2024 语言课程设计实验报告

一、 题目（把题目复制一遍。如果是自拟的题目，把题目具体内容写在这里）

模拟一个手机银行的基本功能

二、 设计思路（这里分几部分：1、C++部分，包括几个类，各个类的功能，以及类的相互关系；2、Qt 部分，用了什么框架，设计了几个 ui，每个 ui 控件的功能，以及他们的相互关系；3、程序所完成的功能）

1. C++部分

（1）Customer 类

定义了一个客户类，涵盖了客户的基本信息、交易记录、账户操作和存款相关的功能。以下是每个方法的功能描述：

- 构造函数：Customer(QString acc, QString n, QString pwd, QString addr, QString depType, double rate, QString empID)：用于初始化一个客户对象，设置账户号码、姓名、密码、地址、存款类型、利率和员工 ID。
- 基本信息获取与设置方法：
  - QString getAccountNumber() const 和 void setAccountNumber(QString &acc)：获取和设置账户号码。
  - QString getName() const 和 void setName(const QString &name)：获取和设置客户姓名。
  - QString getPassword() const 和 void setPassword(const QString &pwd)：获取和设置账户密码。
  - QString getAddress() const 和 void setAddress(const QString &addr)：获取和设置客户地址。
  - QString getDepositType() const：获取存款类型。
  - double getInterestRate() const：获取利率。
  - bool isLost() const 和 void setLost(bool lost)：判断客户账户是否被报告丢失及设置状态。
  - QDate getReportLostDate() const：获取报告丢失的日期。
- 报告丢失：void reportLoss(QDate date)：报告账户丢失并记录日期。
- 交易记录管理：
  - void addTransaction(const Transaction &transaction)：添加交易记录。
  - QList<Transaction> getTransactions() const：获取所有交易记录。
- 账户余额和利息计算：
  - double calculateInterest() const：计算利息。
  - double getBalance() const 和 void setBalance(double balance)：获取和设置账户余额。
  - double calculateCurrentAccountBalance() const：计算活期

账户余额。

- void updateDeposits()：更新到期的定期存款利息。

- 存款、取款和转账操作：
  - void deposit(double amount, QString term, QString employeeID)：存款操作。
  - bool withdraw(double amount, QString employeeID)：取款操作。
  - bool transfer(double amount, Customer &receiver, QString employeeID)：转账操作。

这些方法为客户的账户管理提供了全面的支持，包括信息维护、交易处理、账户操作和丢失报告等功能

（2）Customermanager 类

onlineBankLogin(const QString& username, const QString& password)：验证线上银行用户的登录凭证，返回是否登录成功。

offlineBankLogin(const QString& username, const QString& password)：验证线下银行用户的登录凭证，返回是否登录成功。

addCustomer(const Customer &customer)：将一个新客户添加到客户管理列表中。

findCustomerByAccountNumber(const QString &accountNumber)：通过账户号码查找并返回对应的客户指针。

getAllCustomers() const：获取所有客户的信息列表。

removeCustomer(const QString& accountNumber)：根据账户号码删除客户记录，返回是否删除成功。

updateCustomerFile()：更新客户文件，以保存最新的客户数据

（3）Transaction 类

定义了一个交易记录类，涵盖了交易的基本信息以及一些额外的功能。以下是每个方法和属性的功能描述：

- **构造函数**：Transaction(QDate d, double amt, QString type, QString empID, QString term, double rate, const QDate &maturityDate = QDate())：初始化一个交易记录对象，设置交易日期、金额、交易类型、员工 ID、存款期限、利率以及到期日期（默认为空日期）。
- **获取方法**：
  - QDate getDate() const：获取交易日期。
  - double getAmount() const：获取交易金额。
  - QString getType() const：获取交易类型（如存款、取款）。
  - QString getEmployeeID() const：获取操作该交易的员工 ID。
  - QString getTerm() const：获取存款期限（如活期、1 年、3 年等）。
  - double getInterestRate() const：获取每笔存款的利率。
  - QDate getMaturityDate() const：获取到期日期。
- **设置方法**：
  - void setMaturityDate(const QDate &date)：设置到期日期。
  - void setAmount(double amt)：设置交易金额。
  - void setType(const QString &type)：设置交易类型。
  - void setTerm(const QString &term)：设置存款期限。

- void setInterestRate(double interestRate)：设置利率。

- **比较运算符：**
  - bool operator==(const Transaction& other) const：用于比较两个交易记录是否相同。
- **状态属性：**
  - bool getIsConverted() const：获取交易记录是否已转换的状态。
  - void setIsConverted(bool converted)：设置交易记录是否已转换的状态。

这个类涵盖了交易记录的基本信息和一些附加属性，如到期日期和转换状态。它支持对交易记录的详细管理和操作。

（4）Transactionmanager 类

定义了一个用于管理交易的类，负责执行交易操作。以下是每个部分的功能描述：

- **构造函数**：TransactionManager(CustomerManager &custManager)：构造函数接收一个 CustomerManager 对象的引用，用于在交易操作中访问和管理客户信息。
- **performTransaction 方法**：bool performTransaction(const QString &accountNumber, const Transaction &transaction)：执行指定账户的交易操作。根据账户号码和交易记录执行相应的交易操作（如存款、取款等），返回操作是否成功。
- **私有成员变量：**
  - CustomerManager &customerManager：引用 CustomerManager 对象，以便在交易过程中进行客户信息的检索和更新。

