文件: " << serial\_file << endl;//有些文件打不开是正常的

return 1;

}

int total\_orders;

serial\_input >> total\_orders;

total\_orders--;

if (total\_orders == 0)

{

cout << "暂时没有订单";

return 0;

}

serial\_input.close();

cook::Dish dishes[500]; // 固定大小数组

int dish\_count = 0;

// 遍历所有订单文件

for (int i = 1; i <= total\_orders && dish\_count < 500; ++i) {

string order\_file = folder\_path + "/" + to\_string(i) + ".txt";

dish\_count = readOrderFile(order\_file, to\_string(i), dishes, dish\_count);

}

// 输出所有正餐菜品

cout << left << setw(5) << "序号" << setw(10) << "订单编号" << setw(10) << "桌号"

<< setw(30) << "菜品名" << setw(20) << "特殊要求" << setw(10) << "状态" << endl;

cout << string(85, '-') << endl;

for (int i = 0; i < dish\_count; ++i) {

cout << left << setw(5) << (i + 1) << setw(10) << dishes[i].order\_number

<< setw(10) << dishes[i].table\_number << setw(30) << dishes[i].name

<< setw(20) << dishes[i].special\_requests << setw(10) << dishes[i].status << endl;

}

// 修改菜品状态

int dish\_index;

cout << "\n请输入要更新状态的菜品序号（输入 0 结束程序）: ";

while (cin >> dish\_index && dish\_index != 0) {

if (dish\_index < 1 || dish\_index > dish\_count) {

cout << "无效的序号，请重新输入。" << endl;

}

else {

Dish& selected\_dish = dishes[dish\_index - 1];

if (selected\_dish.status == "未完成") {

string filepath = folder\_path + "/" + selected\_dish.order\_number + ".txt";

updateOrderStatus(filepath, selected\_dish);

selected\_dish.status = "已出菜"; // 更新内存中的状态

}

else {

cout << "该菜品已出菜，无法重复更新状态。" << endl;

}

}

cout << "\n请输入要更新状态的菜品序号（输入 0 结束程序）: ";

}

return 0;

}

}

厨师.main

#pragma once

#include"厨师.h"

namespace cook

{

using namespace std;

void cook()

{

sys::password("厨师");

while(true)

{

cout << "请选择你要需要的功能" << endl;

cout << "1：出菜" << endl;

cout << "2：修改菜品状态（修改是否售罄）" << endl;

cout << "3:修改密码" << endl;

cout << "0：返回上一级" << endl;

int n = int\_input(3);

switch (n)

{

case 1:cooker\_cook(); break;

case 2:changemunu("数据存储/menu.txt"); break;

case 3:sys::cpass2("厨师");

case 0:return;

default: break;

}

}

}

}

厨师结构体.h

#pragma once

// 菜品信息结构体(厨师）

namespace cook

{

using namespace std;

struct Dish {

string name;

string table\_number;

string order\_number;

string special\_requests;

string type;

string status;

};

}

#pragma once

#include"服务员上菜.h"

#include"服务员修改订单.h"

服务员main.h

namespace server

{

using namespace std;

void server()

{

sys::password("服务员");

while(1)

{

cout << "请选择你需要的功能" << endl;

cout << "1、上菜" << endl;

cout << "2、订单添加菜品" << endl;

cout << "3、订单删除菜品" << endl;

cout << "4、修改菜品状态（是否售罄）" << endl;

cout << "5、修改密码" << endl;

cout << "0、退出程序" << endl;

int n = ::int\_input(5);

switch (n)

{

case 1:waiter\_serve(); break;

case 2:serveradd(); break;

case 3:serverDelete(); break;

case 4: changemunu("数据存储/menu.txt"); break;

case 5:sys::cpass2("服务员"); break;

case 0:return;

}

}

}

}

服务员上菜.h

#pragma once

namespace server

{

using namespace std;

// 从文件读取服务员需要处理的菜品数据

int readWaiterFile(const string& filepath, cook::Dish dishes[], int start\_index) {

ifstream file(filepath);

if (!file.is\_open()) {

// cerr << "无法打开文件: " << filepath << endl;//有些文件打不开是正常的

return start\_index;

}

string line, table\_number;

while (getline(file, line)) {

if (line.find("桌号:") != string::npos) {

table\_number = line.substr(line.find(":") + 1); // 提取桌号

}

// 读取菜品表头之后的内容

if (line.find("菜品名") != string::npos) {

while (getline(file, line) && !line.empty()) {

stringstream ss(line);

string name, price, quantity, special\_requests, type, status;

// 按固定宽度读取各列

ss >> setw(30) >> name >> setw(10) >> price >> setw(10) >> quantity

>> setw(20) >> special\_requests >> setw(10) >> type >> setw(10) >> status;

// 如果种类是正餐但不是未完成，或者是饮料或小吃，则保存菜品信息

if (((type == "正餐" && status != "未完成") || type == "饮料或小吃") && start\_index < 500) {

dishes[start\_index++] = { name, table\_number, "", special\_requests, type, status };

}

}

}

}

file.close();

return start\_index;

}

void markDishAsCompleted(const string& filepath, const cook::Dish& dish) {

ifstream file(filepath);

if (!file.is\_open()) {

//cerr << "无法打开文件: " << filepath << endl;//有些文件打不开是正常的

return;

}

stringstream buffer;

string line;

bool updated = false;

while (getline(file, line)) {

if (line.find(dish.name) != string::npos &&

line.find(dish.status) != string::npos &&

(line.find("未完成") != string::npos || line.find("已出菜") != string::npos)) {

line.replace(line.find(dish.status), dish.status.length(), "已完成");

updated = true;

}

buffer << line << "\n";

}

file.close();

if (updated) {

ofstream output\_file(filepath, ios::trunc);

output\_file << buffer.str();

output\_file.close();

cout << "状态已更新为“已完成”：桌号 " << dish.table\_number << "，菜品名 " << dish.name << endl;

}

else {

cout << "未找到需要更新的菜品记录。" << endl;

}

}

// 更新菜品状态

int readWaiterFile(const string& filepath, cook::Dish dishes[], int start\_index, const string& order\_number) {

ifstream file(filepath);

if (!file.is\_open()) {

// cerr << "无法打开文件: " << filepath << endl;//有些文件打不开是正常的

return start\_index;

}

string line, table\_number;

while (getline(file, line)) {

if (line.find("桌号:") != string::npos) {

table\_number = line.substr(line.find(":") + 1); // 提取桌号

}

// 读取菜品表头之后的内容

if (line.find("菜品名") != string::npos) {

while (getline(file, line) && !line.empty()) {

stringstream ss(line);

string name, price, quantity, special\_requests, type, status;

// 按固定宽度读取各列

ss >> setw(30) >> name >> setw(10) >> price >> setw(10) >> quantity

>> setw(20) >> special\_requests >> setw(10) >> type >> setw(10) >> status;

// 如果种类是正餐但不是未完成，或者是饮料或小吃，则保存菜品信息

if (((type == "正餐" && status != "未完成") || type == "饮料或小吃") && start\_index < 500) {

dishes[start\_index++] = { name, table\_number, order\_number, special\_requests, type, status };

}

}

}

}

file.close();

return start\_index;

}

int waiter\_serve() {

string folder\_path = "数据存储/订单数据";

string serial\_file = folder\_path + "/序号.txt";

// 读取序号文件，确定订单总数

ifstream serial\_input(serial\_file);

if (!serial\_input.is\_open()) {

cerr << "无法读取序号文件: " << serial\_file << endl;

return 1;

}

int total\_orders;

serial\_input >> total\_orders;

total\_orders--;

if (total\_orders == 0) {

cout << "暂时没有订单";

return 0;

}

serial\_input.close();

cook::Dish dishes[500]; // 固定大小数组

int dish\_count = 0;

