

**《面向对象程序设计实践（C++）》课程设计报告**



**姓 名 张梓良**

**学 院 计算机学院**

**专 业 计算机科学与技术**

**班 级 2021211304**

**学 号 2021212484**

**任课教师 胡博**

**2023年 6月**

**目录**

[1.引言 4](#_Toc137156850)

[2.实验环境和工具 4](#_Toc137156851)

[3.基础实验 4](#_Toc137156852)

[3.1 C++基础知识实验 4](#_Toc137156853)

[3.1.1 概述 4](#_Toc137156854)

[3.1.2 实验内容 5](#_Toc137156855)

[3.1.3 实验步骤和代码分析 5](#_Toc137156856)

[3.1.4 实验源代码 8](#_Toc137156857)

[3.1.5 实验结果展示 12](#_Toc137156858)

[3.2 类与对象实验——“圆形” 12](#_Toc137156859)

[3.2.1 概述 12](#_Toc137156860)

[3.2.2 实验内容 12](#_Toc137156861)

[3.2.3 实验步骤和代码分析 13](#_Toc137156862)

[3.2.4 实验源代码 15](#_Toc137156863)

[3.2.5 实验结果展示 19](#_Toc137156864)

[3.3 类与对象实验——“矩阵” 20](#_Toc137156865)

[3.3.1 概述 20](#_Toc137156866)

[3.3.2 实验内容 21](#_Toc137156867)

[3.3.3 实验步骤和代码分析 21](#_Toc137156868)

[3.3.4 实验源代码 27](#_Toc137156869)

[3.3.5 实验结果展示 32](#_Toc137156870)

[3.4 继承与派生实验 34](#_Toc137156871)

[3.4.1 概述 34](#_Toc137156872)

[3.4.2 实验内容 34](#_Toc137156873)

[3.4.3 实验步骤和代码分析 34](#_Toc137156874)

[3.4.4 实验源代码 36](#_Toc137156875)

[3.4.5 实验结果展示 40](#_Toc137156876)

[3.5 I/O 流实验 42](#_Toc137156877)

[3.5.1 概述 42](#_Toc137156878)

[3.5.2 实验内容 42](#_Toc137156879)

[3.5.3 实验步骤和代码分析 42](#_Toc137156880)

[3.5.4 实验源代码 43](#_Toc137156881)

[3.5.5 实验结果展示 45](#_Toc137156882)

[3.6 重载实验——“虚函数” 45](#_Toc137156883)

[3.6.1 概述 45](#_Toc137156884)

[3.6.2 实验内容 46](#_Toc137156885)

[3.6.3 实验步骤和代码分析 46](#_Toc137156886)

[3.6.4 实验源代码（主要差异在于.h文件，上面代码中已经展示了shape.h的修改内容，因此不再单独附上代码） 47](#_Toc137156887)

[3.6.5 实验结果展示 47](#_Toc137156888)

[3.7 重载实验——“对Point类重载++和--运算符” 48](#_Toc137156889)

[3.7.1 概述 48](#_Toc137156890)

[3.7.2 实验内容 49](#_Toc137156891)

[3.7.3 实验步骤和代码分析 49](#_Toc137156892)

[3.7.4 实验源代码 49](#_Toc137156893)

[3.7.5 实验结果展示 52](#_Toc137156894)

[4.综合实验 52](#_Toc137156895)

[4.1 任务描述 52](#_Toc137156896)

[4.2 功能需求 52](#_Toc137156897)

[4.3 模块介绍 54](#_Toc137156898)

[4.3.1模块划分 54](#_Toc137156899)

[4.3.2 服务器端 54](#_Toc137156900)

[4.3.3 客户端 64](#_Toc137156901)

[5.心得体会 79](#_Toc137156902)

# 1.引言

本报告旨在总结和展示《面向对象程序设计实践（C++）》课程设计的基础实验和综合实验。该课程旨在通过实践，帮助学生巩固和应用C++编程语言的基础知识，培养面向对象程序设计的能力。

在本次实验中，需要完成基础实验要求中的多个任务，包括矩阵操作、类与对象的使用、继承与派生、I/O流操作以及重载等内容。通过这些实验，我们将理论知识与实际编程相结合，深入理解了面向对象程序设计的核心概念和原则。同时通过完成综合实验：单词消除游戏系统设计与开发，让我们将所学知识融会贯通，进一步加强我们对面向对象语言C++的掌握，提升我们的问题分析、需求分析、工程问题解决的能力。

本报告将详细介绍每个实验任务的具体内容、代码实现和调试过程。将分析实验结果，讨论遇到的问题和解决方案，并总结在实验中的收获和体会。通过这份报告，希望能够全面展示我在《面向对象程序设计实践（C++）》课程中的学习成果和实践经验。

# 2.实验环境和工具

* windowsSdkVersion: 10.0.22000.0
* Visual Studio Code1.77
* cStandard: c17
* Qt Creator 9.0.2
* Qt 6.4.3 (MSVC 2019, x86\_64)

# 3.基础实验

## 3.1 C++基础知识实验

### 3.1.1 概述

在本实验中，我完成了基于C++编程语言的矩阵功能实现。矩阵是一种重要的数据结构，在计算机科学和数学领域广泛应用。通过本实验，旨在加深对C++语言的理解，并运用面向对象编程的原则和技巧来实现矩阵操作的各项功能。

### 3.1.2 实验内容

编写C++程序完成“矩阵”以下功能：

（1） 假定矩阵大小为 4×5（整型）；

（2） 矩阵空间采用 new 动态申请，保存在指针中；

（3） 定义矩阵初始化函数，可以从 cin 中输入矩阵元素；

（4） 定义矩阵输出函数，将矩阵格式化输出到 cout；

（5） 定义矩阵相加的函数，实现两个矩阵相加的功能，结果保存在另一个矩阵中；

（6） 定义矩阵相减的函数，实现两个矩阵相减的功能，结果保存在另一个矩阵中；

（7） 动态申请三个矩阵：A1、A2、A3；

（8） 初始化 A1、A2；

（9） 计算并输出 A3 = A1 加 A2，A3 = A1 减 A2；

（10） 释放矩阵空间。

### 3.1.3 实验步骤和代码分析

1. 定义矩阵的数据结构

* 用二维指针指向的内存空间存储矩阵的元素。

1. **struct** matrix
2. {
3. **int** \*\*elem;
4. };
5. 矩阵初始化函数

* 传入一个矩阵变量的引用，使用new关键字为此结构体变量中的elem元素动态申请内存空间。
* 此处需要注意对内存分配失败的情况进行特殊判断。并且由于矩阵相当于一个二维数组，需要两次用到new关键字进行动态申请矩阵空间，因此两次都需要判断内存申请是否成功。
* 同时此处使用new (std::nothrow)，在内存不足时，并不会抛出异常，而是将指针置NULL。

1. // 初始化函数
2. **void** InitMatrix(matrix &m)
3. {
4. m.elem = **new** (**nothrow**) **int** \*[4];
5. **if** (m.elem == nullptr)
6. {
7. cout << "初始化失败" << endl;
8. **return**;
9. }
10. **for** (**int** i = 0; i < N; i++)
11. {
12. m.elem[i] = **new** (**nothrow**) **int**[M];
13. **if** (m.elem[i] == nullptr)
14. {
15. cout << "初始化失败" << endl;
16. **return**;
17. }
18. }
19. }
20. 矩阵输入函数

* 传入一个矩阵变量的引用，通过输入流(cin)获取用户输入的矩阵元素，并将其存储到矩阵中。

1. // 输入函数
2. **void** InputMatrix(matrix &m)
3. {
4. cout << "请输入一个大小为4×5的矩阵:" << endl;
5. **for** (**int** i = 0; i < N; i++)
6. **for** (**int** j = 0; j < M; j++)
7. cin >> m.elem[i][j];
8. }
9. 矩阵输出函数

* 传入一个矩阵常量，确保函数不会对矩阵常量的内容进行修改。同时使用循环遍历矩阵中的元素，并通过输出流(cout)格式化输出矩阵。

1. // 输出函数
2. **void** OutputMatrix(**const** matrix m)
3. {
4. **for** (**int** i = 0; i < N; i++)
5. {
6. **for** (**int** j = 0; j < M; j++)
7. cout << m.elem[i][j] << " ";
8. cout << endl;
9. }
10. }
11. 矩阵相加函数

* 传入2个矩阵常量作为加数，同时传入一个矩阵变量的引用作为结果矩阵。使用循环遍历两个矩阵中的对应元素，将其相加并存储到结果矩阵中。

1. // 相加的函数
2. // m3 = m1 + m2
3. **void** AddMatrix(**const** matrix m1, **const** matrix m2, matrix &m3)
4. {
5. **for** (**int** i = 0; i < N; i++)
6. **for** (**int** j = 0; j < M; j++)
7. m3.elem[i][j] = m1.elem[i][j] + m2.elem[i][j];
8. }
9. 矩阵相减函数

* 传入2个矩阵常量作为被减数和减数，同时传入一个矩阵变量的引用作为结果矩阵。使用循环遍历两个矩阵中的对应元素，将其相减并存储到结果矩阵中。

1. // 相减的函数
2. // m3 = m1 - m2
3. **void** SubMatrix(**const** matrix m1, **const** matrix m2, matrix &m3)
4. {
5. **for** (**int** i = 0; i < N; i++)
6. **for** (**int** j = 0; j < M; j++)
7. m3.elem[i][j] = m1.elem[i][j] - m2.elem[i][j];
8. }
9. 矩阵空间释放函数

* 使用delete关键字释放给矩阵变量动态分配的内存空间。
* 同时在释放指针所指向的空间时，首先要对指针进行非空判断，防止出错，在释放完内存空间后，要手动将指针指向空，防止野指针问题。
* 在为矩阵动态分配内存空间时，我们是从外到内动态分配，所以释放时应该按照从内到外的释放顺序，否则会造成内存丢失。

1. // 释放矩阵空间
2. **void** ReleaseMatrix(matrix &m)
3. {
4. **for** (**int** i = 0; i < N; i++)
5. {
6. **if** (m.elem[i] != nullptr)
7. {
8. **delete**[] m.elem[i];
9. m.elem[i] = nullptr;
10. }
11. }
12. **if** (m.elem != nullptr)
13. {
14. **delete**[] m.elem;
15. m.elem = nullptr;
16. }
17. }
18. 动态申请矩阵并初始化

* 定义三个矩阵变量A1，A2，A3。A1，A2作为加减法运算的操作数，A3用以保存运算结果。并通过矩阵初始化函数初始化A1，A2，A3，通过矩阵输入函数输入矩阵A1，A2的元素。

1. InitMatrix(A1);
2. InitMatrix(A2);
3. InitMatrix(A3);
5. InputMatrix(A1);
6. InputMatrix(A2);
7. 计算并输出矩阵相加和相减的结果

* 分别调用矩阵相加函数和矩阵相减函数，并通过矩阵输出函数将结果格式化输出。

1. AddMatrix(A1, A2, A3);
2. cout << "A1加A2的结果为：" << endl;
3. OutputMatrix(A3);
5. SubMatrix(A1, A2, A3);
6. cout << "A1减A2的结果为：" << endl;
7. OutputMatrix(A3);
8. 释放矩阵空间

* 对矩阵A1，A2，A3分别调用矩阵释放函数。

1. ReleaseMatrix(A1);
2. ReleaseMatrix(A2);
3. ReleaseMatrix(A3);

### 3.1.4 实验源代码

matrix.h

1. #ifndef MATRIX\_H
2. #define MATRIX\_H
4. **const** **int** N = 4;
5. **const** **int** M = 5;
7. **struct** matrix
8. {
9. **int** \*\*elem;
10. };
12. // 初始化函数
13. **void** InitMatrix(matrix &m);
15. // 输入函数
16. **void** InputMatrix(matrix &m);
18. // 输出函数
19. **void** OutputMatrix(**const** matrix m);
21. // 相加的函数
22. // m3 = m1 + m2
23. **void** AddMatrix(**const** matrix m1, **const** matrix m2, matrix &m3);
25. // 相减的函数
26. // m3 = m1 - m2
27. **void** SubMatrix(**const** matrix m1, **const** matrix m2, matrix &m3);
29. // 释放矩阵空间
30. **void** ReleaseMatrix(matrix &m);
32. #endif