这个类将负责处理交易的执行，并利用 CustomerManager 来访问和管理客户账户信息。

（5）Filemanager 类

定义了一个用于处理数据存储和加载的类，支持将客户数据保存到文件和从文件加载数据。以下是每个部分的功能描述：

- **构造函数**：FileManager()：初始化 FileManager 对象。
- **saveData 方法**：bool saveData(const QString &fileName, const CustomerManager &customerManager)：将 CustomerManager 中的客户数据保存到指定的文件中。这个方法返回一个布尔值，表示数据是否成功保存。
- **loadData 方法**:bool loadData(const QString &fileName, CustomerManager &customerManager)：从指定的文件中加载客户数据，并更新 CustomerManager 对象。这个方法返回一个布尔值，表示数据是否成功加载。

2.Qt 类

一共 16 个 ui

（1）Adminpanel.ui

定义了一个管理员面板，用于通过用户界面对客户进行管理操作。以下是每个部分的功能描述：

- 构造函数：AdminPanel(CustomerManager *manager, QWidget *parent = nullptr)：管理员面板的构造函数，接收一个 CustomerManager 指针用于管理客户信息，并初始化 UI。
- 析构函数：~AdminPanel()：用于释放 UI 资源。
- 私有槽函数：

- void on_searchButton_clicked()：当用户点击 "搜索" 按钮时触发，可能用于根据某种条件查找客户。
- void on_editButton_clicked()：当用户点击 "编辑" 按钮时触发，允许管理员对选中的客户信息进行编辑。
- void on_backToLoginButton_clicked()：当用户点击 "返回登录" 按钮时触发，返回到登录界面。

- 私有成员变量：
  - Ui::AdminPanel *ui：指向 UI 界面的指针，用于管理界面元素。
  - CustomerManager *manager：指向 CustomerManager 对象的指针，用于访问和管理客户信息。
  - Customer* currentCustomer：用于保存当前选中的客户信息，在搜索或编辑时使用。

该类的目的是提供一个图形用户界面，允许管理员进行客户的查找和编辑等操作。每个按钮都有对应的槽函数，用于处理用户的操作请求。

（2）Bankselectiondialog.ui

定义了一个用于选择银行模式的对话框，用户可以在选择线上或线下银行之后进行进一步操作。以下是每个部分的功能描述：

- 构造函数：BankSelectionDialog(CustomerManager* manager, QWidget *parent = nullptr)：用于初始化银行选择对话框，接受一个 CustomerManager 指针，用于后续的客户管理操作，并初始化 UI 界面。
- 析构函数：~BankSelectionDialog()：用于释放 UI 资源。
- 私有槽函数：
  - void on_OfflineButton_clicked()：当用户点击 "线下银行" 按钮时触发，可能打开线下银行登录界面。
  - void on_OnlineButton_clicked()：当用户点击 "线上银行" 按钮时触发，可能打开线上银行登录界面，并根据需要禁用某些功能（如存款和取款）。
- 私有成员变量：
  - Ui::BankSelectionDialog *ui：指向 UI 界面的指针，用于管理界面元素。
  - CustomerManager *manager：指向 CustomerManager 对象的指针，用于管理客户登录和相关操作。

该类提供了一个简单的界面，让用户选择使用线下或线上银行的服务。根据用户的选择，系统会进入相应的操作模式，例如在线上银行中禁用某些功能。

（3）Converttocurrentdialog.ui

定义了一个用于将定期存款转换为活期存款的对话框。以下是每个部分的功能描述：

- 构造函数：ConvertToCurrentDialog(QWidget *parent = nullptr, Customer *customer = nullptr)：用于初始化转换对话框，接收一个父窗口指针和一个 Customer 指针，后者用于获取和操作客户的存款信息。
- 析构函数：~ConvertToCurrentDialog()：用于释放 UI 资源。
- 私有槽函数：
  - void on_convertButton_clicked()：当用户点击 "转换" 按钮时触

发，执行将选定的定期存款转换为活期存款的操作。

- void on_pushButton_clicked()：用于其他按钮的操作，例如关闭对话框或返回上一级界面。
- void populateDepositList()：将客户的定期存款信息加载到界面上（例如列表视图），以供用户选择要转换的存款。

- 私有成员变量：
  - Ui::ConvertToCurrentDialog *ui：指向 UI 界面的指针，用于管理界面元素。
  - Customer *customer：用于存储客户信息的指针，便于获取客户的定期存款信息并进行操作。

这个类提供了一个功能界面，允许用户选择并将定期存款转换为活期存款。populateDepositList() 方法可以用于填充用户的定期存款列表，方便用户选择要转换的存款。on_convertButton_clicked() 负责处理转换逻辑，例如更新客户账户中的存款类型。

（4）Converttofixeddialog.ui

定义了一个用于将活期存款转换为定期存款的对话框。以下是每个部分的功能描述：

- 构造函数：ConvertToFixedDialog(QWidget *parent = nullptr, Customer *customer = nullptr)：初始化转换对话框，接收一个父窗口指针和一个 Customer 指针，后者用于获取和操作客户的存款信息。
- 析构函数：~ConvertToFixedDialog()：用于释放 UI 资源。
- 私有槽函数：
  - void on_convertButton_clicked()：当用户点击 "转换" 按钮时触发，执行将选定的活期存款转换为定期存款的操作。
  - void on_pushButton_clicked()：用于处理关闭对话框或返回上一级界面等操作。
- 私有成员变量：
  - Ui::ConvertToFixedDialog *ui：指向 UI 界面的指针，用于管理界面元素。
  - Customer *customer：用于存储客户信息的指针，用于访问客户的活期存款信息并进行操作。

