

北京邮电大学

c++面向对象程序设计实践 期 末 课 程 报 告



姓 名 杨晨
学 院 计算机学院
专 业 计算机科学与技术
班 级 2021211304
学 号 2021212171
任课教师 胡博

2023 年 6 月

目 录

第一章 引言	1
1.1 课程目的和要求	1
1.2 报告内容和框架	1
第二章 基础题目	3
2.1 C++基础知识实验	3
2.1.1 矩阵类的定义和实现	3
2.1.2 矩阵的输入与输出	3
2.1.3 矩阵运算	4
2.1.4 体会感想	5
2.2 类与对象实验	5
2.2.1 Point 类、Circle 类的定义与实现	5
2.2.2 程序 2-1 运行结果分析	5
2.2.3 Matrix 类的定义与实现	6
2.2.4 程序 2-2 运行结果分析	7
2.2.5 体会感想	9
2.3 基类与派生类实验	9
2.3.1 基类和派生类的定义	9
2.3.2 构造和析构函数的调用顺序	10
2.3.3 面积计算	11
2.3.4 心得体会	11
2.4 I/O 流实验	11
2.4.1 商品类和猜价游戏类的定义	11
2.4.2 游戏流程和判断输入的合法性	12
2.4.3 C++输入输出的运用	13
2.4.4 心得体会	13
2.5 运算符重载实验	13
2.5.1 虚函数与抽象类	13
2.5.2 Point 类及运算符重载	13
2.5.3 测试程序	14
第三章 基础题目源代码	15
3.1 C++基础知识实验	15
3.2 类与对象实验	17
3.2.1 “圆形”	17

3.2.2 “矩阵”	20
3.3 继承和派生实验.....	24
3.4 I/O 流实验	27
3.5 类与对象实验.....	29
3.5.1 虚函数	29
3.5.2 对 Point 类重载++和--运算符.....	32
第四章 综合实验	35
4.1 项目设计.....	35
4.1.1 需求分析	35
4.1.2 结构设计	35
4.1.3 功能模块	35
4.1.4 数据库设计	36
4.2 项目实施.....	37
4.2.1 技术选型	37
4.2.2 功能试验	37
4.3 项目测试与部署.....	46
4.3.1 开发环境准备	46
4.3.2 测试用例设计	46
4.3.3 测试执行和结果	47
4.3.4 部署和发布	50
4.4 项目总结.....	51
4.4.1 项目收获	51
4.4.2 项目展望	51
第五章 总结.....	52

第一章 引言

1.1 课程目的和要求

本课程的主要目的是通过一系列实验任务，培养学生扎实的 C++ 语言技能和软件设计与开发能力。

课程要求学生完成以下内容：

C++ 基础知识实验：通过矩阵和点类定义，理解 C++ 基础语法，掌握构造函数、析构函数、运算符重载等概念。

类与对象实验：通过圆形类定义，理解 C++ 类与对象、继承的概念。通过矩阵类定义，练习构造函数、析构函数、拷贝构造函数的调用。

继承与派生实验：通过形状基类及其派生类的定义，理解继承与多态的概念，熟练掌握构造/析构函数的调用顺序。

I/O 流实验：通过猜价格小游戏的实现，理解 C++ 的 I/O 操作与与 C 语言 I/O 的区别。

运算符重载实验：通过形状类和点类运算符重载的实现，理解运算符重载的概念与用法。

综合题目一：设计单词游戏的单机版本，理解软件系统的设计与开发思路。

综合题目二：扩展单机版本为网络多人游戏，使用客户端/服务器模式，理解网络编程的概念和方法。

综合题目三：在题目二的基础上实现支持并发连接的网络平台，加深对 C/S 模式的理解。

扩展功能：可以扩展双人对战模式，在线玩家列表和玩家挑战功能等。

通过上述一系列实验与项目的完成，学生可以全面掌握 C++ 语言和面向对象程序设计概念，理解网络编程模式，具有一定的软件分析与设计能力，为后续课程打下扎实的基础。课程目的就是培养学生的编程思维，不断提高技术与解决问题的能力。

1.2 报告内容和框架

本报告分为 5 个章节：

第一章 引言：介绍本课程学习目的与要求，阐述报告的主要内容与框架。

第二章 基础知识与理论：介绍 C++ 面向对象程序设计的相关理论知识，包括类与对象、继承与派生、运算符重载等。同时分析几个基础实验题目，掌握相关概念与应用。

第三章 项目设计与实现：以 C++ 开发的简单单词消除游戏项目为例，从需求分析、结构设计、功能模块设计与数据库设计等角度，阐述项目的设计思路与实现方法。

第四章 项目测试与部署：对项目进行测试环境准备、测试用例设计与执行，记录并分析测试结果。阐述项目的部署与发布步骤与要求。

第五章 项目总结：总结项目的心得与收获，提出项目的未来改进方向与展望。同时总结报告的内容与心得，完成报告的编写。

本报告按照理论与实践相结合的原则编写。第二章详细介绍 C++ 面向对象程序设计

的相关理论知识，并实例分析几个基础实验题目加深理解。第三章至第五章以项目实践为主，依据第二章理论知识进行项目的设计、实现、测试与总结。通过理论的学习与项目实践相结合，使知识的理解更加全面与扎实，同时提高实际软件开发的能力与水平。

第二章 基础题目

2.1 C++基础知识实验

2.1.1 矩阵类的定义和实现

定义了一个矩阵类 Matrix，包含矩阵的行数 rows 和列数 cols，以及存储矩阵元素的二维数组 data。构造函数中动态申请 data 二维数组的空间，析构函数中释放该空间。包含矩阵输入、输出、相加和相减四个成员函数。

```
// 矩阵类定义
class Matrix
{
public:
    Matrix(int rows, int cols);
    ~Matrix();
    void inputMatrix();
    void printMatrix() const;
    bool addMatrix(const Matrix &m1, const Matrix &m2);
    bool subMatrix(const Matrix &m1, const Matrix &m2);
private:
    int **data;
    int rows, cols;
};

// 构造函数
Matrix::Matrix(int rows, int cols) : rows(rows), cols(cols)
{
    data = new int*[rows]; // 申请 rows 个指针
    for (int i = 0; i < rows; i++)
    {
        data[i] = new int[cols]; // 每个指针指向一个数组
    }
}

// 析构函数
Matrix::~~Matrix()
{
    for (int i = 0; i < rows; i++)
    {
        delete[] data[i]; // 释放每个数组
    }
    delete[] data; // 释放指针数组
}
```

2.1.2 矩阵输入与输出

矩阵输入函数 inputMatrix()通过 cin 读取输入并存储在 data 中。矩阵输出函数 printMatrix()则将 data 中的元素格式化输出到 cout，实现矩阵数据的输入与输出。

```
// 输入矩阵
void Matrix::inputMatrix()
```

```

{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (!(cin >> data[i][j])) { // 输入无效
                cerr << "错误: 输入不是整数" << endl;
                cin.clear(); // 清空输入流
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // 跳过当前行
                return;
            }
        }
    }
}

// 输出矩阵
void Matrix::printMatrix() const
{
    for (int i = 0; i < rows; i++)
    {
        cout << setw(4) << data[i][0];
        for (int j = 1; j < cols; j++)
        {
            cout << " " << setw(4) << data[i][j];
        }
        cout << endl;
    }
}

```

2.1.3 矩阵运算

矩阵相加函数 `addMatrix()` 首先检查两个矩阵是否大小相同,如果相同则对应位置元素相加,结果保存在调用对象中。具体实现是利用双重 `for` 循环遍历两个矩阵,将对应元素相加后存储在调用对象的 `data` 中。矩阵相减函数 `subMatrix()` 同理,只是对应元素相减。这两个函数实现了矩阵的加减运算。

```

bool Matrix::addMatrix(const Matrix &m1, const Matrix &m2)
{
    if (m1.rows != m2.rows || m1.cols != m2.cols)
    {
        cout << "矩阵大小不同,无法相加!" << endl;
        return false;
    }
    for (int i = 0; i < m1.rows; i++)
    {
        for (int j = 0; j < m1.cols; j++)
        {
            data[i][j] = m1.data[i][j] + m2.data[i][j];
        }
    }
}

```

```

    return true;
}

```

2.1.4 体会感想

目前矩阵中只包含 int 类型，可以扩展为包含 double 等其他类型，实现浮点数矩阵的运算。在浮点数运算时需要注意精度问题。

对这个实验程序，我觉得矩阵运算可以进一步扩展，可以增加乘法运算，还可以对特殊矩阵如单位矩阵、对角矩阵等进行优化。数据输入方面可以增加判断输入是否正确。总体来说，这个 C++ 基础实验对掌握 C++ 对象、类的定义与实现，以及运算与算法的设计都很有帮助，是编程能力的很好锻炼。