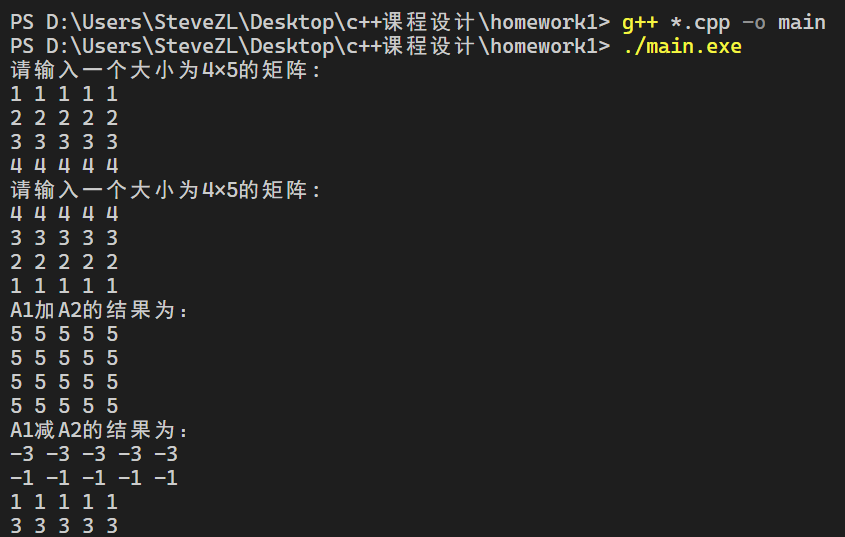
matrix.cpp

1. #include "matrix.h"
2. #include <iostream>
3. #include <new>
4. **using** **namespace** std;
6. // 初始化函数
7. **void** InitMatrix(matrix &m)
8. {
9. m.elem = **new** (**nothrow**) **int** \*[4];
10. **if** (m.elem == nullptr)
11. {
12. cout << "初始化失败" << endl;
13. **return**;
14. }
15. **for** (**int** i = 0; i < N; i++)
16. {
17. m.elem[i] = **new** (**nothrow**) **int**[M];
18. **if** (m.elem[i] == nullptr)
19. {
20. cout << "初始化失败" << endl;
21. **return**;
22. }
23. }
24. }
26. // 输入函数
27. **void** InputMatrix(matrix &m)
28. {
29. cout << "请输入一个大小为4×5的矩阵:" << endl;
30. **for** (**int** i = 0; i < N; i++)
31. **for** (**int** j = 0; j < M; j++)
32. cin >> m.elem[i][j];
33. }
35. // 输出函数
36. **void** OutputMatrix(**const** matrix m)
37. {
38. **for** (**int** i = 0; i < N; i++)
39. {
40. **for** (**int** j = 0; j < M; j++)
41. cout << m.elem[i][j] << " ";
42. cout << endl;
43. }
44. }
46. // 相加的函数
47. // m3 = m1 + m2
48. **void** AddMatrix(**const** matrix m1, **const** matrix m2, matrix &m3)
49. {
50. **for** (**int** i = 0; i < N; i++)
51. **for** (**int** j = 0; j < M; j++)
52. m3.elem[i][j] = m1.elem[i][j] + m2.elem[i][j];
53. }
55. // 相减的函数
56. // m3 = m1 - m2
57. **void** SubMatrix(**const** matrix m1, **const** matrix m2, matrix &m3)
58. {
59. **for** (**int** i = 0; i < N; i++)
60. **for** (**int** j = 0; j < M; j++)
61. m3.elem[i][j] = m1.elem[i][j] - m2.elem[i][j];
62. }
64. // 释放矩阵空间
65. **void** ReleaseMatrix(matrix &m)
66. {
67. **for** (**int** i = 0; i < N; i++)
68. {
69. **if** (m.elem[i] != nullptr)
70. {
71. **delete**[] m.elem[i];
72. m.elem[i] = nullptr;
73. }
74. }
75. **if** (m.elem != nullptr)
76. {
77. **delete**[] m.elem;
78. m.elem = nullptr;
79. }
80. }

main.cpp

1. #include "matrix.h"
2. #include <iostream>
3. **using** **namespace** std;
5. **int** main()
6. {
7. matrix A1, A2, A3;
9. InitMatrix(A1);
10. InitMatrix(A2);
11. InitMatrix(A3);
13. InputMatrix(A1);
14. InputMatrix(A2);
16. AddMatrix(A1, A2, A3);
17. cout << "A1加A2的结果为：" << endl;
18. OutputMatrix(A3);
20. SubMatrix(A1, A2, A3);
21. cout << "A1减A2的结果为：" << endl;
22. OutputMatrix(A3);
24. ReleaseMatrix(A1);
25. ReleaseMatrix(A2);
26. ReleaseMatrix(A3);
28. **return** 0;
29. }

### 3.1.5 实验结果展示



## **3.2 类与对象实验——“圆形**”

### 3.2.1 概述

本实验旨在通过编写C++程序实现圆形类的功能，包括圆心坐标和半径的输入、判断两个圆是否相交，并观察构造函数和析构函数的调用顺序。通过本实验，我们可以深入理解类的设计和实现，以及构造函数和析构函数的使用。

### 3.2.2 实验内容

编写C++程序完成“圆形”以下功能：

（1） 定义一个 Point 类，其属性包括点的坐标，提供计算两点之间距离的方法；（要求当用户不输入坐标数据时，能够默认为坐标原点（0，0））

（2） 定义一个圆形类，其属性包括圆心和半径；

（3） 创建两个圆形对象，提示用户输入圆心坐标和半径，判断两个圆是否相交，并输出结果；

（4） 观察圆形对象以及 Point 类成员的构造函数与析构函数的调用。（提示及要求：1，可通过在构造与析构函数中加入输出提示信息的方式观察相关调用；可以使用 system("pause")进行程序的暂停；2，能够理解并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及其原因）

### 3.2.3 实验步骤和代码分析

1. 定义Point类：

* 在Point类中，定义公有成员变量x和y来表示点的坐标。
* 提供构造函数用于设置点的坐标。
* 提供公有成员函数用于计算两个点之间的距离。

1. **class** Point
2. {
3. **public**:
4. Point(**double** x1, **double** y1);
5. ~Point();
7. // 计算两点之间的距离
8. **double** CalDis(**const** Point &point) **const**;
10. **double** x, y; // 点的坐标
11. };
12. 定义Circle类：

* 在Circle类中，定义私有成员变量center和r来表示圆心和半径。
* 提供构造函数用于获取和设置圆心和半径，同时将圆心的坐标设置默认参数(0.0,0.0)。
* 提供公有成员函数用于判断两个圆是否相交。

1. **class** Circle
2. {
3. **public**:
4. Circle(**double** r1, **double** x1 = 0.0, **double** y1 = 0.0);
5. ~Circle();
7. // 判断两个圆是否相交
8. **bool** IsIntersect(**const** Circle c) **const**;
10. **private**:
11. **double** r;     // 半径
12. Point center; // 圆心
13. };
14. 观察构造函数和析构函数的调用顺序：

* 在Point类和Circle类的构造函数和析构函数中添加输出提示信息，以便观察调用顺序和对象的创建和销毁过程。
* 运行程序，并观察控制台中的输出结果。

在Point类的构造函数和析构函数中使用标准输出流（cout）输出提示信息：

1. Point::Point(**double** x1, **double** y1) : x(x1), y(y1)
2. {
3. cout << "创建了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
4. }
6. Point::~Point()
7. {
8. cout << "销毁了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
9. }

在Circle类的构造函数和析构函数中使用标准输出流（cout）输出提示信息：

1. Circle::Circle(**double** r1, **double** x1, **double** y1) : r(r1), center(x1, y1)
2. {
3. cout << "创建了一个圆心为 (" << center.x << ", " << center.y
4. << ") ,半径为 " << r << " 的圆" << endl;
5. }
7. Circle::~Circle()
8. {
9. cout << "销毁了一个圆心为 (" << center.x << ", " << center.y
10. << ") ,半径为 " << r << " 的圆" << endl;
11. }
12. 计算两点之间的距离的函数

* 传入一个Point类的常量对象的引用。
* 限定函数为const，确保函数不会修改调用对象的成员变量。
* 使用<cmath>库中的sqrt函数进行距离求解。

1. // 计算两点之间的距离
2. **double** Point::CalDis(**const** Point &point) **const**
3. {
4. **return** sqrt((x - point.x) \* (x - point.x) + (y - point.y) \* (y - point.y));
5. }
6. 判断两个圆是否相交的函数

* 传入一个Circle类的常量对象。
* 限定函数为const，确保函数不会修改调用对象的成员变量。

1. // 判断两个圆是否相交
2. **bool** Circle::IsIntersect(**const** Circle c) **const**
3. {
4. **double** delta = center.CalDis(c.center) - r - c.r;
5. **if** (delta > 1e-6)
6. **return** **false**;
7. **else**
8. **return** **true**;
9. }
10. 创建圆形对象并进行操作：

* 在主函数中，创建两个Circle对象，分别表示两个圆形。
* 提示用户输入第一个圆的圆心坐标和半径，并设置对应的属性值。
* 提示用户输入第二个圆的圆心坐标和半径，并设置对应的属性值。
* 调用Circle类的判断相交方法，判断两个圆是否相交。
* 在主函数中适当位置添加system("pause")，以便观察构造函数和析构函数的调用情况。

### 3.2.4 实验源代码

circle.h

1. #ifndef CIRCLE\_H
2. #define CIRCLE\_H
4. **class** Point
5. {
6. **public**:
7. Point(**double** x1, **double** y1);
8. ~Point();
10. // 计算两点之间的距离
11. **double** CalDis(**const** Point &point) **const**;
13. **double** x, y; // 点的坐标
14. };
16. **class** Circle
17. {
18. **public**:
19. Circle(**double** r1, **double** x1 = 0.0, **double** y1 = 0.0);
20. ~Circle();
22. // 判断两个圆是否相交
23. **bool** IsIntersect(**const** Circle c) **const**;
25. **private**:
26. **double** r;     // 半径
27. Point center; // 圆心
28. };
30. #endif

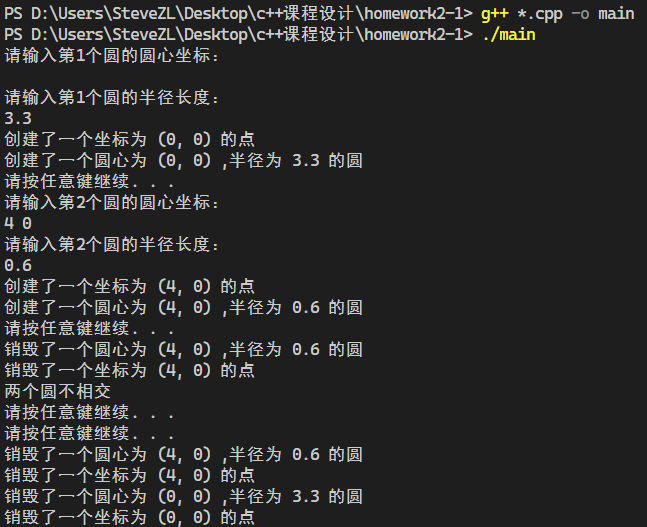
circle.cpp

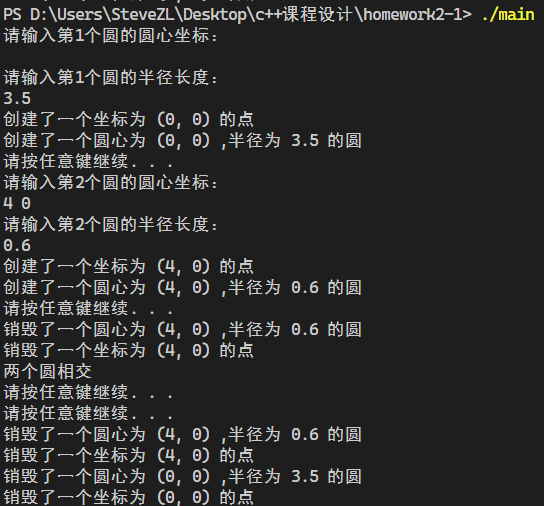
1. #include "circle.h"
2. #include <iostream>
3. #include <cmath>
4. **using** **namespace** std;
6. Point::Point(**double** x1, **double** y1) : x(x1), y(y1)
7. {
8. cout << "创建了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
9. }
11. Point::~Point()
12. {
13. cout << "销毁了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
14. }
16. // 计算两点之间的距离
17. **double** Point::CalDis(**const** Point &point) **const**
18. {
19. **return** sqrt((x - point.x) \* (x - point.x) + (y - point.y) \* (y - point.y));
20. }
22. Circle::Circle(**double** r1, **double** x1, **double** y1) : r(r1), center(x1, y1)
23. {
24. cout << "创建了一个圆心为 (" << center.x << ", " << center.y
25. << ") ,半径为 " << r << " 的圆" << endl;
26. }
28. Circle::~Circle()
29. {
30. cout << "销毁了一个圆心为 (" << center.x << ", " << center.y
31. << ") ,半径为 " << r << " 的圆" << endl;
32. }
34. // 判断两个圆是否相交
35. **bool** Circle::IsIntersect(**const** Circle c) **const**
36. {
37. **double** delta = center.CalDis(c.center) - r - c.r;
38. **if** (delta > 1e-6)
39. **return** **false**;
40. **else**
41. **return** **true**;
42. }

main.cpp

1. #include "circle.h"
2. #include <iostream>
3. **using** **namespace** std;
5. **int** main()
6. {
7. **double** x1 = 0.0, y1 = 0.0, r1;
8. **double** x2 = 0.0, y2 = 0.0, r2;
10. cout << "请输入第1个圆的圆心坐标：" << endl;
11. **if** (cin.peek() != '\n')
12. {
13. cin >> x1 >> y1;
14. }
15. cout << "请输入第1个圆的半径长度：" << endl;
16. cin >> r1;
17. **while** (r1 <= 0)
18. {
19. cout << "请输入一个值大于0的半径：" << endl;
20. cin >> r1;
21. }
22. Circle c1(r1, x1, y1);
23. system("pause");
25. getchar();
27. cout << "请输入第2个圆的圆心坐标：" << endl;
28. **if** (cin.peek() != '\n')
29. {
30. cin >> x2 >> y2;
31. }
32. cout << "请输入第2个圆的半径长度：" << endl;
33. cin >> r2;
34. **while** (r2 <= 0)
35. {
36. cout << "请输入一个值大于0的半径：" << endl;
37. cin >> r2;
38. }
39. Circle c2(r2, x2, y2);
40. system("pause");
42. **if** (c1.IsIntersect(c2))
43. cout << "两个圆相交" << endl;
44. **else**
45. cout << "两个圆不相交" << endl;
46. system("pause");
48. system("pause");
49. **return** 0;
50. }

### 3.2.5 实验结果展示





观察实验结果，我们可以看到构造函数在对象创建时被调用，析构函数在对象销毁时被调用。构造函数的调用顺序是先调用成员对象的构造函数，然后是本类的构造函数；析构函数的调用顺序与构造函数相反，先调用本类的析构函数，然后是成员对象的析构函数。

同时可以发现结果中有一次多调用的析构函数，这是由于在IsIntersect函数中参数为Circle类的对象，会调用默认的拷贝构造函数为其创建一个对象的副本，而该副本在函数执行完后会自动调用其析构函数~Circle进行析构。