该类提供了一个转换界面，允许用户将其活期存款转换为定期存款。on_convertButton_clicked() 槽函数中应包含具体的转换逻辑，例如更新客户的存款类型、存款期限等。

（5）Depositdialog.ui

定义了一个用于处理存款操作的对话框，用户可以通过此对话框输入存款金额、选择存款期限，并获取存款账户信息。以下是每个部分的功能描述：

- 构造函数：DepositDialog(QWidget *parent = nullptr)：用于初始化存款对话框，接收一个父窗口指针，创建 UI 元素。
- 析构函数：~DepositDialog()：用于释放 UI 资源。
- 公有方法：
  - double getDepositAmount() const：返回用户输入的存款金额。
  - QString getDepositTerm() const：返回用户选择的存款期限（如活期、定期 1 年等）。

- QString getAccountNumber() const：返回用户输入的存款账户号码。

- 私有成员变量：
  - Ui::DepositDialog *ui：指向 UI 界面的指针，用于管理对话框的界面元素。

该类的主要作用是通过提供一个界面，允许用户输入相关存款信息，并通过 getDepositAmount()、getDepositTerm() 和 getAccountNumber() 方法获取这些信息，以便后续处理存款操作。

（6）Depositselectiondialog.ui

定义了一个对话框，用于让用户从定期存款列表中选择一个存款。以下是各个部分的功能描述：

- 构造函数：DepositSelectionDialog(const QList<Transaction>& deposits, QWidget *parent = nullptr)：
  - 接收一个定期存款列表 deposits，用于显示在对话框中，供用户选择。
  - 接收一个父窗口指针 parent，用于初始化 UI。
- 析构函数：~DepositSelectionDialog()：用于释放 UI 资源。
- 公有方法：
  - Transaction getSelectedDeposit() const：返回用户从列表中选择的定期存款交易记录 (Transaction)。
    - 这个方法会在用户确认选择后被调用，用于获取选定的存款信息。
- 私有成员变量：
  - Ui::DepositSelectionDialog *ui：指向 UI 界面的指针，用于管理对话框的界面元素。
  - QListWidget *listWidget：一个用于显示定期存款列表的 QListWidget，用户可以从中选择一个存款。
  - QList<Transaction> depositList：存储传入的定期存款列表，用于显示在 listWidget 中。

功能：

该类的主要功能是提供一个选择界面，让用户从多个定期存款中选择一个。通过 getSelectedDeposit() 方法，用户可以获取其选择的存款信息，之后可以进一步进行定期转活期等操作。

（7）Editcustomerdialog.ui

定义了一个用于编辑客户信息的对话框，包括更改密码的功能。以下是各个部分的功能描述：

- 构造函数：EditCustomerDialog(Customer *customer, QWidget *parent = nullptr)：
  - 接收一个 Customer 指针 customer，用于加载和更新客户信息。
  - 接收一个父窗口指针 parent，用于初始化 UI。
- 析构函数：~EditCustomerDialog()：用于释放 UI 资源。
- 私有槽函数：
  - void on_changePasswordButton_clicked()：当用户点击 "更改密码

″ 按钮时触发，处理更改密码的操作。这通常包括验证原密码和设置新密码的逻辑。

- void on_cancelButton_clicked()：当用户点击 ″取消″ 按钮时触发，关闭对话框或返回到前一个界面，放弃对客户信息的更改。

- 私有成员变量：
  - Ui::EditCustomerDialog *ui：指向 UI 界面的指针，用于管理对话框的界面元素。
  - Customer *customer：用于存储客户信息的指针，便于加载和更新客户信息。

功能：

该类提供了一个界面，允许管理员或用户对客户信息进行编辑。最常见的操作包括更改密码，通过 on_changePasswordButton_clicked() 方法处理这些操作。on_cancelButton_clicked() 方法则用于取消当前操作，关闭对话框。

（8）Edituserdialog.ui

定义了一个用于编辑客户信息的对话框，主要用于保存用户的更改。以下是各个部分的功能描述：

- **构造函数**：EditUserDialog(Customer* customer, QWidget *parent = nullptr)：
  - 接收一个指向 Customer 的指针 customer，用于加载当前用户信息并进行编辑。
  - 接收一个父窗口指针 parent，用于初始化对话框的 UI 元素。
- **析构函数**：~EditUserDialog()：用于释放 UI 资源。
- **私有槽函数**：
  - void on_saveButton_clicked()：当用户点击 ″保存″ 按钮时触发，处理保存用户信息的操作。通常包括将对用户信息的更改（如用户名、地址等）应用到 Customer 对象中。
- **私有成员变量**：
  - Ui::EditUserDialog *ui：指向 UI 界面的指针，用于管理界面元素。
  - Customer* customer：指向需要编辑的客户信息的指针，用于加载和修改用户的详细信息。

功能：

该类提供一个界面供用户编辑自己的信息，并通过点击 ″保存″ 按钮来应用更改。在 on_saveButton_clicked() 函数中,通常会将用户的修改保存回 Customer 对象中。