// 遍历所有订单文件

for (int i = 1; i <= total\_orders && dish\_count < 500; ++i) {

string order\_file = folder\_path + "/" + to\_string(i) + ".txt";

string order\_number = to\_string(i); // 获取订单号

dish\_count = readWaiterFile(order\_file, dishes, dish\_count, order\_number);

}

// 输出所有需要处理的菜品

cout << left << setw(5) << "序号" << setw(10) << "桌号"

<< setw(30) << "菜品名" << setw(20) << "特殊要求" << setw(10) << "状态" << endl;

cout << string(75, '-') << endl;

for (int i = 0; i < dish\_count; ++i) {

cout << left << setw(5) << (i + 1) << setw(10) << dishes[i].table\_number

<< setw(30) << dishes[i].name << setw(20) << dishes[i].special\_requests

<< setw(10) << dishes[i].status << endl;

}

// 修改菜品状态

int dish\_index;

cout << "\n请输入要更新状态的菜品序号（输入 0 结束程序）: ";

while (cin >> dish\_index && dish\_index != 0) {

if (dish\_index < 1 || dish\_index > dish\_count) {

cout << "无效的序号，请重新输入。" << endl;

}

else {

cook::Dish& selected\_dish = dishes[dish\_index - 1];

if (selected\_dish.status == "未完成" || selected\_dish.status == "已出菜") {

string filepath = folder\_path + "/" + selected\_dish.order\_number + ".txt";

markDishAsCompleted(filepath, selected\_dish);

selected\_dish.status = "已完成"; // 更新内存中的状态

}

else {

cout << "该菜品已完成，无需更新状态。" << endl;

}

}

cout << "\n请输入要更新状态的菜品序号（输入 0 结束程序）: ";

}

return 0;

}

}

服务员添加菜品.h

#pragma once

namespace server

{

void serverAddDishesToOrder(const string& order\_file, const MenuItem menu[], int menu\_size) {

// 打开订单文件

fstream file(order\_file, ios::in | ios::out);

if (!file.is\_open()) {

cout << "无法打开订单文件: " << order\_file << endl;

return;

}

// 读取订单内容并显示

string line;

cout << "当前订单内容:" << endl;

while (getline(file, line)) {

cout << line << endl;

}

// 添加菜品

int item\_number;

cout << "\n请输入要添加的菜品编号: ";

cin >> item\_number;

if (item\_number < 1 || item\_number > menu\_size) {

cout << "无效的菜品编号。" << endl;

return;

}

const MenuItem& selected\_item = menu[item\_number - 1];

cout << "已选择菜品: " << selected\_item.name << "，价格: " << selected\_item.price << " 元\n";

int quantity;

string special\_requests = "";

cout << "请输入数量: ";

cin >> quantity;

cin.ignore(); // 清除输入缓冲区中的换行符

special\_requests = costomer::getSpecialRequests(menu[item\_number - 1].type);

// 定位文件末尾并写入新菜品

file.clear(); // 清除文件流状态

file.seekp(0, ios::end);

file << left << setw(30) << selected\_item.name

<< setw(10) << selected\_item.price

<< setw(10) << quantity

<< setw(20) << special\_requests

<< setw(20) << costomer::class\_dishes(selected\_item.type)

<< setw(20) << "未完成" << endl;

cout << "菜品已成功添加到订单。" << endl;

file.close();

}

void serveradd()

{

displayAvailableOrders();

ifstream ss("数据存储/订单数据/序号.txt");

int n\_max;

ss >> n\_max;

ss.close();

::MenuItem menu[MAX\_MENU\_ITEMS];

int menu\_num = ::loadMenuFromFile("数据存储/menu.txt", menu);

showMenu(menu, menu\_num);

while (1)

{

cout << "请输入你想要添加的订单,按0退出";

int in = ::int\_input(n\_max, 0);

if (in == 0)

return;

string filename = "数据存储/订单数据/" + to\_string(in) + ".txt";

serverAddDishesToOrder(filename, menu, menu\_num);

}

}

void serverDeleteDishFromOrder(const string& order\_file) {

// 打开订单文件

ifstream file(order\_file);

if (!file.is\_open()) {

cout << "无法打开订单文件: " << order\_file << endl;

return;

}

// 读取订单内容并保存到数组

string lines[150];

int line\_count = 0;

string line;

cout << "当前订单内容:" << endl;

while (getline(file, line) && line\_count < 150) {

lines[line\_count++] = line;

cout << line << endl;

}

::separator(80);

file.close();

cout << endl << endl;

// 查找菜品部分的起始行

int dish\_start = -1;

for (int i = 0; i < line\_count; ++i) {

if (lines[i].find("菜品名") != string::npos) {

dish\_start = i + 1;

break;

}

}

if (dish\_start == -1 || dish\_start >= line\_count) {

// cerr << "订单中未找到菜品部分。" << endl;

return;

}

// 显示菜品列表

cout << "\n菜品列表:" << endl;

for (int i = dish\_start; i < line\_count; ++i) {

cout << i - dish\_start + 1 << ". " << lines[i] << endl;

}

// 选择要删除的菜品

int dish\_number;

cout << "\n请输入要删除的菜品序号 (输入0退出): ";

cin >> dish\_number;

if (dish\_number <= 0 || dish\_start + dish\_number - 1 >= line\_count) {

cout << "无效的菜品序号。" << endl;

return;

}

// 删除指定菜品

for (int i = dish\_start + dish\_number - 1; i < line\_count - 1; ++i) {

lines[i] = lines[i + 1];

}

--line\_count;

cout << "菜品已成功删除。" << endl;

// 写回文件

ofstream outfile(order\_file, ios::trunc);

if (!outfile.is\_open()) {

cout << "无法写入订单文件: " << order\_file << endl;

return;

}

for (int i = 0; i < line\_count; ++i) {

outfile << lines[i] << endl;

}

outfile.close();

}

void serverDelete() {

displayAvailableOrders();

ifstream ss("数据存储/订单数据/序号.txt");

int n\_max;

ss >> n\_max;

ss.close();

while (1) {

cout << "请输入你想要删除菜品的订单编号, 按0退出: ";

int in = ::int\_input(n\_max, 0);

if (in == 0)

return;

string filename = "数据存储/订单数据/" + to\_string(in) + ".txt";

serverDeleteDishFromOrder(filename);

}

}

void displayAvailableOrders(const std::string& order\_directory) {

using namespace std;

// 打开序号文件获取最大订单号

ifstream seq\_file(order\_directory + "/序号.txt");

if (!seq\_file.is\_open()) {

cerr << "无法打开序号文件: " << order\_directory + "/序号.txt" << endl;

return;

}

int max\_order\_number;

seq\_file >> max\_order\_number;

seq\_file.close();

if (max\_order\_number <= 0) {

cout << "当前没有可用的订单。" << endl;

return;

}

cout << left << setw(15) << "订单编号" << setw(15) << "餐桌号" << setw(20) << "姓名" << endl;

cout << string(50, '-') << endl;

for (int i = 1; i <= max\_order\_number; ++i) {

string file = order\_directory + "/" + to\_string(i) + ".txt";

ifstream infile(file);

if (!infile.is\_open()) {

continue; // 跳过不存在的文件

}

string line, table\_number = "未知", name = "未知";

// 读取餐桌号和姓名信息

while (getline(infile, line)) {

if (line.find("餐桌号:") != string::npos) {

table\_number = line.substr(line.find(":") + 1);

table\_number.erase(0, table\_number.find\_first\_not\_of(" "));

}

else if (line.find("姓名:") != string::npos) {

name = line.substr(line.find(":") + 1);

name.erase(0, name.find\_first\_not\_of(" "));

break;

}

}

infile.close();

cout << left << setw(15) << i

<< setw(15) << table\_number

<< setw(20) << name << endl;

}

}

}

顾客.h

using namespace std;

// 从文件加载菜单

// 获取有效数量

int getValidQuantity() {

int quantity=int\_input(MAX\_MENU\_ITEMS);

return quantity;