## 3.3 类与对象实验——“矩阵”

### 3.3.1 概述

本实验旨在通过编写C++程序，定义一个矩阵类，实现矩阵的基本操作，包括矩阵的大小定义、动态内存分配、输入输出、矩阵相加和相减等功能。通过本次实验，我们将深入理解类与对象的概念，并掌握C++中类的设计和实现。

### 3.3.2 实验内容

编写C++程序完成“矩阵”类以下功能：

（1） 用类来实现矩阵，定义一个矩阵的类，属性包括：

* 矩阵大小，用 lines, rows（行、列来表示）；
* 存贮矩阵的数组指针，根据矩阵大小动态申请（new）。

（2） 矩阵类的方法包括：

构造函数，参数是矩阵大小，需要动态申请存贮矩阵的数组；

* 析构函数，需要释放矩阵的数组指针；
* 拷贝构造函数，需要申请和复制数组；
* 输入，可以从 cin 中输入矩阵元素；
* 输出，将矩阵格式化输出到 cout；
* 矩阵相加的函数，实现两个矩阵相加的功能，结果保存在另一个矩阵类，但必须矩阵大小相同；
* 矩阵相减的函数，实现两个矩阵相减的功能，结果保存在另一个矩阵类，但必须矩阵大小相同。

（3） 定义三个矩阵：A1、A2、A3；

（4） 初始化 A1、A2；

（5） 计算并输出 A3 = A1 加 A2，A3=A1 减 A2；（要求及提示：最好能实现对赋值操作符“=”的重载；注意检查“自赋值”、释放“旧元素”）

（6） 用 new 动态创建三个矩阵类的对象：pA1、pA1、pA3；

（7） 初始化 pA1、pA2；

（8） 计算并输出 pA3=pA1 加 pA2，pA3=pA1 减 pA2；

（9） 释放 pA1、pA2、pA3。

### 3.3.3 实验步骤和代码分析

1. 定义一个Matrix类

* 在Matrix类中，定义公有成员变量lines和rows来表示矩阵的行数和列数。
* 在Matrix类中，定义公有成员变量二维指针elem来指向矩阵元素在内存中的地址。
* 实现构造函数，在构造函数中动态申请内存来存储矩阵元素，同时完成构造函数的重载。
  1. 实现无参构造函数
  2. 实现有参构造函数
* 实现析构函数，在析构函数中释放动态分配的内存。
* 实现拷贝构造函数，进行矩阵的复制。
* 提供公有的输入和输出函数，使得用户能够输入和输出矩阵。
* 实现“+、-、=”运算操作符对矩阵类的重载。

1. #ifndef MATRIX\_H
2. #define MATRIX\_H
4. **class** Matrix
5. {
6. **public**:
7. // 构造函数
8. Matrix();
9. Matrix(**int** r, **int** l);
10. Matrix(**const** Matrix &m);
11. // 析构函数
12. ~Matrix();
14. **void** input();
15. **void** output();
17. // 对运算操作符“=、+、-”的重载
18. // 注意：类内操作符重载参数个数-1
19. Matrix &operator=(**const** Matrix &m);
20. Matrix operator+(**const** Matrix &m) **const**;
21. Matrix operator-(**const** Matrix &m) **const**;
22. **int** rows, lines; // 行、列
23. **int** \*\*elem;      // 矩阵元素
24. };
26. #endif
27. Matrix类的无参构造和有参构造函数

* 无参构造函数通过赋值列表将成员变量lines和rows赋值为0，将elem赋值为空。
* 有参构造函数根据参数r和l，通过赋值列表将成员变量lines和rows分别赋值为l和r，并使用new关键字动态申请内存空间创建一个大小为r\*l的矩阵。
  + 此处需要注意对内存分配失败的情况进行特殊判断。并且由于矩阵相当于一个二维数组，需要两次用到new关键字进行动态申请矩阵空间，因此两次都需要判断内存申请是否成功。
  + 同时此处使用new (std::nothrow)，在内存不足时，并不会抛出异常，而是将指针置NULL。

1. Matrix::Matrix() : lines(0), rows(0), elem(nullptr)
2. {
3. }
5. Matrix::Matrix(**int** r, **int** l) : rows(r), lines(l)
6. {
7. elem = **new** (**nothrow**) **int** \*[rows];
8. **if** (elem == nullptr)
9. {
10. cout << "动态分配内存失败" << endl;
11. **return**;
12. }
13. **for** (**int** i = 0; i < rows; i++)
14. {
15. elem[i] = **new** (**nothrow**) **int**[lines];
16. **if** (elem == nullptr)
17. {
18. cout << "动态分配内存失败" << endl;
19. **return**;
20. }
21. }
22. }
23. Matrix类的拷贝构造函数

* 传入一个Matrix类的常量对象m的引用，通过该常量对象的成员变量m.lines和m.rows对this->lines和this->rows赋值。并使用new关键字动态申请内存空间创建一个大小为this->rows\*this->lines的矩阵。

1. Matrix::Matrix(**const** Matrix &m) : lines(m.lines), rows(m.rows)
2. {
3. elem = **new** (**nothrow**) **int** \*[rows];
4. **if** (elem == nullptr)
5. {
6. cout << "动态分配内存失败" << endl;
7. **return**;
8. }
9. **for** (**int** i = 0; i < rows; i++)
10. {
11. elem[i] = **new** (**nothrow**) **int**[lines];
12. **if** (elem == nullptr)
13. {
14. cout << "动态分配内存失败" << endl;
15. **return**;
16. }
17. **for** (**int** j = 0; j < lines; j++)
18. elem[i][j] = m.elem[i][j];
19. }
20. }
21. Matrix类的析构函数

* 使用delete关键字释放给矩阵变量动态分配的内存空间。
* 同时在释放指针所指向的空间时，首先要对指针进行非空判断，防止出错，在释放完内存空间后，要手动将指针指向空，防止野指针问题。
* 在为矩阵动态分配内存空间时，我们是从外到内动态分配，所以释放时应该按照从内到外的释放顺序，否则会造成内存丢失。

1. Matrix::~Matrix()
2. {
3. **for** (**int** i = 0; i < rows; i++)
4. {
5. **if** (elem[i] != nullptr)
6. {
7. **delete**[] elem[i];
8. elem[i] = nullptr;
9. }
10. }
11. **if** (elem != nullptr)
12. {
13. **delete**[] elem;
14. elem = nullptr;
15. }
16. }
17. Matrix类的输入函数

* 通过标准输入流(cin)获取用户输入的矩阵元素，并将其存储到成员变量elem所指向的内存空间中。

1. Matrix类的输出函数

* 使用循环遍历矩阵中的元素，并通过标准输出流(cout)格式化输出矩阵。

1. Matrix类的“+”运算符重载

* 传入一个Matrix类的常量对象m的引用，同时限定重载函数为const，确保不会对this的成员变量的值进行修改。
* 需要对进行相加操作的两个矩阵大小是否相同进行判断，不同时需要输出错误信息，并终止程序。
* 返回值为一个Matrix类的对象，该对象中存储着运算的结果。

1. Matrix Matrix::operator+(**const** Matrix &m) **const**
2. {
3. **if** (rows != m.rows || lines != m.lines)
4. {
5. cout << "发生错误：大小不相同的两个矩阵无法做加法" << endl;
6. exit(1);
7. }
9. Matrix result(rows, lines);
10. **for** (**int** i = 0; i < rows; i++)
11. **for** (**int** j = 0; j < lines; j++)
12. {
13. result.elem[i][j] = elem[i][j] + m.elem[i][j];
14. }
15. **return** result;
16. }
17. Matrix类的“-”运算符重载

* 传入一个Matrix类的常量对象m的引用，同时限定重载函数为const，确保不会对this的成员变量的值进行修改。
* 需要对进行相减操作的两个矩阵大小是否相同进行判断，不同时需要输出错误信息，并终止程序。
* 返回值为一个Matrix类的对象，该对象中存储着运算的结果。

1. Matrix Matrix::operator-(**const** Matrix &m) **const**
2. {
3. **if** (rows != m.rows || lines != m.lines)
4. {
5. cout << "发生错误：大小不相同的两个矩阵无法做减法" << endl;
6. exit(1);
7. }
9. Matrix result(rows, lines);
10. **for** (**int** i = 0; i < rows; i++)
11. **for** (**int** j = 0; j < lines; j++)
12. result.elem[i][j] = elem[i][j] - m.elem[i][j];
13. **return** result;
14. }
15. Matrix类的“=”运算符重载

* 传入一个Matrix类的常量对象m的引用，同时判断&m是否等于this，避免自赋值的情况。
* 若不是自赋值的情况，先将this析构，释放掉可能已经为this中成员变量动态分配的内存，再为this中的成员变量赋值和动态分配内存。此处仍要对动态分配内存的结果进行判断。
* 最后返回\*this。

1. Matrix &Matrix::operator=(**const** Matrix &m)
2. {
3. **if** (**this** != &m)
4. {
5. **this**->~Matrix(); // 先释放空间
6. rows = m.rows;
7. lines = m.lines;
8. elem = **new** (**nothrow**) **int** \*[rows];
9. **if** (elem == nullptr)
10. {
11. cout << "动态分配内存失败" << endl;
12. exit(1);
13. }
14. **for** (**int** i = 0; i < rows; i++)
15. {
16. elem[i] = **new** (**nothrow**) **int**[lines];
17. **if** (elem == nullptr)
18. {
19. cout << "动态分配内存失败" << endl;
20. exit(1);
21. }
22. **for** (**int** j = 0; j < lines; j++)
23. elem[i][j] = m.elem[i][j];
24. }
25. }
26. // 自赋值直接返回
27. **return** \***this**;
28. }
29. 创建Matrix对象和指针

* 在主函数中，创建三个Matrix对象：A1、A2和A3，并进行初始化。
* 使用输入函数从标准输入流中输入矩阵元素，并使用输出函数以格式化的方式将矩阵输出到标准输出流。
* 调用矩阵相加和相减的函数，计算A1加A2和A1减A2的结果，并将结果保存在A3中。
* 使用输出函数将A3以格式化的方式输出到标准输出流。
* 使用new关键字创建三个Matrix对象的指针：pA1、pA2和pA3，并进行初始化。
* 使用输入函数从标准输入流中输入矩阵元素，并使用输出函数以格式化的方式将矩阵输出到标准输出流。
* 调用矩阵相加和相减的函数，计算pA1加pA2和pA1减pA2的结果，并将结果保存在pA3中。
* 使用输出函数将pA3以格式化的方式输出到标准输出流。
* 释放为pA1、pA2、pA3动态分配的内存。

### 3.3.4 实验源代码

matrix.h

1. #ifndef MATRIX\_H
2. #define MATRIX\_H
4. **class** Matrix
5. {
6. **public**:
7. // 构造函数
8. Matrix();
9. Matrix(**int** r, **int** l);
10. Matrix(**const** Matrix &m);
11. // 析构函数
12. ~Matrix();
14. **void** input();
15. **void** output();
17. // 对运算操作符“=、+、-”的重载
18. // 注意：类内操作符重载参数个数-1
19. Matrix &operator=(**const** Matrix &m);
20. Matrix operator+(**const** Matrix &m) **const**;
21. Matrix operator-(**const** Matrix &m) **const**;
22. **int** rows, lines; // 行、列
23. **int** \*\*elem;      // 矩阵元素
24. };
26. #endif