（9）LoadingDialog.ui

定义了一个用于显示加载动画的对话框,常用于在程序执行耗时操作时告知用户加载状态。以下是各个部分的功能描述：

- **构造函数**：LoadingDialog(QWidget *parent = nullptr)：
  - 接收一个父窗口指针 parent，用于初始化加载对话框。
  - 构造函数中通常会设置加载动画或进度条，并显示在对话框中。
- **析构函数**：~LoadingDialog()：用于释放 UI 资源，确保当加载完成时正确销毁对话框。
- **私有成员变量**：

- Ui::LoadingDialog *ui：指向 UI 界面的指针，用于管理加载对话框的界面元素。

功能：

LoadingDialog 主要用于显示一个加载中的状态，通常会结合一个动画（例如 QMovie）来让用户看到进度或动画指示正在加载的内容。常用于网络请求、文件读取等耗时操作时的反馈。

（10）Login.ui

定义了一个用户登录界面，用于管理用户通过线上或线下银行的登录操作。以下是类的功能描述：

构造函数：

- Login(CustomerManager* manager, bool isOnline, QWidget *parent = nullptr)：
  - 接收一个指向 CustomerManager 的指针 manager，用于管理客户数据（如验证登录信息）。
  - isOnline 布尔值用于区分是线上银行登录还是线下银行登录。
  - parent 是父窗口指针，用于初始化界面元素。

析构函数：

- ~Login()：用于释放 UI 资源。

私有槽函数：

- void on_loginButton_clicked()：
  - 处理用户点击 "登录" 按钮的逻辑。通常包含用户身份验证过程，通过与 CustomerManager 交互来验证输入的用户名和密码。
  - 根据 isOnline 的值决定调用线上或线下银行的登录逻辑。
- void on_registerButton_clicked()：
  - 处理用户点击 "注册" 按钮的操作，通常会跳转到用户注册界面，允许用户创建新账户。

私有成员变量：

- Ui::Login *ui：指向 UI 界面的指针，用于管理登录窗口的界面元素。
- CustomerManager* manager：指向 CustomerManager 的指针，用于管理登录时所需的客户信息和登录验证。
- bool isOnline：标识当前是线上银行还是线下银行登录，影响登录逻辑和可能的功能可用性。

功能：

该类提供一个界面，让用户可以通过输入账号和密码进行登录。根据 isOnline 的值，决定是进行线上银行的登录还是线下银行的登录，并且支持用户注册新账户的功能。

（11）Mainwindow.ui

MainWindow 类是银行系统的主界面，提供各种客户操作功能，如存款、取款、查询、挂失等。MainWindow 类是银行系统的核心操作界面，提供了多个按钮和选项，允许客户进行各种账户管理和操作。通过 setCurrentCustomer 方法，可以在用户登录或选择账户后，将相关信息传递给界面，以便进行存款、取款、转账等操作。各个按钮点击事件会调用相应的槽函数处理具体的业务逻辑。以下是类的功能描述：

构造函数：

- MainWindow(CustomerManager* manager, bool isOnline, QWidget *parent = nullptr)：
    - 接收 CustomerManager 指针 manager，用于管理系统中的客户信息和操作。
    - 布尔值 isOnline 用于标识当前是否是线上银行登录，以调整可用的功能和显示。
    - parent 是父窗口指针，通常用于 UI 初始化。

析构函数：

- ~MainWindow()：销毁窗口并释放 UI 资源。

公共方法：

- void setCurrentCustomer(Customer* customer)：
    - 设置当前操作的客户信息 currentCustomer,用于执行操作时指定客户。

私有槽函数：

- void on_exitButton_clicked()：
    - 处理点击 "退出" 按钮的逻辑，通常会关闭程序或返回登录界面。
- void on_depositButton_clicked()：
    - 处理点击 "存款" 按钮的逻辑，弹出存款对话框，让用户进行存款操作。
- void on_withdrawButton_clicked()：
    - 处理点击 "取款" 按钮的逻辑，执行取款操作。
- void on_queryButton_clicked()：
    - 处理点击 "查询" 按钮的逻辑，查询客户账户的相关信息或交易记录。
- void on_reportLostButton_clicked()：
    - 处理点击 "挂失" 按钮的逻辑，执行账户挂失操作。
- void on_settingsButton_clicked()：
    - 处理点击 "设置" 按钮的逻辑，进入客户账户的设置界面。
- void on_convertToFixedButton_clicked()：
    - 处理点击 "转为定期存款" 按钮的逻辑，允许用户将活期存款转换为定期存款。
- void on_convertToCurrentButton_clicked()：
    - 处理点击 "转为活期存款" 按钮的逻辑，允许用户将定期存款转换为活期存款。
- void on_transferButton_clicked()：
    - 处理点击 "转账" 按钮的逻辑，允许用户执行账户间的资金转账操作。
- void on_registerButton_clicked()：
    - 处理点击 "注册" 按钮的逻辑，允许用户注册新账户。
- void on_inquireButton_clicked()：
    - 处理点击 "询问" 按钮的逻辑，执行客户信息或账户状态的查询。
- void on_closeAccountButton_clicked()：
    - 处理点击 "销户" 按钮的逻辑，允许用户关闭或注销账户。