2.2 类与对象实验

2.2.1 Point 类、Circle 类的定义与实现

定义了一个 Point 类，包含 x 和 y 两个坐标，构造函数可以设置默认原点坐标(0, 0)，类方法 distance() 用于计算两点之间的距离。

```

// Point 类定义
class Point
{
public:
    Point(double x = 0, double y = 0);
    ~Point();
    // 静态成员函数不属于某个对象，因此没有 this 指针
    static double distance(const Point& p1, const Point& p2);
private:
    double x, y;
};

```

定义了一个 Circle 类，包含中心点 center 和半径 radius，构造函数初始化 center 和 radius，方法 intersects() 判断两个圆是否相交。

```

// Circle 类定义
class Circle
{
public:
    Circle(const Point& center, double radius);
    ~Circle();
    bool intersects(const Circle& other) const;
private:
    Point center;
    double radius;
};

```

2.2.2 程序 2-1 运行结果分析

main() 函数中创建两个 Circle 对象，调用 intersects() 方法判断是否相交，并输出结果。同时，在构造和析构函数中添加输出语句，可以清晰观察到 Point 和 Circle 对象的构造

和析构顺序。

```

D:\program\c\c++\problem\basic\2-1.exe
输入第一个圆的圆心坐标和半径:
1 1 1
输入第二个圆的圆心坐标和半径:
2 2 1
Point构造函数被调用
Point构造函数被调用
Circle构造函数被调用
Circle构造函数被调用
两个圆相交
Circle析构函数被调用
Point析构函数被调用
Circle析构函数被调用
Point析构函数被调用
Point析构函数被调用
Point析构函数被调用

```

1. 两次 Point 构造函数被调用:

分别对应于创建 center1 和 center2 两个 Point 对象的构造。
这两个对象表示输入的两个圆的圆心。

2. 两次 Circle 构造函数被调用:

分别对应于创建 circle1 和 circle2 两个 Circle 对象的构造。
这两个对象表示输入的两个圆。

接下来是析构函数的调用。

析构函数的调用顺序与构造函数相反，这是因为对象的生命周期是从创建到销毁，
因此先创建的对象后销毁，后创建的对象先销毁。

1. 第一个 Circle 析构函数被调用: 对应于销毁 circle2 对象。

2. 第一个 Point 析构函数被调用: 对应于销毁 circle2 内部的 center2 对象。

3. 第二个 Circle 析构函数被调用: 对应于销毁 circle1 对象。

4. 第二个 Point 析构函数被调用: 对应于销毁 circle1 内部的 center1 对象。

5. 最后两次 Point 析构函数被调用: 分别对应于销毁 center1 和 center2 两个 Point 对象。

综上所述，对象的创建顺序是: center1、center2、circle1、circle2; 销毁顺序正好相反。

2.2.3 Matrix 类的定义与实现

定义了一个 Matrix 类，包含行数 lines 和列数 rows，以及存储矩阵元素的二维数组 data。构造函数中动态申请 data 空间，析构函数释放空间。包含矩阵输入、输出、相加和相减 4 个方法，以及赋值运算符和加减法运算符的重载。

```
// 矩阵类定义
class Matrix
{
public:
    Matrix(int lines, int rows);
    Matrix(const Matrix& other); // 拷贝构造函数
    ~Matrix();
    Matrix& operator=(const Matrix& other); // 重载赋值运算符
    void inputMatrix();
    void printMatrix() const;
    Matrix operator+(const Matrix& other) const; // 重载加法运算符
    Matrix operator-(const Matrix& other) const; // 重载减法运算符
private:
    int **data;
    int lines, rows;
};

// 构造函数
Matrix::Matrix(int lines, int rows) : lines(lines), rows(rows)
{
    data = new int*[lines]; // 申请 lines 个指针
    for (int i = 0; i < lines; i++)
    {
        data[i] = new int[rows]; // 每个指针指向一个数组
    }
}

// 析构函数
Matrix::~~Matrix()
{
    for (int i = 0; i < lines; i++)
    {
        delete[] data[i]; // 释放每个数组
    }
    delete[] data; // 释放指针数组
    data = NULL; // 防止野指针
}
```

2.2.4 程序 2-2 运行结果与分析

main()函数中定义三个Matrix对象A1,A2和A3。输入A1和A2,然后计算A3=A1+A2和A3=A1-A2,并输出结果。

```

D:\program\c\c++\problem\basic\2-2.exe
请输入第一个矩阵的行数和列数: 3 3
请输入第一个矩阵的元素:
0 0 1
0 1 2
0 2 3
请输入第二个矩阵的行数和列数: 3 3
请输入第二个矩阵的元素:
9 6 6
2 3 3
4 1 8
矩阵相加的结果为:
  9   6   7
  2   4   5
  4   3  11
矩阵相减的结果为:
 -9  -6  -5
 -2  -2  -1
 -4   1  -5

```

同样定义三个 Matrix 指针 pA1, pA2 和 pA3, 输入 pA1 和 pA2, 计算 $pA3=pA1+pA2$ 和 $pA3=pA1-pA2$, 并输出结果。最后释放三个指针。

```

请输入第一个矩阵的元素:
9 6 6
2 3 3
4 1 8
请输入第二个矩阵的元素:
0 0 1
0 1 2
0 2 3
赋值运算符被调用
指针矩阵相加的结果为:
  9   6   7
  2   4   5
  4   3  11
赋值运算符被调用
指针矩阵相减的结果为:
  9   6   5
  2   2   1
  4  -1   5

```

通过这个程序可以观察到 Matrix 对象的构造、拷贝构造、赋值以及析构的调用过程。当进行 $A3=A1+A2$ 时, 会调用 A3 的拷贝构造函数; 当进行 $pA3=pA1+pA2$ 时, 会调用 pA3 的赋值运算符。

然而, 运行结果中没有看到“拷贝构造函数被调用”输出, 可能是因为编译器优化

了这些调用

2.2.5 体会感想

2-1 这个程序运用了对象之间的组合关系，Point 对象作为 Circle 对象的一个组成部分。Circle 对象可以访问并使用 Point 对象，加强了代码的复用性。同时通过构造析构函数的观察，可以加深对 C++对象生命周期的理解。

2-2 这个程序，让我意识到运算符重载可以改变运算符的行为，这会对我的程序逻辑产生重大影响。比如在 Matrix 类中，+运算符的重载使其返回一个引用，这就导致后续的=运算符行为变成了简单的引用指向改变，避免了拷贝构造函数的调用。这提醒我运算符重载的强大效果，但也需要仔细考虑其影响。

对象的生命周期和临时性对拷贝构造函数的调用至关重要。仅当我需要从一个临时对象复制值到另一个对象时，拷贝构造函数才会被调用。如果运算符重载使原本应生成临时对象的表达式生成非临时对象，拷贝构造函数就不会被调用。这让我对临时对象有了更深的认识。

拷贝构造函数的调用与否并不决定表达式的意义上是否进行了“拷贝”。我重载的=运算符本质上也完成了矩阵元素的拷贝，只是使用了引用而不是调用拷贝构造函数。这提醒我不要被表象所迷惑，要深入理解语义。

编译器会在必要时生成默认的运算符行为以保证程序正确性，但这可能导致效率下降。理解编译器的这种默认行为有助于我在需要时选择重载相应的运算符。

总的来说，通过这两个简单的 C++程序，让我对对象的组合与继承、生命周期与临时性、运算符重载与拷贝构造等概念有了更深入的理解。不仅在编程上有很大提高，也在理论知识上有很大的收获。这再一次证明，实践与理论的结合可以产生非常好的学习效果。

2.3 继承与派生实验

2.3.1 基类和派生类的定义

定义了一个基类 Shape，包含一个虚函数 area()用于计算面积。从 Shape 派生出 Rectangle 和 Circle 两个类，Rectangle 再派生出 Square 类。各类都包含构造和析构函数。

```
// 基类 Shape 定义
class Shape
{
public:
    virtual ~Shape() {cout << "Shape 析构函数被调用" << endl;}
    virtual double area() const = 0; // 纯虚函数,计算面积
};

// 矩阵类 Rectangle 定义
class Rectangle : public Shape
{
public:
```

```

    Rectangle(double width, double height);
    ~Rectangle();
    double area() const;
protected:
    double width, height;
};


// 圆类 Circle 定义
class Circle : public Shape
{
public:
    Circle(double radius);
    ~Circle();
    double area() const;
private:
    double radius;
};

//正方形类 Square 定义
class Square : public Rectangle
{
public:
    Square(double width);
    ~Square();
};

```

2.3.2 构造和析构函数的调用顺序

通过在各构造和析构函数中添加输出语句，可以清晰地观察到对象构造和析构的调用顺序。构造函数的调用顺序是从基类到派生类，而析构函数的调用顺序是从派生类到基类。

 D:\program\c\c++\problem\basic\3.exe

```

Rectangle构造函数被调用
Circle构造函数被调用
Rectangle构造函数被调用
Square构造函数被调用
第1个形状的面积是12
Rectangle析构函数被调用
Shape析构函数被调用
第2个形状的面积是78.5398
Circle析构函数被调用
Shape析构函数被调用
第3个形状的面积是36
Square析构函数被调用
Rectangle析构函数被调用
Shape析构函数被调用