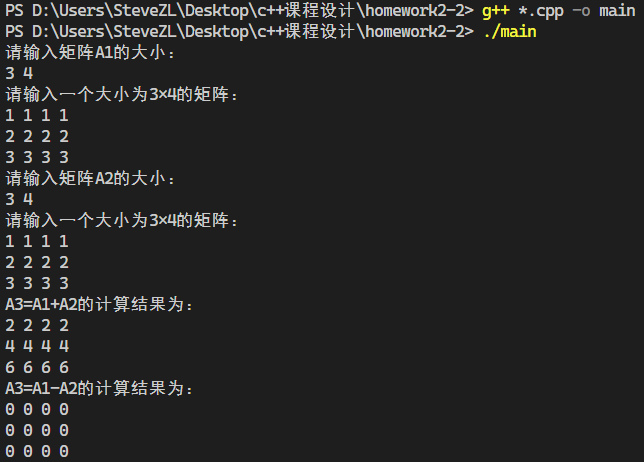
matrix.cpp

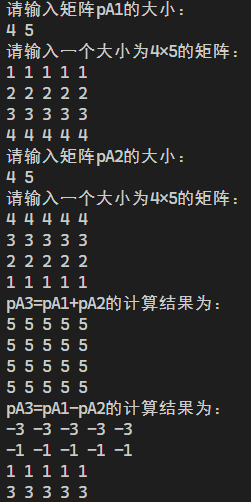
1. #include "matrix.h"
2. #include <iostream>
3. **using** **namespace** std;
5. Matrix::Matrix() : lines(0), rows(0), elem(nullptr)
6. {
7. }
9. Matrix::Matrix(**int** r, **int** l) : rows(r), lines(l)
10. {
11. elem = **new** (**nothrow**) **int** \*[rows];
12. **if** (elem == nullptr)
13. {
14. cout << "动态分配内存失败" << endl;
15. **return**;
16. }
17. **for** (**int** i = 0; i < rows; i++)
18. {
19. elem[i] = **new** (**nothrow**) **int**[lines];
20. **if** (elem == nullptr)
21. {
22. cout << "动态分配内存失败" << endl;
23. **return**;
24. }
25. }
26. }
28. Matrix::Matrix(**const** Matrix &m) : lines(m.lines), rows(m.rows)
29. {
30. elem = **new** (**nothrow**) **int** \*[rows];
31. **if** (elem == nullptr)
32. {
33. cout << "动态分配内存失败" << endl;
34. **return**;
35. }
36. **for** (**int** i = 0; i < rows; i++)
37. {
38. elem[i] = **new** (**nothrow**) **int**[lines];
39. **if** (elem == nullptr)
40. {
41. cout << "动态分配内存失败" << endl;
42. **return**;
43. }
44. **for** (**int** j = 0; j < lines; j++)
45. elem[i][j] = m.elem[i][j];
46. }
47. }
49. Matrix::~Matrix()
50. {
51. **for** (**int** i = 0; i < rows; i++)
52. {
53. **if** (elem[i] != nullptr)
54. {
55. **delete**[] elem[i];
56. elem[i] = nullptr;
57. }
58. }
59. **if** (elem != nullptr)
60. {
61. **delete**[] elem;
62. elem = nullptr;
63. }
64. }
66. **void** Matrix::input()
67. {
68. cout << "请输入一个大小为" << rows << "×" << lines << "的矩阵：" << endl;
69. **for** (**int** i = 0; i < rows; i++)
70. **for** (**int** j = 0; j < lines; j++)
71. cin >> elem[i][j];
72. }
74. **void** Matrix::output()
75. {
76. **for** (**int** i = 0; i < rows; i++)
77. {
78. **for** (**int** j = 0; j < lines; j++)
79. cout << elem[i][j] << " ";
80. cout << endl;
81. }
82. }
84. Matrix Matrix::operator+(**const** Matrix &m) **const**
85. {
86. **if** (rows != m.rows || lines != m.lines)
87. {
88. cout << "发生错误：大小不相同的两个矩阵无法做加法" << endl;
89. exit(1);
90. }
92. Matrix result(rows, lines);
93. **for** (**int** i = 0; i < rows; i++)
94. **for** (**int** j = 0; j < lines; j++)
95. {
96. result.elem[i][j] = elem[i][j] + m.elem[i][j];
97. }
98. **return** result;
99. }
101. Matrix Matrix::operator-(**const** Matrix &m) **const**
102. {
103. **if** (rows != m.rows || lines != m.lines)
104. {
105. cout << "发生错误：大小不相同的两个矩阵无法做减法" << endl;
106. exit(1);
107. }
109. Matrix result(rows, lines);
110. **for** (**int** i = 0; i < rows; i++)
111. **for** (**int** j = 0; j < lines; j++)
112. result.elem[i][j] = elem[i][j] - m.elem[i][j];
113. **return** result;
114. }
116. Matrix &Matrix::operator=(**const** Matrix &m)
117. {
118. **if** (**this** != &m)
119. {
120. **this**->~Matrix(); // 先释放空间
121. rows = m.rows;
122. lines = m.lines;
123. elem = **new** (**nothrow**) **int** \*[rows];
124. **if** (elem == nullptr)
125. {
126. cout << "动态分配内存失败" << endl;
127. exit(1);
128. }
129. **for** (**int** i = 0; i < rows; i++)
130. {
131. elem[i] = **new** (**nothrow**) **int**[lines];
132. **if** (elem == nullptr)
133. {
134. cout << "动态分配内存失败" << endl;
135. exit(1);
136. }
137. **for** (**int** j = 0; j < lines; j++)
138. elem[i][j] = m.elem[i][j];
139. }
140. }
141. // 自赋值直接返回
142. **return** \***this**;
143. }

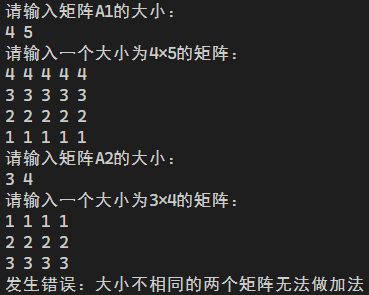
main.cpp

1. #include "matrix.h"
2. #include <iostream>
3. **using** **namespace** std;
5. **int** main()
6. {
7. **int** r1, l1;
8. **int** r2, l2;
10. cout << "请输入矩阵A1的大小：" << endl;
11. cin >> r1 >> l1;
12. Matrix A1(r1, l1);
13. A1.input();
15. cout << "请输入矩阵A2的大小：" << endl;
16. cin >> r2 >> l2;
17. Matrix A2(r2, l2);
18. A2.input();
20. Matrix A3 = A1 + A2;
21. cout << "A3=A1+A2的计算结果为：" << endl;
22. A3.output();
23. A3 = A1 - A2;
24. cout << "A3=A1-A2的计算结果为：" << endl;
25. A3.output();
27. cout << "请输入矩阵pA1的大小：" << endl;
28. cin >> r1 >> l1;
29. Matrix \*pA1 = **new** Matrix(r1, l1);
30. pA1->input();
32. cout << "请输入矩阵pA2的大小：" << endl;
33. cin >> r2 >> l2;
34. Matrix \*pA2 = **new** Matrix(r2, l2);
35. pA2->input();
37. Matrix \*pA3 = **new** Matrix;
38. \*pA3 = \*pA1 + \*pA2;
39. cout << "pA3=pA1+pA2的计算结果为：" << endl;
40. pA3->output();
41. \*pA3 = \*pA1 - \*pA2;
42. cout << "pA3=pA1-pA2的计算结果为：" << endl;
43. pA3->output();
45. // 释放 pA1、pA2、pA3
46. **delete** pA1;
47. **delete** pA2;
48. **delete** pA3;
50. **return** 0;
51. }

### 3.3.5 实验结果展示







(两个大小不相同的矩阵进行运算时发生错误)

## 3.4 继承与派生实验

### 3.4.1 概述

本实验旨在通过通过编写C++程序，定义和实现Shape类及其派生类Rectangle、Circle和Square。我们将学习如何在C++中使用继承和派生，以及掌握虚函数的概念和使用方法。本次实验将展示不同形状的面积计算和对象的构造与析构过程。

### 3.4.2 实验内容

编写C++程序完成“形状”的以下功能：

（1） 声明一个基类 Shape（形状），其中包含一个方法来计算面积；

（2） 从 Shape 派生两个类：矩形类和圆形类；

（3） 从矩形类派生正方形类；

（4） 分别实现派生类构造函数、析构函数和其他方法；

（5） 创建派生类的对象，观察构造函数、析构函数调用次序；

（提示及要求：1，可通过在构造与析构函数中加入输出提示信息的方式观察相关调用；可以使用 system("pause")进行程序的暂停；2，能够理解并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及其原因）

（6） 对不同对象计算面积。

### 3.4.3 实验步骤和代码分析

1. 定义Shape类及其派生类

* 定义Shape类作为基类，并声明构造函数和析构函数。
* 在Rectangle类中，继承Shape类，并添加height和width成员变量，声明构造函数和析构函数。
* 在Circle类中，继承Shape类，并添加radius成员变量，声明构造函数和析构函数。
* 在Square类中，继承Rectangle类，声明构造函数和析构函数。

1. // 形状类
2. **class** Shape
3. {
4. **public**:
5. Shape();
6. ~Shape();
7. };
9. // Shape类的派生：矩形类
10. **class** Rectangle : **public** Shape
11. {
12. **protected**:
13. **double** height, width;
15. **public**:
16. Rectangle(**double** h, **double** w);
17. ~Rectangle();
18. **double** area() **const**;
19. };
21. // Shape类的派生：圆形类
22. **class** Circle : **public** Shape
23. {
24. **private**:
25. **double** radius;
27. **public**:
28. Circle(**double** r);
29. ~Circle();
30. **double** area() **const**;
31. };
33. // 矩形类的派生：正方形类
34. **class** Square : **public** Rectangle
35. {
36. **public**:
37. Square(**double** side);
38. ~Square();
39. };
40. 实现类的构造函数和析构函数

* 在shape.cpp源文件中，实现Shape类及其派生类的构造函数和析构函数，并输出相应的构造和析构信息。

1. 实现area()函数

* 实现Rectangle和Circle类中的area()函数，计算不同形状的面积，Square类的面积计算可以调用其基类Rectangle的area()函数实现。

1. **double** Rectangle::area() **const**
2. {
3. **return** width \* height;
4. }
5. **double** Circle::area() **const**
6. {
7. **return** PI \* radius \* radius;
8. }
9. 主函数中的对象创建和调用

* 在main.cpp源文件中，创建不同形状的对象，包括Rectangle、Circle和Square，并调用相应的成员函数计算面积。
* 由于矩形和正方形的边长都应>0,同时圆形的半径也应>0，所以应该对于用户输入的非正数进行告警，要求用户重新输入正数作为边长或者半径。

### 3.4.4 实验源代码

shape.h

1. #ifndef SHAPE\_H
2. #define SHAPE\_H
4. **const** **double** PI = 3.14159265358979323846;
6. // 形状类
7. **class** Shape
8. {
9. **public**:
10. Shape();
11. ~Shape();
12. };
14. // Shape类的派生：矩形类
15. **class** Rectangle : **public** Shape
16. {
17. **protected**:
18. **double** height, width;
20. **public**:
21. Rectangle(**double** h, **double** w);
22. ~Rectangle();
23. **double** area() **const**;
24. };
26. // Shape类的派生：圆形类
27. **class** Circle : **public** Shape
28. {
29. **private**:
30. **double** radius;
32. **public**:
33. Circle(**double** r);
34. ~Circle();
35. **double** area() **const**;
36. };
38. // 矩形类的派生：正方形类
39. **class** Square : **public** Rectangle
40. {
41. **public**:
42. Square(**double** side);
43. ~Square();
44. };
46. #endif

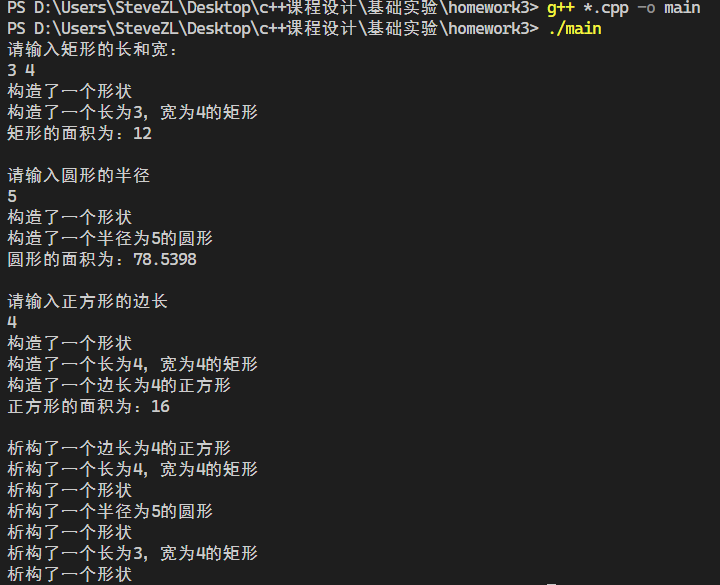
shape.cpp

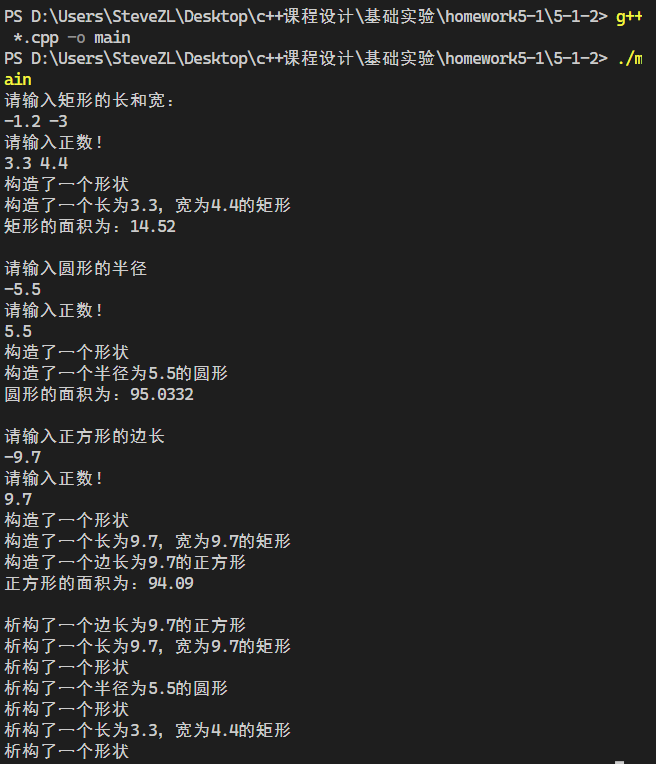
1. #include "shape.h"
2. #include <cmath>
3. #include <iostream>
4. **using** **namespace** std;
6. Shape::Shape()
7. {
8. cout << "构造了一个形状" << endl;
9. }
11. Shape::~Shape()
12. {
13. cout << "析构了一个形状" << endl;
14. }
16. Rectangle::Rectangle(**double** h, **double** w) : height(h), width(w)
17. {
18. cout << "构造了一个长为" << height << "，宽为" << width << "的矩形" << endl;
19. }
21. Rectangle::~Rectangle()
22. {
23. cout << "析构了一个长为" << height << "，宽为" << width << "的矩形" << endl;
24. }
26. **double** Rectangle::area() **const**
27. {
28. **return** width \* height;
29. }
31. Circle::Circle(**double** r) : radius(r)
32. {
33. cout << "构造了一个半径为" << radius << "的圆形" << endl;
34. }
36. Circle::~Circle()
37. {
38. cout << "析构了一个半径为" << radius << "的圆形" << endl;
39. }
41. **double** Circle::area() **const**
42. {
43. **return** PI \* radius \* radius;
44. }
46. Square::Square(**double** side) : Rectangle(side, side)
47. {
48. cout << "构造了一个边长为" << height << "的正方形" << endl;
49. }
51. Square::~Square()
52. {
53. cout << "析构了一个边长为" << height << "的正方形" << endl;
54. }

main.cpp

1. #include "shape.h"
2. #include <iostream>
3. **using** **namespace** std;
5. **int** main()
6. {
7. **double** h, w;
8. **double** r;
9. **double** s;
11. cout << "请输入矩形的长和宽：" << endl;
12. cin >> h >> w;
13. **while** (h <= 0 || w <= 0)
14. {
15. cout << "请输入正数！" << endl;
16. cin >> h >> w;
17. }
18. Rectangle rec(h, w);
19. cout << "矩形的面积为：" << rec.area() << endl;
20. cout << endl;
22. cout << "请输入圆形的半径" << endl;
23. cin >> r;
24. **while** (r <= 0)
25. {
26. cout << "请输入正数！" << endl;
27. cin >> r;
28. }
29. Circle cir(r);
30. cout << "圆形的面积为：" << cir.area() << endl;
31. cout << endl;
33. cout << "请输入正方形的边长" << endl;
34. cin >> s;
35. **while** (s <= 0)
36. {
37. cout << "请输入正数！" << endl;
38. cin >> s;
39. }
40. Square squ(s);
41. cout << "正方形的面积为：" << squ.area() << endl;
42. cout << endl;
44. **return** 0;
45. }