私有成员变量：

- Ui::MainWindow *ui：指向 UI 界面的指针，用于管理主窗口的界面元素。
- CustomerManager* manager：用于管理客户的相关操作，如登录、添加客户、查找客户等。
- Customer* currentCustomer：用于保存当前登录或操作的客户。
- bool isOnline:用于标识当前是线上还是线下银行登录,影响功能的可用性。

私有方法：

- void updateUI()：
  - 更新主界面的显示内容，根据当前登录的客户和功能状态调整界面。

功能：

MainWindow 类提供了一系列用户操作的入口，包括存款、取款、转账、挂失、账户查询和设置等。不同功能按钮对应不同的客户操作。根据 isOnline 参数，可以控制某些功能的启用与禁用，适应线上或线下银行的不同需求。

（12）Querydialog.ui

QueryDialog 类是用于显示客户账户信息和交易记录的对话框窗口。以下是该类的功能描述：

构造函数：

- QueryDialog(Customer *customer, QWidget *parent = nullptr)：
  - 接收一个 Customer 指针 customer，用于获取当前客户的账户信息和交易记录。
  - parent 是父窗口指针，用于 UI 初始化。

析构函数：

- ~QueryDialog()：用于销毁窗口并释放资源。

私有成员变量：

- Ui::QueryDialog *ui：指向 UI 界面的指针，用于管理 QueryDialog 的界面元素。
- Customer *currentCustomer：用于保存传入的当前客户信息的指针，以便在界面上显示相关信息。

私有方法：

- void displayCustomerInfo()：
  - 用于在对话框中显示当前客户的详细信息，如账户余额、存款类型、利率、挂失状态等。
  - 该方法可以通过 currentCustomer 对象获取客户的详细信息，并将其更新到 UI 界面中显示。

功能：

QueryDialog 类的主要功能是展示与当前客户相关的账户信息。它能够通过 Customer 类对象获取客户的个人资料、账户余额、交易历史等信息，并将这些信息通过 UI 界面显示给用户。

这个类主要适用于银行系统中客户查询功能，帮助客户查看账户详情、余额以及其他重要的账户信息。

（13）Registerdialog.ui

RegisterDialog 类用于银行系统中的客户注册功能，允许用户创建新账户并生成唯一的账号。以下是该类的功能描述：

构造函数：

- RegisterDialog(CustomerManager* manager, QWidget *parent = nullptr)：
  - 接收 CustomerManager 指针 manager，用于管理客户信息的操作。
  - parent 参数用于指定父窗口。

析构函数：

- ~RegisterDialog()：用于释放资源并销毁对话框。

公有成员变量：

- CustomerManager* manager：指向客户管理器对象的指针，用于管理和处理客户数据。
- QString accountNumber：用于保存新生成的唯一账号。

公有方法：

- QString generateUniqueAccountNumber()：
  - 生成一个唯一的账号字符串，用于新注册的客户。
  - 通常通过某种逻辑（如随机数、时间戳等）确保账号的唯一性。
- void displayAccountNumber()：
  - 将生成的账号显示在注册对话框的 UI 上，方便用户确认。

私有槽函数：

- void on_buttonBox_accepted()：
  - 当用户在对话框中点击确认按钮时，执行的槽函数。
  - 负责获取用户输入的信息，创建新的 Customer 对象，并将其添加到 CustomerManager 中。
  - 还可以调用 generateUniqueAccountNumber() 生成新账号。

私有成员变量：

- Ui::RegisterDialog *ui：指向 UI 界面对象的指针，用于管理对话框的界面元素。

信号：

- void registrationSuccess()：
  - 注册成功时发射的信号，可能用于通知主界面更新或执行其他操作。

功能：

- RegisterDialog 类提供了注册新客户的界面和功能，用户可以通过填写表单信息创建一个新的银行账户。
- 该类还具有生成唯一账号的功能，确保每个客户都有独立的账户号码。
- 注册完成后，通过信号通知其他窗口或类更新系统状态。

此类在银行管理系统中用于客户注册功能，确保用户能够顺利创建账户，并且生成的账号是唯一的。

（14）Reportlostdialog.ui

ReportLostDialog 类提供了一个挂失对话框界面，用于处理银行账户的挂失操作。以下是该类的功能描述：

构造函数：

- ReportLostDialog(QWidget *parent = nullptr)：
  - 初始化对话框对象，parent 参数用于指定父窗口。默认值为 nullptr，即没有父窗口。

析构函数：

- ~ReportLostDialog()：用于释放资源并销毁对话框。

私有槽函数：

- void on_confirmButton_clicked()：
  - 当用户点击“确认”按钮时执行的槽函数。
  - 该函数处理账户挂失操作，通常会将用户的账户状态设置为挂失。
- void on_cancelButton_clicked()：
  - 当用户点击“取消”按钮时执行的槽函数。
  - 关闭对话框，不进行任何挂失操作。

私有成员变量：

- Ui::ReportLostDialog *ui：指向 UI 界面对象的指针，用于管理挂失对话框的界面元素。

功能：

- ReportLostDialog 类为用户提供了一个简洁的界面,以便进行账户挂失操作。
- 用户可以通过点击“确认”按钮来执行挂失流程,或者点击“取消”按钮来中止挂失。
- 在挂失成功后，系统可以禁用某些功能（如存款、取款等），确保账户安全。