```

这是因为在创建一个派生类对象时，需要先构造基类部分，然后才能构造派生类部

分。而在销毁一个派生类对象时，需要先销毁派生类部分，然后再销毁基类部分。

2.3.3 面积计算

主函数中创建三个 Shape 指针，指向 Rectangle, Circle 和 Square 对象。通过调用这三个对象的 area() 方法，计算并输出其面积。由于 area() 在基类中定义为虚函数，最终调用的是三个派生类中重写的 area() 方法。

```
int main()
{
    Shape* shapes[3];
    shapes[0] = new Rectangle(3, 4);
    shapes[1] = new Circle(5);
    shapes[2] = new Square(6);

    for (int i = 0; i < 3; i++) {
        cout << " 第 " << i + 1 << " 个 形 状 的 面 积 是 "
        << shapes[i]->area() << endl;
        delete shapes[i];
    }
}
```

2.3.4 心得体会

这个程序很好地演示了 C++ 继承与派生的概念。通过构造析构函数的调用顺序，我可以真切地理解基类对象在派生类对象中的位置。虚函数的多态调用也使我明白，编译时类型可以与运行时类型不同，程序的行为依赖于运行时类型。

2.4 I/O 流实验

2.4.1 商品类和猜价游戏类的定义

定义了一个商品类 Product，包含一个随机生成的价格。定义了一个猜价游戏类 GuessPriceGame，包含一个 Product 对象和 startGame() 方法用于开始游戏。

```
class Product
{
public:
    Product() { price = rand() % 1000 + 1; } // 1-1000 之间的随机整数
    int getPrice() const { return price; }
private:
    int price;
};

class GuessPriceGame
{
public:
    void startGame();
private:
    Product product;
};
```

2.4.2 游戏流程和判断输入的合法性

startGame()方法首先获取商品的随机价格。然后提示用户输入猜测价格，并对输入进行判断。如果输入不合法(小于 1 或大于 1000)，提示重新输入。如果输入合法，则与商品价格比较，给出相应提示。循环直至猜中为止。

```
// 开始游戏
void GuessPriceGame::startGame()
{
    int userGuess;
    int productPrice = product.getPrice();

    cout << "猜猜商品价格是多少？" << endl;

    do
    {
        cout << "请输入您猜测的商品价格 (1-1000): ";
        cin >> userGuess;
        if (cin.fail()) // 捕获输入流错误
        {
            cin.clear(); // 清除输入流的状态标志
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            // 忽略输入缓冲区中的所有字符
            throw invalid_argument("Error: 输入必须是一个整数!");
        }
        if (!isInputValid(userGuess))
        {
            cout << "输入不合法，请重新输入！" << endl;
            continue;
        }
        else
        {
            if (userGuess > productPrice)
            {
                cout << "您猜的价格太高了！" << endl;
            }
            else if (userGuess < productPrice)
            {
                cout << "您猜的价格太低了！" << endl;
            }
            else
            {
                cout << "恭喜您，猜对了！" << endl;
            }
        }
    } while (userGuess != productPrice);
    cout << "游戏结束！" << endl;
}
```


为确保输入的合法性，定义了 `isValid()` 方法对输入价格进行判断。这体现了输入验证的重要性，可以避免后续逻辑出错。

```
// 判断输入是否合法
bool GuessPriceGame::isValid(int price) const
{
    return price >= 1 && price <= 1000;
}
```

2.4.3 C++输入输出的运用

这个程序很好地运用了 C++ 的输入输出功能 `cin` 和 `cout`。通过这两个对象，程序可以与用户进行交互，实现输入输出的目的。

相比 C 语言的 `scanf` 和 `printf`，C++ 的输入输出方式更加面向对象，也更加安全和易于使用。熟练掌握 `cin` 和 `cout` 对 C++ 学习至关重要。

2.4.4 心得体会

这个小游戏通过随机数和用户交互实现了一个简单的猜价效果。更重要的是，它让我熟悉了 C++ 的输入输出方式，以及如何进行输入验证。输入输出和交互是程序开发中最基础的技能，为我后续设计单词消除游戏，打下了基础

2.5 运算符重载实验

2.5.1 虚函数与抽象类

在 `Shape` 基类中，将 `area()` 方法定义为虚函数，并比较与之前的程序差异。将 `Shape` 基类定义为抽象类，并比较差异。

```
class Shape
{
public:
    virtual ~Shape() { cout << "Shape 析构函数被调用" << endl; }
    virtual double area() const = 0; // 虚函数, 计算面积
};
```

定义为虚函数后，`area()` 的调用会根据运行时类型确定实际执行的方法。这实现了动态绑定，增强了程序的灵活性。

将 `Shape` 定义为抽象类后，`area()` 变为纯虚函数。抽象类不能被实例化，必须由具体的派生类实现纯虚函数。这实现了要求派生类实现某些方法的目的。

2.5.2 Point 类及运算符重载

定义了一个 `Point` 类，表示坐标点。重载了 `++`，`--` 运算符，用于增加或减少 `Point` 的 `x` 和 `y` 坐标。分别重载了前置和后置递增递减运算符。

```
class Point
{
    // ...
public:
```



```
Point &operator++(); // 前置++
Point &operator--(); // 前置--
Point operator++(int); // 后置++
Point operator--(int); // 后置--
};
```

运算符重载是 C++ 的一个强大功能，可以改变标准运算符的行为，实现自定义类型的运算。这大大增强了 C++ 的可扩展性和灵活性。

2.5.3 测试程序

主程序中创建 Point 对象，并调用不同的 ++, -- 运算符，观察其对 Point 坐标的影响。这验证了我们定义的运算符重载行为是正确的

```
int main()
{
    try {
        Point p(1, 1);
        cout << (++p) << endl; // (2, 2)

        cout << (--p) << endl; // (1, 1)

        cout << (p++) << endl; // (1, 1)

        cout << p << endl; // (2, 2)

        cout << (p--) << endl; // (2, 2)

        cout << p << endl; // (1, 1)
    }
}
```

这个实例让我熟悉了虚函数、抽象类和运算符重载的概念与用法。这些是 C++ 泛型编程和扩展性的基石，值得我深入理解和运用。

第三章 基础题目源代码

3.1 C++ 基础知识实验

```
#include <iostream>
#include <iomanip>
#include <limits> // for numeric_limits
using namespace std;

// 矩阵类定义
class Matrix
{
public:
    Matrix(int rows, int cols);
    ~Matrix();
    void inputMatrix();
    void printMatrix() const;
    bool addMatrix(const Matrix &m1, const Matrix &m2);
    bool subMatrix(const Matrix &m1, const Matrix &m2);
private:
    int **data;
    int rows, cols;
};

// 构造函数
Matrix::Matrix(int rows, int cols) : rows(rows), cols(cols)
{
    data = new int*[rows]; // 申请 rows 个指针
    for (int i = 0; i < rows; i++)
    {
        data[i] = new int[cols]; // 每个指针指向一个数组
    }
}

// 析构函数
Matrix::~~Matrix()
{
    for (int i = 0; i < rows; i++)
    {
        delete[] data[i]; // 释放每个数组
    }
    delete[] data; // 释放指针数组
}

// 输入矩阵
void Matrix::inputMatrix()
{
    for (int i = 0; i < rows; i++)
```

```
{
    for (int j = 0; j < cols; j++)
    {
        if (!(cin >> data[i][j])) { // 输入无效
            cerr << "错误: 输入不是整数" << endl;
            cin.clear(); // 清空输入流
            cin.ignore(numeric_limits<streamsize>::max(), '\\
n'); // 跳过当前行
            return;
        }
    }
}

// 输出矩阵
void Matrix::printMatrix() const
{
    for (int i = 0; i < rows; i++)
    {
        cout << setw(4) << data[i][0];
        for (int j = 1; j < cols; j++)
        {
            cout << " " << setw(4) << data[i][j];
        }
        cout << endl;
    }
}

// 矩阵相加
bool Matrix::addMatrix(const Matrix &m1, const Matrix &m2)
{
    if (m1.rows != m2.rows || m1.cols != m2.cols)
    {
        cout << "矩阵大小不同, 无法相加!" << endl;
        return false;
    }
    rows = m1.rows, cols = m1.cols;
    for (int i = 0; i < m1.rows; i++)
    {
        for (int j = 0; j < m1.cols; j++)
        {
            data[i][j] = m1.data[i][j] + m2.data[i][j];
        }
    }
    return true;
}

// 矩阵相减
bool Matrix::subMatrix(const Matrix &m1, const Matrix &m2)
```

```
{
    if (m1.rows != m2.rows || m1.cols != m2.cols)
    {
        cout << "矩阵大小不同, 无法相减!" << endl;
        return false;
    }
    rows = m1.rows, cols = m1.cols;
    for (int i = 0; i < m1.rows; i++)
    {
        for (int j = 0; j < m1.cols; j++)
        {
            data[i][j] = m1.data[i][j] - m2.data[i][j];
        }
    }
    return true;
}

int main()
{
    Matrix A1(4, 5), A2(4, 5), A3(4, 5);
    cout << "输出矩阵 A1:" << endl;
    A1.inputMatrix();
    cout << "输出矩阵 A2:" << endl;
    A2.inputMatrix();

    if (A3.addMatrix(A1, A2))
    {
        cout << "输出矩阵 A3=A1+A2:" << endl;
        A3.printMatrix();
    }

    if (A3.subMatrix(A1, A2))
    {
        cout << "输出矩阵 A3=A1-A2:" << endl;
        A3.printMatrix();
    }

    return 0;
}
```

