

北 京 邮 电 大 学

课 程 设 计 报 告

课程名称__面向对象程序设计实践（C++）__

__计算机__学院_2020211307__班

姓名__陈可儿__

_2022__年__5_月__3_日

一、基础实验

1、C++基础知识实验

实验笔记：

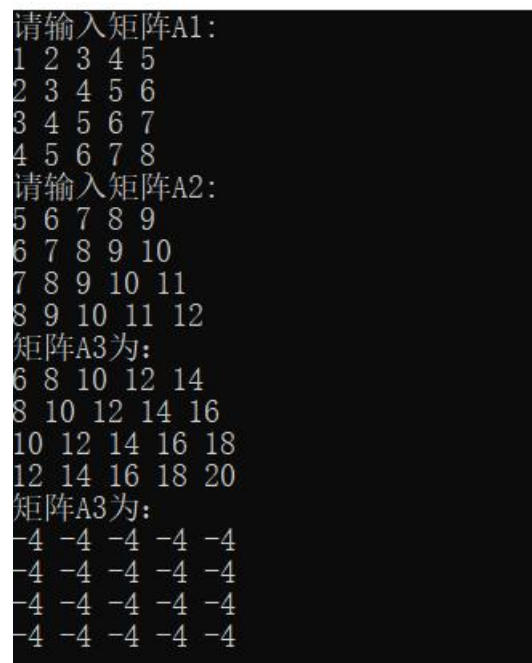
本次实验考察的是 C++ 输入输出，动态申请空间 new 的用法，以及释放所申请的空間的操作

动态申请矩阵块空间：使用 `new int* [ROW]`，将矩阵的每一行看作一个整体，new 出来的一段空间用来存放每行的首地址，矩阵 A1 (A2,A3) 存放这段空间的首地址，之后再使用 for 循环，为每个 A1[i]，即矩阵的每行动态申请空间，这时则需要申请一段长度为 col 的空间，`A1[i] = new int[COL]`，此时 A1[i] 即存放了每行的首地址。

矩阵相加相减和输入输出：使用两层 for 循环即可，外层循环行，内层循环列

释放矩阵空间：与动态申请的顺序相反，先 delete 每行的首地址 A1[i]，再 delete 存放这些首地址空间的首地址 A1.

运行结果截图：（先输出的为相加结果）



```
请输入矩阵A1:
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
请输入矩阵A2:
5 6 7 8 9
6 7 8 9 10
7 8 9 10 11
8 9 10 11 12
矩阵A3为:
6 8 10 12 14
8 10 12 14 16
10 12 14 16 18
12 14 16 18 20
矩阵A3为:
-4 -4 -4 -4 -4
-4 -4 -4 -4 -4
-4 -4 -4 -4 -4
-4 -4 -4 -4 -4
```

源代码：

```
#include<iostream>
using namespace std;

#define ROW 4
#define COL 5
//初始化
void initMatrix(int**& A1, int**& A2, int**& A3);
//矩阵相加
void addMatrix(int** A1, int** A2, int** A3);
//矩阵相减
```

```

void subMatrix(int** A1, int** A2, int** A3);
//输出
void outputMatrix(int** A);
//释放空间
void releaseMatrix(int**& A);
int main()
{
    int** A1, ** A2, ** A3;
    initMatrix(A1, A2, A3);

    addMatrix(A1, A2, A3);
    outputMatrix(A3);

    subMatrix(A1, A2, A3);
    outputMatrix(A3);

    releaseMatrix(A1);
    releaseMatrix(A2);
    releaseMatrix(A3);
    return 0;
}

void initMatrix(int**& A1, int**& A2, int**& A3)
{
    //开辟矩阵空间
    A1 = new int* [ROW];
    A2 = new int* [ROW];
    A3 = new int* [ROW];
    for (int i = 0; i < ROW; i++)
    {
        A1[i] = new int[COL];
        A2[i] = new int[COL];
        A3[i] = new int[COL];
    }
    cout << "请输入矩阵A1:" << endl;
    for (int i = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++)
            cin >> A1[i][j];
    }
    cout << "请输入矩阵A2:" << endl;
    for (int i = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++)

```

```

        cin >> A2[i][j];
    }
}

void addMatrix(int** A1, int** A2, int** A3)
{
    for (int i = 0; i < ROW; i++)
        for (int j = 0; j < COL; j++)
        {
            A3[i][j] = A1[i][j] + A2[i][j];
        }
}

void subMatrix(int** A1, int** A2, int** A3)
{
    for (int i = 0; i < ROW; i++)
        for (int j = 0; j < COL; j++)
        {
            A3[i][j] = A1[i][j] - A2[i][j];
        }
}

void outputMatrix(int** A)
{
    cout << "矩阵A3为: " << endl;
    for (int i = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++)
            cout << A[i][j] << " ";
        cout << endl;
    }
}

void releaseMatrix(int**& A)
{
    for (int i = 0; i < ROW; i++)
    {
        delete[] A[i];
    }
    delete[] A;
}

```

2、类与对象实验

(1)

实验笔记：

本实验考察类与对象的定义、初始化，构造析构函数等。

由题设，Point 类需要有两个成员变量（int 型）代表横纵坐标，需要一个成员函数（返回值为 double）计算两点间的距离，需要两种构造函数，一种默认初始化成员变量为 0，一种可供用户自行输入。由于题目要求说明每一次构造与析构函数调用是哪个对象的调用，并观察和解释相关调用顺序，所以在 Point 类的两个构造函数中加入相关输出信息，用以观察。圆形类，则需要圆心和半径两个成员变量，其中圆心就需要用到 Point 类，半径为 int 型，默认的构造函数和析构函数。

实验结果截图：

```
Point类默认构造函数
Circle类构造函数
Point类默认构造函数
Circle类构造函数
请输入第一个圆的圆心坐标(x, y):
4 5
请输入第一个圆的半径:
6
请输入第二个圆的圆心坐标(x, y):
10 12
请输入第二个圆的半径:
2
两圆相离
Circle类析构函数
Point类析构函数
Circle类析构函数
Point类析构函数
```

结论：当类中成员（Circle 类中成员圆心是 Point 类）是其他类对象时（即类中成员有对象成员），会先调用对象成员的构造，再调用本类构造，而析构顺序与构造相反

源代码：

Point 类：

.h:

```
class Point
{
public:
    int x, y;
    double getDistance(int x1, int y1, int x2, int y2);
    Point();
    Point(int a, int b);
```

```

    ~Point();
};

.cpp:
#include<iostream>
#include<cmath>
#include "Point.h"
using namespace std;
Point::Point()
{
    this->x = 0;
    this->y = 0;
    cout << "Point类默认构造函数" << endl;
}
Point::Point(int a, int b)
{
    this->x = a;
    this->y = b;
    cout << "Point类含参构造函数" << endl;
}
Point::~~Point()
{
    cout << "Point类析构函数" << endl;
}
double Point::getDistance(int x1, int y1, int x2, int y2)
{
    return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}

```

Circle 类:

.h:

```

class Circle
{
public:
    Point o;
    int r;
    Circle();
    ~Circle();
};

```

.cpp:

```

#include<iostream>
#include "Circle.h"
using namespace std;
Circle::Circle()
{
    this->o.x = 0;

```

```

        this->o.y = 0;
        cout << "Circle类构造函数" << endl;
    }
Circle::~~Circle()
{
    cout << "Circle 类析构函数" << endl;
}
main:
#pragma once
#include<iostream>
#include"Circle.h"
#include"Point.h"
using namespace std;
int main()
{
    Circle c1, c2;
    cout << "请输入第一个圆的圆心坐标(x, y):" << endl;
    cin >> c1.o.x >> c1.o.y;
    cout << "请输入第一个圆的半径:" << endl;
    cin >> c1.r;
    cout << "请输入第二个圆的圆心坐标(x, y):" << endl;
    cin >> c2.o.x >> c2.o.y;
    cout << "请输入第二个圆的半径:" << endl;
    cin >> c2.r;
    double distance = c1.o.getDistance(c1.o.x, c1.o.y, c2.o.x, c2.o.y);
    if (distance > c1.r + c2.r)
        cout << "两圆相离" << endl;
    else if (distance == c1.r + c2.r)
        cout << "两圆外切" << endl;
    else if (distance == abs(c1.r - c2.r))
        cout << "两圆内切" << endl;
    else if (distance < abs(c1.r - c2.r))
        cout << "两圆内含" << endl;
    else
        cout << "两圆相交" << endl;
    return 0;
}

```

(2)

实验笔记:

在实验一的基础上，定义矩阵类，构造函数用来动态申请矩阵空间（new，方法同实验一），析构函数释放矩阵空间（delete，方法同实验一），拷贝构造函数，需要完成申请空间和复制的功能，成员函数包括，输入矩阵元素，输出矩阵，矩阵相加和相减。