}

// 选择菜单项

int selectMenuItems(MenuItem menu[], int itemCount, Order& order) {

cout << "请选择菜品 (输入菜品编号和数量, 输入q退出): " << endl;

while (true) {

string input;

cout << "菜品编号: ";

cin >> input;

if (input == "q") break;

stringstream ss(input);

int id;

if (ss >> id && id > 0 && id <= itemCount&&menu[id-1].type%10==1) {

cout << "数量: ";

int quantity = getValidQuantity();

string special\_requests= getSpecialRequests(menu[id - 1].type);

order.items[order.item\_count++] = { menu[id - 1], quantity, special\_requests };

if (order.item\_count >= MAX\_MENU\_ITEMS) {

cout << "订单项已达上限, 无法添加更多菜品." << endl;

break;

}

}

else {

cout << "无效的菜品编号, 请重新输入." << endl;

}

}

return order.item\_count;

}

顾客main.h

namespace costomer

{

using namespace std;

#include"特殊需求.h"

#include"小票.h"

#include"顾客.h"

#include"选位子.h"

#include"投诉与建议.h"

int updateSerialNumber(const string& filename) //当前到哪里了

{

int serial\_number = 1; // 默认值

// 打开文件读取当前的序号

ifstream input\_file(filename);

if (input\_file.is\_open()) {

input\_file >> serial\_number; // 读取文件中的序号

input\_file.close();

}

else {

//cout << "文件不存在，将创建新文件。" << endl;

}

// 将序号加一

// 以覆盖模式写入文件

ofstream output\_file(filename, ios::out);

if (output\_file.is\_open()) {

output\_file << serial\_number + 1;

output\_file.close();

}

else {

cerr << "无法打开文件进行写入。" << endl;

}

return serial\_number;

}

int customer() {

int table = select\_table();//分配位子模块

bool is\_break = 1;

while (is\_break)

{

cout << "1:点菜\n"

"2:投诉与建议\n"

"0:返回上级"<<endl;

int n = int\_input(2);

switch (n)

{

case 1:is\_break = 0; break;

case 2:saveFeedback(); break;

case 0:return 0;

}//投诉与建议模块

}

int number = updateSerialNumber("数据存储/订单数据/序号.txt");

MenuItem menu[MAX\_MENU\_ITEMS];

int itemCount = loadMenuFromFile("数据存储/menu.txt", menu);

Order order;

getUserDetails(order);

order.table\_count = table;

//int n=customer\_order(menu,\*order, itemCount);

if (itemCount == 0) {

cout << "菜单为空或加载失败" << endl;

exit(0);

return 0;

}

order.item\_count = 0;

cout << "欢迎来到餐厅点餐系统!" << endl;

showMenu(menu, itemCount);

int fn = selectMenuItems(menu, itemCount, order);

calculateTotal(order);

printOrderSummary(order);

saveOrderToFile(order, number);

while (1) {

cout <<

"1:添加菜品\n"

"2:投诉与建议\n"

"0:返回上级\n";

int n = int\_input(2);

switch (n)

{

case 1:addMenuItem(order, menu, itemCount); saveOrderToFile(order, number); break;

case 2:saveFeedback(); break;

case 0:; return 0;

}

}

return 0;

}

int book\_customer() {

int table = select\_table();//分配位子模块

bool is\_break = 1;

while (is\_break)

{

cout << "您可以在本系统预定餐位并点餐" << endl;

cout << "1:点菜\n"

"2:投诉与建议\n"

"0:不点餐" << endl;

int n = int\_input(2);

switch (n)

{

case 1:is\_break = 0; break;

case 2:saveFeedback(); break;

case 0:return 0;

}//投诉与建议模块

}

int number = updateSerialNumber("数据存储/订单数据/序号.txt");

MenuItem menu[MAX\_MENU\_ITEMS];

int itemCount = loadMenuFromFile("数据存储/menu.txt", menu);

Order order;

getUserDetails(order);

order.table\_count = table;

//int n=customer\_order(menu,\*order, itemCount);

if (itemCount == 0) {

cout << "菜单为空或加载失败" << endl;

exit(0);

return 0;

}

order.item\_count = 0;

cout << "欢迎来到餐厅点餐系统!" << endl;

showMenu(menu, itemCount);

int fn = selectMenuItems(menu, itemCount, order);

calculateTotal(order);

printOrderSummary(order);

saveOrderToFile(order, number);

cout << "您的预定餐位已经记录，您仍然可以在到达后添加菜品";

暂停;

return 0;

}

void book\_customer\_arrival() {

cout << "请输入你的订单号";

int in = ::int\_input(0x7fffffff,0);

::MenuItem menu[MAX\_MENU\_ITEMS];

int itemCount = loadMenuFromFile("数据存储/menu.txt", menu);

showMenu(menu, itemCount);

int menu\_num = ::loadMenuFromFile("数据存储/menu.txt", menu);

string filename = "数据存储/订单数据/" + to\_string(in) + ".txt";

server::serverAddDishesToOrder(filename, menu, menu\_num);

暂停;

return ;

}

}

结构体.h

#pragma once

#define MAX\_MENU\_ITEMS 100

#define MAX\_NUMBER 100

using namespace std;

// 菜单项结构体

struct MenuItem {

int id; // 菜单项编号

string name; // 菜单项名称

double price; // 菜单项价格

int type; // 菜单项类型

};

// 订单项结构体

struct OrderItem {

MenuItem item; // 菜单项信息

int quantity; // 数量

string special\_requests; // 特殊要求

};

// 订单结构体

struct Order {

string customer\_name; // 顾客姓名

string contact\_info; // 顾客联系方式

string remarks; // 订单备注

int assigned\_tables; // 分配的桌子编号数组

int table\_count; // 分配的桌子数量

OrderItem items[MAX\_MENU\_ITEMS]; // 订单包含的菜品项

int item\_count; // 订单中的菜品总数

double total\_price; // 订单总价

};

struct Table {

int id; // 桌子的编号

int capacity; // 桌子的容量（最多可以容纳多少人）

int status; // 桌子状态：1 表示可用，2 表示已占用

};

特殊需求.h

#pragma once

string special\_requests\_hot(int n)

{

switch (n)

{

case 1:return "不辣";

case 2:return "微辣";

case 3:return "中辣";

case 4:return "麻辣";

}

return "-1";

}

string getSpecialRequests(int type) {

if (type == 101) {

cout << "请输入辣椒程度:用数字1-4表示，1是不辣，2是微辣，3是中辣，4是麻辣:";

int spiceLevel = int\_input(4,1);

return special\_requests\_hot(spiceLevel);

}

return "无";

}

投诉与建议.h

#pragma once

void saveFeedback() {

int choice;

string content;

// 提示用户选择投诉或建议

cout << "请选择要提交的内容:" << endl;

cout << "1. 投诉" << endl;

cout << "2. 建议" << endl;

cout << "请输入选择 (1 或 2): ";

cin >> choice;

cin.ignore(); // 清除缓冲区的换行符

// 提示用户输入内容

cout << "请输入您的内容 (最多500个字符): ";

getline(cin, content);

ofstream file;

// 根据用户选择将内容写入不同的文件

if (choice == 1) {

file.open("数据存储/投诉.txt", ios::app); // 以追加模式打开投诉文件

if (!file) {

cout << "无法打开投诉文件！" << endl;

return;

}

file << "投诉内容:\n" << content << "\n\n";

cout << "您的投诉已提交，谢谢！" << endl;

}

else if (choice == 2) {

file.open("数据存储/建议.txt", ios::app); // 以追加模式打开建议文件

if (!file) {

cout << "无法打开建议文件！" << endl;

return;

}

file << "建议内容:\n" << content << "\n\n";

cout << "您的建议已提交，谢谢！" << endl;

}

else {

cout << "无效的选择，请重新选择！" << endl;

return;

}

file.close(); // 关闭文件

}

小票.h

#pragma once

void calculateTotal(Order& order) {

order.total\_price = 0.0;

for (int i = 0; i < order.item\_count; ++i) {

order.total\_price += order.items[i].item.price \* order.items[i].quantity;