此类是银行管理系统中挂失功能的核心部分，负责处理用户发起的挂失请求，并相应更新账户状态。

（15）Transferdialog.ui

TransferDialog 类是一个处理银行转账功能的对话框界面，用于用户输入转账相关信息，如接收方账号和转账金额。以下是该类的功能描述：

构造函数：

- TransferDialog(QWidget *parent = nullptr)：
  - 初始化对话框对象,parent 参数用于指定父窗口,默认为 nullptr。
  - 提供转账界面供用户输入信息，如接收方 ID 和转账金额。

析构函数：

- ~TransferDialog()：用于销毁对话框对象并释放资源。

公共方法：

- QString getReceiverID() const：
  - 返回用户输入的接收方账户 ID。
  - 该方法用于获取用户输入的转账接收方账户号。
- double getTransferAmount() const：
  - 返回用户输入的转账金额。
  - 该方法用于获取用户输入的转账金额值。

私有槽函数：

- void on_okButton_clicked()：
  - 当用户点击“确认”按钮时执行的槽函数。
  - 验证用户输入的账户号和转账金额是否有效,并在成功验证后关闭对话框。
  - 若输入无效，可能会弹出提示框警告用户。

私有成员变量：

- Ui::TransferDialog *ui：指向 UI 界面对象的指针，用于管理界面元素。

功能：

- TransferDialog 类提供了一个用户友好的界面，用于执行转账操作。
- 用户在对话框中输入接收方账号和转账金额，点击确认后触发验证并进行下一步操作。
- 如果输入有误，系统会通过提示框告知用户错误信息，以便重新输入。

此类为银行系统中的转账功能提供了界面支持。

（16）Withdrawdialog.ui

WithdrawDialog 类是一个用于银行系统中提现操作的对话框界面，允许用户输入提现金额以及账号信息。其功能简述如下：

构造函数：

- WithdrawDialog(QWidget *parent = nullptr)：
  - 初始化对话框对象，parent 参数用于指定父窗口，默认为 nullptr。
  - 提供提现的输入界面，用户可以在其中输入提现的相关信息。

析构函数：

- ~WithdrawDialog()：
  - 析构函数，用于销毁对话框对象并释放相关资源。

公共方法：

- double getWithdrawAmount() const：
  - 获取用户输入的提现金额。
  - 该方法返回一个 double 类型的金额值，用于后续的提现操作。
- QString getAccountNumber() const：
  - 获取用户输入的账户号码。
  - 该方法返回一个 QString 类型的账号，用于确定从哪个账户进行提现操作。

私有成员变量：

- Ui::WithdrawDialog *ui：指向 UI 界面对象的指针，用于管理界面元素。

功能：

- WithdrawDialog 类为提现操作提供了界面支持，用户可以通过此界面输入提现金额和账户信息。
- 通过 getWithdrawAmount() 和 getAccountNumber() 两个方法，系统能够获取用户输入的数据，随后进行提现处理。

三、 文件格式描述（详细描述所保存和读取的文件格式）

以 txt 格式存储数据

1. 存储的客户信息格式：

每一行表示一个客户的信息，字段用逗号分隔，字段内容包括：

账户号码（Account Number）
姓名（Name）
密码（Password）
地址（Address）
存款类型（Deposit Type）
利率（Interest Rate）

挂失状态（Lost Status）：布尔值

挂失日期（Report Lost Date）：如果账户未挂失，日期字段为 "0000-00-00"；否则，为实际挂失日期

余额（Balance）

交易记录数量（Transaction Count）

2. 存储的交易记录格式：

客户信息行之后，紧跟着该客户的交易记录。每条交易记录占用一行，字段通过逗号分隔，字段内容包括：

交易日期（Date）

交易金额（Amount）

交易类型（Type）

员工 ID（Employee ID）

存款期限（Term）

利率（Interest Rate）

到期日期（Maturity Date）

文本文件格式：每个客户信息和交易记录按行保存，每个客户的基本信息和交易记录分开，且客户信息优先保存。

四、 具体实现过程（详细描述按照什么顺序进行程序搭建，在实现中碰到的问题以及解决的方法）

银行系统的具体实现过程：

在构建一个基于 Qt 的银行系统时，我们需要考虑用户的交互流程、界面的组织以及后台逻辑。以下是该银行系统的详细实现过程，包括从结构搭建、类设计、功能实现到问题解决的步骤。

1. 项目架构设计

由于这是一个银行管理系统，主要包含几个核心模块：

- 用户管理模块 (CustomerManager, Customer)
- 交易管理模块 (Transaction, TransactionManager)
- 文件管理模块 (FileManager)
- 界面交互模块 （多个 QDialog 类，例如 MainWindow, DepositDialog, WithdrawDialog 等)

核心类设计：

- Customer 类：用于表示每个用户的信息及其操作，如存款、取款、修改密码等。
- CustomerManager 类：用于管理所有用户，包括登录、查找、添加、删除用户等操作。
- Transaction 类：用于表示每笔交易的详细信息。
- TransactionManager 类：用于处理交易并记录到对应的用户上。
- FileManager 类：负责从文件加载/保存用户信息（后期可以改为数据库）。

2. 界面设计和交互

我们使用 Qt Designer 来设计界面，并通过编写 C++ 代码处理界面背后的逻辑。

主窗口 (MainWindow)