### 3.4.5 实验结果展示





（输入负数时，程序给出相应告警提示）

观察实验结果，我们可以看到构造函数在对象创建时被调用，析构函数在对象销毁时被调用。构造函数的调用顺序是先调用基类的构造函数，再调用成员对象的构造函数，最后才是本类的构造函数；析构函数的调用顺序与构造函数相反，先调用本类的析构函数，然后是成员对象的析构函数，最后才是基类的析构函数。

## 3.5 I/O 流实验

### 3.5.1 概述

本实验旨在通过编写C++程序，实现一个猜价格的游戏。我们将学习如何使用C++的I/O流，包括输入和输出流，以及如何进行输入验证和异常处理。

### 3.5.2 实验内容

编写 C++程序完成猜价格游戏的以下功能：

（1） 假定有一件商品，程序用随机数指定该商品的价格（1-1000 的整数）；

（2） 提示用户猜价格，并输入：若用户猜的价格比商品价格高或低，对用户作出相应的提示；

（3） 直到猜对为止，并给出提示。

（提示及要求：1，要求使用 C++的输入输出方式（cin, cout），不能使用 C 语言的 printf 等；2，注意检查输入的合法性）

### 3.5.3 实验步骤和代码分析

1. 生成随机价格

* 使用default\_random\_engine和uniform\_int\_distribution类生成一个1-1000的随机整数，作为商品的价格。

1. // 随机产生一个1-1000的整数
2. default\_random\_engine e;
3. uniform\_int\_distribution<**int**> u(1, 1000);
4. e.seed(time(0));
5. price = u(e);
6. 实现猜价格的游戏逻辑

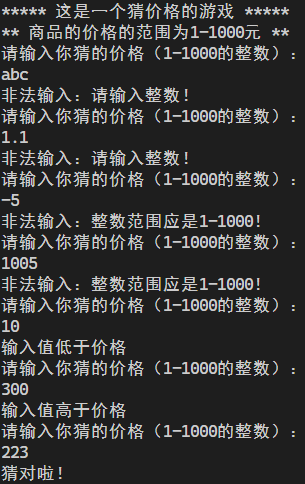
* 在主函数中，使用循环结构，当用户的猜测价格不等于实际价格时，继续循环。
* 提示用户输入猜测的价格，使用cin流接收用户的输入。
* 对用户的输入进行验证：
  + 检查输入是否为整数，若不是整数则输出“非法输入：请输入整数！”的错误提示。。
  + 检查输入是否在合法范围内（1-1000），若不在范围内则输出“非法输入：整数范围应是1-1000！”错误信息。
* 根据用户的猜测结果给予相应的提示：
  + 若猜测价格高于实际价格，则输出提示信息：“输入值高于价格”。
  + 若猜测价格低于实际价格，则输出提示信息：“输入值低于价格”。
  + 若猜测价格等于实际价格，则输出提示信息：“猜对啦！”。

### 3.5.4 实验源代码

main.cpp

1. #include <iostream>
2. #include <ctime>
3. #include <random>
4. **using** **namespace** std;
6. **int** main()
7. {
8. **double** guess = 0, price;
10. // 随机产生一个1-1000的整数
11. default\_random\_engine e;
12. uniform\_int\_distribution<**int**> u(1, 1000);
13. e.seed(time(0));
14. price = u(e);
15. // cout << price << endl;
17. cout << "\*\*\*\*\* 这是一个猜价格的游戏 \*\*\*\*\* " << endl;
18. cout << "\*\* 商品的价格的范围为1-1000元 \*\* " << endl;
20. **while** (guess != price)
21. {
22. cout << "请输入你猜的价格（1-1000的整数）：" << endl;
23. cin >> guess;
25. // 输入非数字
26. **if** (cin.fail())
27. {
28. cin.clear();            // 清除cin的错误状态
29. cin.ignore(1024, '\n'); // 忽略缓冲区的内容，直到EOF
30. cout << "非法输入：请输入整数！" << endl;
31. }
32. // 输入小数
33. **else** **if** (guess - **int**(guess))
34. cout << "非法输入：请输入整数！" << endl;
35. // 超出范围
36. **else** **if** (guess < 1 || guess > 1000)
37. cout << "非法输入：整数范围应是1-1000！" << endl;
39. **else** **if** (guess > price)
40. cout
41. << "输入值高于价格" << endl;
42. **else** **if** (guess < price)
43. cout
44. << "输入值低于价格" << endl;
45. **else**
46. cout << "猜对啦！" << endl;
47. }
49. **return** 0;
50. }

### 3.5.5 实验结果展示



可以发现当输入abc，1.1，-5等非法输入时，程序都进行了相应的非法输入提示。

## 3.6 重载实验——“虚函数”

### 3.6.1 概述

本实验旨在通过对题目3中的“形状”类进行改进，学习虚函数的概念和用法，并比较其与非虚函数的差异。此外，我们还将尝试将基类定义为抽象类，以比较抽象类和普通类之间的差异。

### 3.6.2 实验内容

针对题目3的“形状”类，编写C++程序完成以下功能：

（1） 将【形状】 中的基类计算面积的方法定义为虚函数，比较与【形状（A）】程

序的差异；

（2） 将【形状】中的基类定义为抽象类，比较与【形状（A）】程序的差异。

### 3.6.3 实验步骤和代码分析

1. 虚函数定义

* 在“形状”类中，将计算面积的方法定义为虚函数。这样做可以使得在使用多态性时能正确调用派生类的方法，实现多态的效果。
* 创建一个“形状”类对象并调用计算面积的方法，观察结果。
* 比较虚函数与普通函数的差异。

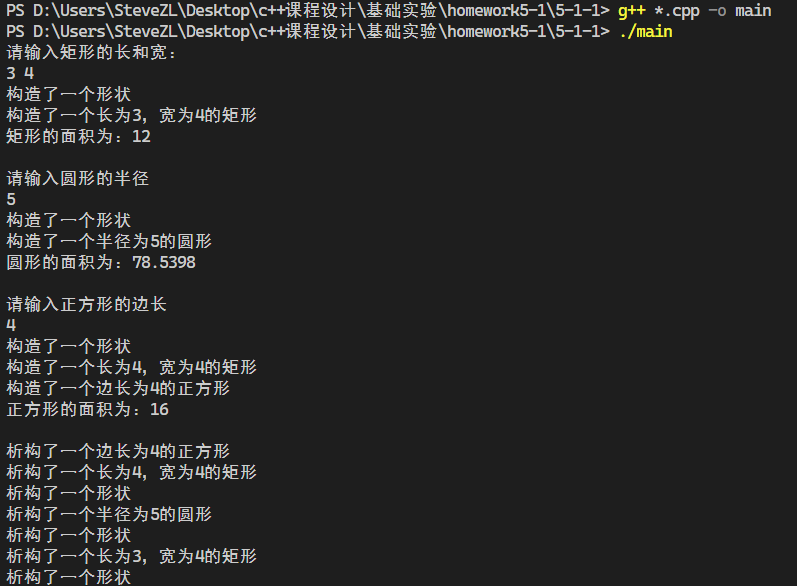
1. // 形状类
2. **class** Shape
3. {
4. **public**:
5. Shape();
6. ~Shape();
7. // 虚函数
8. // 计算面积
9. **virtual** **double** area();
10. };
11. 将基类定义为抽象类

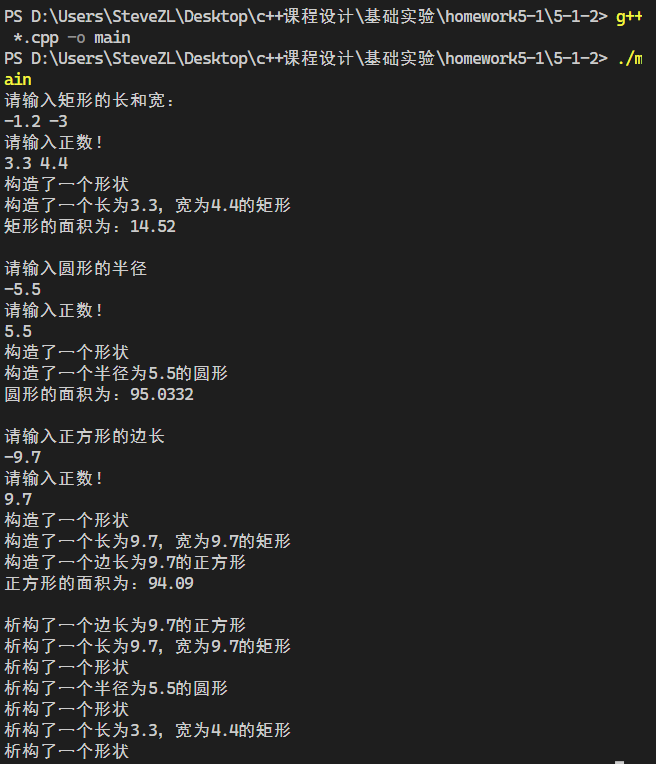
* 修改“形状”类，使其成为抽象类，即在基类中添加一个纯虚函数。
* 创建一个抽象类对象并调用计算面积的方法，观察结果。
* 比较抽象类与普通类的差异。

1. // 形状类
2. // 抽象类
3. **class** Shape
4. {
5. **public**:
6. Shape();
7. ~Shape();
8. // 纯虚函数
9. // 计算面积
10. **virtual** **double** area() **const** = 0;
11. };

### 3.6.4 实验源代码（主要差异在于.h文件，上面代码中已经展示了shape.h的修改内容，因此不再单独附上代码）

### 3.6.5 实验结果展示





## 3.7 重载实验——“对Point类重载++和--运算符”

### 3.7.1 概述

本实验旨在通过对Point类进行改进，理解运算符重载的概念和用法，学习如何重载自增（++）和自减（--）运算符，并比较其不同形式的使用。

### 3.7.2 实验内容

编写C++程序完成以下功能：

（1） Point 类的属性包括点的坐标（x，y）；

（2） 实现 Point 类重载＋＋和――运算符：

* ++p，--p，p++，p--；
* ++和--分别表示 x，y 增加或减少 1

### 3.7.3 实验步骤和代码分析

1. 运算符重载的实现

* 在Point类中，重载自增（++）和自减（--）运算符。
* 分别实现前置和后置形式的自增和自减运算符。

1. 运算符的使用

* 在主函数中，创建一个Point对象，并对其进行自增和自减运算符的操作。
* 输出运算符操作后的结果，观察运算符的使用效果。

1. 输出运算符重载

* 将输出运算符重载函数定义为Point类的友元函数，以便访问Point类的私有成员变量。
* 可以直接用“cout<<”以“(x,y)”的格式输出Point类的对象的私有成员变量x和y。

1. ostream &operator<<(ostream &cout, **const** Point &p)
2. {
3. cout << "(" << p.x << "," << p.y << ")";
4. **return** cout;
5. }

### 3.7.4 实验源代码

point.h

1. #ifndef POINT\_H
2. #define POINT\_H
3. #include <iostream>
4. **using** **namespace** std;
6. **class** Point
7. {
8. **friend** ostream &operator<<(ostream &cout, **const** Point &p);
10. **public**:
11. Point(**double** x1, **double** y1);
12. ~Point();
13. Point(Point &p);
15. Point &operator++();
16. Point &operator--();
17. Point operator++(**int**);
18. Point operator--(**int**);
20. **private**:
21. **double** x, y; // 点的坐标
22. };
24. #endif

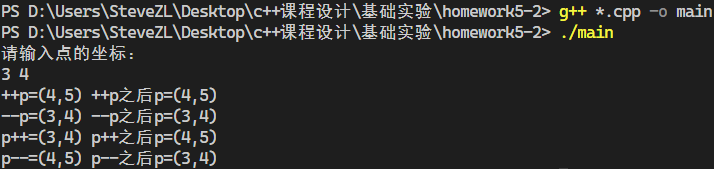
point.cpp

1. #include "point.h"
3. Point::Point(**double** x1, **double** y1) : x(x1), y(y1)
4. {
5. // cout << "构造了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
6. }
8. Point::~Point()
9. {
10. // cout << "析构了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
11. }
13. Point::Point(Point &p)
14. {
15. x = p.x;
16. y = p.y;
17. // cout << "拷贝构造了一个坐标为 (" << x << ", " << y << ") 的点" << endl;
18. }
20. Point &Point::operator++()
21. {
22. x++, y++;
23. **return** \***this**;
24. }
26. Point &Point::operator--()
27. {
28. x--, y--;
29. **return** \***this**;
30. }
32. Point Point::operator++(**int**)
33. {
34. Point temp = \***this**;
35. x++, y++;
36. **return** temp;
37. }
39. Point Point::operator--(**int**)
40. {
41. Point temp = \***this**;
42. x--, y--;
43. **return** temp;
44. }
46. ostream &operator<<(ostream &cout, **const** Point &p)
47. {
48. cout << "(" << p.x << "," << p.y << ")";
49. **return** cout;
50. }

main.cpp

1. #include "point.h"
3. **int** main()
4. {
5. **double** x, y;
7. cout << "请输入点的坐标：" << endl;
8. cin >> x >> y;
9. Point p(x, y);
11. cout << "++p=" << ++p << " "
12. << "++p之后p=" << p << endl;
13. cout << "--p=" << --p << " "
14. << "--p之后p=" << p << endl;
15. cout << "p++=" << p++ << " "
16. << "p++之后p=" << p << endl;
17. cout << "p--=" << p-- << " "
18. << "p--之后p=" << p << endl;
20. **return** 0;
21. }