}

}

string class\_dishes(int n)

{

if (n == 101 || n == 201)

return"正餐";

else

return"饮料或小吃";

}

// 获取客户信息

void getUserDetails(Order& order) {

cout << "请输入您的姓名: ";

cin.ignore();

getline(cin, order.customer\_name);

cout << "请输入联系方式 (如果不愿透露可直接按回车): ";

getline(cin, order.contact\_info);

cout << "对订单有任何备注吗? (如特殊要求, 过敏信息等): ";

getline(cin, order.remarks);

}

// 打印订单总结

void printOrderSummary(const Order& order) {

cout << "订单总结:" << endl;

cout << "姓名: " << order.customer\_name << endl;

cout << "联系方式: " << order.contact\_info << endl;

cout << "备注: " << order.remarks << endl;

cout << left << setw(30) << "菜品名" << setw(10) << "单价" << setw(10) << "数量"

<< setw(20) << "特殊要求" <<setw(20)<<"种类" << endl;

separator(80);

for (int i = 0; i < order.item\_count; ++i) {

cout << left << setw(30) << order.items[i].item.name

<< setw(10) << order.items[i].item.price

<< setw(10) << order.items[i].quantity

<< setw(20) << order.items[i].special\_requests

<< setw(20)<<class\_dishes(order.items[i].item.type)<<endl;

}

//cout << "------------------------------------------" << endl;

cout << "总价: " << order.total\_price << "元" << endl;

}

#pragma once//总价

// 获取客户信息

// 打印订单总结

void addMenuItem(Order& order, const MenuItem menu[],int itemCount) {

cout << "请输入你要添加的菜品编号";

int n=int\_input(itemCount);

int quantity;

string special\_requests;

MenuItem menu\_item = menu[n - 1];

cout << "您选择的菜品是: " << menu\_item.name << "，价格: " << menu\_item.price << "元\n";

cout << "请输入数量: ";

cin >> quantity;

cin.ignore(); // 清除换行符

special\_requests= getSpecialRequests(menu[n-1].type);

OrderItem new\_item = { menu\_item, quantity, special\_requests };

order.items[order.item\_count++]=new\_item;

cout << "菜品已添加到订单。\n";

}

void saveOrderToFile(const Order& order, int order\_number) {

string filename = "数据存储/订单数据/"+to\_string(order\_number) + ".txt";

ofstream order\_file(filename, ios::out);

if (!order\_file.is\_open()) {

cerr << "无法创建订单文件" << endl;

return;

}

// 写入订单信息

order\_file << "餐桌号: " << order.table\_count << "\n";

order\_file << "姓名: " << order.customer\_name << "\n";

order\_file << "联系方式: " << order.contact\_info << "\n";

order\_file << "备注: " << order.remarks << "\n\n";

order\_file << left << setw(30) << "菜品名" << setw(10) << "单价" << setw(10) << "数量"

<< setw(20) << "特殊要求" << setw(20)<<"种类"<<setw(20)<<"状态" << "\n";

//order\_file << "----------------------------------------------\n";

for (int i = 0; i < order.item\_count;i++) {

order\_file << left << setw(30) << order.items[i].item.name

<< setw(10) << order.items[i].item.price

<< setw(10) << order.items[i].quantity

<< setw(20) << order.items[i].special\_requests

<<setw(20)<<class\_dishes(order.items[i].item.type)

<<setw(20)<<"未完成"<<endl;

}

//order\_file << "------------------------------------------\n";

order\_file.close();

cout << "订单已保存到文件: " << filename << endl;

}

选位子.h

#pragma once

int max\_table;

bool cmd1(const Table& a, const Table& b) {

return a.capacity < b.capacity;

}

int loadTablesFromFile(const string& filename, Table tables[], int maxTables) {

ifstream file(filename);

if (!file.is\_open()) {

cerr << "无法打开位子文件" << endl;

return 0;

}

int tableCount;

file >> tableCount;

file.ignore(); // Ignore newline character

for (int i = 0; i < tableCount && i < maxTables; ++i) {

file >> tables[i].id >> tables[i].capacity >> tables[i].status;

}

file.close();

return tableCount;

}

void saveTablesToFile(const string& filename, Table tables[], int tableCount) {

ofstream file(filename);

if (!file.is\_open()) {

cerr << "无法保存位子文件" << endl;

return;

}

file << tableCount << endl;

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

file << tables[i].id << " " << tables[i].capacity << " " << tables[i].status << endl;

}

file.close();

}

int assignTable(int customerCount, Table tables[], int tableCount, Order& order) {

int selectedTableId = -1; // 用于存储选择的桌子ID

// 按桌子容量升序排序

bubbleSort(tables, tables + tableCount, cmd1);

// 查找第一个容量大于等于顾客人数且状态为可用的桌子

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

if (tables[i].status == 1 && tables[i].capacity >= customerCount) {

selectedTableId = tables[i].id; // 记录桌子ID

tables[i].status = 2; // 标记桌子为已占用

break;

}

}

// 如果找到合适的桌子，则分配给订单

if (selectedTableId != -1) {

order.assigned\_tables = selectedTableId; // 分配的桌子ID

order.table\_count = 1; // 表示只分配了一张桌子

//cout << "分配了桌子编号: " << selectedTableId << endl;

return selectedTableId;

}

else {

// 如果没有找到合适的桌子，提示错误

cerr << "无法满足顾客人数的座位需求，请稍后再试。" << endl;

}

}

void printAssignedTables(const Order& order) {

cout << "分配的餐桌编号: ";

for (int i = 0; i < order.table\_count; ++i) {

cout << order.assigned\_tables << "号 ";

}

cout << endl;

}

int select\_table()

{

const int MAX\_TABLES = 100; // 最大桌子数量

Table tables[MAX\_TABLES];

int tableCount = loadTablesFromFile("数据存储/位子.txt", tables, MAX\_TABLES);

if (tableCount == 0) {

cout << "无法加载位子信息" << endl;

exit(0);

}

Order order;

order.item\_count = 0;

// 询问顾客人数

int customerCount;

cout << "请输入顾客人数: ";

customerCount=int\_input(1000,1);

// 分配餐桌

int table=assignTable(customerCount, tables, tableCount, order);

// 显示分配结果

printAssignedTables(order);

// 保存更新后的桌子状态

saveTablesToFile("数据存储/位子.txt", tables, tableCount);

return table;

}

餐桌管理.h

namespace sys

{

using namespace std;

const int MAX\_TABLES = 100;

struct Table {

int table\_number;

int customer\_count;

int status; // 1: 无人, 2: 有人

};

// 读取餐桌信息文件

bool readTableInfo(const string& filepath, Table tables[], int& table\_count) {

ifstream file(filepath);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filepath << endl;

return false;

}

string line;

if (getline(file, line)) {

table\_count = stoi(line); // 读取餐桌总数

if (table\_count > MAX\_TABLES) {

cerr << "餐桌数量超过最大值 (" << MAX\_TABLES << ")！\n";

return false;

}

}

else {

cerr << "文件格式错误: 无法读取餐桌总数。" << endl;

return false;

}

int index = 0;

while (getline(file, line) && index < table\_count) {

stringstream ss(line);

ss >> tables[index].table\_number >> tables[index].customer\_count >> tables[index].status;

++index;

}

file.close();

return true;

}

// 显示所有餐桌状态

void displayTableStatus(const Table tables[], int table\_count) {

cout << "================ 餐桌状态 ================\n";

cout << left << setw(10) << "桌号" << setw(15) << "就餐人数" << setw(10) << "状态" << "\n";

cout << "------------------------------------------\n";

for (int i = 0; i < table\_count; ++i) {

cout << left << setw(10) << tables[i].table\_number

<< setw(15) << tables[i].customer\_count

<< (tables[i].status == 1 ? "无人" : "有人") << "\n";

}

cout << "==========================================\n";

