项目说明文档

数据结构课程设计

——银行业务系统

作 者 姓 名：张翔

学　　号：2352985

指 导 教 师：张颖

学院、 专业：计算机科学与技术学院 软件工程

同济大学

Tongji University

二〇二四 年 十二 月 七 日

1 项目分析

1.1 项目背景分析

随着社会经济的快速发展，银行已成为人们日常生活中不可或缺的一部分，提供多种金融服务以满足客户多样化的需求。然而，在高峰时段或业务繁忙的情况下，如何有效地管理银行业务窗口的服务流程，提升客户满意度，成为银行亟需解决的问题。由于不同业务窗口的处理能力和效率可能存在差异，这种差异会直接影响顾客的服务体验与业务完成的顺序。

因此，银行业务窗口是客户服务的核心环节，不同窗口的处理效率差异可能导致排队秩序与完成顺序不一致，影响客户体验。为优化服务流程，提高窗口资源利用率，合理设计服务调度机制尤为重要。本项目聚焦于银行业务场景，通过模拟窗口服务差异和调度逻辑，探索如何高效、公平地管理业务完成顺序。

1.2 项目需求分析

基于以上背景分析，本项目需要实现需求如下:

(1)初始化银行业务系统，根据办理业务不同，引导顾客进行排队

(2)根据不同窗口业务办理速度的不同，判断顾客完成办理的顺序

(3)根据情况判断是否要处理下一波顾客

(4)异常处理机制

1.3 项目功能分析

1.3.1 银行排队功能

系统需要模拟顾客根据办理业务的种类，进行合理的排队。根据银行各个窗口的业务类型与处理能力，系统将顾客按业务种类分配至相应窗口，并自动排队。

1.3.2 银行业务处理功能

系统将根据各个窗口的业务办理速度，模拟业务办理过程，判断顾客完成办理的顺序。窗口在办理业务时，系统需要记录每个顾客的业务办理进度，自动更新各窗口的空闲状态。

1.3.3.异常处理功能

此功能需要应对程序运行时可能出现的异常，如顾客人数输入错误，顾客编号发生重复，判断是否要处理下一波顾客时

2 项目设计

2.1 类和结构体设计

2.1.1 MyLinkNode 类的设计

2.1.1.1 概述

MyLinkNode 是一个模板结构体，表示链表节点。它包含两个成员：存储节点的数据和指向下一个节点的指针，它定义了两个构造函数，一个为默认，另一个为有参数输入。

2.1.1.2 类定义

```
template <typename Type>
struct MyLinkNode
{
    Type data;
    MyLinkNode<Type>* link;
    MyLinkNode(MyLinkNode<Type>* ptr = nullptr) : link(ptr) {}
    MyLinkNode(const Type& item, MyLinkNode<Type>* ptr = nullptr) : data(item),
link(ptr) {}
};
```

2.1.2 MyQueue 类的设计

2.1.2.1 概述

MyQueue 是一个基于链表实现的模板队列类，包含 front 和 rear 指针分别指向队列的头部和尾部，count 用于记录队列中元素的个数。该类提供了基本的队列操作，包括构造函数和析构函数。通过 isEmpty()判断队列是否为空,Size()返回队列中元素的数量。enQueue()用于将元素加入队尾，deQueue()从队头移除并返回元素，getHead()获取队头元素。

2.1.2.2 类定义

```
template <typename Type>
class MyQueue
{
private:
    MyLinkNode<Type>* front;
```

```
    MyLinkNode<Type>* rear;
    int count;
public:
    MyQueue() : front(nullptr), rear(nullptr), count(0) {}
    ~MyQueue() { makeEmpty(); }
    bool isEmpty() const;
    void makeEmpty();
    int Size()const;
    void enQueue(const Type& item);
    bool deQueue(Type& item);
    bool getHead(Type& item);
};
```

2.1.3 Bank 类的设计

2.1.3.1 概述

Bank 类模拟了一个银行排队系统，使用两个队列 queueA 和 queueB 来分别存储排队的客户。该类包括构造函数和析构函数，初始化时不做任何操作。lineUp()方法用于将客户排队到相应的队列中，dealWith()方法则用于处理排队中的客户，执行服务操作。receiveCommand()方法接收并处理命令，根据不同的指令控制排队和服务流程。

2.1.3.2 类定义

```
class Bank
{
private:
    MyQueue<int> queueA;
    MyQueue<int> queueB;
public:
    Bank() { ; }
    ~Bank() { ; }
    void lineUp();
    void dealWith();
    bool receiveCommand();
```

};

2.2 项目主体架构设计

项目主题架构设计为:

(1)银行业务处理系统启动;

(2)提示输入顾客总人数和每个人的编号;

(3)进行排队操作和业务处理操作;

(3)如果用户选择继续处理下一批顾客，那么重复(2)(3)操作;