- 这是整个系统的核心界面。它提供了存款、取款、查询、转账等主要功能的入口。
- 用户在登录成功后进入此界面，通过点击不同的按钮执行相应的操作。

登录窗口 (Login)

- Login 类用于区分线上和线下银行的登录。线上银行的登录会限制某些功能（例如禁用存款、取款），而线下银行则提供完整的功能。
- 在此过程中，需要注意密码验证和用户的在线/离线状态管理。

主要功能窗口：

- 存款对话框 (DepositDialog)：用户输入存款金额和存款期限。通过 getDepositAmount 和 getDepositTerm 获取用户输入的数据，并将这些信息传递给对应的用户对象进行存款操作。
- 取款对话框 (WithdrawDialog)：用户输入取款金额和账户号码。通过 getWithdrawAmount 和 getAccountNumber 实现提现功能。
- 转账对话框 (TransferDialog)：实现从一个用户向另一个用户的资金转账，支持输入接收方 ID 和转账金额。
- 挂失对话框 (ReportLostDialog)：允许用户挂失账户，在挂失后该账户无法进行取款、转账等操作。
- 账户查询对话框 (QueryDialog)：显示用户的基本信息以及交易记录。

3. 核心功能实现

3.1. 用户管理模块实现

- Customer 类：
  - 定义了基本的用户信息，如账户号、姓名、地址、密码等。
  - 通过 deposit() 和 withdraw() 等方法实现存取款功能。
  - 通过 addTransaction() 记录每笔交易。
- CustomerManager 类：
  - 负责管理所有用户，保存所有用户的列表 (QList<Customer>)，提供 addCustomer()、findCustomerByAccountNumber()、onlineBankLogin()、offlineBankLogin() 等功能。
  - 登录方法需要通过用户名和密码进行验证，并返回用户对象用于后续的操作。

3.2. 交易管理模块实现

- Transaction 类：
  - 定义了交易的基本信息，如日期、金额、类型、利率等。每个用户的存取款行为都被记录为一个 Transaction 对象。
- TransactionManager 类：
  - 用于处理交易。performTransaction() 方法接收用户的账户号码和交易对象，找到对应用户后执行交易并更新记录。

3.3. 文件管理模块实现

- FileManager 类：
  - 负责从文件加载和保存用户数据。在系统初始化时，从文件中读取用户列表，并在操作完成后保存数据。
  - 由于 Qt 的数据库模块暂未启用，所有用户信息保存在文本文件中，使用 QTextStream 进行读写。

4. 问题解决与优化

4.1. 问题：账户登录与用户验证

- 问题描述：最初的系统中，用户的登录功能仅通过简单的用户名和密码验证，这在多个用户登录时出现冲突。
- 解决方案：通过 CustomerManager 实现了区分线上和线下登录的功能，线上登录禁用存取款等功能。并通过返回登录的 Customer* 指针对象来精确操作不同的用户。

4.2. 问题：交易记录与存款类型

- 问题描述：在实现定期存款和活期存款的转换时，无法区分用户的不同定期存款记录。
- 解决方案：为 Transaction 类添加了 term 和 maturityDate 属性，记录每笔定期存款的期限和到期时间。并通过 DepositSelectionDialog 让用户选择具体哪一笔存款来进行操作。

4.3. 问题：读取数据时会出现错误

- 问题描述：在实现数据读取时为设置的账户余额，日期等会出现空白或乱码
- 解决方案：在 loaddata 中加入了 qdebug<<进行异常检测，一次一次找出读取错误的位置并设置默认数据解决错误

4.4. 问题：文件保存和读取的完整性

- 问题描述：在保存和读取用户数据时，某些交易信息可能未完整记录。
- 解决方案：通过 FileManager 类中的 saveData() 和 loadData() 方法，确保将每个用户的所有信息（包括交易记录）保存到文件中。

4.5. 问题：界面更新与状态同步

- 问题描述：当用户进行某些操作后（如存款、取款、挂失），界面状态未及时更新。
- 解决方案：引入了 updateUI() 方法，在 MainWindow 中调用此方法以更新界面元素，确保每次操作后用户看到的界面是最新的。

4.6. 问题：多窗口之间的数据传递

- 问题描述：多个窗口之间需要传递用户数据，例如从登录界面跳转到主界面时需要将用户信息传递过来。
- 解决方案：通过构造函数传递 CustomerManager 和 Customer 对象，并使用 setter 方法设置当前用户。例如在 MainWindow 中调用 setCurrentCustomer() 方法来设置当前登录用户

4.7. 问题：退出注册页面闪退

- 问题描述：创建用户成功后点击 ok 会闪退。
- 解决方案：在最后的时候使用 try catch 语句进行检测

总结：

构建银行管理系统的核心步骤包括用户管理、交易处理、文件存储以及界面交互。在实现过程中，我们通过设计多个类来组织系统逻辑，并通过 Qt Designer 创建用户界面。在实际开发中，解决了用户登录验证、交易记录管理、存取款操作中的问题，并通过合理的类设计和方法调用实现了整个银行系统的功能。

五、 测试报告（对主要功能进行测试，可以列表表示各项功能的完成情况，

也可以贴图表示）

在对银行管理系统的测试中，主要针对系统的核心功能模块，包括用户登录、存款、取款、转账、账户查询、挂失等操作进行了详细测试。以下是对各项功能的测试结果。

测试环境：

- **操作系统：** Windows 11
- **Qt 版本：** 5.12.14
- **编译器：** MinGW 64-bit
- **数据库：** 文件存储