}

// 修改指定餐桌的状态

void modifyTableStatus(Table tables[], int table\_count, int table\_number, int new\_status) {

for (int i = 0; i < table\_count; ++i) {

if (tables[i].table\_number == table\_number) {

tables[i].status = new\_status;

cout << "桌号 " << table\_number << " 的状态已修改为 "

<< (new\_status == 1 ? "无人" : "有人") << "。\n";

return;

}

}

cerr << "未找到桌号 " << table\_number << "！\n";

}

// 批量修改所有餐桌的状态

void batchModifyTableStatus(Table tables[], int table\_count, int new\_status) {

for (int i = 0; i < table\_count; ++i) {

tables[i].status = new\_status;

}

cout << "所有餐桌的状态已修改为 "

<< (new\_status == 1 ? "无人" : "有人") << "。\n";

}

// 保存餐桌信息回文件

bool saveTableInfo(const string& filepath, const Table tables[], int table\_count) {

ofstream file(filepath);

if (!file.is\_open()) {

cerr << "无法保存文件: " << filepath << endl;

return false;

}

file << table\_count << "\n"; // 写入餐桌总数

for (int i = 0; i < table\_count; ++i) {

file << tables[i].table\_number << " " << tables[i].customer\_count << " " << tables[i].status << "\n";

}

file.close();

return true;

}

// 餐桌管理主函数

void manageTables() {

string filepath = "数据存储/位子.txt";

Table tables[MAX\_TABLES];

int table\_count = 0;

// 读取餐桌信息

if (!readTableInfo(filepath, tables, table\_count)) {

return;

}

int choice;

do {

cout << "\n======== 餐桌管理系统 ========\n";

cout << "1. 显示所有餐桌状态\n";

cout << "2. 修改指定餐桌状态\n";

cout << "3. 批量修改所有餐桌状态\n";

cout << "4. 保存并退出\n";

cout << "请输入选项: ";

cin >> choice;

switch (choice) {

case 1:

displayTableStatus(tables, table\_count);

break;

case 2: {

int table\_number, new\_status;

cout << "请输入桌号: ";

cin >> table\_number;

cout << "请输入新状态 (1: 无人, 2: 有人): ";

cin >> new\_status;

if (new\_status == 1 || new\_status == 2) {

modifyTableStatus(tables, table\_count, table\_number, new\_status);

}

else {

cerr << "无效状态！\n";

}

break;

}

case 3: {

int new\_status;

cout << "请输入新状态 (1: 无人, 2: 有人): ";

cin >> new\_status;

if (new\_status == 1 || new\_status == 2) {

batchModifyTableStatus(tables, table\_count, new\_status);

}

else {

cerr << "无效状态！\n";

}

break;

}

case 4:

if (saveTableInfo(filepath, tables, table\_count)) {

cout << "餐桌信息已保存。\n";

}

break;

default:

cerr << "无效选项！\n";

break;

}

} while (choice != 4);

}

}

查看统计数据.h

#pragma once

namespace sys

{

using namespace std;

// 去除字符串中的所有空格

string removeSpaces(const string& str) {

string result = str;

result.erase(remove(result.begin(), result.end(), ' '), result.end());

return result;

}

// 处理单个文件内容

void processFile(const string& filename, DishStats dishStatistics[], int& dishCount, double& totalRevenue) {

ifstream receiptFile(filename);

if (!receiptFile.is\_open()) {

//cerr << "无法打开文件: " << filename << endl;

return;

}

string line;

while (getline(receiptFile, line)) {

// 处理菜品数据行

if (line.find("菜品名") != string::npos) {

// 跳过表头和分隔线

getline(receiptFile, line); // 跳过表头

getline(receiptFile, line); // 跳过分隔线

while (!line.empty() && line.find("餐位费") == string::npos && line.find("总价") == string::npos) {

istringstream ss(line);

string name, unitPriceStr, quantityStr, specialRequest, status;

// 解析每一行

ss >> name >> unitPriceStr >> quantityStr >> specialRequest >> status;

if (ss.fail()) {

//cerr << "行解析失败，跳过: " << line << endl;

break;

}

// 去掉空格

name = removeSpaces(name);

quantityStr = removeSpaces(quantityStr);

status = removeSpaces(status);

// 检查数量是否为数字

if (!all\_of(quantityStr.begin(), quantityStr.end(), ::isdigit)) {

//cerr << "无效数量字段: " << quantityStr << endl;

break;

}

int quantity = stoi(quantityStr);

// 查找或添加菜品统计

bool found = false;

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

if (dishStatistics[i].name == name) {

if (status == "已出菜") {

dishStatistics[i].completed\_count += quantity;

}

else {

dishStatistics[i].pending\_count += quantity;

}

found = true;

break;

}

}

// 如果菜品不存在，新增一条统计信息

if (!found) {

if (dishCount >= 500) {

cerr << "菜品统计数量超过限制，无法添加新菜品: " << name << endl;

break;

}

dishStatistics[dishCount++] = { name, (status == "已出菜" ? quantity : 0), (status != "已出菜" ? quantity : 0) };

}

// 读取下一行

if (!getline(receiptFile, line)) break;

}

}

// 处理总价行

if (line.find("总价:") != string::npos) {

string totalPriceStr = line.substr(line.find(":") + 1);

try {

totalRevenue += stod(totalPriceStr);

}

catch (const invalid\_argument&) {

cerr << "无效总价字段: " << totalPriceStr << endl;

}

}

}

receiptFile.close();

}

// 主函数

void analyzeReceipts(const string& folderPath) {

DishStats dishStatistics[500] = {}; // 最大支持 500 种菜品

int dishCount = 0; // 当前统计的菜品数量

double totalRevenue = 0.0; // 总收入

int totalReceipts = 0; // 小票总数

// 遍历文件名 1.txt 到 500.txt

for (int i = 1; i <= 500; ++i) {

string filename = folderPath + "/" + to\_string(i) + ".txt";

ifstream testFile(filename);

if (testFile.is\_open()) {

++totalReceipts; // 成功打开文件

processFile(filename, dishStatistics, dishCount, totalRevenue);

}

}

// 输出统计结果

cout << "================ 小票统计数据 ================\n";

cout << "总小票数: " << totalReceipts << "\n";

cout << "总收入: " << fixed << setprecision(2) << totalRevenue << " 元\n";

cout << "---------------------------------------------\n";

cout << left << setw(30) << "菜品名"

<< setw(15) << "完成数量"

<< setw(15) << "未完成数量" << "\n";

cout << "---------------------------------------------\n";

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

cout << left << setw(30) << dishStatistics[i].name

<< setw(15) << dishStatistics[i].completed\_count

<< setw(15) << dishStatistics[i].pending\_count << "\n";

}

cout << "=============================================\n";

system("pause");

}

}

获取修改菜单.h

#pragma once

#define max\_int 0x7fffffff

using namespace std;

int loadMenuFromFile(const string& filename, MenuItem menu[]) {

ifstream file(filename);

if (!file.is\_open()) {

cerr << "无法打开菜单文件" << endl;

return 0;

}

size\_t itemCount;

file >> itemCount;

file.ignore(); // 忽略换行符

string line;

size\_t loadedItems = 0;

for (size\_t i = 0; i < itemCount && i < MAX\_MENU\_ITEMS; ++i) {

MenuItem item;

getline(file, line);

item.id = stoi(line);

getline(file, item.name);

getline(file, line);

item.price = stod(line);

getline(file, line);

item.type = stoi(line);

menu[loadedItems++] = item;

}

file.close();

return itemCount;

}

string showmenutype(int n)

{

if (n == 101 || n == 201)

return"正餐";

else if (n == 301)

return "小吃或零食";

else

return "已售罄";

}

// 显示菜单

void showMenu(MenuItem menu[], int itemCount)

{

cout << "当前菜单:" << endl;

cout <<left<<setw(10) <<"编号" <<setw(30)<< "名称"<<setw(20) << "价格" << setw(20)<< "种类"<<endl;

separator(80);

cout << endl;