(4)如果用户选择退出，则跳出循环，程序运行结束并退出。

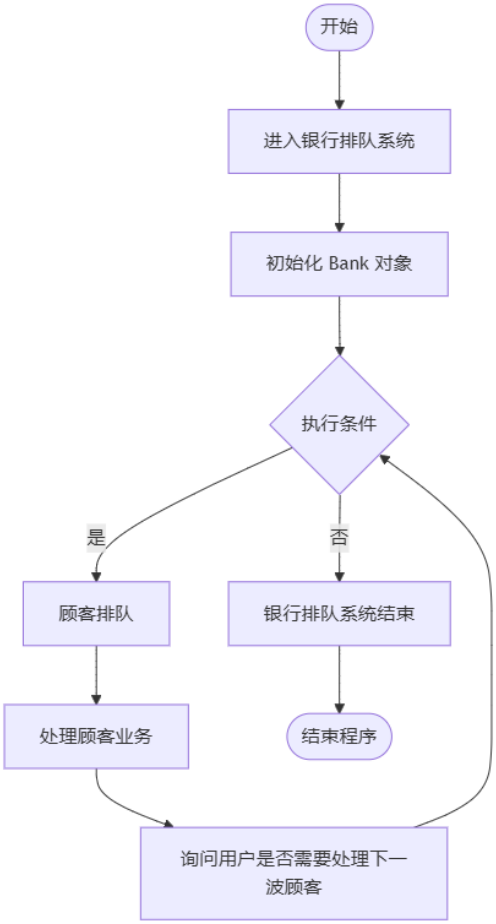


图 2.3.1 项目主体架构设计流程图

3 项目功能实现

3.1 项目主体架构实现

3.1.1 项目主体框架实现思路

这段 main()函数的具体实现思路如下:

(1)通过输出欢迎信息提示用户进入银行排队系统。

(2)创建一个 Bank 类的对象 bank，并进入一个循环。在循环中，首先调用 bank.lineUp()方法让客户排队，接着调用 bank.dealWith()方法处理排队中的客户。

(3)循环继续进行，直到 bank.receiveCommand()返回 false，即用户选择结束操作。最后，输出结束信息，表示银行排队系统结束。

3.1.2 项目主体框架核心代码

```
int main()
{
    std::cout << std::endl << ">>> 欢迎进入银行排队系统!" << std::endl;
    Bank bank;
    do {
        bank.lineUp();
        bank.dealWith();
    } while (bank.receiveCommand());
    std::cout << std::endl << ">>> 银行排队系统结束，感谢使用!" << std::endl;
    return 0;
}
```

3.2 银行排队功能实现

3.2.1 银行排队功能实现思路

Bank::lineUp()函数的具体实现思路如下:

(1)函数提示用户输入排队的顾客人数,并通过 inputInteger()函数获取一个有效的顾客人数。

(2)程序提示用户依次输入每个顾客的编号，并确保输入的是正整数。如果输入无效，程序会提示错误并重新输入。

(3)在输入顾客编号后，程序根据顾客编号的奇偶性将顾客分别排入两个不同的队列：奇数编号的顾客进入队列 A，偶数编号的顾客进入队列 B。

3.2.2 银行排队功能核心代码

```
void Bank::lineUp()
  {
      std::cout << std::endl;
      int count = inputInteger(1, 1000, "顾客人数");
      std::cout << "请依次输入顾客编号(正整数):";
      for (int i = 0; i < count; i++) {
          unsigned int customerNum;
          std::cin >> customerNum;
          while (std::cin.fail()) {
              std::cin.clear();
              std::cerr << "输入无效，请输入一个正整数:" << std::endl;
              std::cin >> customerNum;
          }
          if (customerNum % 2 == 1) {
              queueA.enQueue(customerNum);
          }
          else {
              queueB.enQueue(customerNum);
          }
      }
      std::cin.clear();
      std::cin.ignore(INT_MAX, '\n');
  }
```

3.3 银行业务处理功能实现

3.3.1 银行业务处理实现思路

Bank::dealWith()函数的具体实现思路如下:

(1)输出提示信息，表示开始处理顾客队列。

(2)进入一个循环，持续处理两个队列中的顾客，直到所有顾客都被处理完毕。在每次循环中，先处理队列 A 中的顾客，通过 deQueue()移除顾客并输出其编号。如果队列 A 为空且队列 B 还有顾客，程序继续处理队列 B 中的顾客。(处理过程中，顾客编号之间以空格分隔，最后如果所有队列都为空，则输出换行。)

(3)循环持续进行，直到两个队列都为空，最后输出提示信息，表示所有顾客已处理完毕，当前没有顾客排队。

3.3.2 银行业务处理核心代码

```
void Bank::dealWith()
{
    std::cout << std::endl << ">>> 开始处理顾客队列..." << std::endl;
    std::cout << std::endl << ">>> ";
    while (!queueA.isEmpty() || !queueB.isEmpty()) {
        for (int i = 0; i < PROCESSSPEED_A && !queueA.isEmpty(); i++) {
            int customer;
            queueA.deQueue(customer);
            std::cout << customer;
            if (queueA.isEmpty() && queueB.isEmpty()) {
                std::cout << std::endl;
                break;
            }
            else {
                std::cout << " ";
            }
        }
        for (int i = 0; i < PROCESSSPEED_B && !queueB.isEmpty(); i++) {
            int customer;
            queueB.deQueue(customer);
            std::cout << customer;
            if (queueA.isEmpty() && queueB.isEmpty()) {
```

```
                std::cout << std::endl;
                break;
            }
            else {
                std::cout << " ";
            }
        }
    }
    std::cout << std::endl << ">>> 已处理完毕，当前没有顾客排队。" << std::endl;
}
```