3.2 类与对象实验

3.2.1 “圆形”

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

```
// Point 类定义
class Point
{
public:
    Point(double x = 0, double y = 0);
    ~Point();
    // 静态成员函数不属于某个对象，因此没有 this 指针
    static double distance(const Point& p1, const Point& p2);
private:
    double x, y;
};

// Point 类实现

// 构造函数
Point::Point(double x, double y) : x(x), y(y)
{
    cout << "Point 构造函数被调用" << endl;
}

// 析构函数
Point::~~Point()
{
    cout << "Point 析构函数被调用" << endl;
}

// 计算两点之间的距离
double Point::distance(const Point& p1, const Point& p2)
{
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return sqrt(dx * dx + dy * dy);
}

// Circle 类定义
class Circle
{
public:
    Circle(const Point& center, double radius);
    ~Circle();
    bool intersects(const Circle& other) const;
private:
    Point center;
    double radius;
};

// Circle 类实现
```

```
// 构造函数
Circle::Circle(const Point& center, double radius) : center(center), radius(radius)
{
    cout << "Circle 构造函数被调用" << endl;
}

// 析构函数
Circle::~~Circle()
{
    cout << "Circle 析构函数被调用" << endl;
}

// 判断两个圆是否相交
bool Circle::intersects(const Circle& other) const
{
    return Point::distance(center, other.center) <= radius + other.radius;
}

int main()
{
    double x1, y1, r1, x2, y2, r2;

    cout << "输入第一个圆的圆心坐标和半径: " << endl;
    if (!(cin >> x1 >> y1 >> r1)) { // 输入无效
        cerr << "错误: 输入不是数字" << endl;
        return 1;
    }

    cout << "输入第二个圆的圆心坐标和半径: " << endl;
    if (!(cin >> x2 >> y2 >> r2)) { // 输入无效
        cerr << "错误: 输入不是数字" << endl;
        return 1;
    }

    // 检查半径是否为非负数
    if (r1 < 0 || r2 < 0) {
        cerr << "错误: 半径不能为负数" << endl;
        return 1;
    }

    Point center1(x1, y1);
    Point center2(x2, y2);
    Circle c1(center1, r1);
    Circle c2(center2, r2);
}
```

```

    if (c1.intersects(c2))
    {
        cout << "两个圆相交" << endl;
    }
    else
    {
        cout << "两个圆不相交" << endl;
    }

    return 0;
}

```

3.2.2 “矩阵”

```

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

// 矩阵类定义
class Matrix
{
public:
    Matrix(int lines, int rows);
    Matrix(const Matrix& other); // 拷贝构造函数
    ~Matrix();
    Matrix& operator=(const Matrix& other); // 重载赋值运算符
    void inputMatrix();
    void printMatrix() const;
    Matrix operator+(const Matrix& other) const; // 重载加法运算符
    Matrix operator-(const Matrix& other) const; // 重载减法运算符
private:
    int **data;
    int lines, rows;
};

// 构造函数
Matrix::Matrix(int lines, int rows) : lines(lines), rows(rows)
{
    data = new int*[lines]; // 申请 lines 个指针
    for (int i = 0; i < lines; i++)
    {
        data[i] = new int[rows]; // 每个指针指向一个数组
    }
}

// 拷贝构造函数
Matrix::Matrix(const Matrix& other) : lines(other.lines), rows(o
her.rows)

```

```
{
    cout << "拷贝构造函数被调用" << endl;
    data = new int*[lines]; // 申请 lines 个指针
    for (int i = 0; i < lines; i++)
    {
        data[i] = new int[rows]; // 每个指针指向一个数组
        for (int j = 0; j < rows; j++)
        {
            data[i][j] = other.data[i][j];
        }
    }
}

// 析构函数
Matrix::~~Matrix()
{
    for (int i = 0; i < lines; i++)
    {
        delete[] data[i]; // 释放每个数组
    }
    delete[] data; // 释放指针数组
    data = NULL; // 防止野指针
}

// 重载赋值运算符
Matrix& Matrix::operator=(const Matrix& other)
{
    cout << "赋值运算符被调用" << endl;
    if (this == &other) // 检查自赋值
    {
        return *this;
    }

    // 释放原有资源
    for (int i = 0; i < lines; i++)
    {
        delete[] data[i];
    }
    delete[] data;

    // 分配新资源并复制
    lines = other.lines;
    rows = other.rows;
    data = new int*[lines];
    for (int i = 0; i < lines; i++)
    {
        data[i] = new int[rows];
        for (int j = 0; j < rows; j++)
        {
```



```
        data[i][j] = other.data[i][j];
    }
}
return *this;
}

// 输入矩阵
void Matrix::inputMatrix()
{
    for (int i = 0; i < lines; i++)
    {
        for (int j = 0; j < rows; j++)
        {
            cin >> data[i][j];
            if (cin.fail()) { // 处理输入格式错误
                cerr << "输入数据格式错误，请重新输入！" << endl;
                cin.clear();
                cin.ignore();
                j--;
            }
        }
    }
}

// 输出矩阵
void Matrix::printMatrix() const
{
    for (int i = 0; i < lines; i++)
    {
        cout << setw(4) << data[i][0];
        for (int j = 1; j < rows; j++)
        {
            cout << " " << setw(4) << data[i][j];
        }
        cout << endl;
    }
}

// 重载加法运算符
Matrix Matrix::operator+(const Matrix& other) const
{
    if (lines != other.lines || rows != other.rows)
    {
        cerr << "矩阵大小不同，无法相加!" << endl;
        throw "Error: 矩阵大小不同，无法相加!";
    }
    Matrix result(lines, rows);
    for (int i = 0; i < lines; i++)
    {
        for (int j = 0; j < rows; j++)
        {
```

```
        result.data[i][j] = data[i][j] + other.data[i][j];
    }
}
return result;
}

// 重载减法运算符
Matrix Matrix::operator-(const Matrix& other) const
{
    if (lines != other.lines || rows != other.rows)
    {
        cerr << "矩阵大小不同，无法相减!" << endl;
        throw "Error: 矩阵大小不同，无法相减!";
    }
    Matrix result(lines, rows);
    for (int i = 0; i < lines; i++)
    {
        for (int j = 0; j < rows; j++)
        {
            result.data[i][j] = data[i][j] - other.data[i][j];
        }
    }
    return result;
}

int main()
{
    int lines, rows;
    cout << "请输入第一个矩阵的行数和列数: ";
    cin >> lines >> rows;
    Matrix A1(lines, rows);
    cout << "请输入第一个矩阵的元素: " << endl;
    A1.inputMatrix();

    cout << "请输入第二个矩阵的行数和列数: ";
    cin >> lines >> rows;
    Matrix A2(lines, rows);
    cout << "请输入第二个矩阵的元素: " << endl;
    A2.inputMatrix();

    try {
        Matrix A3 = A1 + A2;
        cout << "矩阵相加的结果为: " << endl;
        A3.printMatrix();
    } catch (const char* msg) { // 捕获异常
        cerr << "异常信息: " << msg << endl;
    }

    try {
        Matrix A3 = A1 - A2;
```

```
        cout << "矩阵相减的结果为: " << endl;
        A3.printMatrix();
    } catch (const char* msg) { // 捕获异常
        cerr << "异常信息: " << msg << endl;
    }

    Matrix* pA1 = new Matrix(lines, rows);
    cout << "请输入第一个矩阵的元素: " << endl;
    pA1->inputMatrix();

    Matrix* pA2 = new Matrix(lines, rows);
    cout << "请输入第二个矩阵的元素: " << endl;
    pA2->inputMatrix();

    try {
        Matrix* pA3 = new Matrix(lines, rows);
        *pA3 = *pA1 + *pA2;
        cout << "指针矩阵相加的结果为: " << endl;
        pA3->printMatrix();
        delete pA3;
    } catch (const char* msg) { // 捕获异常
        cerr << "异常信息: " << msg << endl;
        delete pA1;
        delete pA2;
        return -1;
    }

    try {
        Matrix* pA3 = new Matrix(lines, rows);
        *pA3 = *pA1 - *pA2;
        cout << "指针矩阵相减的结果为: " << endl;
        pA3->printMatrix();
        delete pA3;
    } catch (const char* msg) { // 捕获异常
        cerr << "异常信息: " << msg << endl;
        delete pA1;
        delete pA2;
        return -1;
    }

    delete pA1;
    delete pA2;

    return 0;
}
```