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

cout <<left<< setw(10) << menu[i].id

<< setw(30) << menu[i].name

<< setw(20)<< menu[i].price <<setw(20)<<showmenutype(menu[i].type) << endl;

}

}

bool cmd1(MenuItem a, MenuItem b)

{

return a.id < b.id;

}

void changemunu(const string& filename) {

MenuItem menu[MAX\_MENU\_ITEMS];

int loadedItems = loadMenuFromFile(filename, menu);

if (loadedItems == 0) {

cerr << "菜单加载失败！" << endl;

return;

}

bubbleSort(menu, menu + loadedItems-1, cmd1);

showMenu(menu, loadedItems);

cout << "请输入你想要改为售罄/未售罄的编号: ";

int n = int\_input(loadedItems) - 1;

// 切换菜品的售罄状态

if (menu[n].type %10== 1) {

menu[n].type += 1; // 未售罄 -> 售罄

cout << "已将 " << menu[n].name << " 改为 售罄。" << endl;

}

else if (menu[n].type%10 == 2) {

menu[n].type -= 1; // 售罄 -> 未售罄

cout << "已将 " << menu[n].name << " 改为 未售罄。" << endl;

}

else {

cerr << "错误: 菜品状态非法！" << endl;

return;

}

// 写回文件

ofstream file(filename);

if (!file.is\_open()) {

cerr << "无法写入文件: " << filename << endl;

return;

}

file << loadedItems << endl;

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

file<<menu[i].id+1<<endl;

file << menu[i].name << endl;

file << menu[i].price << endl;

file << menu[i].type << endl;

}

}

namespace sys

{

using namespace std;

struct MenuItem {

int id;

string name;

int price;

int status;

};

const int MAX\_ITEMS = 500;

MenuItem menu[MAX\_ITEMS];

int itemCount = 0;

// 读取菜单

void loadMenu(const string& filename) {

ifstream file(filename);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filename << endl;

return;

}

file >> itemCount;

file.ignore(); // 忽略换行符

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

file >> menu[i].id;

file.ignore(); // 忽略换行符

getline(file, menu[i].name);

file >> menu[i].price;

file >> menu[i].status;

file.ignore(); // 忽略换行符

}

file.close();

}

// 保存菜单

void saveMenu(const string& filename) {

ofstream file(filename);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filename << endl;

return;

}

file << itemCount << endl;

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

file << menu[i].id << endl;

file << menu[i].name << endl;

file << menu[i].price << endl;

file << menu[i].status << endl;

}

file.close();

}

// 显示菜单

void displayMenu() {

cout << "\n当前菜单:\n";

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

cout << "编号: " << menu[i].id

<< ", 名称: " << menu[i].name

<< ", 价格: " << menu[i].price

<< ", 状态: " << menu[i].status << endl;

}

}

// 添加菜品

void addMenuItem() {

if (itemCount >= MAX\_ITEMS) {

cout << "菜单已满，无法添加更多菜品。\n";

return;

}

MenuItem newItem;

cout << "请输入新菜品信息:\n";

cout << "编号: ";

cin >> newItem.id;

cin.ignore();

cout << "名称: ";

getline(cin, newItem.name);

cout << "价格: ";

cin >> newItem.price;

cout << "状态 (例如: 101, 201): ";

cin >> newItem.status;

menu[itemCount++] = newItem;

cout << "菜品添加成功!\n";

}

// 删除菜品

void deleteMenuItem() {

int idToDelete;

cout << "请输入要删除的菜品编号: ";

cin >> idToDelete;

int index = -1;

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

if (menu[i].id == idToDelete) {

index = i;

break;

}

}

if (index == -1) {

cout << "未找到指定编号的菜品。\n";

return;

}

for (int i = index; i < itemCount - 1; ++i) {

menu[i] = menu[i + 1];

}

--itemCount;

cout << "菜品删除成功!\n";

}

int change\_menu() {

string filename = "数据存储/menu.txt";

loadMenu(filename);

int choice;

do {

cout << "\n菜单管理系统\n";

cout << "1. 显示菜单\n";

cout << "2. 添加菜品\n";

cout << "3. 删除菜品\n";

cout << "4. 保存并退出\n";

cout << "请输入你的选择: ";

cin >> choice;

switch (choice) {

case 1:

displayMenu();

break;

case 2:

addMenuItem();

break;

case 3:

deleteMenuItem();

break;

case 4:

saveMenu(filename);

cout << "更改已保存，正在退出...\n";

break;

default:

cout << "无效选择，请重试。\n";

}

} while (choice != 4);

return 0;

}

}

投诉与建议.h

void readSuggestions() {

string filePath = "数据存储/建议.txt";

readFileByLine(filePath);

system("pause");

}

// 读取投诉文件

void readComplaints() {

string filePath = "数据存储/投诉.txt";

readFileByLine(filePath);

system("pause");

}

调试功能模块.h

#pragma once

namespace administrator

{

using namespace std;

void test()

{

while (1) {

cout << "请选择你需要调试的模块" << endl;

cout << "1、厨师模块" << endl;

cout << "2、服务员模块" << endl;

cout << "3、顾客模块" << endl;

cout << "4、老板模块" << endl;

cout << "5、收营员模块" << endl;

cout << "0:返回上一级"<<endl;

int n = ::int\_input(5);

system("cls");

switch (n)

{

case 1:cook::cook(); break;

case 2:server::server(); break;

case 3:costomer::customer(); break;

case 4: ; break;

case 5: cashier::cashier(); break;

case 0:return;

default: break;

}

system("cls");

}

}

}

管理员.h

#pragma once

#include"调试功能模块.h"

#include"管理账号.h"

namespace administrator

{

using namespace std;

void administrator()

{

sys::password("管理员");

while (1) {

cout << "请选择你需要的功能" << endl;

cout << "1、订单生成小票" << endl;

cout << "2、订单添加菜品" << endl;

cout << "3、订单删除菜品" << endl;

cout << "4、修改菜品状态（是否售罄）" << endl;

cout << "5、统计小票数据" << endl;

cout << "6、修改餐桌状态" << endl;

cout << "7、修改菜单" << endl;

cout << "8、展示建议" << endl;

cout << "9、展示投诉" << endl;

cout << "10、调试功能模块" << endl;

cout << "11、设置餐位费" << endl;

cout << "12、修改密码" << endl;

cout << "0、返回上一级" << endl;

int n = ::int\_input(11);

switch (n)

{

case 1:cashier::cashier\_recipet(); break;

case 2:server::serveradd(); break;

case 3:server::serverDelete(); break;

case 4: ::changemunu("数据存储/menu.txt"); break;

case 5: sys::analyzeReceipts("数据存储/小票存储/"); break;

case 6:sys::manageTables(); break;

case 7:sys::change\_menu(); break;

case 8: ::readSuggestions(); break;

case 9: ::readComplaints(); break;

case 10:test(); break;

case 11:set\_feel(); break;

case 12:admin(); break;

case 0:return;

default: break;

}

system("cls");

}

}

void set\_feel()

{

cout << "请输入你要设置的餐位费";

int n = ::int\_input();

cout << "设置成功";

::writeNumberToFile("数据存储/餐位费.txt",n);

}

}

账号管理.h

#pragma once

#include"调试功能模块.h"

namespace administrator

{

using namespace std;

void admin()

{

sys::password("管理员");

while (1) {

cout << "请选择你需要修改的密码" << endl;

cout << "1、修改厨师密码" << endl;

cout << "2、修改老板密码" << endl;

cout << "3、修改服务员密码" << endl;

cout << "4、修改管理员密码" << endl;

cout << "5、修改收营员密码" << endl;

int n = ::int\_input(5);

switch (n)

{

case 1:sys::cpass1("厨师"); break;

case 2:sys::cpass1("老板"); break;

case 3:sys::cpass1("服务员"); break;

case 4:sys::cpass1("管理员"); break;

case 5:sys::cpass1("收营员"); break;

case 0:return;

default: break;

}

system("cls");

}

}

}

老板main.h

#pragma once

namespace boss

{

using namespace std;

void boss()