### 3.7.5 实验结果展示



可以发现成功实现了p++，++p，p--，--p，完成了对++，--运算符的重载。

# 4.综合实验

## 4.1 任务描述

使用 C++ 语言，基于面向对象的程序设计方法，设计并实现一个简单的单词消除小游戏，提供用户登录、注册、闯关、出题、查看排行榜、多人对战等功能。

## 4.2 功能需求

**用户管理**

* 实现闯关者，出题者本地的注册、登录（闯关者与出题者有共同的基类）。
* 程序支持多人注册。
* 闯关者每闯过一关，增加一定经验值。经验值会根据闯过的该关卡的关卡号、该关的闯关耗费时间共同决定。当经验值累计到一定程度闯关者等级增加。闯关失败需要重新闯该关。
* 每成功出题一次，更新该出题者的出题数目。出题者等级根据出题者成功出题数目来升级。

**单词管理**

* 游戏自带词库，而且已经注册的出题者可以为系统出题，即增加词库的新词，已经存在的单词不能再次添加。
* 每次出题时，系统从该单词池中按照关卡难度随机的选择相应长度的单词

**游戏管理**

* 实现游戏规则：出题者增加游戏中使用单词。游戏每一关，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词， 如果闯关者输入正确则为通过。
* 单词难度可以递增或者持平（即长度加长或不变）。
* 进行轮数增多（即单词数目增加，如：前三关仅仅通过一个单词就过关，后续需要通过两个、三个甚至更多才过关）。
* 单词显示时间缩短（随着关卡的增加显示时间越来越短）。
* 任何角色均可查询所有闯关者、出题者，按照属性查找相应闯关者、出题者。
* 可以根据闯关者闯过关卡数、等级对闯关者排名，根据出题者出题数目、等级对出题者排名。
* 双人对战游戏，要求参与闯关者均已经登录，双人同时面对一个单词，最先打出正确单词者获得经验增长，在双人对战中获胜所获得的经验增长要高于

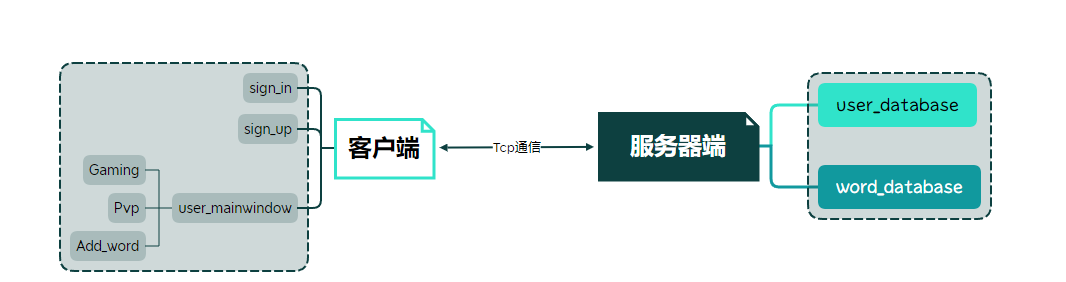
从同等难度的单人游戏中所获得的经验增长，失败者则需要扣除一定经验值。

**客户端和服务器管理**

* 将所有闯关者、出题者信息保存在服务器。
* 使用 socket 进行通信。
* 客户端可以启动多个同时与服务器交互，要求服务器具有并发处理能力。

## 4.3 模块介绍

### 4.3.1模块划分



单词消除小游戏采用C/S架构，由客户端和服务端两部分组成。客户端模块根据用户的操作生成相应的网络报文，并且通过socket通信与服务器端进行交互。

服务器端与客户端通信、解析客户端发送来的报文请求，并从数据库中抓取相应信息生成回复报文回复客户端。

### 4.3.2 服务器端

根据需求，服务器端需要对客户端的以下13种请求进行分析并回复报文以响应：

1. checkPassword 检测用户名和密码是否匹配
2. getUser 获取用户信息
3. findUser 查找用户
4. addUser 添加用户
5. updateUser 更新用户信息
6. get\_all\_userInfo 获取所有用户信息
7. resetPassword 重置密码
8. addWord 添加单词
9. getWord 随机获取满足条件的单词
10. pvp 多人对战
11. cancel\_pvp 取消匹配
12. end\_pvp 对战结束
13. quit\_midway 中途退出pvp

服务器端通过访问用户数据库user.db和单词数据库word.db，获取客户端请求的信息进行回复。

#### 4.3.2.1 用户管理

请求1-7属于用户管理功能，在User\_Database类中实现。该类中实现的函数有：

* 数据库初始化

1. **static** **bool** init();

* 按照用户名查询用户，查询到返回true

1. **bool** findUser(**const** QString& username);

* 输出数据库所有记录

1. **void** printAllRecords() **const**;

* 清除数据库信息

1. **void** clearAllRecords();

* 根据用户信息向数据库添加用户

1. **void** addUser(**const** User& user, **int** role, QString question, QString answer);

* 检验用户名和密码是否匹配

1. **bool** checkPassword(**const** QString& username, **const** QString& password);

* 更新用户信息

1. **void** updateUser(**const** QString& username, **const** **int**& level, **const** **int**& experience, **const** **int**& questionCount, **const** **int**& passedLevels);

* 获取用户信息

1. **void** get\_User(**const** QString& username, **int**& role, **int**& level, **int**& experience, **int**& questionCount, **int**& passedLevels);

* 向json文件中写入所有用户信息

1. **bool** writeToJsonFile(QJsonArray& json) **const**;

* 重置密码

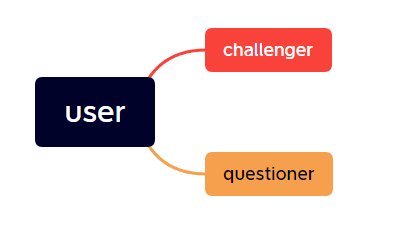
1. **void** resetPassword(**const** QString username, **const** QString password)**const**;

通过以上函数综合实现用户的登录、注册、找回密码、获取用户信息等功能。同时用户信息保存在数据库中，数据库采用SQLite，利用QSqlDatabase类和QSqlQuery类，实现对数据库表项的增删和修改操作。

数据库中user表的格式如下：

1. "username varchar(30) primary key," // 用户名
2. "password varchar(30),"             // 密码
3. "role int,"                         // 角色
4. "level int,"                        // 等级
5. "experience int,"                   // 经验
6. "questionCount int,"                // 出题数目
7. "passedLevels int,"                 // 已闯关卡数
8. "get\_back\_pwd\_que varchar(200),"    // 找回密码问题
9. "get\_back\_pwd\_ans varchar(200)"     // 找回密码答案

用户类采用用户基类-用户子类继承体系，其主要结构如下：



其中user为用户基类，challenger为用户基类派生出的挑战者类，questioner是用户基类派生出的出题者类。

类的具体定义如下：

1. **class** User
2. {
3. **public**:
4. User(QString username, QString password, **int** level);
6. // 获取用户名
7. QString getUsername() **const**;
8. // 获取用户等级
9. **int** getLevel() **const**;
10. // 获取用户密码
11. QString getPassword() **const**;
13. **private**:
14. QString m\_username; // 用户名
15. QString m\_password; // 密码
16. **int** m\_level;        // 等级
17. };
19. **class** Challenger : **public** User
20. {
21. **public**:
22. Challenger(QString username, QString password, **int** level, **int** experience, **int** passedLevels);
24. **int** getExperience() **const**;
25. **int** getPassedLevels() **const**;
27. **private**:
28. **int** m\_experience;
29. **int** m\_passedLevels;
30. };
32. **class** Questioner : **public** User
33. {
34. **public**:
35. Questioner(QString username, QString password, **int** level, **int** questionCount);
37. **int** getQuestionCount() **const**;
39. **private**:
40. **int** m\_questionCount;
41. };

#### 4.3.2.2 单词管理

请求8-9属于单词管理功能，Word\_Database类中实现。该类中实现的函数有：

* 数据库初始化

1. **static** **bool** init();

* 向数据库中添加单词

1. **void** addWord(**const** QString& word);

* 向数据库中查询单词

1. **bool** findWord(**const** QString& word);

* 输出数据库所有记录

1. **void** printAllRecords() **const**;

* 清除数据库信息

1. **void** clearAllRecords();

* 获取指定长度范围的单词

1. QString getWord(**int** len1, **int** len2);

通过以上函数综合实现单词的添加、查找和获取等功能。同时单词信息保存在数据库中，数据库采用SQLite，利用QSqlDatabase类和QSqlQuery类，实现对数据库表项的增删和修改操作。

数据库word表的格式如下：

1. "word varchar(30) primary key," // 单词
2. "len int" // 单词长度

#### 4.3.2.3 网络通信与报文解析

服务器端的Tcp\_Connection类的功能是通过TCP协议与客户端通信，将收到的报文解析为json格式，并且将返回的json格式报文转化为字节流。网络通信服务器将socket绑定至本地的某个端口，并且监听此端口，当有报文到达时进行相应的解析和处理。

报文解析是将收到的TCP报文从字节流转化成json格式，报文的info\_type字段指明了请求的类型，其余字段指明了请求的内容，服务器端根据请求的类型进行相应的处理。对于每种请求，服务器端都需要从请求的内容中提取出需要的信息，并根据请求的不同类型构造回复报文，构造后的回复报文为json格式，需要转化成字节流，之后通过原本的TCP连接发回给客户端。