3.3 继承与派生实验

```
#include <iostream>
#include <cmath>
#include <stdexcept>
#define PI acos(-1.0)
using namespace std;

// 基类 Shape 定义
class Shape
{
public:
    virtual ~Shape() {cout << "Shape 析构函数被调用" << endl;}
    virtual double area() const = 0; // 纯虚函数，计算面积
};

// 矩阵类 Rectangle 定义
class Rectangle : public Shape
{
public:
    Rectangle(double width, double height);
    ~Rectangle();
    double area() const;
protected:
    double width, height;
};

// 矩阵类 Rectangle 实现
Rectangle::Rectangle(double width, double height): width(width)
, height(height)
{
    if (width <= 0 || height <= 0) { // 处理非法参数
        throw invalid_argument("Error: 矩形的长和宽必须为正数!");
    }
    cout << "Rectangle 构造函数被调用" << endl;
}

// 析构函数
Rectangle::~~Rectangle()
{
    cout << "Rectangle 析构函数被调用" << endl;
}

// 计算矩形面积
double Rectangle::area() const
{
    return width * height;
}

// 圆类 Circle 定义
```

```
class circle : public Shape
{
public:
    circle(double radius);
    ~circle();
    double area() const;
private:
    double radius;
};

// 圆类 Circle 实现
circle::circle(double radius) : radius(radius)
{
    if (radius <= 0) { // 处理非法参数
        throw invalid_argument("Error: 圆的半径必须为正数!");
    }
    cout << "circle 构造函数被调用" << endl;
}

// 析构函数
circle::~~circle()
{
    cout << "circle 析构函数被调用" << endl;
}

// 计算圆面积
double circle::area() const
{
    return PI * radius * radius;
}

// 正方形类 Square 定义
class Square : public Rectangle
{
public:
    Square(double width);
    ~Square();
};

// 正方形类 Square 实现
Square::Square(double width) : Rectangle(width, width)
{
    cout << "Square 构造函数被调用" << endl;
}

// 析构函数
Square::~~Square()
{
    cout << "Square 析构函数被调用" << endl;
}
```

```
}

int main()
{
    Shape* shapes[3];
    try {
        shapes[0] = new Rectangle(3, 4);
        shapes[1] = new Circle(5);
        shapes[2] = new Square(6);

        for (int i = 0; i < 3; i++)
        {
            cout << " 第 " << i + 1 << " 个 形 状 的 面 积 是 "
            << shapes[i]->area() << endl;
            delete shapes[i];
        }
    } catch (const invalid_argument& e) { // 捕获异常
        cerr << "异常信息: " << e.what() << endl;
    }

    return 0;
}
```

3.4 I/O 流实验

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <stdexcept>
#include <limits>

using namespace std;

// 商品类
class Product
{
public:
    Product();
    int getPrice() const;
private:
    int price;
};

// 商品类构造函数
Product::Product()
{
    srand(time(NULL)); // 初始化随机数种子
    price = rand() % 1000 + 1; // 随机生成价格
}
```

```
// 获取商品价格
int Product::getPrice() const
{
    return price;
}

// 猜价游戏类
class GuessPriceGame
{
public:
    GuessPriceGame();
    void startGame();
private:
    Product product;
    bool isValidInput(int price) const;
};

// 猜价游戏类构造函数
GuessPriceGame::GuessPriceGame()
{
    product = Product();
}

// 判断输入是否合法
bool GuessPriceGame::isValidInput(int price) const
{
    return price >= 1 && price <= 1000;
}

// 开始游戏
void GuessPriceGame::startGame()
{
    int userGuess;
    int productPrice = product.getPrice();

    cout << "猜猜商品价格是多少？" << endl;

    do
    {
        cout << "请输入您猜测的商品价格 (1-1000): ";
        cin >> userGuess;
        if (cin.fail()) // 捕获输入流错误
        {
            cin.clear(); // 清除输入流的状态标志
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        // 忽略输入缓冲区中的所有字符
        throw invalid_argument("Error: 输入必须是一个整数!");
    }
}
```

```
        if (!isInputValid(userGuess))
        {
            cout << "输入不合法，请重新输入！" << endl;
            continue;
        }
        else
        {
            if (userGuess > productPrice)
            {
                cout << "您猜的价格太高了！" << endl;
            }
            else if (userGuess < productPrice)
            {
                cout << "您猜的价格太低了！" << endl;
            }
            else
            {
                cout << "恭喜您，猜对了！" << endl;
            }
        }
    } while (userGuess != productPrice);
    cout << "游戏结束！" << endl;
}

int main()
{
    GuessPriceGame game = GuessPriceGame();
    try {
        game.startGame();
    } catch (const invalid_argument& e) { // 捕获异常
        cerr << "异常信息：" << e.what() << endl;
    }
    return 0;
}
```

3.5 重载实验

3.5.1 虚函数

```
#include <iostream>
#include <cmath>
#include <stdexcept>
#define PI acos(-1.0)
using namespace std;

// 基类 Shape 定义
class Shape
{
public:
    virtual ~Shape() { cout << "Shape 析构函数被调用" << endl; }
```



```
        virtual double area() const = 0; // 纯虚函数，计算面积
};

// 矩阵类 Rectangle 定义
class Rectangle : public Shape
{
public:
    Rectangle(double width, double height);
    ~Rectangle();
    double area() const;
protected:
    double width, height;
};

// 矩阵类 Rectangle 实现
Rectangle::Rectangle(double width, double height) : width(width)
, height(height)
{
    if (width < 0 || height < 0)
    {
        throw invalid_argument("Error: 矩形宽度或高度不能为负数!");
    }
    cout << "Rectangle 构造函数被调用" << endl;
}

// 析构函数
Rectangle::~~Rectangle()
{
    cout << "Rectangle 析构函数被调用" << endl;
}

// 计算矩形面积
double Rectangle::area() const
{
    return width * height;
}

// 圆类 Circle 定义
class Circle : public Shape
{
public:
    Circle(double radius);
    ~Circle();
    double area() const;
private:
    double radius;
};

// 圆类 Circle 实现
```

```
Circle::Circle(double radius) : radius(radius)
{
    if (radius < 0)
    {
        throw invalid_argument("Error: 圆的半径不能为负数!");
    }
    cout << "Circle 构造函数被调用" << endl;
}

// 析构函数
Circle::~~Circle()
{
    cout << "Circle 析构函数被调用" << endl;
}

// 计算圆面积
double Circle::area() const
{
    return PI * radius * radius;
}

// 正方形类 Square 定义
class Square : public Rectangle
{
public:
    Square(double width);
    ~Square();
};

// 正方形类 Square 实现
Square::Square(double width) : Rectangle(width, width)
{
    cout << "Square 构造函数被调用" << endl;
}

// 析构函数
Square::~~Square()
{
    cout << "Square 析构函数被调用" << endl;
}

int main()
{
    Shape* shapes[3];
    try {
        shapes[0] = new Rectangle(3, 4);
        shapes[1] = new Circle(5);
        shapes[2] = new Square(6);
    }
}
```

```

    catch (const invalid_argument& e) {
        cerr << "异常信息: " << e.what() << endl;
        return 1;
    }

    for (int i = 0; i < 3; i++)
    {
        cout << " 第 " << i + 1 << " 个 形 状 的 面 积 是 "
        << shapes[i]->area() << endl;
        delete shapes[i];
    }
    return 0;
}

```

3.5.2 对 Point 类重载++和--运算符

```

#include <iostream>
#include <cmath>
#include <stdexcept>
using namespace std;

// Point 类定义
class Point
{
    friend ostream &operator<<(ostream &cout, const Point &p);

public:
    Point(double x = 0, double y = 0);
    ~Point();
    Point &operator++(); // 前置++
    Point &operator--(); // 前置--
    Point operator++(int); // 后置++
    Point operator--(int); // 后置--
private:
    double x, y;
};

// Point 类实现

// 构造函数
Point::Point(double x, double y) : x(x), y(y)
{
    if (isnan(x) || isnan(y) || isinf(x) || isinf(y))
    {
        throw domain_error("Error: x 或 y 坐标不是有限数!");
    }
}

```

```
// 析构函数
Point::~~Point()
{
}

// 前置++
Point &Point::operator++()
{
    ++x, ++y;
    return *this;
}

// 前置--
Point &Point::operator--()
{
    --x, --y;
    return *this;
}

// 后置++
Point Point::operator++(int)
{
    Point old = *this;
    ++(*this); // 调用前置++
    return old;
}