{

sys::password("老板");

while (1) {

cout << "请选择你需要的功能" << endl;

cout << "1、订单生成小票" << endl;

cout << "2、订单添加菜品" << endl;

cout << "3、订单删除菜品" << endl;

cout << "4、修改菜品状态（是否售罄）" << endl;

cout << "5、统计数据" << endl;

cout << "6、修改餐桌状态" << endl;

cout << "7、修改菜单" << endl;

cout << "8、展示建议" << endl;

cout << "9、展示投诉" << endl;

cout << "10、修改密码" << endl;

cout << "0、返回上一级" << endl;

int n = ::int\_input(10);

switch (n)

{

case 1:cashier::cashier\_recipet(); break;

case 2:server::serveradd(); break;

case 3:server::serverDelete(); break;

case 4: ::changemunu("数据存储/menu.txt"); break;

case 5: sys::analyzeReceipts("数据存储/小票存储/"); break;

case 6:sys::manageTables(); break;

case 7:sys::change\_menu(); break;

case 8: ::readSuggestions(); break;

case 9: ::readComplaints(); break;

case 10: sys::cpass2("老板"); break;

case 0:return;

default: break;

}

system("cls");

}

}

}

收营员main.h

#pragma once

#include"收营员结算小票.h"

namespace cashier

{

void cashier()

{

sys::password("收营员");

while(1){

cout << "请选择你需要的功能" << endl;

cout << "1、订单结算并生成小票" << endl;

cout << "2、订单添加菜品" << endl;

cout << "3、订单删除菜品" << endl;

cout << "4、修改菜品状态（是否售罄）" << endl;

cout << "5、统计小票数据" << endl;

cout << "6、修改餐桌状态"<<endl;

cout << "7、修改密码" << endl;

cout << "0、退出程序" << endl;

int n = ::int\_input(7);

switch (n)

{

case 1:cashier\_recipet(); break;

case 2:server::serveradd(); break;

case 3:server::serverDelete(); break;

case 4: ::changemunu("数据存储/menu.txt"); break;

case 5: sys::analyzeReceipts("数据存储/小票存储/"); break;

case 6:sys::manageTables(); break;

case 7:sys::cpass2("收营员"); break;

case 0:return;

default: break;

}

system("cls");

}

}

}

收营员结算小票.h

#pragma once

namespace cashier

{

#define MAX\_item 500

// 读取订单文件并提取菜品信息

int readWaiterFile(const string& filepath, cashier::Dish dishes[], int& start\_index, int& customer\_count) {

ifstream file(filepath);

if (!file.is\_open()|| file.tellg() == 0) {

cerr << "无法打开文件: " << filepath << endl;

return -1;

}

string line, table\_number;

while (getline(file, line)) {

// 提取餐桌号

if (line.find("餐桌号:") != string::npos) {

table\_number = line.substr(line.find(":") + 1);

table\_number.erase(remove(table\_number.begin(), table\_number.end(), ' '), table\_number.end());

}

// 提取顾客人数

else if (line.find("姓名:") != string::npos) {

customer\_count = stoi(line.substr(line.find(":") + 1));

}

// 查找菜品表头

if (line.find("菜品名") != string::npos) {

// 跳过表头行和分隔线

getline(file, line);

while (getline(file, line) && !line.empty()) {

stringstream ss(line);

string name, special\_requests, type, status;

double price;

int quantity;

// 按固定宽度解析各列

name = line.substr(0, 30);

name.erase(remove(name.begin(), name.end(), ' '), name.end());

string price\_str = line.substr(30, 10);

price = stod(price\_str);

string quantity\_str = line.substr(40, 10);

quantity = stoi(quantity\_str);

special\_requests = line.substr(50, 20);

special\_requests.erase(remove(special\_requests.begin(), special\_requests.end(), ' '), special\_requests.end());

type = line.substr(70, 20);

type.erase(remove(type.begin(), type.end(), ' '), type.end());

status = line.substr(90, 10);

status.erase(remove(status.begin(), status.end(), ' '), status.end());

// 如果菜品状态是 "已完成"，则存储到数组

if (status == "已完成" && start\_index < MAX\_item) {

dishes[start\_index++] = { name, table\_number, price, quantity, special\_requests, type, status };

}

}

}

}

file.close();

return 0;

}

// 清空文件内容

void clearFileContent(const string& filepath) {

ofstream file(filepath, ofstream::out | ofstream::trunc);

if (!file.is\_open()) {

cerr << "无法清空文件: " << filepath << endl;

}

file.close();

}

// 从文件读取餐位费

double readSeatingFee(const string& filepath) {

ifstream file(filepath);

if (!file.is\_open()) {

cerr << "无法打开餐位费文件: " << filepath << endl;

return 0.0;

}

double seating\_fee = 0.0;

file >> seating\_fee;

file.close();

return seating\_fee;

}

// 预览小票

void previewReceipt(const Dish dishes[], int count, int customer\_count, double seating\_fee) {

double total\_price = 0.0;

// 计算总价，仅计算状态为 "已完成" 的菜品

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

if (dishes[i].status == "已完成") {

total\_price += dishes[i].price \* dishes[i].quantity;

}

}

// 如果总价不足500元，增加餐位费

if (total\_price < 500) {

total\_price += customer\_count \* seating\_fee;

}

// 输出小票内容到控制台

cout << "================ 小票预览 ================\n";

cout << "餐桌号: " << (count > 0 ? dishes[0].table\_number : "未知") << "\n\n";

cout << left << setw(30) << "菜品名" << setw(10) << "单价" << setw(10) << "数量" << setw(20) << "特殊要求" << setw(10) << "状态" << "\n";

cout << "---------------------------------------------\n";

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

cout << left << setw(30) << dishes[i].name

<< setw(10) << fixed << setprecision(2) << dishes[i].price

<< setw(10) << dishes[i].quantity

<< setw(20) << dishes[i].special\_requests

<< setw(10) << dishes[i].status << "\n";

}

cout << "---------------------------------------------\n";

if (total\_price < 500) {

cout << "餐位费: " << fixed << setprecision(2) << customer\_count \* seating\_fee << " 元\n";

}

cout << "总价: " << fixed << setprecision(2) << total\_price << " 元\n";

cout << "=============================================\n";

}

// 生成小票并写入文件

void generateReceipt(const string& outputPath, const Dish dishes[], int count, int customer\_count, const string& payment\_method, double cash, double seating\_fee) {

ofstream receipt(outputPath);

if (!receipt.is\_open()) {

cerr << "无法生成小票文件: " << outputPath << endl;

return;

}

double total\_price = 0.0;

// 计算总价，仅计算状态为 "已完成" 的菜品

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

if (dishes[i].status == "已完成") {

total\_price += dishes[i].price \* dishes[i].quantity;

}

}

// 如果总价不足500元，增加餐位费

double seating\_fee\_total = 0.0;

if (total\_price < 500) {

seating\_fee\_total = customer\_count \* seating\_fee;

total\_price += seating\_fee\_total;

}

double total\_due = total\_price;

double change = 0.0;

// 如果是现金支付，计算找零

if (payment\_method == "现金") {

while (cash < total\_due) {

cerr << "现金不足以支付总额！请重新输入" << endl;

cin >> cash;

}

change = cash - total\_due;

}

// 输出小票内容

receipt << "================ 小票 ================\n";

receipt << "餐桌号: " << (count > 0 ? dishes[0].table\_number : "未知") << "\n";

receipt << "支付方式: " << payment\_method << "\n";

if (payment\_method == "现金") {

receipt << "实收金额: " << fixed << setprecision(2) << cash << " 元\n";

receipt << "找零金额: " << fixed << setprecision(2) << change << " 元\n";

}

receipt << "\n";

receipt << left << setw(30) << "菜品名" << setw(10) << "单价" << setw(10) << "数量" << setw(20) << "特殊要求" << setw(10) << "状态" << "\n";

receipt << "---------------------------------------------\n";