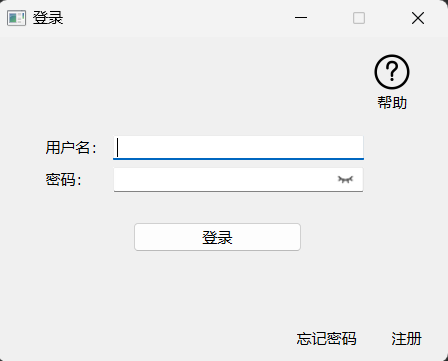
下面附上详细的服务器报文解析过程：

1. // 判断是哪个客户端发出了数据
2. QTcpSocket\* current\_client = qobject\_cast<QTcpSocket\*>(sender());
3. // 获取json文件
4. QByteArray array = current\_client->readAll();
5. QJsonObject receive = QJsonDocument::fromJson(array).object();
6. QString info\_type = receive["info\_type"].toString();
8. // 根据收到的客户端信息类型向客户端回复对应json文件
9. QJsonObject reply;
10. User\_Database user\_db;
11. Word\_Database word\_db;
12. // 检测用户名和密码是否匹配
13. **if**(info\_type == "checkPassword")
14. {
15. qDebug() << "checkPassword";
16. **if**(user\_db.checkPassword(receive["username"].toString(), receive["password"].toString()))
17. {
18. **int** flag = 1;
19. reply["status"] = 1;
20. // 判断是否已经在线
21. **for**(**int** i = 0; i < connections.size(); i++)
22. {
23. **if**(connections[i] != current\_client && **this**->socket\_user[connections[i]] == receive["username"].toString())
24. {
25. reply["status"] = 0;
26. flag = 0;
27. }
28. }
29. // 创建新的连接到用户之间的映射
30. **if**(flag)
31. {
32. socket\_user[current\_client] = receive["username"].toString();
33. }
34. }
35. **else**
36. {
37. reply["status"] = -1;
38. }
39. }
41. // 获取用户信息
42. **else** **if**(info\_type == "getUser")
43. {
44. qDebug() << "getUser";
45. **int** role, level, experience, questionCount, passedLevels;
46. user\_db.get\_User(receive["username"].toString(), role, level, experience, questionCount, passedLevels);
47. reply["role"] = role;
48. reply["level"] = level;
49. reply["experience"] = experience;
50. reply["questionCount"] = questionCount;
51. reply["passedLevels"] = passedLevels;
52. }
54. // 查找用户
55. **else** **if**(info\_type == "findUser")
56. {
57. qDebug() << "findUser";
58. **if**(user\_db.findUser(receive["username"].toString()))
59. {
60. reply["user\_exist"] = **true**;
61. }
62. **else**
63. {
64. reply["user\_exist"] = **false**;
65. }
66. }
68. // 添加用户
69. **else** **if**(info\_type == "addUser")
70. {
71. qDebug() << "addUser";
72. User user(receive["username"].toString(), receive["password"].toString(), receive["level"].toInt());
73. user\_db.addUser(user, receive["role"].toInt(), receive["question"].toString(), receive["answer"].toString());
74. }
76. // 更新用户信息
77. **else** **if**(info\_type == "updateUser")
78. {
79. qDebug() << "updateUser";
80. user\_db.updateUser(receive["username"].toString(), receive["level"].toInt(), receive["experience"].toInt(), receive["questionCount"].toInt(), receive["passedLevels"].toInt());
81. }
83. // 获取所有用户信息
84. **else** **if**(info\_type == "get\_all\_userInfo")
85. {
86. qDebug() << "get\_all\_userInfo";
87. QJsonArray array;
88. **if**(user\_db.writeToJsonFile(array))
89. {
90. qDebug() << "Succeed in geting all users' information.";
91. reply["all\_userInfo"] = array;
92. }
93. **else**
94. {
95. qDebug() << "Failed in geting all users' information.";
96. }
97. }
99. // 添加单词
100. **else** **if**(info\_type == "addWord")
101. {
102. qDebug() << "addWord";
103. // 判断单词是否已经存在
104. **if**(!word\_db.findWord(receive["word"].toString()))
105. {
106. word\_db.addWord(receive["word"].toString());
107. reply["word\_exist"] = **false**;
108. }
109. **else**
110. {
111. reply["word\_exist"] = **true**;
112. }
113. }
115. // 随机获取满足条件的单词
116. **else** **if**(info\_type == "getWord")
117. {
118. qDebug() << "getWord";
119. reply["word"] = word\_db.getWord(receive["min\_len"].toInt(), receive["max\_len"].toInt());
120. }
122. // pvp对战
123. **else** **if**(info\_type == "pvp")
124. {
125. **bool** pvp\_success = **false**;
126. qDebug() << "pvp";
127. QString user = receive["username"].toString();
128. **int** flag = 0;
129. **if**(pvp.find(user) != pvp.end())
130. {
131. // 已经成功匹配
132. qDebug() << "in pvp";
133. pvp\_success = **true**;
134. reply["opponent"] = pvp[user];
135. }
136. **for**(**int** i = 0; i < waitList.size(); i++)
137. {
138. **if**(waitList[i] == user)
139. {
140. flag = 1;
141. **break**;
142. }
143. }
144. **if**(!flag)
145. {
146. waitList.push\_back(user);
147. // 匹配成功
148. **if**(waitList.size() >= 2)
149. {
150. qDebug() << "matching success";
151. pvp\_success = **true**;
152. waitList.removeOne(user);
153. QString opponent = waitList[0];
154. reply["opponent"] = opponent;
155. waitList.pop\_front();
156. pvp[user] = opponent;
157. pvp[opponent] = user;
159. // 获取10个不同的单词
160. **for**(**int** i = 0; i < 5;)
161. {
162. QString word = word\_db.getWord(4, 8);
163. **int** flag = 0; // 判重标志
164. qDebug() << word;
165. **for**(**int** j = 0; j < i; j++)
166. {
167. **if**(word == words[user][j])
168. {
169. flag = 1;
170. **break**;
171. }
172. }
173. **if**(flag == 0)
174. {
175. words[user].push\_back(word);
176. words[opponent].push\_back(word);
177. i++;
178. }
179. }
180. }
181. }
182. **if**(pvp\_success)
183. {
184. qDebug() << "pvp success";
185. reply["pvp"] = **true**;
186. QJsonArray array;
187. QJsonObject object;
188. **for**(**int** i = 0; i < 5; i++)
189. {
190. object["word"] = words[user][i];
191. array.append(object);
192. }
193. reply["words"] = array;
194. }
195. **else**
196. {
197. reply["pvp"] = **false**;
198. }
199. }
201. // 取消匹配
202. **else** **if**(info\_type == "cancel\_pvp")
203. {
204. qDebug() << "cancel\_pvp";
205. QString user = receive["username"].toString();
206. waitList.removeOne(user);
207. }
209. // 对战结束
210. **else** **if**(info\_type == "end\_pvp")
211. {
212. QString user = receive["username"].toString();
213. qDebug() << "end\_pvp";
214. QString opponent = pvp[user];
215. pvp.erase(pvp.find(user));
216. **if**(win.find(opponent) != win.end())
217. {
218. win.erase(win.find(opponent));
219. reply["win"] = **false**;
220. }
221. **else**
222. {
223. win[user] = **true**;
224. reply["win"] = **true**;
225. }
226. }
228. // 中途退出pvp
229. **else** **if**(info\_type == "quit\_midway")
230. {
231. QString user = receive["username"].toString();
232. qDebug() << "quit\_midway";
233. **if**(pvp.find(user) != pvp.end())
234. pvp.erase(pvp.find(user));
235. }
237. // 重置密码
238. **else** **if**(info\_type == "resetPassword")
239. {
240. qDebug() << "resetPassword";
241. user\_db.resetPassword(receive["username"].toString(), receive["password"].toString());
242. }
244. // 将要发送的json数据流转换为byteArray类型
245. QByteArray byteArray = QJsonDocument(reply).toJson();
246. // 发送数据
247. current\_client->write(byteArray);
248. // 手动清空缓冲区
249. current\_client->flush();

### 4.3.3 客户端

#### 4.3.3.1 登录

下面是登录的图形化界面:



登录时用户输入用户名以及密码。当用户信息输入不完全时会发出相应提示：

1. **if**(username == "" && password == "")
2. {
3. QMessageBox::information(**this**, "提示", "请输入用户名和密码!");
4. }
5. **else** **if**(username == "")
6. {
7. QMessageBox::information(**this**, "提示", "请输入用户名!");
9. }
10. **else**
11. {
12. QMessageBox::information(**this**, "提示", "请输入密码!");
13. }

当用户信息输入完全时，客户端会向服务器发出登录请求并判断用户名是否存在，当用户名不存在时，会发出错误信息，当用户名存在时，再去判断密码是否正确。

用户名不存在：

1. //用户不存在
2. **else**
3. {
4. QMessageBox::critical(**this**, "错误", "用户不存在！");
5. }

判断密码是否正确：

1. **int** check = userdb.checkPassword(username, password);

check的返回值代表密码是否正确，check=1时代表密码正确，check=-1时代表密码错误。

若不正确，会发出相应的错误提示，若正确，则切换到用户主界面。

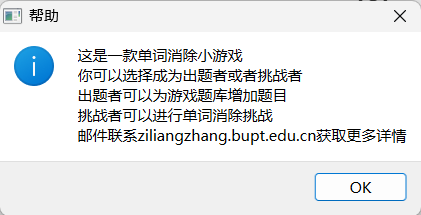
同时考虑到多个客户端同时与服务器进行交互，可能存在已经登录的用户再次登录的情况，因此需要向服务器发出请求判断是否已经登录，通过check的返回值对此判断，check=0时代表用户已经登录。

1. // 用户已登录
2. **else**
3. {
4. QMessageBox::critical(**this**, "错误", "用户已登录！");
5. }

当点击登录界面的注册按钮时，会触发相应信号，切换到注册界面。

1. // 点击注册按钮
2. connect(ui->signup\_btn, &QPushButton::clicked, **this**, [ = ]()
3. {
4. // QMessageBox::information(this, "提示", "注册!");
5. emit change\_to\_sign\_up();
6. });

当点击登录界面的帮助按钮时，会弹出帮助信息消息框。



当点击登录界面的忘记密码按钮时，会切换到重置密码界面。

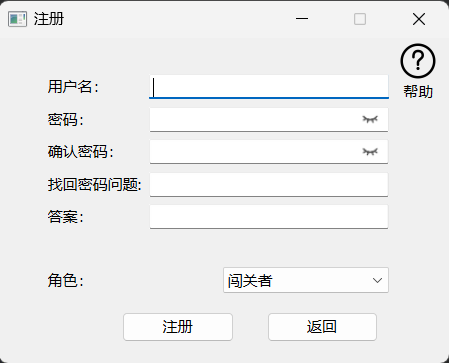
当点击输入密码框右侧的小眼睛时，会切换密码的显示状态：





#### 4.3.3.2 注册

下面是注册的图形化界面：



注册时需要用户输入用户名、密码、确认密码、找回密码问题、答案等五类信息，并需要用户选择角色，默认角色是闯关者。

当输入信息缺省、用户名已经被注册或者两次输入的密码不一致时，会发出相应提示信息。

输入信息缺省：

1. //输入信息不全
2. **else**
3. {
4. QMessageBox::information(**this**, "提示", "请填写所有注册信息!");
5. }

用户名已经被注册：

1. **if**(userdb.findUser(username))
2. {
3. QMessageBox::critical(**this**, "错误", "用户名已存在！");
4. }

两次输入的密码不一致：

1. **else**
2. {
3. QMessageBox::critical(**this**, "错误", "两次输入的密码不一致！");
4. }

找回密码问题和答案是为了重置密码而设定的，会在重置密码模块详细介绍。

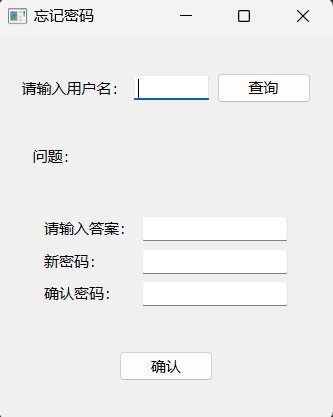
当用户注册信息填写完整无误后，会将新用户的信息发送到服务器进行存储。

点击返回按钮或者关闭注册界面都会返回到登录界面。

点击密码信息栏右侧的小眼睛以及帮助按钮效果和登录界面相同。

#### 4.3.3.3 重置密码

下面是重置密码的图形化界面：



重置密码时，首先需要输入用户名进行查询，若用户名存在则问题栏会显示注册时录入的问题信息。否则会输出用户名不存在的相应提示。



用户对问题输入相应预留的答案，答案正确时，新密码被发送到服务器，将存储的用户密码更新，用户密码被重置为新密码。答案错误时，输出相应错误提示。

#### 4.3.3.3 用户主界面

下面是用户主界面的图形化界面：



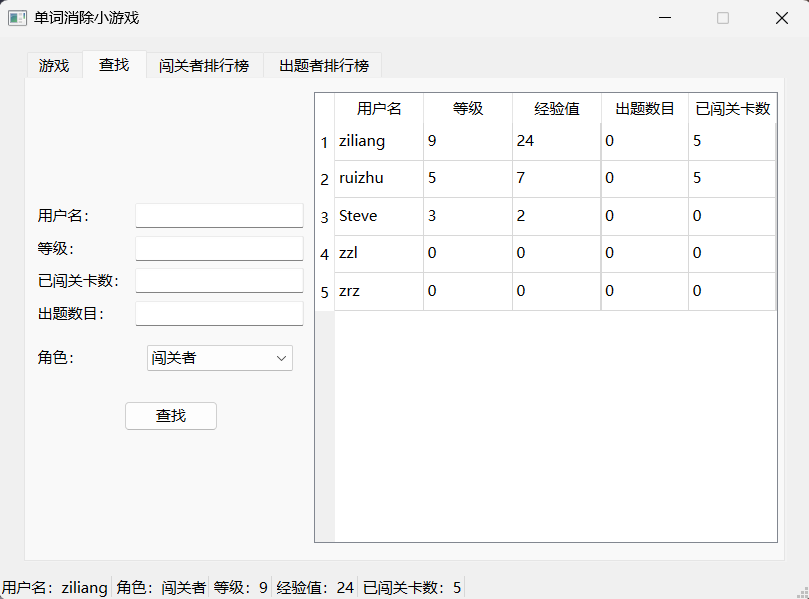
左上角的选项卡可以让用户在游戏、查找、排行榜界面之间进行切换。

底侧的状态栏显示当前登录的用户信息，且会随用户进行游戏或者出题而引起的等级经验值等的变化而实时更新。

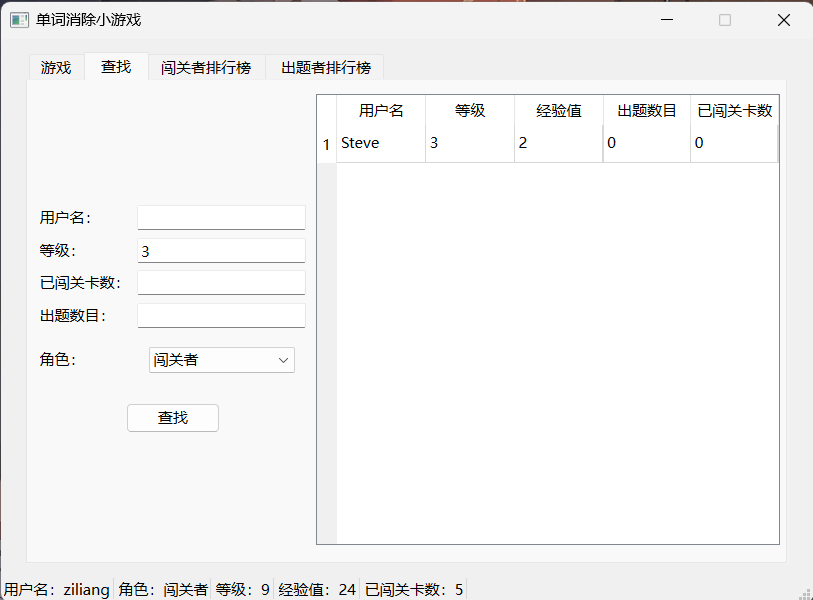
游戏界面允许闯关者进行新游戏、继续游戏和多人对战，允许出题者进行开始出题，同时不允许的功能按钮会被禁用，此处登录用户为ziliang，为闯关者，所以开始出题按钮被禁用。同时继续游戏时允许用户进行选择关卡，选择范围为1—已闯关卡数+1。

查找界面允许用户进行信息缺省查找，未输入内容的栏不作为查找条件。

只以闯关者这一个关键词进行查找：



以闯关者以及等级3这两个关键词进行查找：



排行榜界面允许用户查看所有闯关者和出题者的排名，排名以等级和已闯关卡数进行排序或者以等级和出题数目进行排序。下面附上以已闯关卡数进行排序的闯关者排行榜。

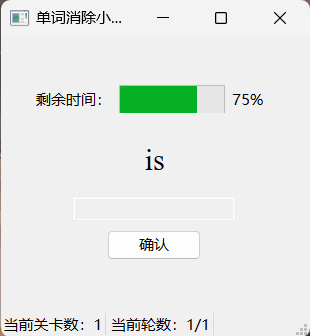


排行榜的实现是依靠QsqlTableModel类。

1. QSqlTableModel\* pModel = **new** QSqlTableModel(**this**);
3. // 设置表格格式
4. pModel->setTable("user");
5. pModel->setEditStrategy(QSqlTableModel::OnManualSubmit);
6. pModel->setFilter(QObject::tr("role = %1").arg(CHALLENGER));
7. pModel->select();
8. pModel->removeColumn(1);
9. pModel->removeColumn(1);
10. pModel->removeColumn(3);
11. pModel->removeColumn(4);
12. pModel->removeColumn(4);
13. pModel->setHeaderData(0, Qt::Horizontal, "用户名");
14. pModel->setHeaderData(1, Qt::Horizontal, "等级");
15. pModel->setHeaderData(2, Qt::Horizontal, "经验值");
16. pModel->setHeaderData(3, Qt::Horizontal, "已闯关卡数");
17. pModel->sort(key, Qt::DescendingOrder);
18. tableView->horizontalHeader()->setSectionsClickable(**false**);
19. tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
20. tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
21. tableView->setModel(pModel);