指向矩阵类的指针 pA1, pA2, pA3, 用 new 和矩阵类的拷贝构造函数完成初始化，用解引用得到矩阵对象进行相加相减。

```

22 23 24 25 26 27
0 0 0 0 0 0
12 24 38 56 82 4
6 -1 14 7 15 16
9 11 13 15 17 19
矩阵相减得到A3:
-6 -7 -8 -9 -10 -11
-4 -5 -6 -7 -8 -9
-2 -4 -6 -8 -10 -12
-8 -16 -22 -24 -18 124
10 15 -2 3 -7 -10
-3 -1 1 3 5 7
下面为利用拷贝构造函数进行初始化的A1、A2、A3
矩阵相加得到A3=: :
8 9 10 11 12 13
22 23 24 25 26 27
0 0 0 0 0 0
12 24 38 56 82 4
6 -1 14 7 15 16
9 11 13 15 17 19
矩阵相减得到A3=:
-6 -7 -8 -9 -10 -11
-4 -5 -6 -7 -8 -9
-2 -4 -6 -8 -10 -12
-8 -16 -22 -24 -18 124
10 15 -2 3 -7 -10
-3 -1 1 3 5 7

```

源代码

.h:

```

class Matrix
{
public:
    int** matrix = nullptr;
    int row, col;
    Matrix(int row, int col);
    ~Matrix();
    Matrix(Matrix& matrix);
    void inputMatrix();
    void outputMatrix();
    void addMatrix(Matrix A1, Matrix A2);
    void subMatrix(Matrix A1, Matrix A2);
};

```

.cpp:

```

#include<iostream>

```



```

#include<cstdlib>
#include "Matrix.h"

using namespace std;

//构造函数，申请矩阵空间
Matrix::Matrix(int row, int col)
{
    this->row = row;
    this->col = col;
    this->matrix = new int* [row];
    for (int i = 0; i < row; i++)
    {
        this->matrix[i] = new int[col];
    }
}

//析构函数，释放矩阵空间
Matrix::~Matrix()
{
    if (matrix != nullptr)
    {
        for (int i = 0; i < this->row; i++)
        {
            delete[]matrix[i];
        }
        delete[]matrix;
        this->matrix = nullptr;
    }
}

//拷贝构造函数, 申请和复制数组
Matrix::Matrix(Matrix& m)
{
    this->row = m.row;
    this->col = m.col;
    this->matrix = new int* [row];
    for (int i = 0; i < row; i++)
    {
        this->matrix[i] = new int[col];
    }
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            {

```

```

        this->matrix[i][j] = m.matrix[i][j];
    }
}

void Matrix::inputMatrix()
{
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
        {
            cin >> this->matrix[i][j];
        }
}

void Matrix::outputMatrix()
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cout << this->matrix[i][j] << " ";
        }
        cout << endl;
    }
}

void Matrix::addMatrix(Matrix A1, Matrix A2)
{
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            this->matrix[i][j] = A1.matrix[i][j] + A2.matrix[i][j];
}

void Matrix::subMatrix(Matrix A1, Matrix A2)
{
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            this->matrix[i][j] = A1.matrix[i][j] - A2.matrix[i][j];
}

main:
#pragma once
#include<iostream>
#include"Matrix.h"
using namespace std;

```

```

int main()
{
    int row, col;
    cout << "请输入行数: ";
    cin >> row;
    cout << "请输入列数: ";
    cin >> col;
    Matrix A1(row, col);
    Matrix A2(row, col);
    Matrix A3(row, col);
    cout << "请输入矩阵A1: " << endl;
    A1.inputMatrix();
    cout << "请输入矩阵A2: " << endl;
    A2.inputMatrix();

    A3.addMatrix(A1, A2);
    cout << "矩阵相加得到A3=: " << endl;
    A3.outputMatrix();
    A3.subMatrix(A1, A2);
    cout << "矩阵相减得到A3: " << endl;
    A3.outputMatrix();

    //使用拷贝构造函数初始化
    Matrix* pA1 = new Matrix(A1);
    Matrix* pA2 = new Matrix(A2);
    Matrix* pA3 = new Matrix(A3);
    cout << "下面为利用拷贝构造函数进行初始化的A1、A2、A3" << endl;
    pA3->addMatrix(*pA1, *pA2); //解引用得到对象
    cout << "矩阵相加得到A3=: " << endl;
    pA3->outputMatrix();
    pA3->subMatrix(*pA1, *pA2);
    cout << "矩阵相减得到A3: " << endl;
    pA3->outputMatrix();

    delete pA1;
    delete pA2;
    delete pA3;

    return 0;
}

```

3、继承与派生实验

实验笔记：

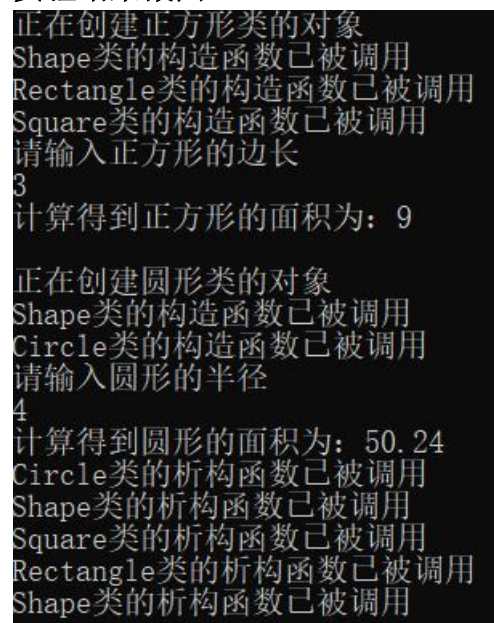
需要构造父类 Shape，Shape 类包括成员变量面积 area，成员函数用来计算形状的面积，由于有两种基本性质（矩形和圆形），需要对 getArea 函数进行重载，构造函数和析构函数用来输出相关信息进行调用的提示。

子类一 Rectangle 类，为 Shape 的派生，成员变量为矩形的长宽，构造函数和析构函数输出相关调提示。

子类二 Circle，为 Shape 的派生，成员变量为圆的半径，构造函数和析构函数输出相关调提示。

子类三 Square，为 Rectangle 的派生，成员变量为正方形边长，构造函数和析构函数输出相关调提示。

实验结果截图：



```
正在创建正方形类的对象
Shape类的构造函数已被调用
Rectangle类的构造函数已被调用
Square类的构造函数已被调用
请输入正方形的边长
3
计算得到正方形的面积为：9

正在创建圆形类的对象
Shape类的构造函数已被调用
Circle类的构造函数已被调用
请输入圆形的半径
4
计算得到圆形的面积为：50.24
Circle类的析构函数已被调用
Shape类的析构函数已被调用
Square类的析构函数已被调用
Rectangle类的析构函数已被调用
Shape类的析构函数已被调用
```

可以看出：创建正方形对象时，先调用了父类 Rectangle 的父类 Shape 的构造函数，再调用父类 Rectangle 的构造函数，最后才是子类 Square 的构造函数，而析构函数的顺序则相反。

源代码：

```
#include<iostream>
#define PI 3.14
using namespace std;
class Shape
{
public:
    double area;
    void getArea(int r);
    void getArea(int x, int y);
    Shape()
    {
        cout << "Shape类的构造函数已被调用" << endl;
```

```

    }
    ~Shape()
    {
        cout << "Shape类的析构函数已被调用" << endl;
    }
};

class Rectangle :public Shape
{
public:
    int x;
    int y;
    Rectangle()
    {
        cout << "Rectangle类的构造函数已被调用" << endl;
    }
    ~Rectangle()
    {
        cout << "Rectangle类的析构函数已被调用" << endl;
    }
};

class Circle :public Shape
{
public:
    int r;
    Circle()
    {
        cout << "Circle类的构造函数已被调用" << endl;
    }
    ~Circle() { cout << "Circle类的析构函数已被调用" << endl;
    }
};

class Square :public Rectangle
{
public:
    int a;
    Square()
    {
        cout << "Square类的构造函数已被调用" << endl;
    }
    ~Square()
    {

```

```

        cout << "Square类的析构函数已被调用" << endl;
    }
};

int main()
{
    cout << "正在创建正方形类的对象" << endl;
    Square square;
    cout << "请输入正方形的边长" << endl;
    cin >> square.a;
    square.getArea(square.a, square.a);
    cout << "计算得到正方形的面积为: " << square.area << endl;
    cout << endl;
    cout << "正在创建圆形类的对象" << endl;
    Circle circle;
    cout << "请输入圆形的半径" << endl;
    cin >> circle.r;
    circle.getArea(circle.r);
    cout << "计算得到圆形的面积为: " << circle.area << endl;
    return 0;
}

void Shape::getArea(int r)
{
    this->area = PI * r * r;
}

void Shape::getArea(int x, int y)
{
    this->area = x * y;
}

```

4、I/O 流实验

实验笔记:

使用 `srand()` 为初始化随机数发生器，用于设置 `rand()` 产生随机数时的种子，`srand((unsigned)time(NULL)); price = (rand() % 1001)` 来生成 `[0,1000]` 范围内的随机数作为价格。

`cin.fail()` 表示输入数据超过 `int_max`，用来判断一些不合法输入，`cin.clear()` 清除 `std::cin` 的错误状态，`cin.sync()` 和 `cin.ignore(10000, '\n')` 清空输入缓冲区

运行结果截图:

```
倒霉蛋，恭喜您被挑选参与竞猜价格游戏！
请输入您猜的价格：999
您猜的价格过高了！
请再次输入竞猜价格：777
您猜的价格过高了！
请再次输入竞猜价格：666
您猜的价格过高了！
请再次输入竞猜价格：200
您猜的价格过高了！
请再次输入竞猜价格：100
您猜的价格过高了！
请再次输入竞猜价格：10
您猜的价格过低了！
请再次输入竞猜价格：80
您猜的价格过高了！
请再次输入竞猜价格：60
您猜的价格过低了！
请再次输入竞猜价格：70
您猜的价格过高了！
请再次输入竞猜价格：66
您猜的价格过高了！
请再次输入竞猜价格：62
您猜的价格过低了！
请再次输入竞猜价格：63
您终于猜对了。
```