// 后置--
Point Point::operator--(int)
{
    Point old = *this;
    --(*this); // 调用前置--
    return old;
}

// 友元函数
ostream &operator<<(ostream &cout, const Point &p)
{
    cout << " p = (" << p.x << ", " << p.y << ")";
    return cout;
}

int main()
{
    try {
        Point p(1, 1);
        cout << (++p) << endl; // (2, 2)
    }
}
```

```
    cout << (--p) << endl; // (1, 1)

    cout << (p++) << endl; // (1, 1)

    cout << p << endl; // (2, 2)

    cout << (p--) << endl; // (2, 2)

    cout << p << endl; // (1, 1)
}
catch (const domain_error& e) {
    cerr << "异常信息: " << e.what() << endl;
    return 1;
}

return 0;
}
```

第四章 综合实验

4.1 项目设计

4.1.1 需求分析

本项目要实现一个简单的单词消除游戏，包括用户系统、游戏系统和排行榜系统。用户系统用于登录、注册和存储用户信息，游戏

系统用于生成关卡、倒计时和判断单词是否正确，排行榜系统用于展示玩家排名和成绩。整个游戏过程包括:用户登录-选择玩家或出题者角色-开始关卡-提交单词-下一关或游戏结束。

4.1.2 结构设计

本项目采用客户端-服务器架构，包含 Server 服务端和 Client 客户端。

Server 端:包含 Game 类、Server 类和 User 类，用于管理在线用户、游戏信息和服务器。

Client 端:包含多个界面类，用于实现用户交互和向 Server 端发送请求。声明结构如下:

Server 端:

- Server.h / Server.cpp:服务器类
- Game.h / Game.cpp:游戏逻辑驱动类
- User.h / User.cpp:用户信息类

Client 端:

- Client.h / Client.cpp:用于连接服务器的客户端类
- LoginWidget.h / LoginWidget.cpp:登录界面类
- MainWidget.h / MainWidget.cpp:主界面类
- RoleSelectionWidget.h / RoleSelectionWidget.cpp:选择角色界面类
- PlayerChoiceWidget.h / PlayerChoiceWidget.cpp: 玩家选择界面类
- QuestionerWidget.h / QuestionerWidget.cpp:出题者界面类
- GameWidget.h / GameWidget.cpp:游戏界面类
- LeaderboardWidget.h / LeaderboardWidget.cpp:排行榜界面类
- MatchmakingWidget.h / MatchmakingWidget.cpp:匹配界面类
- OnlineBattleWidget.h / OnlineBattleWidget.cpp:在线对战界面类

4.1.3 功能模块

本游戏主要包含以下功能模块:

- 用户系统:登录、注册、获取用户信息等。
- 游戏系统:选择角色，获取关卡信息，提交单词和判断正确性，计时器等。
 - 单人游戏:玩家选择闯关者，进入游戏阶段，生成关卡并判断单词是否正确；玩家选择出题者，进入出题界面，向单词库中添加自定义的单词

- 在线游戏:通过匹配系统匹配到其他玩家,双方竞速闯关,并根据用时长短判断是否成功。
- 排行榜系统:获取并展示排行榜信息,包括个人最高闯关记录、最高出题数目和最高等级经验。
- 匹配系统:匹配玩家进行在线对战,成功后进入在线游戏系统。
- 添加单词系统:出题者添加单词,提交至服务器存储到数据库。
- 算法模块:随机选择单词、计算经验值、用户等级和出题难度等。

各功能模块主要由 Server 端的 Game 类实现,Client 端的各界面类通过向 Server 发送请求来使用相应功能。例如:

- 用户登录:Client 的 LoginWidget 向 Server 的 Game 类发送登录请求。
- 单人游戏:Client 的 PlayerChoiceWidget 向 Server 的 Game 类发送进入游戏请求,获取关卡信息并判断单词。
- 在线游戏:Client 的 MatchmakingWidget 向 Server 的 Game 类发送匹配请求,成功后 Client 的 OnlineBattleWidget 与其他玩家进行在线对战。
- 添加单词:Client 的 QuestionerWidget 向 Server 的 Game 类发送添加单词请求。
- 排行榜:Client 的 LeaderboardWidget 向 Server 的 Game 类发送获取排行榜请求。

4.1.4 数据库设计

本项目使用 Microsoft Access 数据库

我选择 Access 数据库,理由如下

轻量级和简单:Access 数据库采用文件型数据库,无需安装复杂的数据库服务器,使用简单,学习成本低,方便快速上手开发。对于一些应用数据量不太大的桌面应用程序来说,Access 数据库是一种简单实用的解决方案。

易于分发和部署:Access 数据库采用的 ACCDB 格式文件,可以直接拷贝发布,无需安装额外软件即可运行,这使其非常适合作为桌面应用程序的嵌入式数据库。对用户来说,使用简单方便。

可视化界面:Access 内置了表、查询、表单、报表等对象的可视化设计界面,可以通过拖放实现简单的数据库开发,适合小型项目快速开发。

本项目,包含两个表:

用户表 Users:

用户 ID	用户名	密码	等级	经验值	当前已 闯关卡 数	最高闯 关卡数	出题数 目
1	Tom	123	27	28	2	8	1
3	Bob	123	87	10	5	7	10
5	Bowing	god	11	0	0	9	0
6	yyy	rk1	91	0	0	10	9

用户表 Users 存储用户信息,包括用户名、密码、等级、经验值和游戏进度等。Access

数据库使用用户 ID 作为主键。

单词表 Words:

单词 ID	单词内容	单词长度	添加者用户名
1	abandon	7	admin
2	banana	6	admin
3	cup	3	admin
4	dog	3	admin
5	elephant	8	admin
6	baby	4	admin
7	hello	5	admin
8	apple	5	admin

单词表 Words 存储所有出题的单词信息，包括内容、长度和添加者。Access 数据库使用单词 ID 作为主键。

我的用户表和单词表，选择这种表格式的主要原因和优点如下

简单和实用:两个表的结构简单清晰，只包含基本必要的字段，易于理解和使用，符合项目的实际需求。对于一个小型单词游戏应用来说，这种简单的表结构已经能满足基本的数据存储和管理要求，没有过于复杂的设计。

便于学习和理解:简单的表结构不会对开发者的数据库技能提出太高要求，比较容易学习掌握和理解，更适用于学习者在项目中实践运用知识。对于一个综合实验项目来说，这也有利于主要关注应用逻辑的开发和学习。

方便数据管理和查询:两个简单的表，通过用户 ID 和单词 ID 建立联系，可以比较容易实现各种数据的添加、删除、修改和检索。基于这两个核心表，一般的查询和管理要求可以得到满足。这也为项目的开发奠定了良好的数据基础。

4.2 项目实施

4.2.1 技术选型

本项目使用 C++语言和 Qt 框架进行开发，IDE 选用 Qt Creator，数据库使用 Access 实现对象持久化存储。

4.2.2 功能实现

实现用户注册与登录功能。系统使用 Access 数据库实现用户信息的持久化存储。支持多用户同时登录和在线。

```
bool Game::connectToDatabase()
{
    if (QSqlDatabase::contains()) {
        // 默认连接已存在，重用它
        m_db = QSqlDatabase::database();
    } else {
        // 创建新的默认连接
```



```

        m_db = QSqlDatabase::addDatabase("QODBC");
    }
    m_db.setDatabaseName(QString("DRIVER={Microsoft Access Driver
(*.mdb, *.accdb)};FIL={MS Access};DBQ=%1").arg("D:\\program\\qt\\
\\complex\\account.accdb"));

    if (!m_db.open())
    {
        qDebug() << "Error: Failed to connect to database.";
        qDebug() << m_db.lastError();
        return false;
    }

    return true;
}

void Game::closeDatabase()
{
    if (m_db.isOpen())
    {
        m_db.close();
    }
}

bool Game::login(const QString& username, const QString& password)
{
    QSqlQuery query(m_db);
    query.prepare("SELECT 用户 ID, 用户名, 密码, 等级, 经验值, 当前已闯
关卡数, 最高闯关数, 出题数目 FROM users WHERE 用户
名 = :username AND 密码 = :password");
    query.bindValue(":username", username);
    query.bindValue(":password", password);

    if (!query.exec())
    {
        qDebug() << "Error: Failed to execute login query.";
        qDebug() << query.lastError();
        return false;
    }

    if (query.next())
    {
        m_currentUser = createUserFromQuery(query); // 使用查询结果
        创建用户对象
        return true; // 登录成功
        emit userLoggedIn();
    }

    return false; // 登录失败, 未找到匹配的用户
}

```

实现游戏功能，包括单人游戏、双人对战等。游戏信息和成绩存储在数据库中。服务器端实现游戏逻辑。