#### 4.3.3.4 游戏界面

下面是游戏的图形化界面：

****

底侧状态栏显示当前关卡数以及轮数。

剩余时间通过一个电量条来显示，当时间耗尽时单词消失，输入框被启用，答题计时启动，用户可以输入单词开始答题。

1. m\_timer->stop();
2. // 单词消失
3. ui->show\_word\_label->hide();
4. // 允许用户输入
5. ui->input\_word\_lineEdit->setEnabled(**true**);
6. ui->input\_word\_lineEdit->setFocus();
7. cost\_time++;

同时单词输入框利用正则表达式进行限定，

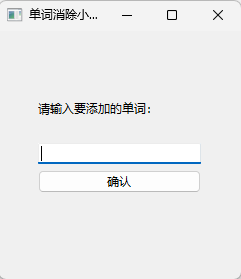
1. // 使用正则表达式限制输入只能为大小写英文字母
2. QRegularExpression regExp("[A-Za-z]+");
3. QValidator\* validator = **new** QRegularExpressionValidator(regExp, **this**);
4. ui->input\_word\_lineEdit->setValidator(validator);

若输入单词正确则会进入下一轮或者下一关，输入单词错误则闯关失败。每成功通过一关，会增加用户的经验值，经验值增加机制与闯过的关卡号以及闯关时间有关。

1. // 增加闯关者经验值
2. **int** add\_exp;
3. **if**(round\_time[**this**->current\_level] \* total\_rounds[**this**->current\_level] - **this**->cost\_time > 0)
4. add\_exp = **this**->current\_level \* 10 + round\_time[**this**->current\_level] \* total\_rounds[**this**->current\_level] - **this**->cost\_time;
5. **else**
6. add\_exp = **this**->current\_level \* 10;

#### 4.3.3.5 出题界面

下面是出题的图形化界面：



当输入为空或者添加的单词已经存在时，会输出相应的提示信息。

输入为空：

1. **else**
2. {
3. QMessageBox::information(**this**, "提示", "请输入单词!");
4. }

添加的单词已经存在：

1. **else**
2. {
3. QMessageBox::critical(**this**, "错误", "单词已存在!");
4. }

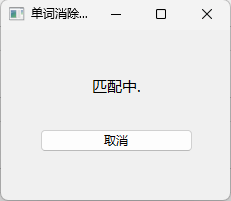
同时输入框采用正则表达式限制只能输入大小写字母。

1. // 使用正则表达式限制输入只能为大小写英文字母
2. QRegularExpression regExp("[A-Za-z]+");
3. QValidator\* validator = **new** QRegularExpressionValidator(regExp, **this**);
4. ui->addword\_lineEdit->setValidator(validator);

当输入不为空时，客户端会将输入内容传送到服务器，当单词不是重复的单词时，服务器会将新单词存储到单词数据库中。

#### 4.3.3.6 多人对战

多人对战前，需要进行匹配，点击多人对战按钮，客户端会向服务器发送相应请求。匹配会在已经登录的用户中随机匹配。



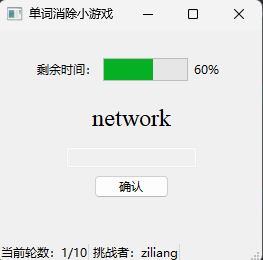
客户端发起匹配请求的代码如下：

1. // 开始匹配
2. connect(**this**, &User\_MainWindow::begin\_pvp, **this**->matching, [ & ]()
3. {
4. **this**->matching->show();
5. **this**->matching\_success = **false**;
6. // 发送匹配请求
7. connect(**this**->pvp\_timer, &QTimer::timeout, **this**, [ & ]()
8. {
9. **if**(matching\_success == **false**)
10. {
11. QJsonObject send\_data;
12. QJsonObject receive\_data;
13. send\_data["info\_type"] = "pvp";
14. qDebug() << **this**->username;
15. send\_data["username"] = username;
16. receive\_data = m\_tcp.sendData(send\_data);
17. **if**(receive\_data["pvp"].toBool())
18. {
19. qDebug() << "matching success";
20. matching\_success = **true**;
21. **this**->matching->hide();
22. **this**->hide();
23. QString words[5];
24. QJsonArray get\_word = receive\_data["words"].toArray();
25. **for**(**int** i = 0; i < 5; i++)
26. {
27. QJsonObject object = get\_word[i].toObject();
28. words[i] = object["word"].toString();
29. }
30. qDebug() << words;
31. qDebug() << username;
32. qDebug() << receive\_data["opponent"].toString();
33. **this**->pvp->init(words, username, receive\_data["opponent"].toString());
34. **this**->pvp->show();
35. }
36. }
37. });

匹配过程中点击取消按钮，客户端会向服务器发送取消匹配的请求。

1. // 取消匹配
2. connect(**this**->matching, &Matching::cancel, **this**, [ & ]()
3. {
4. QJsonObject send\_data;
5. QJsonObject receive\_data;
6. send\_data["info\_type"] = "cancel\_pvp";
7. qDebug() << ui->usr\_lineEdit->text();
8. send\_data["username"] = username;
9. m\_tcp.sendData(send\_data);
10. matching\_success = **true**;
11. **this**->matching->hide();
12. });;

当匹配到对手时，会切换到对战界面。



对战界面和游戏界面类似。

底部显示当前对战轮数，以及对方用户名。此处登录的是ruizhu的账户，匹配到的对手是ziliang。

多人对战根据谁先闯完所有轮数判定胜负，胜负由服务器来判定，会在用户结束对战后，发送给用户。

1. // 对战结束
2. **else** **if**(info\_type == "end\_pvp")
3. {
4. QString user = receive["username"].toString();
5. qDebug() << "end\_pvp";
6. QString opponent = pvp[user];
7. pvp.erase(pvp.find(user));
8. **if**(win.find(opponent) != win.end())
9. {
10. win.erase(win.find(opponent));
11. reply["win"] = **false**;
12. }
13. **else**
14. {
15. win[user] = **true**;
16. reply["win"] = **true**;
17. }
18. }

#### 4.3.3.7 网络通信与报文解析

客户端的Tcp\_Connection类的功能是通过TCP协议与服务器端通信。通过调用不同的客户端函数，向服务器发出不同的请求并接收服务器回传的报文。然后对回传报文进行解析，提取出其中有用的内容。

下面附上详细的客户端发送报文以及解析回传报文的过程：

1. **bool** User\_Database::findUser(**const** QString& username)
2. {
3. QJsonObject send\_data;
4. QJsonObject receive\_data;
5. send\_data["info\_type"] = "findUser";
6. send\_data["username"] = username;
7. receive\_data = m\_tcp.sendData(send\_data);
8. **if**(receive\_data["received"].toBool())
9. **return** receive\_data["user\_exist"].toBool();
10. **else**
11. **return** **false**;
12. }
14. **void** User\_Database::addUser(**const** User& user, **int** role, QString question, QString answer)
15. {
16. QJsonObject send\_data;
17. QJsonObject receive\_data;
18. send\_data["info\_type"] = "addUser";
19. send\_data["username"] = user.getUsername();
20. send\_data["password"] = user.getPassword();
21. send\_data["level"] = user.getLevel();
22. send\_data["role"] = role;
23. send\_data["question"] = question;
24. send\_data["answer"] = answer;
25. receive\_data = m\_tcp.sendData(send\_data);
26. }
28. // 检验用户名和密码是否匹配
29. **int** User\_Database::checkPassword(**const** QString& username, **const** QString& password)
30. {
31. QJsonObject send\_data;
32. QJsonObject receive\_data;
33. send\_data["info\_type"] = "checkPassword";
34. send\_data["username"] = username;
35. send\_data["password"] = password;
36. receive\_data = m\_tcp.sendData(send\_data);
37. **if**(receive\_data["received"].toBool())
38. **return** receive\_data["status"].toInt();
39. qDebug() << "Disconnected to the server.";
40. **return** -1;
41. }
43. // 更新用户信息
44. **void** User\_Database::updateUser(**const** QString username, **const** **int** level, **const** **int** experience, **const** **int** questionCount, **const** **int** passedLevels)**const**
45. {
46. QJsonObject send\_data;
47. QJsonObject receive\_data;
48. send\_data["info\_type"] = "updateUser";
49. send\_data["username"] = username;
50. send\_data["level"] = level;
51. send\_data["experience"] = experience;
52. send\_data["questionCount"] = questionCount;
53. send\_data["passedLevels"] = passedLevels;
54. m\_tcp.sendData(send\_data);
55. }
57. // 获取用户信息
58. **void** User\_Database::getUser(**const** QString& username, **int**& role, **int**& level, **int**& experience, **int**& questionCount, **int**& passedLevels)**const**
59. {
60. QJsonObject send\_data;
61. QJsonObject receive\_data;
62. send\_data["info\_type"] = "getUser";
63. send\_data["username"] = username;
64. receive\_data = m\_tcp.sendData(send\_data);
65. **if**(receive\_data["received"].toBool())
66. {
67. role = receive\_data["role"].toInt();
68. level = receive\_data["level"].toInt();
69. experience = receive\_data["experience"].toInt();
70. questionCount = receive\_data["questionCount"].toInt();
71. passedLevels = receive\_data["passedLevels"].toInt();
72. }
73. **else**
74. qDebug() << "Disconnected to the server.";
75. }
77. **void** User\_Database::get\_all\_userInfo()
78. {
79. QJsonObject send\_data;
80. QJsonObject receive\_data;
81. send\_data["info\_type"] = "get\_all\_userInfo";
82. receive\_data = m\_tcp.sendData(send\_data);
83. **if**(receive\_data["received"].toBool())
84. {
85. QSqlQuery query(m\_db);
86. // 先清空数据库
87. **if** (!query.exec("delete from user"))
88. {
89. qDebug() << "Failed to clear." << query.lastError();
90. }
91. QJsonArray array = receive\_data["all\_userInfo"].toArray();
92. **for**(**int** i = 0; i < array.size(); i++)
93. {
94. QJsonObject object = array[i].toObject();
95. query.prepare("insert into user values(:username, :password, :role, :level, :experience, :questionCount, :passedLevels, :get\_back\_pwd\_que, :get\_back\_pwd\_ans)");
96. query.bindValue(":username", object["username"].toString());
97. query.bindValue(":password", object["password"].toString());
98. query.bindValue(":role", object["role"].toInt());
99. query.bindValue(":level", object["level"].toInt());
100. query.bindValue(":experience", object["experience"].toInt());
101. query.bindValue(":questionCount", object["questionCount"].toInt());
102. query.bindValue(":passedLevels", object["passedLevels"].toInt());
103. query.bindValue(":get\_back\_pwd\_que", object["get\_back\_pwd\_que"].toString());
104. query.bindValue(":get\_back\_pwd\_ans", object["get\_back\_pwd\_ans"].toString());
105. **if** (!query.exec())
106. {
107. qDebug() << "Failed to add user." << query.lastError();
108. }
109. }
110. }
111. **else**
112. qDebug() << "Disconnected to the server.";
113. }
115. **bool** User\_Database::checkAns(QString username, QString ans)
116. {
117. QSqlQuery query(m\_db);
118. query.prepare("select \* from user where username = :username and get\_back\_pwd\_ans = :get\_back\_pwd\_ans");
119. query.bindValue(":username", username);
120. query.bindValue(":get\_back\_pwd\_ans", ans);
121. **if** (!query.exec())
122. {
123. qDebug() << "Failed to check." << query.lastError();
124. }
125. **else** **if** (query.next())
126. {
127. **return** **true**;
128. }
129. **return** **false**;
130. }
132. QString User\_Database::getQue(QString username)
133. {
134. QSqlQuery query(m\_db);
135. query.prepare("select \* from user where username = :username");
136. query.bindValue(":username", username);
137. **if** (!query.exec())
138. {
139. qDebug() << "Failed to find." << query.lastError();
140. }
141. **else** **if** (query.next())
142. {
143. **return** query.value(7).toString();
144. }
145. **return** "";
146. }
148. // 重置密码
149. **void** User\_Database::resetPassword(**const** QString username, **const** QString password)**const**
150. {
151. QJsonObject send\_data;
152. QJsonObject receive\_data;
153. send\_data["info\_type"] = "resetPassword";
154. send\_data["username"] = username;
155. send\_data["password"] = password;
156. m\_tcp.sendData(send\_data);
157. }

# 5.心得体会

通过完成这些实验，我不仅加深了对C++语言和面向对象编程的理解，还提高了我的编程能力和解决问题的能力。我深入研究了每个实验任务的要求，分析了实验结果，并就所学到的知识进行了总结和体会。

我的收获主要有以下几个方面：

* 巩固了课堂上学习到的C++面向对象知识，增强了C++编程能力。
* 学习了Qt框架的相关知识，增加了对其的了解，初步掌握运用Qt开发 GUI的能力。
* 初步掌握数据库的使用方法，熟悉了SQLite的使用。
* 对面向对象编程有了更深的理解，增强了面向对象架构和设计能力。
* 在阅读Qt帮助文档的过程中，增强信息检索能力和英文文献、文档阅读能力。

此次课程设计使我收获良多，积累了很多开发经验，同时也让我发现了自身现阶段的一些能力不足的地方，我将在以后的学习中努力提升不足之处，不断提高自己软件开发的能力和水平。