源代码：

```
#include<iostream>
#include<ctime>
#include<cstdlib>
#include<string>
using namespace std;
int main()
{
    cout << "倒霉蛋，恭喜您被挑选参与竞猜价格游戏！" << endl;
    srand((unsigned)time(NULL));
    int price = (rand() % 1001); //生成[0, 1000]范围内的随机数作为价格
    int guess_price;
    string trynumbera;
    cout << "请输入您猜的价格：";
    cin >> guess_price;
    if (cin.fail()) //输入数据不合法
    {
        cin.clear(); //清除std::cin的错误状态
        cin.sync(); //清空输入缓冲区
        cin.ignore(10000, '\n');
        cout << "您输入的不是合法数字！" << endl;
        cout << "请再次输入竞猜价格：";
        cin >> guess_price;
    }
}
```

```

while (guess_price != price)
{
    if (guess_price > price && guess_price < 1001 && guess_price > 0)
        cout << "您猜的价格过高了!" << endl;
    else if (guess_price < price && guess_price < 1001 && guess_price > 0)
        cout << "您猜的价格过低了!" << endl;
    else cout << "您猜的价格不在可能的商品价格区间[1, 1000]内!" << endl;
    cout << "请再次输入竞猜价格: ";
    cin >> guess_price;
}
cout << "您终于猜对了。" << endl;

return 0;
}

```

5、重载实验

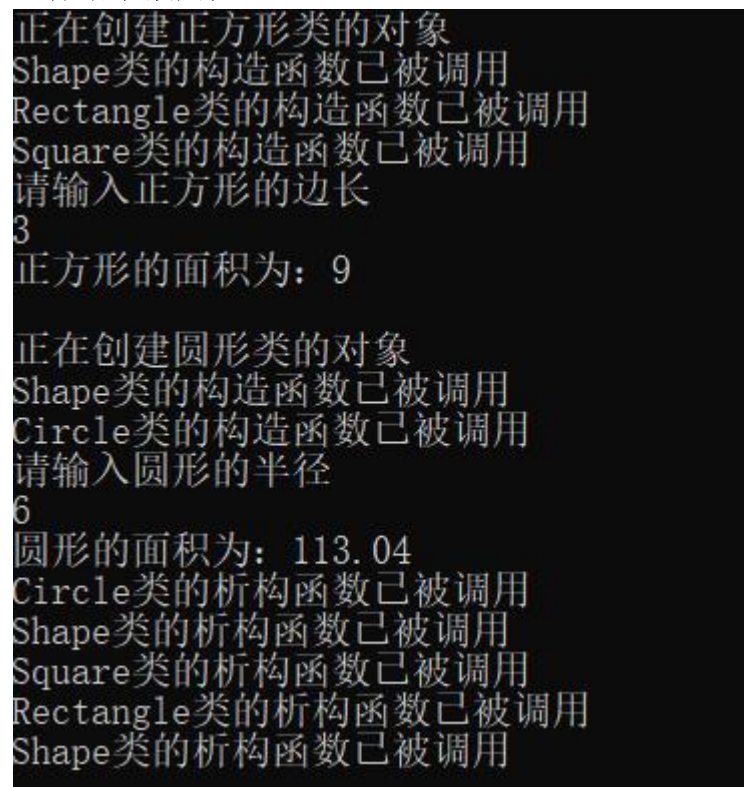
(1)

虚函数:

实验笔记:

在实验三基础上，将 Shape 类中 getArea 改成虚函数。

运行结果截图:



```

正在创建正方形类的对象
Shape类的构造函数已被调用
Rectangle类的构造函数已被调用
Square类的构造函数已被调用
请输入正方形的边长
3
正方形的面积为: 9

正在创建圆形类的对象
Shape类的构造函数已被调用
Circle类的构造函数已被调用
请输入圆形的半径
6
圆形的面积为: 113.04
Circle类的析构函数已被调用
Shape类的析构函数已被调用
Square类的析构函数已被调用
Rectangle类的析构函数已被调用
Shape类的析构函数已被调用

```

源代码:

```
#include<iostream>
```



```

#define PI 3.14
using namespace std;
class Shape
{
public:
    double area;
    virtual void getArea(int r);
    virtual void getArea(int x, int y);
    Shape()
    {
        cout << "Shape类的构造函数已被调用" << endl;
    }
    ~Shape()
    {
        cout << "Shape类的析构函数已被调用" << endl;
    }
};

class Rectangle :public Shape
{
public:
    int x;
    int y;
    Rectangle()
    {
        cout << "Rectangle类的构造函数已被调用" << endl;
    }
    ~Rectangle()
    {
        cout << "Rectangle类的析构函数已被调用" << endl;
    }
};

class Circle :public Shape
{
public:
    int r;
    Circle()
    {
        cout << "Circle类的构造函数已被调用" << endl;
    }
    ~Circle() {
        cout << "Circle类的析构函数已被调用" << endl;
    }
}

```

```

};

class Square :public Rectangle
{
public:
    int a;
    Square()
    {
        cout << "Square类的构造函数已被调用" << endl;
    }
    ~Square()
    {
        cout << "Square类的析构函数已被调用" << endl;
    }
};

int main()
{
    cout << "正在创建正方形类的对象" << endl;
    Square square;
    cout << "请输入正方形的边长" << endl;
    cin >> square.a;
    square.getArea(square.a, square.a);
    cout << "正方形的面积为: " << square.area << endl;

    cout << endl;
    cout << "正在创建圆形类的对象" << endl;
    Circle circle;
    cout << "请输入圆形的半径" << endl;
    cin >> circle.r;
    circle.getArea(circle.r);
    cout << "圆形的面积为: " << circle.area << endl;
    return 0;
}

void Shape::getArea(int r)
{
    this->area = PI * r * r;
}

void Shape::getArea(int x, int y)
{
    this->area = x * y;
}

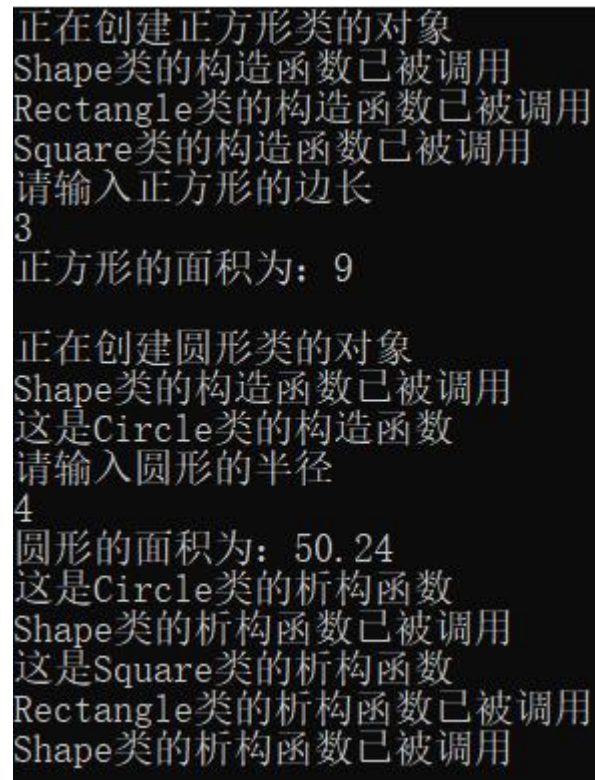
```

抽象类：

实验笔记：

抽象类为含有纯虚函数的类（在基类中仅仅给出声明，不对虚函数实现定义，而是在派生类中实现，纯虚函数需要在声明之后加个=0）。抽象类只能作为派生类的基类，不能定义对象，但可以定义指针。在派生类实现该纯虚函数后，定义抽象类对象的指针，并指向或引用子类对象。

运行结果截图：



正在创建正方形类的对象
Shape类的构造函数已被调用
Rectangle类的构造函数已被调用
Square类的构造函数已被调用
请输入正方形的边长
3
正方形的面积为：9

正在创建圆形类的对象
Shape类的构造函数已被调用
这是Circle类的构造函数
请输入圆形的半径
4
圆形的面积为：50.24
这是Circle类的析构函数
Shape类的析构函数已被调用
这是Square类的析构函数
Rectangle类的析构函数已被调用
Shape类的析构函数已被调用

源代码：

```
#include <iostream>
#include <cmath>
using namespace std;
#define PI 3.14

class Shape
{
public:
    virtual double getArea()=0;
    Shape()
    {
        cout << "Shape类的构造函数已被调用" << endl;
    }
    ~Shape()
    {
        cout << "Shape类的析构函数已被调用" << endl;
    }
};
```

```

    }
};

class Rectangle :virtual public Shape
{
public:
    int w, h;
    Rectangle(int _w = 0, int _h = 0) :w(_w), h(_h)
    {
        cout << "Rectangle类的构造函数已被调用" << endl;
    }
    ~Rectangle()
    {
        cout << "Rectangle类的析构函数已被调用" << endl;
    }
    double getArea()
    {
        return w * h;
    }
};