```

#ifndef GAME_H
#define GAME_H

#include <QString>
#include <SqliteDatabase>
#include <vector>
#include <random>
#include <QTimer>
#include <SqlQuery>
#include <SqlError>
#include <SqlRecord>
#include <QDebug>
#include <QTime>
#include <CoreApplication>
#include <algorithm>
#include <random>
#include <chrono>
#include <QVector>
#include <QJsonDocument>
#include <JsonObject>
#include <JsonArray>
#include "User.h"

class Game: public QObject
{
    Q_OBJECT
public:
    Game();
    ~Game();
    bool connectToDatabase(); // 连接到数据库
    void closeDatabase(); // 关闭数据库连接
    User* getCurrentUser() const; // 获取用户信息
    int getCurrentStage() const; // 当前关卡号
    int getCurrentWordCount() const; // 当前关卡有多少个单词
    void moveToNextStage(); // 下一关
    QString getWordAt(int wordIndex) const; // 获取单词
    double getRemainingTime() const; // 获取剩余时间
    bool login(const QString& userId, const QString& password);
    bool registerUser(const QString& username, const QString& password);

    void logout(); // 登出

    void selectRole(bool isPlayer);

    void startStage(int stage); // 闯关者从第 stage 关卡开始
    bool submitword(const QString& word); // 闯关者提交单词

```

```

bool addword(const QString& word); // 出题者添加单词

// 其他你可能需要的功能函数
double getwordDisplayTime() const; // 获取当前关卡单词展示时间
int calculateExpGain() const;
void startCountdown(); // 启动倒计时
void stopCountdown(); // 停止倒计时

bool updateUserInDatabase(); // 更新数据库信息

void fetchLeaderboardData();
void sortLeaderboardData(int sortBy);

struct LeaderboardEntry // 排行榜信息
{
    int id;
    QString username;
    int level;
    int exp;
    int currentLevel;
    int maxLevel;
    int questions;
};
QJsonArray getLeaderboardData() const;

// 从词库中随机选择单词，参数为关卡和单词数目
std::vector<QString> chooseWordsFromPool(int stage, int wordCount) const;

signals:
    void countdownUpdated(double remainingTime); // 倒计时更新信号
    void userLoggedIn();

private:
    User* m_currentUser;
    User* createUserFromQuery(const QSqlQuery& query);
    // 随机数生成器
    std::default_random_engine m_randomGenerator;
    QSqlDatabase m_db;
    bool m_isPlayer;
    int m_currentStage; // 当前关卡
    int m_currentWordIndex;
    int m_wordCount; // 当前关卡单词数
    double m_wordDisplayTime; // 当前关卡时间

    QTimer * m_countdownTimer; // 倒计时计时器

```

```

double m_remainingTime;    // 停止倒计时

std::vector<QString> m_words; // 当前关卡的单词列表

int getWordCountForCurrentStage() const; // 获取当前关卡所需单词
数目

std::pair<int, int> getMinMaxWordLength() const;

QVector<LeaderboardEntry> m_leaderboardData;

};

#endif // GAME_H

```

客户端显示界面并获取用户输入并发送至服务器

```

#ifndef CLIENT_H
#define CLIENT_H

#include <QObject>
#include <QTcpSocket>
#include <QJsonDocument>
#include <QJsonObject>
#include <QDebug>
#include <QJsonArray>
#include <QStringList>

class Client : public QObject
{
    Q_OBJECT

public:
    explicit Client(QObject *parent = nullptr);
    ~Client();

    void connectToServer(const QString &hostname, int port);
    void disconnectFromServer();
    void sendLoginRequest(const QString &username, const QString
&password);
    void sendRegisterRequest(const QString &username, const QStri
ng &password);
    void sendGetUserInfoRequest();
    void sendAddWordRequest(const QString &word);
    void sendCheckWordRequest(const QString &inputword);
    void sendGetNewWordRequest(int wordIndex);
    void sendMoveToNextStageRequest();
    void sendGetRemainingTimeRequest();
    void sendStartGame(int stage);

```

```
void sendStartCountdown();
void sendSortLeaderboardRequest(int sortOrder);
void sendGetOnlinePlayersRequest();

void sendChallengeRequest(const QString &challengedPlayer);
void sendChallengeResponse(const QString &challenger, bool accepted);

void sendBattleStart(const QString &opponent);
void sendGetBattleword();
void sendBattleInput(const QString &word, const QString &opponentName);
void sendBattleEnd(bool won);

signals:
void loginSucceeded();
void loginFailed(const QString& message);
void registerSucceeded();
void registerFailed(const QString& message);
void userInfoReceived(const QJsonObject userInfo);
void wordAddedSuccessfully();
void wordAdditionFailed(const QString& message);
void wordAccept();
void wordWrong(const QString& message);
void getNewWord(const QString& newWord);
void stageChanged();
void remainingTimeChanged(int timeLeft);
void stageInfoUpdated(int currentStage, int currentWordCount);
void startStage();
void startCountdown();
void sortedData(QJsonArray sortedLeaderboardData);
void onlinePlayersReceived(QStringList onlinePlayers);

void challengeReceived(const QString &challenger);
void challengeResponseReceived(const QString &challengedPlayer, bool accepted);

void battleStartReceived(const QJsonObject &battleInfo);
void battlewordReceived(const QString &word);
void battleResultReceived(const QJsonObject &resultInfo);
void battleEndReceived(const QJsonObject &endInfo);

private slots:
void handleReadyRead();
void handleServerResponse(const QJsonObject &response);

private:
    QTcpSocket *m_socket;
};

#endif // CLIENT_H
```

实现排行榜功能。系统可以根据玩家的游戏成绩和经验值计算排名，信息存储在数据库中，所有用户都可以查询整个排行榜。

```
void Game::fetchLeaderboardData()
{
    m_leaderboardData.clear();

    QSqlQuery query("SELECT * FROM users");
    while (query.next())
    {
        QSqlRecord record = query.record();
        LeaderboardEntry entry;
        entry.id = record.value("用户 ID").toInt();
        entry.username = record.value("用户名").toString();
        entry.level = record.value("等级").toInt();
        entry.exp = record.value("经验值").toInt();
        entry.currentLevel = record.value("当前已闯关卡数").toInt();
        entry.maxLevel = record.value("最高闯关数").toInt();
        entry.questions = record.value("出题数目").toInt();

        m_leaderboardData.append(entry);
    }
}

void Game::sortLeaderboardData(int sortBy)
{
    auto compareByLevelExp = [](const LeaderboardEntry &a, const
LeaderboardEntry &b) {
        if (a.level == b.level)
        {
            if (a.exp == b.exp)
            {
                return a.username < b.username;
            }
            return a.exp > b.exp;
        }
        return a.level > b.level;
    };

    switch (sortBy)
    {
        case 0: // 按等级经验排序
            std::sort(m_leaderboardData.begin(), m_leaderboardData.en
d(), compareByLevelExp);
            break;
        case 1: // 按最高闯关数排序
            std::sort(m_leaderboardData.begin(), m_leaderboardData.en
d(), [&](const LeaderboardEntry &a, const LeaderboardEntry &b) {
                if (a.maxLevel == b.maxLevel)
                {

```

```

        return compareByLevelExp(a, b);
    }
    return a.maxLevel > b.maxLevel;
});
break;
case 2: // 按出题数目排序
    std::sort(m_leaderboardData.begin(), m_leaderboardData.end(), [&](const LeaderboardEntry &a, const LeaderboardEntry &b) {
        if (a.questions == b.questions)
        {
            return compareByLevelExp(a, b);
        }
        return a.questions > b.questions;
    });
break;
}
}

QJsonArray Game::getLeaderboardData() const
{
    QJsonArray leaderboardData;

    for (const auto &entry : m_leaderboardData)
    {
        QJsonObject entryObj;
        entryObj["id"] = entry.id;
        entryObj["username"] = entry.username;
        entryObj["level"] = entry.level;
        entryObj["exp"] = entry.exp;
        entryObj["currentLevel"] = entry.currentLevel;
        entryObj["maxLevel"] = entry.maxLevel;
        entryObj["questions"] = entry.questions;

        leaderboardData.append(entryObj);
    }

    return leaderboardData;
}

```

实现 C/S 架构。服务端使用 QTcpServer 实现网络连接监听和管理，并使用 QMap 将每个客户端连接与一个 Game 实例关联，以处理连接对应的游戏逻辑。服务端定义了对战开始、获取单词、输入单词、对战结束等请求处理函数，实现在线对战功能。客户端显示界面并实现用户交互，将输入发送至服务端。

服务端还使用 QThreadPool 实现了自己的线程池，可以发起多个线程同时处理客户端请求，每个 Game 实例运行在线程池的一个线程中，可以并发处理多个客户端的游戏逻辑。所以，服务端具有较强的并发处理能力，可以支持大量客户端同时连接和游戏。

服务端使用 QTcpServer 监听和管理网络连接，使用 QMap 将每个连接与一个 Game 实例关联，该关联是 1:1 的，每个连接有一个独立的 Game 实例进行处理。

Game 实例中封装了游戏逻辑，运行在 QThreadPool 的线程中，多个 Game 实例可以并发运行以处理不同连接的游戏。