1. **用户登录测试**

功能	测试用例	期望结果	实际结果	结论
用户登录	输入正确的用户名和密码	成功登录并进入主界面	成功	通过
用户登录（错误）	输入错误的用户名或密码	显示错误提示并保持在登录界面	成功	通过
线上银行登录	选择线上银行并成功输入正确的用户名和密码	登录成功并禁用存款取款功能	成功	通过
线下银行登录	选择线下银行并成功输入正确的用户名和密码	登录成功并启用所有功能	成功	通过

2. **存款功能测试**

功能	测试用例	期望结果	实际结果	结论
活期存款	输入存款金额并选择活期存款	存款成功，账户余额增加	成功	通过
定期存款	输入存款金额并选择定期存款，输入期限	存款成功，定期账户增加相应记录	成功	通过
存款记录查询	存款后在查询功能中查看交易记录	交易记录显示正确的存款信息	成功	通过

3. **取款功能测试**

功能	测试用例	期望结果	实际结果	结论
活期取款	输入取款金额并选择活	取款成功，账户余额减	成功	通过

	期存款	少		
定期取款	选择定期存款并尝试取款	显示错误提示（定期存款不能直接取款）	成功	通过
定期转活期取款	将定期存款转换为活期后进行取款	取款成功，定期余额减少，活期增加	成功	通过

4. **转账功能测试**

功能	测试用例	期望结果	实际结果	结论
转账功能	输入接收方账户号码和转账金额	转账成功，发送方余额减少，接收方余额增加	成功	通过
错误账户转账	输入不存在的接收方账户号码	显示错误提示并拒绝转账	成功	通过

5. **账户查询功能测试**

功能	测试用例	期望结果	实际结果	结论
基本信息查询	登录成功后点击查询按钮，查看用户基本信息	显示用户名、账户余额、定期存款等详细信息	成功	通过
交易记录查询	查询账户中的所有交易记录（存款、取款、转账）	显示完整交易记录	成功	通过

6. **账户挂失功能测试**

功能	测试用例	期望结果	实际结果	结论
账户挂失	选择挂失账户并确认挂失	账户挂失成功，禁用存款、取款功能	成功	通过
挂失后操作	尝试对挂失账户进行取款或转账操作	显示错误提示并拒绝操作	成功	通过

7. **账户管理功能测试**

功能	测试用例	期望结果	实际结果	结论

修改密码	输入原密码并设置新密码	密码修改成功	成功	通过
关闭账户	成功登录后关闭账户	账户关闭，所有数据清除	成功	通过

六、 心得体会和相关建议（课程小结和对课程改进的建议）

通过这个银行管理系统的开发，我深刻体会到 Qt 框架在 C++开发中强大的跨平台 UI 开发能力和高效性。这个项目不仅让我掌握了如何设计和实现一个实际应用中的 GUI 系统，还在以下几个方面给了我很大的收获：

- 项目规划与模块化设计：从项目需求的分析、模块划分，到具体的功能实现，我学会了如何在一个大型项目中规划功能模块，并合理安排开发顺序。这帮助我提升了项目管理和代码组织能力。
- 面向对象设计与编程：通过设计多个类（如 CustomerManager、DepositDialog、MainWindow 等），我更加熟悉了面向对象编程的原则和实践。这些类的设计让我理解了如何通过类与类之间的交互来实现复杂的业务逻辑。
- Qt 信号与槽机制的运用：Qt 独特的信号与槽机制让事件处理变得非常灵活，我通过这个项目加深了对信号与槽的理解，掌握了如何在 GUI 应用程序中处理用户输入和界面更新。
- 调试与问题解决能力的提升：在开发过程中遇到的各种问题，如 UI 更新延迟、数据传输错误等，都是实际开发中常见的挑战。通过调试工具（如 qDebug()）和自定义错误处理逻辑，我提升了问题诊断和解决的能力。

相关建议

在课程内容和项目开发过程中，我有以下一些改进建议：

- 课程内容多样化：课程中主要教授了 Qt 应用开发和 C++编程，但可以进一步增加其他技术栈（如 Python 和 Web 开发）的内容，帮助学生拓展技术视野。
- 更多的实践项目：实际项目开发是巩固编程能力的最佳途径，课程可以引入更多与现实生活密切相关的项目作为练习，不仅提升学习的趣味性，还能让学生具备应对真实世界开发挑战的能力。
- 数据库与安全性：在开发过程中，我发现数据存储方式以及安全性（如用户密码加密、数据校验等）对于一个银行系统尤为重要，课程可以进一步深入讲解如何引入 SQLite 数据库、数据加密等高级主题，以提升学生的全栈开发能力。
- 项目后期维护与优化：除了功能实现外，课程可以增加项目优化与维护的内容，例如如何提升系统性能、减少内存泄漏、代码重构等，让学生在编写代码的同时也关注代码的质量和可维护性。
-

总结

通过这个项目，我不仅掌握了 Qt 框架的基本使用方法，还提升了软件开发的整体素质。我非常感谢课程带来的丰富知识和实践经验，希望未来可以有更多更深入的课程和项目，帮助学生提升技术能力，为实际工作打下坚实的基础。