```

```

class Square :public Rectangle
{
public:
    int a;
    Square(int _a = 0) :a(_a)
    {
        cout << "Square类的构造函数已被调用" << endl;
    }
    ~Square()
    {
        cout << "这是Square类的析构函数" << endl; }
    double getArea()
    {
        return a * a;
    }
};

```

```

class Circle :virtual public Shape
{
public:
    int r;
    Circle(int _r = 0) :r(_r)
    {

```

```

        cout << "这是Circle类的构造函数" << endl;
    }
    ~Circle()
    {
        cout << "这是Circle类的析构函数" << endl;
    }
    double getArea()
    {
        return PI * r * r;
    }
};

int main() {
    cout << "正在创建正方形类的对象" << endl;
    Square square;
    cout << "请输入正方形的边长" << endl;
    cin >> square.a;
    cout << "正方形的面积为: " << square.getArea() << endl;

    cout << endl;
    cout << "正在创建圆形类的对象" << endl;
    Circle circle;
    cout << "请输入圆形的半径" << endl;
    cin >> circle.r;
    cout << "圆形的面积为: " << circle.getArea() << endl;
    return 0;
}

```

(2)

实验笔记:

前置++和--,需要在重载操作符的时候先对 Point 类的成员变量 x,y 加一减一,再返回*this; 后置++, --,需要用 tmp 存放*this,再实现 this->x,this->y 的加一减一,最后返回的是 tmp。最后通过对 p1.x 观察判断是否正确实现前置++和后置++的重载,通过对 p1.y 的值来判断是否正确实现前置--和后置--的重载。

运行结果截图:

```
请输入p1的坐标(x, y):2 4
p1的x坐标: 2
p1++的x坐标: 2
p1的x坐标: 3
++p1的x坐标: 4
p1的x坐标: 4
p1的y坐标: 6
p1--的y坐标: 6
p1的y坐标: 5
--p1的y坐标: 4
p1的y坐标: 4
```

源代码:

```
#include<iostream>
using namespace std;

class Point
{
public:
    int x;
    int y;

    //前置++
    Point& operator++()
    {
        this->x += 1;
        this->y += 1;
        return *this;
    }

    //前置--
    Point& operator--()
    {
        this->x-=1;
        this->y-=1;
        return *this;
    }

    //后置++
    Point operator++(int)
    {
        Point tmp = *this;//先使用原数（tmp），然后+1
        this->x += 1;
        this->y += 1;
        return tmp;
    }
};
```

```

    }
    //后置--
    Point operator--(int)
    {
        Point tmp = *this;
        this->x-=1;
        this->y-=1;
        return tmp;
    }
};

int main()
{
    Point p1;
    cout << "请输入p1的坐标(x, y):";
    cin >> p1.x >> p1.y;
    cout << "p1的x坐标: " << p1.x << endl;
    cout << "p1++的x坐标: " << p1++.x << endl;
    cout << "p1的x坐标: " << p1.x << endl;
    cout << "++p1的x坐标: " << (++p1).x << endl;
    cout << "p1的x坐标: " << p1.x << endl;

    cout << "p1的y坐标: " << p1.y << endl;
    cout << "p1--的y坐标: " << p1--.y << endl;
    cout << "p1的y坐标: " << p1.y << endl;

    cout << "--p1的y坐标: " << (--p1).y << endl;
    cout << "p1的y坐标: " << p1.y<< endl;
    return 0;
}

```

综合题目：

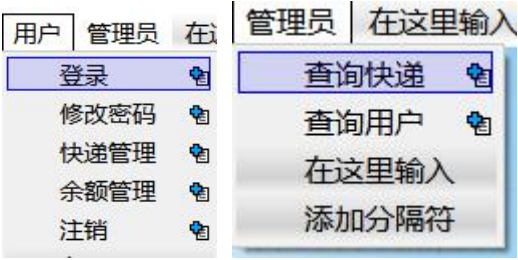
物流业务管理子系统和用户管理系统子系统：

设计思路：

为了方便用户交互，使用了 Qt 作为开发环境。

首先设计了交互界面，使用 Qt 的设计师类，主体框架为 `stackedWidget`，设置了多个页面，可以进行翻页。

上方菜单栏用来供用户手动选择要完成的功能。



首页（第 0 页）：登录界面（loginpage）

包括一个 `comboBox` 下拉菜单栏用来供用户选择身份(管理员、普通用户)和一个窗口 `widget`，主要包括用户名密码的标签和输入框，登录和注册的按钮。

loginpage	QWidget
comboBox	QComboBox
widget	QWidget
LoginButton	QPushButton
RegisterButton	QPushButton
label	QLabel
le_name	QLineEdit
le_pw	QLineEdit
name	QLabel
password	QLabel

设计效果如图，更改了样式表的背景图片。



核心思路:

登录和注册主要涉及到文件的读取和写入,为了方便程序准确读取到想要的信息,我将用户的用户名和密码以及余额按照固定格式存放在名为 user.txt (D:\\userdata\\user.txt) 的文本中,管理员的存在 manager.txt (D:\\userdata\\manager.txt) 中,每一行为一个用户账号的信息记录,从左到右依次为用户账号名,密码,余额,中间用空格隔开。



登录:

从输入框读取到用户输入的账号名和密码之后,先判断是否合法(是否为空,是否包含空格),若合法,则从 comboBox 下拉菜单的索引号判断用户现在选择的身份是管理员还是普通用户,根据此索引号设置要读取文件的路径。

读文件的操作比较重要,也是我反复修改之后才确定用哪一种的(因为真的很多),首先用 Qt 的文本流类 QTextStream 读取文件,声明 QString 变量 l,r 用来存放读取到的账户名和密码, double c 用来存余额。

外层循环 !in.atEnd() 用来判断是否读到文件尾, in.readLine 表示每次循环读一行, line.section 读取每行用空格分隔开的内容, l 存放分隔符空格隔开的第 0 个字段(即用户名), r 存放第 1 个字段(密码), c 存放第 2 个字段(余额)。

若读取到的账户名和用户输入的匹配,则跳出循环,否则要一直循环读到文件尾。

```
QTextStream in(&file);
QString l="",r="";
double c=0;
while(!in.atEnd()){
    QString line=in.readLine();
    l=line.section(" ",0,0);
    r=line.section(" ",1,1);
    c=line.section(" ",2,2).toDouble();
    if(l==a){
        break;
    }
}
```

若账户名密码都匹配成功，则跳转到登录成功界面。并将 `mainwindow` 类的成员变量用户指针根据用户选择的身份初始化。

```
if(l==a&&r==p){
    ui->stackedWidget->setCurrentIndex(3);
    ui->label_4->setText(a);
    state=1;
    if(index==0){
        pr=new realUser(a,p,c,0);
    }else{
        pr=new manager(a,p,c,1);
    }
}else{
    QMessageBox::warning(this,"警告","账号或密码错误");
}
```

涉及到的类:

基类:

```
class User : public QObject
{
    Q_OBJECT

public:
    User(QString a, QString p, double b, int t);
    virtual int getUserType()const=0;
    QString account; //账户
    double balance; //余额
    QString password; //密码
private:
    int type; //用户类型
};
#endif // USER_H
```

含参构造函数，四个参数分别为用户名，密码，余额，用户类型

子类:

普通用户:

```
class realUser : public User
{
public:
    realUser(QString a, QString p, double b, int t);
    virtual int getUserType()const;
};
```

管理员:

```
class manager : public User
{
public:
    manager(QString a, QString p, double b, int t);
    virtual int getUserType()const;
};
```

登录成功会跳转到成功提示:



注册:

和登录基本一致, 但需要判断账号名是否被注册过:

```
QTextStream in(&file);
QString l="";
while(!in.atEnd()){
    QString line=in.readLine(0);
    int i=line.indexOf(" ");
    l=line.left(i);
    qDebug()<<l;
    if(l==a){
        break;
    }
}
if(l==a){
    QMessageBox::warning(this,"警告","此账号已被注册");
    return;
}
file.close();
```

i 为某一行中第一个
空格的索引号, l 读取
i 左边的所有内容, 即
用户账号名。

如果在行读取的过程中, 某
某行的账号名和用户输入的一
致, 则提示已被注册

写入文件&更新数据库中的用户表格:

```
//写入文件
QTextStream out(&file);
out<<a<<" "<<p<<" "<<"0"<<endl;
out.flush();
file.close();
//写入数据库
UserInfo info;
info.account=a;
auto ptr = usersql::getInstance();
ptr->addUser(info);
QMessageBox::information(this,"提示","注册成功");
}else{
    QMessageBox::warning(this,"警告","账号和密码不能为空");
}
}
```