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

receipt << left << setw(30) << dishes[i].name

<< setw(10) << fixed << setprecision(2) << dishes[i].price

<< setw(10) << dishes[i].quantity

<< setw(20) << dishes[i].special\_requests

<< setw(10) << dishes[i].status << "\n";

}

receipt << "---------------------------------------------\n";

if (seating\_fee\_total > 0) {

receipt << "餐位费: " << fixed << setprecision(2) << seating\_fee\_total << " 元\n";

}

receipt << "总价: " << fixed << setprecision(2) << total\_due << " 元\n";

receipt << "=============================================\n";

receipt.close();

}

// 收银功能入口

void cashier\_recipet() {

string inputDir = "数据存储/订单数据/";

string outputDir = "数据存储/小票存储/";

string feeFile = "数据存储/餐位费.txt";

double seating\_fee = readSeatingFee(feeFile);

if (seating\_fee <= 0) {

cerr << "餐位费文件内容无效，操作取消。" << endl;

return;

}

::readallfile();

cout << "请输入要处理的订单编号: ";

string orderNumber;

cin >> orderNumber;

string inputFile = inputDir + orderNumber + ".txt";

// 检查订单文件是否存在

ifstream checkFile(inputFile);

if (!checkFile.is\_open()) {

cerr << "订单文件 " << inputFile << " 不存在！" << endl;

return;

}

checkFile.close();

Dish dishes[MAX\_item];

int dish\_count = 0;

int customer\_count = 0; // 顾客人数

// 读取订单文件

if (readWaiterFile(inputFile, dishes, dish\_count, customer\_count) == 0) {

// 展示小票预览

previewReceipt(dishes, dish\_count, customer\_count, seating\_fee);

// 确认是否生成小票

cout << "是否确认生成小票？(输入 Y 确认，其他键取消): ";

string userInput;

cin >> userInput;

if (userInput != "Y" && userInput != "y") {

cout << "操作已取消。" << endl;

return;

}

// 选择支付方式

string payment\_method;

double cash\_amount = 0.0;

cout << "请选择支付方式 (1: 现金, 2: 支付宝, 3: 微信, 4: 信用卡): ";

int payment\_choice;

cin >> payment\_choice;

switch (payment\_choice) {

case 1:

payment\_method = "现金";

cout << "请输入现金支付金额: ";

cin >> cash\_amount;

break;

case 2:

payment\_method = "支付宝";

break;

case 3:

payment\_method = "微信";

break;

case 4:

payment\_method = "信用卡";

break;

default:

cerr << "无效的支付方式！操作已取消。" << endl;

return;

}

string outputFile = outputDir + orderNumber + ".txt";

// 生成小票

generateReceipt(outputFile, dishes, dish\_count, customer\_count, payment\_method, cash\_amount, seating\_fee);

// 清空订单文件

clearFileContent(inputFile);

cout << "订单 " << orderNumber << " 已生成小票: " << outputFile << endl;

}

else {

cout << "没有需要处理的菜品，或订单为空！" << endl;

}

}

}

directory.h

void directory()

{

while (1)

{

cout << "请选择你的身份\n";

cout << "1:顾客点餐\n"

"2:收营员\n"

"3:厨师\n"

"4:服务员\n"

"5:餐厅老板\n"

"6:系统管理员\n"

"7:关于本系统\n"

"8、顾客预定餐位\n"

"9、预定餐位兑现\n"

"0:退出程序\n";

int n = int\_input(9);

system("cls");

switch (n)

{

case 1:costomer::customer(); break;

case 2:cashier::cashier(); break;

case 3:cook::cook(); break;

case 4:server::server(); break;

case 5:boss::boss(); break;

case 6:administrator::administrator(); break;

case 7:about::about(); break;

case 8:costomer::book\_customer();break;

case 9:costomer::book\_customer\_arrival(); break;

case 0:return ;

default:

break;

}

system("cls");

}

}

读取文件.h

#pragma once

using namespace std;

void readFileByLine(string filePath) {

ifstream file(filePath);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filePath << endl;

return;

}

string line;

while (getline(file, line)) {

cout << line << endl;

}

file.close();

}

//读取数字

int readNumberFromFile(const string& filename) {

ifstream inputFile(filename);

if (!inputFile) {

cerr << "无法打开文件：" << filename << endl;

return -1; // 返回-1表示错误

}

int number;

inputFile >> number;

if (inputFile.fail()) {

cerr << "文件中没有有效数字或读取失败。" << endl;

return -1; // 返回-1表示错误

}

inputFile.close();

return number;

}

// 修改文件中的数字

void writeNumberToFile(const string& filename, int newNumber) {

ofstream outputFile(filename);

if (!outputFile) {

cerr << "无法打开文件：" << filename << endl;

return;

}

outputFile << newNumber;

if (outputFile.fail()) {

cerr << "写入文件失败。" << endl;

}

outputFile.close();

}

密码.h

#pragma once

namespace sys

{

using namespace std;

void password(string name)

{

cout << "请输入" << name << "的密码"<<endl;

for (int i = 4; i >= 0; i--)

{

string input;

cin >> input;

if (checkPassword(name,input))

return;

if(i>0)

cout << "密码错误,还有" << i << "次机会";

}

cout << "密码输入次数过多，老弟，程序已被强制退出，请联系管理员重置密码";

exit(0);

}

}

namespace sys {

// 检查密码是否正确

bool checkPassword(const string& username, const string& inputPassword) {

string filepath = "数据存储/账号/" + username + ".txt";

ifstream file(filepath);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filepath << endl;

return false;

}

string storedPassword;

getline(file, storedPassword);

file.close();

return inputPassword == storedPassword;

}

// 修改密码

bool changePassword(const string& username, const string& newPassword) {

string filepath = "数据存储/账号/" + username + ".txt";

ofstream file(filepath, ofstream::out | ofstream::trunc);

if (!file.is\_open()) {

cerr << "无法打开文件: " << filepath << endl;

return false;

}

file << newPassword;

file.close();

return true;

}

void cpass1(const string& username)//管理员端

{

string in;

cout << "请输入新密码";

cin >> in;

changePassword(username, in);

return;

}

void cpass2(const string& username)

{

void password(string username);

string in;

cout << "请输入新密码";

cin >> in;

changePassword(username, in);

return;

}

}

排序,h

#pragma once

template<typename T, typename Compare>

void bubbleSort(T\* head, T\* tail, Compare comp) {

for (T\* i = head; i != tail; ++i) {

for (T\* j = i + 1; j <= tail; ++j) {

if (comp(\*j, \*i)) {

T temp = \*i;

\*i = \*j;

\*j = temp;

}

}

}

}

输入判断.h

#pragma once

void inputnotright() {

cout << "\n请输入正确的数字: ";

cin.clear(); // 重置输入流状态

cin.ignore(max\_int, '\n'); // 清理缓冲区

}

int int\_input(int input\_max,int input\_min)

{

int n;

while(1)

{

if (cin >> n &&cin.good() && n >= 0 && n <= input\_max && n >= input\_min)

{

return n;

}

else

{

inputnotright();

}

}

}

void separator(int n)

{

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

{

cout << "-";

}

cout << endl;

}

展示全部文件.h

#pragma once

namespace fs = std::filesystem;

void readallfile(std::string path) {

if (fs::exists(path) && fs::is\_directory(path)) {

std::cout << "目录中的非空文件: " << path << "\n";

for (const auto& entry : fs::directory\_iterator(path)) {

if (fs::is\_regular\_file(entry.path())) {

// 打开文件并检查是否为空

std::ifstream file(entry.path(), std::ios::ate); // 直接定位到文件末尾

if (file.is\_open() && file.tellg() > 0) { // 文件打开成功且非空

std::cout << entry.path().filename() << '\n';

}

}

}

}

else {

std::cerr << "路径不存在或不是目录: " << path << "\n";

}

}

后记：现在是2025年1月3日星期五0:18，终于写完报告文档了！！！！我开心吗？true.