3.4 异常处理功能

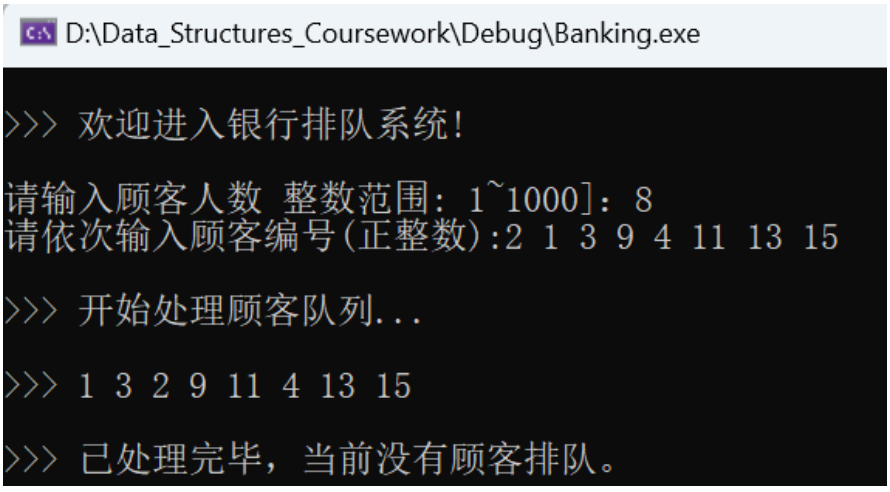
在MyQueue类中创建实例时,程序使用new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL 或 nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1)向标准错误流 std::cerr 输出一条错误消息"Error: Memory allocation failed.";

(2)调用 exit 函数，返回错误码为-1，用于指示内存分配错误，并导致程序退出。

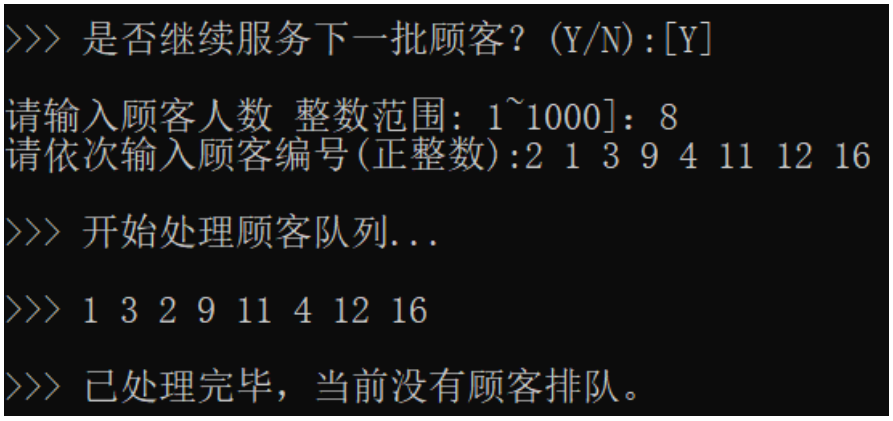
4 项目测试

4.1 银行排队和业务处理测试



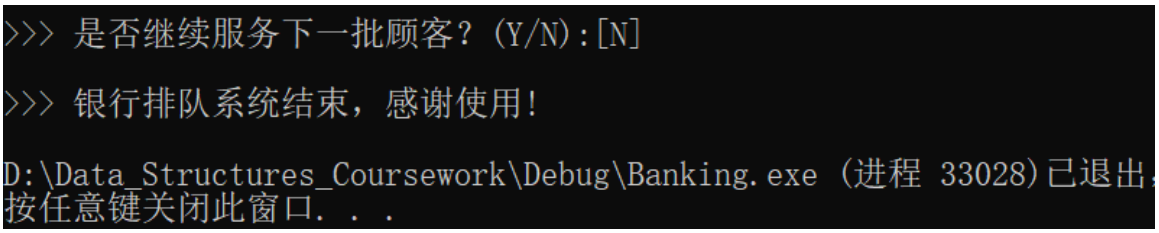
4.1.1 银行排队和业务处理示例

4.2 继续办理业务测试



4.2.1 继续办理业务处理示例

4.3 程序退出测试



4.3.1 程序退出示例

5 集成开发环境与编译运行环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：Microsoft Visual Studio 2022 (Release 模式)

Windows 编译运行环境：本项目适用于 x86 架构和 x64 架构