第 1 页：用户进行快递管理界面（查询、签收、发送）

设计界面：

主体为一个 TableWidget 用来展示用户的快递（包括用户收和发出的所有快件）



核心设计：

签收快递：

定义了一个结构体用来存放快件信息：

```
struct DeliveryInfo
{
    int id;
    QString number;//快递单号
    QString sender;//发件人
    QString receiver;//收件人
    QString sendtime;//发送时间
    QString receivetime;//接收时间
    QString status;//快件状态（待签收/已签收）
    QString detail;//详细描述（易碎品、图书、日用品）
};
```

delivery 表:

id	number	sender	receiver	sendtime	receivetime	status	detail
1	SF202212891	张丽	王小明	2022.4.23	2022.4.25	已签收	书籍
2	YT452911399	张鹏帆	宇宙的花朵	2022.4.23	2022.4.26	已签收	牙刷
3	YT420121038	吴彦祖	刘亦菲	2022.4.24	2022.4.24	已签收	生鲜
4	JD981365182	ckeee	wyx0801	2022.4.26	2022.4.27	已签收	生活用品
5	SF776235013	aa吃饱了	姜姜	2022.4.26	2022.4.28	已签收	生鲜
6	YD481921913	王小明	张丽	2022.4.27	2022.4.30	已签收	服饰
7	JT3829139011	toomany	张丽	2022.4.28	2022.4.29	已签收	美妆
8	YT2838612491	美丽生活服装店	张丽	2022.5.1	2022.5.3	已签收	服饰
9	SF3018192011	张丽	华天	2022.5.2	2022.5.3	已签收	电子产品
10	JD7382915396	京东大药房	张丽	2022.5.2	2022.5.2	已签收	药品
11	ST0390213555	隅田川咖啡自营店	张丽	2022.5.6	2022.5.10	已签收	食品
12	YD4829498174	雅布家居	王小明	2022.5.4	2022.5.6	已签收	家居用品
13	SF3982902102	张鹏帆	张丽	2022.5.27	2022.5.29	已签收	电器
14	efptliqgbcj	刘亦菲	aa吃饱了	2022.5.29		未签收	零食
15	hauoepaqruj	刘亦菲	张丽	2022.5.30	2022-6-30	已签收	礼物

快件 sql 类:

```
class deliverysql : public QObject
{
public:
    static deliverysql *ptrdSql;
    static deliverysql *getInstance()
    {
        if(nullptr == ptrdSql )
            ptrdSql = new deliverysql ;
        return ptrdSql;
    }
    explicit deliverysql(QObject *parent = nullptr);
    //初始化
    void init();
    //获取快递数量
    quint32 getDelCnt();
    //获取快递数据
    QList<DeliveryInfo> getPageDelivery(quint32, quint32);
    //更新表格
    void updateTable(DeliveryInfo);
    //增加快递信息
    bool addDel(DeliveryInfo);
    //用户签收快递
    bool updateDelInfo(DeliveryInfo);
private:
    QSqlDatabase m_db;
};
```


用户在输入框输入签收的快递单号之后点击签收按钮，会触发槽函数：

```
void mainwindow::on_SingforD_clicked()
{
    DeliveryInfo info;
    info.number = ui->d_info->text(); //读取用户输入的快递单号
    auto ptr = deliverysql::getInstance(); //获取一个指向快件 sql 类的指针
    ptr->updateDelInfo(info); //通过指针实现功能更新数据库记录
    QMessageBox::information(nullptr, "信息", "签收成功!"); //提示信息
    updateUDTable(); //更新用户界面上的 TableWidget
}
```

快件 sql 类成员函数 updateDelInfo 的实现：

```
bool deliverysql::updateDelInfo(DeliveryInfo info)
{
    QSqlQuery sql(m_db);
    QString time = QDate::currentDate().toString("yyyy-MM-dd"); //获取当前时间作为签收时间
    QString strSql = QString("update delivery set status = '%1', receivetime = '%2' where number = '%3' ").
        arg("已签收").
        arg(time).
        arg(info.number); //更新数据库
    return sql.exec(strSql);
}
```

用户查询快件：

数据库中 user 表：

name 为真实姓名（寄快件所需，account 是用户名）

id	name	phonenumber	account	address
1	张丽	18045972350	zxllovefish	光启市浦东区幸福之家7-107
2	王小明	13879275088	wxm_	湖北省襄阳市湖西街道办事处8-901
3	刘亦菲	17928993484	feifei	上海市我心区心房之居6-66
4	吴彦祖	15263597406	WuYanzu	北京市朝阳区翻斗花园201室
5	张鹏帆	13809867820	ZhangPfan	江苏省南京市卜值街一二居2-301
6	aa吃饱了	15789572069	aahungry	光启市齐司礼之心0740
7	美丽生活册	820882036	MayLife	美丽的地球村
8	隅田川咖啡	190671	ytCafe	浙江省杭州市光华大道8-1
9	cke	18071070450	caike	湖北省武汉市江岸区黄浦大街武汉天地2-

usersql.h:

结构体:

```
struct UserInfo{
    int id;
    QString name;
    QString phonenumber;
    QString account;
    QString address;
};
```

涉及到的类:

```
class usersql : public QObject
{
public:
    static usersql *ptruSql;
    static usersql *getInstance()
    {
        if(nullptr == ptruSql )
            ptruSql = new usersql ;
        return ptruSql;
    }
    explicit usersql(QObject *parent = nullptr);
    //初始化
    void init();
    //获取用户数量
    quint32 getUserCnt();
    //获取用户数据
    QList<UserInfo> getPageUser(quint32, quint32);
    //更新表格
    void updateUTable(UserInfo);
    //增加用户
    bool addUser(UserInfo);

private:
    QSqlDatabase m_db;
};
```

按下查找按钮触发槽函数:

```
void mainwindow::on_dsearch_clicked()
{
    DeliveryInfo info;
    info.number = ui->d_uinfo->text();
    info.sender = ui->d_uinfo->text();
```

用户输入的信息被存放在快件信息的结构体中,输入的信息可以是快件单号,收件人,发件人,发送时间

```

info.receiver = ui->d_uinfo->text();
info.sendtime = ui->d_uinfo->text();
//准备表格
ui->tableWidget->clear();
ui->tableWidget->setColumnCount(8);
QStringList l;
l<<"序号"<<"单号"<<"发件人"<<"收件人"<<"发件时间"<<" 签收时间"<<"快件状态"
"<<"物品描述";
ui->tableWidget->setHorizontalHeaderLabels(l);

//数据库准备
m_ptrdSql = deliverysql::getInstance();
m_ptrdSql ->init();
m_ptruSql = usersql::getInstance();
m_ptruSql ->init();
auto cnt1 = m_ptrdSql ->getDelCnt();
QList<DeliveryInfo> lEvents = m_ptrdSql->getPageDelivery(0, cnt1);
ui->tableWidget->setRowCount(cnt1);
auto cnt2 = m_ptruSql ->getUserCnt();
QList<UserInfo> lUser = m_ptruSql->getPageUser(0, cnt2);
QString strFilter;
//找到该账号用户所对应的姓名
for(int i = 0; i<lUser.size(); i++)
{
    if(lUser[i].account==pr->account){
        strFilter=lUser[i].name;
        break;
    }
}

int index=0;
for(int i = 0; i<lEvents.size(); i++)
{
    //如果收件人和发件人都不是用户，表格不显示
    if(!((lEvents[i].sender==strFilter||lEvents[i].receiver==strFilter)&&
        (lEvents[i].number==info.number||lEvents[i].sender==info.sender||lEvents[i].receiver==info.receiver||lEvents[i].sendtime==info.sendtime)))
        continue;
    ui->tableWidget->setItem(index,0,new QTableWidgetItem(QString::number(i)));
    ui->tableWidget->setItem(index,1,new QTableWidgetItem(lEvents[i].number));
    ui->tableWidget->setItem(index,2,new QTableWidgetItem(lEvents[i].sender));
    ui->tableWidget->setItem(index,3,new QTableWidgetItem(lEvents[i].receiver));
    ui->tableWidget->setItem(index,4,new QTableWidgetItem(lEvents[i].sendtime));
    ui->tableWidget->setItem(index,5,new QTableWidgetItem(lEvents[i].receivetime));
    ui->tableWidget->setItem(index,6,new QTableWidgetItem(lEvents[i].status));
    ui->tableWidget->setItem(index,7,new QTableWidgetItem(lEvents[i].detail));
    index++;
}

```

将数据库中的快件表格中的
每一项记录都存在 QList 表格
中。

余额管理：

定义一个 QDialog 类的子类用来交互让用户输入充值的量：


```

namespace Ui {
class BalanceDialog;
}

class BalanceDialog : public QDialog
{
    Q_OBJECT

public:
    explicit BalanceDialog(QWidget *parent = nullptr, double b=0);
    ~BalanceDialog();
    QString getBalance();

private:
    Ui::BalanceDialog *ui;
};

```

构造函数，参数为主窗口指针和主窗口当前用户余额，初始化弹出的 BalanceDialog，让对话框显示用户当前余额。

成员函数，返回用户通过输入框输入的充值额

用户根据 menuBar 点击功能->余额管理，触发 QAction:

New 一个 BalanceDialog 类，如果用户在 Dialog 点击了确定，则将输入框读取的充值额转换为 double，通过 ChangeFile 函数改变 txt 文件中存储的用户余额信息。

ChangeFile:

```
void changeFile(int index, int type,/mainwindow *m, QString text)
```

Index 为下拉菜单栏索引号，用以判别用户身份以设置文件路径

指明本次操作修改的是密码还是余额

m->pr->account
指明要修改的用户账号

修改后的密码或者充值后的余额

```

connect(ui->actionbalance,&QAction::triggered,[=]() { //余额管理
    if(state){
        BalanceDialog *mydialog = new BalanceDialog(this,pr->balance);
        Qt::WindowFlags flags=mydialog->windowFlags();
        mydialog->setWindowFlags(flags | Qt::MSWindowsFixedSizeDialogHint); //设置窗口为固定大小
        int ret=mydialog->exec ();
        if (ret==BalanceDialog::Accepted){
            QString text=mydialog->getBalance();
            if(text!=""){
                int index=pr->getUserType();
                pr->balance+=text.toDouble();
                changeFile(index,2,this,QString("%1").arg(pr->balance));
                QMessageBox::information(this,"提示","充值成功");
            }
        }
    }
    else{
        QMessageBox::warning(this,"警告","您还未登录");
    }
});

```

ChangeFile 核心操作

```
QTextStream stream1(&writefile);
if(index==0){
    for(int i=0;i<strlist.count();i++){
        QString line=strlist.at(i);
        if(line.section(" ",0,0)==m->pr->account){
            if(type==1){
                stream1<<m->pr->account<<" "<<text<<" "<<line.section(" ",2,2)<<endl; //替换密码
            }else{
                stream1<<m->pr->account<<" "<<line.section(" ",1,1)<<" "<<text<<endl; //替换余额
            }
        }else{
            stream1<<line<<endl;
        }
    }
}
else{
    stream1<<"admin"<<" "<<12345<<" "<<text;
}
writefile.close();
```

文件流操作修改文件某一行

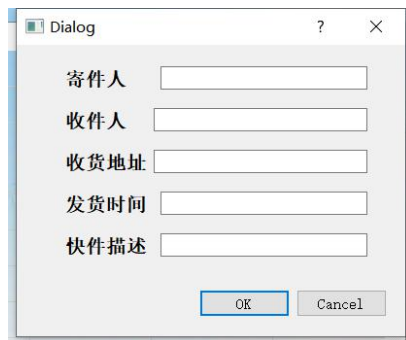
发送快件:

按下发送快件按钮后, 触发槽函数弹出增加快件的对话框, 然后扣除用户余额,

```
void mainwindow::on_send_clicked()
{
    m_adddeliverydialog.exec();
    //修改余额
    pr->balance-=15.00;
    changeFile(0,2,this,QString("%1").arg(pr->balance));
    //读管理员的余额
    QFile file("D:\\userdata\\manager.txt");
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QMessageBox::critical(this,"错误","文件读取出错");
        return;
    }
    QTextStream in(&file);
    double c=15;
    QString line=in.readLine(0);
    c+=line.section(" ",2,2).toDouble();
    changeFile(1,2,this,QString("%1").arg(c));
    updateUTable();
}
```

manager.txt 只有一行, 读取这一行末管理员的余额并加 15, 用 ChangeFile 修改

Adddeliverydialog 为 **QDialog** 子类，用来与用户交互，让用户输入要发送快递的相关信息，包括以下：



类定义：

```
namespace Ui {
class adddeliverydialog;
}

class adddeliverydialog : public QDialog
{
    Q_OBJECT

public:
    explicit adddeliverydialog(QWidget *parent = nullptr);
    ~adddeliverydialog();

private slots:
    void on_buttonBox_accepted();

    void on_buttonBox_rejected();

private:
    Ui::adddeliverydialog *ui;
};
```

槽函数实现：

```
void adddeliverydialog::on_buttonBox_accepted()
{
    DeliveryInfo info;
    info.sender = ui->le_sender->text();
    info.receiver = ui->le_receiver->text();
    info.sendtime = ui->le_time->text();
    info.detail = ui->le_detail->text();
    auto ptr = deliverysql::getinstance();
    ptr->addDel(info);
    QMessageBox::information(nullptr, "信息", "快件创建成功");
}

void adddeliverydialog::on_buttonBox_rejected()
{
    this->hide();
}
```

deliverysql 的成员函数 addDel:


```
bool deliverysql::addDel(DeliveryInfo info)
{
    QString str;           /*声明用来保存随机字符串的str*/
    char c;                /*声明字符c, 用来保存随机生成的字符*/
    /*循环向字符串中添加随机生成的字符*/
    for(int idx = 0; idx < 2; idx++)
    {
        /*rand()%26是取余, 余数为0~25加上'a', 就是字母a~z, 详见asc码表*/
        c = 'A' + rand()%26;
        str.push_back(c);    /*push_back()是string类尾插函数。这里插入随机字符c*/
    }
    for(int j = 0; j < 10; j++)
    {
        /*rand()%10是取余, 余数为0~9加上'1', 就是数字0~9, 详见asc码表*/
        c = '0' + rand()%10;
        str.push_back(c);    /*push_back()是string类尾插函数。这里插入随机字符c*/
    }
    QSqlQuery sql(m_db);
    QString strSql = QString("insert into courier_delivery values (null,'%1','%2','%3','%4','%5','%6','%7')");
    sql.arg(str);
    sql.arg(info.sender);
    sql.arg(info.receiver);
    sql.arg(info.sendtime);
    sql.arg("待揽收");
    sql.arg("待分配");
    sql.arg(info.type);
    return sql.exec(strSql);
}
```

核心为自动生成快递单号:
格式为 2 个字母+10 个数字, 采用循环添加, 通过 rand 随机取余+基数生成符合要求的字符, 添加到字符串尾

快递员任务管理子系统

在题目一的基础上, 增加了一种用户身份——快递员, 相应地, 我在 comboBox 下拉菜单栏增加了一栏快递员, 索引号为 2, 同时增加了文本文件 courier.txt (路径:

D:\\userdata\\courier.txt), 存放快递员的账号密码和余额。格式同题目一, 一行存放一个账户信息, 从左到右依次是账户名, 密码, 余额。

 courier.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
xiaohua xh123 687.00
luckincoffee luckyone 1009.50
wwwxy123 xy188 888.50
aijaychouai zjlyyds 666.00
wxfszn0531 bigSkillyou 999.00
190734519 hlflyyyy 690.50
guanxichen cgxinLA 200.00
```

数据库中增加了快递员信息表: name 为快递员快递单位真实姓名 account 为快递员账号名

id	name	phonenummer	account
1	小华	18076993271	xiaohua
2	瑞幸咖啡	8208820	luckincoffee
3	王星宇	13697885640	wwwxy123
4	周杰伦全球粉丝后援会	15060068710	aijaychouai
5	汪小菲是渣男	16365448210	wxfszn0531
6	何磊	19073451922	190734519
7	我是陈冠希	15843209861	guanxichen

增加类:

Class User 的子类 courier:(和 realuser,manager 一样, 都是 User 派生的)

```
class courier : public User
{
public:
    courier(QString a, QString p, double b, int t);
    virtual int getUserType() const;
};

couriersql:
struct CourierInfo{
    int id;
    QString name;
    QString phonenumber;
    QString account;
};

class couriersql : public QObject
{
public:
    static couriersql *ptrcSql;
    static couriersql *getInstance()
    {
        if(nullptr == ptrcSql )
            ptrcSql = new couriersql ;
        return ptrcSql;
    }
    explicit couriersql(QObject *parent = nullptr);
    //初始化
    void init();
    //获取用户数量
    quint32 getCourierCnt();
    //获取用户数据
    QList<CourierInfo> getPageCourier(quint32, quint32);
    //更新表格
    void updateCTable(CourierInfo);
    //增加用户
    bool addCourier(CourierInfo);

private:
    QSqlDatabase m_db;
};
```

快件类型基类:

```
class package
{
public:
```

```

        package(double p);
        virtual double getPrice();
        double price;
};

```

子类易碎品:

```

class fragile : public package
{
public:
    fragile(double p);
    double getPrice();
};

```

子类图书:

```

class book :public package
{
public:
    book(double p);
    double getPrice();
};

```

子类普通快递:

```

class normal :public package
{
public:
    normal(double p);
    double getPrice();
};

```

管理员端增加功能①: 为快件分配快递员

同样是在 menuBar 增加功能选项, 用 QAction 触发, 跳转到页面 5,

```

connect(ui->actiondistribute,&QAction::triggered,[=]() {
    ui->stackedWidget->setCurrentIndex(5);
    updateDUTable();

});

```

页面 5 设计:

TableWidget 显示出所有没有被分配快递员的快递, 用户复制单号到输入框, 点击确定, 跳转到槽函数,



获取选中快递单号的快递类型，根据快递类型，减去管理员的余额（从用户发快件获取的快递费分一半给快递员）

```
void mainwindow::on_distributeCourier_clicked()
{
    //获取选中的待分配快递单号
    dinfo.number=ui->le_dnumber->text();
    QString type=QString("select type from courier_delivery where number = '%1'").arg(dinfo.number);
    if(type=="易碎品")
        pr->balance-=4;
    else if(type=="图书")
        pr->balance-=1;
    else if(type=="普通快递")
        pr->balance-=2.5;
    changeFile(1,2,this,QString("%1").arg(pr->balance));
    ui->stackedWidget->setCurrentIndex(6);
    updateCTable();
}
```

ChangeFile 改变余额。第一个参数代表读取 manager.txt，第二个参数代表操作数修改余额

跳转到到页面 6，进行快递员的选择。

页面 6 设计：

TableWidget 读取数据库中 courier 表格并进行显示，管理员可以复制快递员姓名到输入框，点击确定分配之后跳转到槽函数，



槽函数读取输入框输入的快递员姓名，并对 `courier_delivery` 表格进行修改:

```
void mainwindow::on_confirmbutton_clicked()
{
    //对选中的快递员进行分配
    dinfo.courier=ui->le_courier->text();
    auto ptr = deliverysql::getinstance();
    DeliveryInfo info;
    info.number=dinfo.number;
    info.courier=dinfo.courier;
    ptr->updateDCInfo(info);
    QMessageBox::information(nullptr,"信息","分配成功!");
}
```

修改记录中的快递员表项为选中的快递员，并将快递状态由待分配改为待揽收

`courier_delivery` 表格是在 `delivery` 表格基础上，增加了快件类型、快递员两个表项:

id	number	sender	receiver	time	status	courier	type
1	SF202212891	张丽	王小明	2022.4.23	已签收	汪小菲是渣男	普通快递
2	YT452911399	张鹏帆	宇宙的花朵	2022.4.23	已签收	小华	易碎品
3	YT420121038	吴彦祖	刘亦菲	2022.4.24	已签收	王星宇	普通快递
4	JD981365182	ckeee	wyx0801	2022.4.26	已签收	汪小菲是渣男	易碎品
5	SF776235013	aa吃饱了	姜姜	2022.4.26	已签收	周杰伦全球粉丝后援会	普通快递
6	YD481921913	王小明	张丽	2022.4.27	已签收	小华	普通快递
7	JT3829139011	toomany	张丽	2022.4.28	已签收	何磊	普通快递
8	YT2838612491	美丽生活家	张丽	2022.5.1	已签收	何磊	图书
9	SF3018192011	张丽	华天	2022.5.2	已签收	何磊	图书
10	JD7382915396	京东大药房	张丽	2022.5.2	已签收	小华	普通快递
11	ST0390213555	隅田川咖啡	张丽	2022.5.6	已签收	周杰伦全球粉丝后援会	普通快递
12	YD4829498174	雅布家居	王小明	2022.5.4	已签收	王星宇	图书
13	SF3982902102	张鹏帆	张丽	2022.5.27	已签收	王星宇	普通快递
14	efptliqbcj	刘亦菲	aa吃饱了	2022.5.29	待揽件	待分配	图书
15	hauoeqaruj	刘亦菲	张丽	2022.5.30	已签收	汪小菲是渣男	普通快递
16	SF8820724310	张丽	刘亦菲	2022.5.31	已签收	小华	易碎品
17	YT9328730291	隅田川咖啡	刘亦菲	2022.6.1	已签收	王星宇	普通快递
18	SC0047047877	张鹏帆	吴彦祖	2022.6.1	已揽收	小华	易碎品
19	YK9182290735	张鹏帆	吴彦祖	2022.6.2	待揽收	待分配	图书
20	AV8437209947	张鹏帆	刘亦菲	2022.6.2	已揽收	周杰伦全球粉丝后援会	普通快递
21	XB2620359932	张鹏帆	刘亦菲	2022.6.3	待揽收	待分配	普通快递
22	YF4654174607	张丽	张鹏帆	2022.06.28	已揽收	何磊	易碎品
23	ZR4296226655	王小明	张鹏帆	2022.06.29	待揽收	待分配	图书

管理员增加功能②: 查询快递员信息，可以根据快递员账户名，真实姓名查找，和查找用户差不多。

将查找到的结果单独显示在表格中。



快递员端：

menuBar 功能①：揽收快递

可以直接点击快递单号和右边确定揽收按钮进行揽收。



点击按钮触发槽函数：

```

void/mainwindow::on_confirmcollect_clicked()
{
    QList<QTableWidgetItem*> items = ui->collectTable->selectedItems();
    DeliveryInfo info;
    auto ptr = deliverysql::getinstance();
    int count = items.count();

    for(int i = 0; i < count; i++)
    {
        QTableWidgetItem *item = items.at(i);
        info.number = item->text(); //获取揽收的单号
        ptr->collectDelInfo(info);
        //将一半的快递费转给快递员
        QString type=QString("select type from courier_delivery where number = '%1'").arg(info.number);
        if(type=="易碎品")
            pr->balance+=4.00;
        else if(type=="图书")
            pr->balance+=1.00;
        else if(type=="普通快递")
            pr->balance+=2.50;
        changeFile(2,2,this,QString("%1").arg(pr->balance));
    }
    QMessageBox::information(nullptr,"信息","揽收成功!");
    updateCDUTable();
}

```

获取用户选择的快递单号（多个或一个）

通过遍历，更新数据库表格内容和 courier.txt 中的余额

揽收之后可以通过查找快递信息功能看到



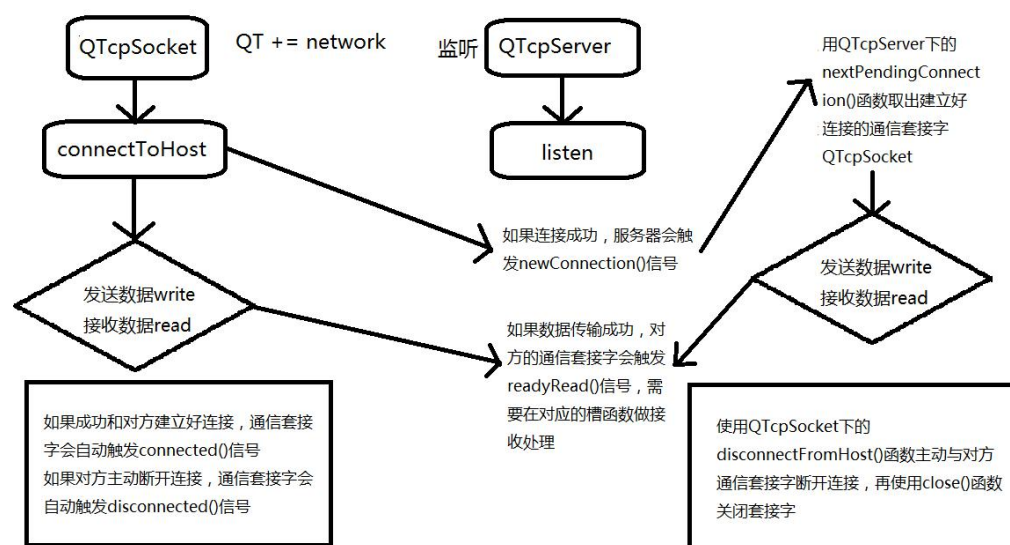
快递员功能②：查找自己派送的快件信息。

可以根据收件人，发件人，快递单号，快递状态，发送时间来查询本用户派送的所有快递。思想和前面的查找类似。

题目三：网络版

在题目二的基础上，将项目分为 client 用户端和 sever 服务器端。Socket 过程：服务器先运行并监听等待客户端连接请求，运行客户端时，客户端自己建立一个套接字 `QTcpSocket`，用这个类下的 `connectToHost` 函数发送连接请求后，如果服务器和客户端成功连接，服务器触发 `newConnection()` 信号，然后服务器用 `QTcpServer` 类的方法 `nextPendingConnection()` 新建立一个用于此连接的通信套接字 `QTcpSocket`。

建立好连接之后，两端的通信套接字都会自动触发 `connected` 信号，客户端和服务端交互完成后，可以使用 `QTcpSocket` 下的 `disconnectFromHost` 函数与服务器的通信套接字断开连接，再使用 `close` 函数关闭套接字，这时服务器端的通信套接字自动触发 `disconnected` 信号断开连接。



思路：

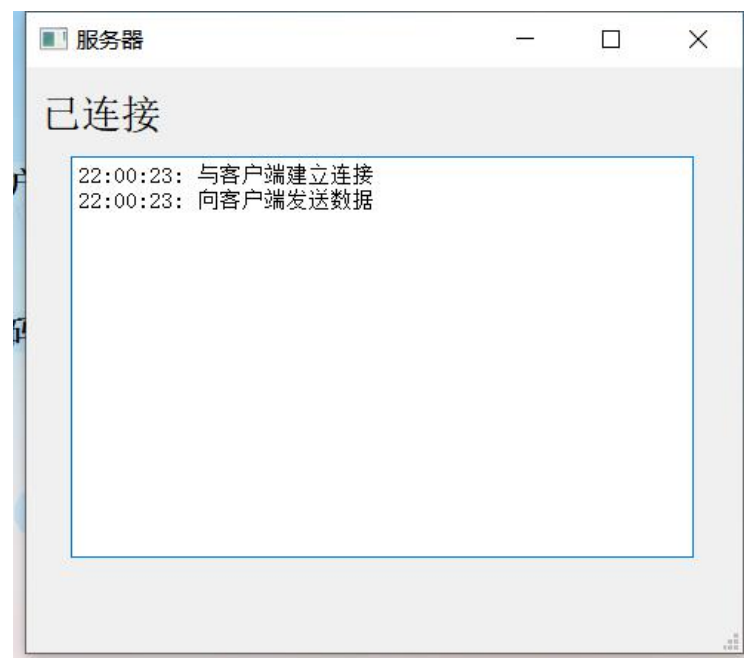
用户端与服务器连接上之后，服务器将数据库文件以字节流的方式全部写入通信套接字，触发用户端的 `readyRead()` 信号，通过此信号，用户可以一次性接收全部数据库文件并自己存储起来，之后对数据库的更改操作则不需要通过服务器来操作了，只有改密码、改余额、注册等涉及到对 txt 文件的更改需要服务器完成，等用户完成了对数据库的所有操作之后（即点击关闭了用户端的窗口），用户端再将完全更改后的数据库文件通过通信套接字发送给服务器。

```
// 发送数据
void sendFile(MainWindow *m)
{
    QFile file("D:\\data.db");
    // 打开文件
    bool ok = file.open(QIODevice::ReadOnly);
    if(!ok){
        QMessageBox::critical(m, "错误", "文件读取出错");
        return;
    }
    // 开始发文件
    QByteArray buf;
    buf = file.readAll();
    m->conn->write(buf);
    file.close();
}
```

```
connect(conn, &QTcpSocket::readyRead, [=]() {
    QByteArray array = conn->readAll();
    if(array.size() > 1000) { // 从服务器收到了数据库相关
        qDebug() << QString("%1").arg(array.size());
        QFile file("D:\\data.db");
        // 打开文件
        bool ok;
        ok = file.open(QIODevice::WriteOnly | QIODevice::Append);
        if(!ok) {
            QMessageBox::critical(this, "错误", "文件读取出错");
            return;
        }
        // 把从服务器读取的数据库写入文件
        file.write(array);
        file.close();
    }
}
```

核心设计:

服务器界面设计:



服务器主要需要实现:

1.建立连接 2.监听 3.发送数据包 4.接收数据包 5.断开连接

```
listen = new QTcpServer
//创建监听
listen->listen(QHostAdd

ui->label->setText("等待连接.....");
```

若有新的连接建立，则触发。

建立连接:

```
connect(listen, &QTcpServer::newConnection, [=](){})
```

若服务器是“等待连接”状态，则建立连接，否则返回

```
if(!connect1){
    connect1=true;
    curtime=QTime::currentTime();
    ui->textEdit->append(QString("%1: 与客户端建立连接").arg(curtime.toString()));
}else{
    return;
}
```

通过监听建立新的套接字

```
conn = listen->nextPendingConnection();
sendFile(this);
curtime=QTime::currentTime();
ui->textEdit->append(QString("%1: 向客户端发送数据").arg(curtime.toString()));
ui->label->setText("已连接");
```

接收数据包：

核心：判断用户发来的是数据库信息还是普通的操作信息（如充值、改密码），若是数据库信息（大小大于 1KB），则表明此时用户完成了更新数据库的所有操作，把更新后的数据库文件写入，若是普通信息，则通过空格分隔符，判断用户请求的操作内容，再进行相关操作。



```
if(array.size(>1024))
{
    qDebug()<< "数据库文件更新";
    if(!flag){
        flag=true;
        curtime=QTime::currentTime();
        ui->textEdit->append(QString("%1: 数据库更新").arg(curtime.toString()));
        receiveFile(array);
    }else{
        receiveFile(array);
    }
}
else{
    QString str = array;
    QString f = str.section(" ",0,0);
    int index = str.section(" ",1,1).toInt();
    QString a = str.section(" ",2,2);
    QString p = str.section(" ",3,3);
    int reply=2;
    double balance=0;
    if(f=="register"){ //注册
        reply = register1(this,index,a,p);
        curtime=QTime::currentTime();
        ui->textEdit->append(QString("%1: 用户注册").arg(curtime.toString()));
    }else if(f=="login"){ //登录
        reply = login(this,index,a,p,&balance);
        curtime=QTime::currentTime();
        ui->textEdit->append(QString("%1: 用户登录").arg(curtime.toString()));
    }else if(f=="balance"){ //改余额
        changeFile(index,2,this,a,p);
        curtime=QTime::currentTime();
        ui->textEdit->append(QString("%1: 用户改余额").arg(curtime.toString()));
    }else if(f=="password"){ //改密码
        changeFile(index,1,this,a,p);
        curtime=QTime::currentTime();
        ui->textEdit->append(QString("%1: 用户改密码").arg(curtime.toString()));
    }
    if(reply!=2){
        conn->write((QString("%1 %2").arg(reply).arg(balance)).toUtf8());
    }
}
```

其中 `changeFile` 文件与前两题大致相同，在此不做赘述，`ui->textEdit` 用来在服务器上显示用户操作的相关信息。

断开连接：


```

connect(conn, &QTcpSocket::disconnected, [=]() {
    ui->label->setText("等待连接.....");
    curtime=QTime::currentTime();
    ui->textEdit->append(QString("%1: 与客户端断开连接").arg(curtime.toString()));
    flag=false;
    connect1=false;
});

```

用户端:

建立断开连接:

```

//与服务器连接
conn = new QTcpSocket(this);
conn->connectToHost(QHostAddress("127.0.0.1"),9999);
connect(conn, SIGNAL(readyRead()), &loop, SLOT(quit()));
connect(conn, SIGNAL(disconnected()), &loop, SLOT(quit()));
//设置连接建立和断开的操作
connect(conn, &QTcpSocket::connected, [=]() {
    ui->statusBar->showMessage("已连接");
    isconnect=true;
});
connect(conn, &QTcpSocket::disconnected, [=]() {
    ui->statusBar->showMessage("等待连接.....");
    isconnect=false;
});

```

接收数据包:

```

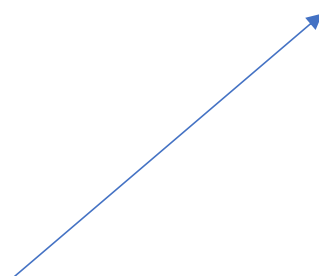
//设置收到信息的操作
connect(conn, &QTcpSocket::readyRead, [=]() {
    QByteArray array = conn->readAll();
    if(array.size()>1000){ //从服务器收到了数据库相关
        qDebug()<<QString("%1").arg(array.size());
        QFile file("D:\\data.db");
        //打开文件
        bool ok;
        ok = file.open(QIODevice::WriteOnly|QIODevice::Append);
        if(!ok){
            QMessageBox::critical(this,"错误","文件读取出错");
            return;
        }
        //把从服务器读取的数据库写入文件
        file.write(array);
        file.close();
        //打开数据库
        QSqlDatabase db = QSqlDatabase::addDatabase("SQLITE");
        db.setDatabaseName("D:\\data.db");
        if(!db.open()){
            QMessageBox::critical(0,"错误","数据库读取出错");
            exit(0);
        }
    }else{
        str=array;
    }
});

```

发送数据包:

主要是通过 `changeFile` 来实现:

发送给服务器信息的格式: 操作主体 用户索引号
用户账号 修改后内容



```

void changeFile(int index, int type,/mainwindow *m, QString account,QString text)
{
    QString s;
    if(type==2){
        s="balance";
    }else{
        s="password";
    }
    m->conn->write(QString("%1 %2 %3 %4 ").arg(s).arg(index).arg(account).arg(text).toUtf8());
    //m->loop.exec(QEventLoop::ExcludeUserInputEvents);
}

```

结束后，将修改后数据库文件发回服务器：

```

delete model;
QSqlDatabase db;
db.setDatabaseName("D:\\data.db");
db.close();
QSqlDatabase::removeDatabase("qt_sql_default_connection");
//向服务器发数据库文件
QFile file("D:\\data.db");
//打开文件
bool ok = file.open(QIODevice::ReadOnly);
if(!ok){
    QMessageBox::critical(this,"错误","文件读取出错");
    return;
}
//开始发文件
QMessageBox::information(nullptr,"信息","发送成功！");
//loop.exec(QEventLoop::ExcludeUserInputEvents);
file.close();
QFile::setPermissions("D:\\data.db",QFile::ReadOther | QFile::WriteOther);
QFile::remove("D:\\data.db");
event->accept();

```

心得&总结：

本次课设对我来说难度很大，不过好在之前数据结构课设需要学习相关 Qt 编程知识，二者之间有一些相似和相关性，为我综合题目的开发带来了帮助。

开发过程中总是不断遇到问题，一开始设计快件类的时候，由于对纯虚函数，抽象类等知识的不熟悉，不停地报错，就把错误一条条粘贴到浏览器去查，在这个 debug 的过程中我学到了很多，比如抽象类为含有纯虚函数的类（在基类中仅仅给出声明，不对虚函数实现定义，而是在派生类中实现，纯虚函数需要在声明之后加个=0）。抽象类只能作为派生类的基类，不能定义对象，但可以定义指针。

还有特别困扰我的就是数据库 sql 语句，为了查找表格中某特定位置的特点字段，研究 select 语句研究了很久才发现，尽管查找的可能只有一个字段，还是需要 while(query.next()) 来遍历获得，因为程序并不知道它查找到的只有一个字段。

最难的就是 Socket 编程相关，这个我学了三个不同来源的网课，但都感觉懵懵懂懂，好像知道大体思路就是服务器一直运行监听，发现用户端主动连接，就建立新套接字，发送数据包，用户端接收，然后二者交互发送数据包这样的过程，但我当时完全不知道如何实现，最后通过对计网课设的学习和计网课设小组之间的交流，有了一些思路，也陆陆续续学了一些 QT cpSocket 类的相关接口，在不断 debug 过程中基本上完成了网络版。

除此之外，界面美化也费了我很大的功夫，一开始我想通过控件类的接口来设置背景色、字体、边框、渲染之类的，对照 rgb 色卡表查了很久，但效果都很丑，最后还是通过添加 resource root 的方法设置背景图片，用了网上的渐变背景图，然后对输入框微调，算是尽我审美的最大可能了 orz

总而言之，本次课设对我来说，挑战性很大，但最后我也是收获了很多理论课学不来的知识，因为纯学理论我可能永远也想不到自己会在哪个奇葩的地方犯错。