```
Server::Server(QObject *parent)
    : QObject(parent), m_server(new QTcpServer(this))
{
    threadPool = new QThreadPool(this);
    threadPool->setMaxThreadCount(10);    // 设置线程池最大线程数为
10
    connect(m_server, &QTcpServer::newConnection, this, &Server::
handleNewConnection);
}
```

服务端定义的请求处理函数，如 handleChallengeRequest、handleBattleStart 等，可以被多个线程并发调用以处理不同客户端的请求。

```
#ifndef SERVER_H
#define SERVER_H

#include <QTcpServer>
#include <QTcpSocket>
#include <QJsonDocument>
#include <QJsonObject>
#include <QThreadPool>
#include "game.h"

class Server : public QObject
{
    Q_OBJECT

public:
    explicit Server(QObject *parent = nullptr);
    ~Server();

    bool start(int port); // 开始监听客户端连接
    void stop(); // 停止监听客户端连接

    QVector<QString> getOnlinePlayers() const; // 获取在线玩家
    void handleClient(QTcpSocket *clientConnection);

private slots:
    void handleIncomingData();
    void handleNewConnection(); // 处理新连接
    void handleClientDisconnected(); // 处理客户端断开连接
    void handleReadyRead(); // 处理客户端发送的数据

private:
    QThreadPool *threadPool;    // 自己的线程池
    int m_maxThreadCount; // 线程池最大线程数
    QHash<QTcpSocket *, QVector<QString>> m_battlewords;
```



```

    QHostAddress m_address;
    quint16 m_port;
    QTcpServer *m_server; // 服务器 socket
    QMap<QTcpSocket *, Game *> m_games; // 每个客户端连接对应一个
Game 实例

    // 解析客户端请求并调用相应的处理函数
    void parseRequest(QTcpSocket *socket, const QByteArray &requestData);

    QTcpSocket* findSocketByUsername(const QString &username);

    void handleChallengeRequest(QTcpSocket *clientSocket, const QString &challengedPlayer);
    void handleChallengeResponse(QTcpSocket *clientSocket, const QString &challengingPlayer, bool accepted);

    void handleBattleStart(QTcpSocket *clientSocket, const QString &opponentPlayer, const QJsonObject &request);
    void handleGetBattleword(QTcpSocket *clientSocket, const QJsonObject &request);
    void handleBattleInput(QTcpSocket *clientSocket, const QJsonObject &request);
    void handleBattleEnd(QTcpSocket *clientSocket, const QJsonObject &request);
    void sendBattleResult(QTcpSocket *clientSocket, bool won);

};

#endif // SERVER_H

```

4.3 项目测试与部署

4.3.1 测试环境准备

在开发完客户端和服务端程序后，需要在测试环境中进行集成测试，确保各个模块之间的调用和交互正常。本项目采用 QT Creator 作为 IDE，QT 作为跨平台 UI 框架，Access 作为数据库。

测试环境需要准备：

Access 数据库，创建用户、游戏和排行榜表，用于持久化存储数据。

安装 QT，用于编译客户端和服务端程序。

安装 QT Creator，用于编写和调试程序。

准备客户端和服务端可执行程序。

4.3.2 测试用例设计

根据项目需求和函数，可以设计以下测试用例：

用户注册登录用例:测试用户能正常注册新账号并登入系统。

单机游戏用例:测试用户可以开始一局单机游戏，答题并取得分数。数据保存到 Access 数据库中。

在线对战用例:两个用户可以发起和参与在线对战，完成一局游戏。游戏数据保存到 Access 数据库中。

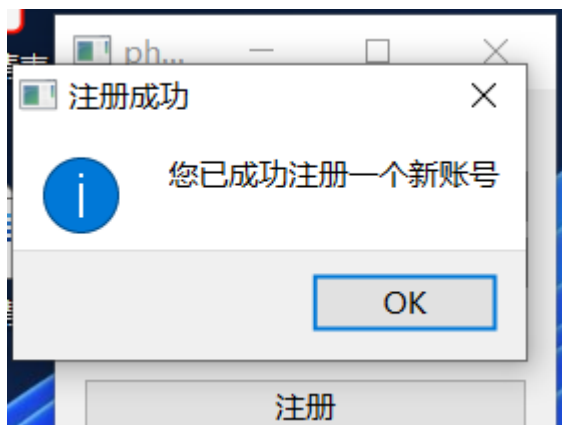
排行榜用例:所有注册用户都可以查询当前的用户排行榜，测试排行榜数据是否正确显示从 Access 数据库中查询的数据。

负载测试用例:多个用户同时访问服务器，测试服务器的负载能力和稳定性。

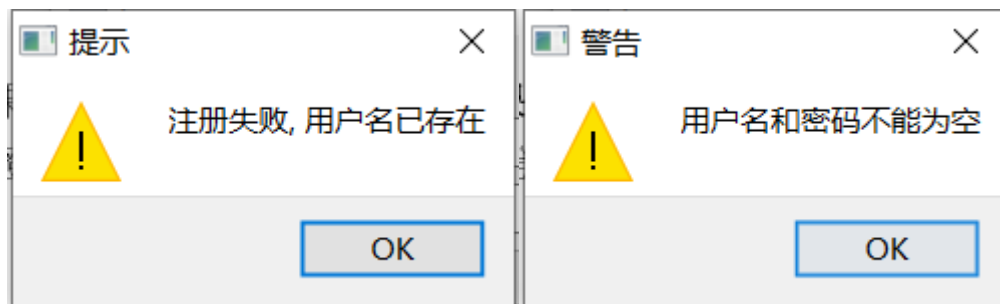
4.3.3 测试执行和结果

按设计的测试用例进行测试，记录测试过程和结果，最后汇总分析如下：

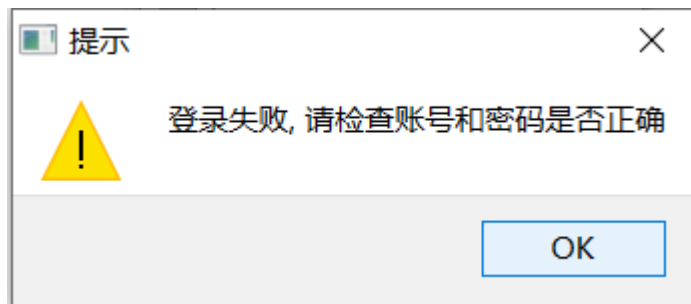
用户注册登录用例:使用多组测试账号进行注册和登录操作，全部成功，未发现问题。



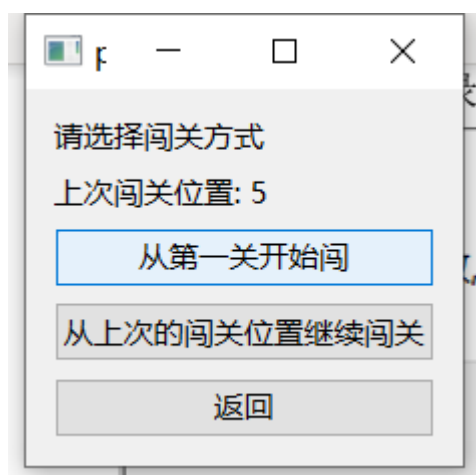
同名，空名字，空密码，无法注册



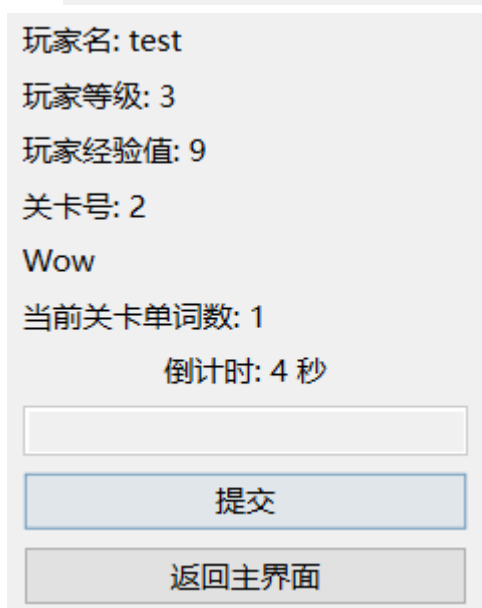
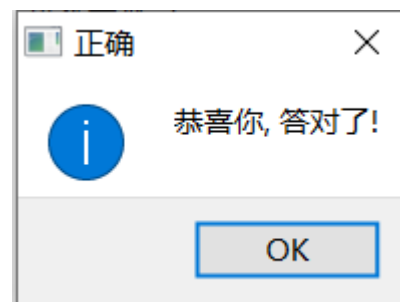
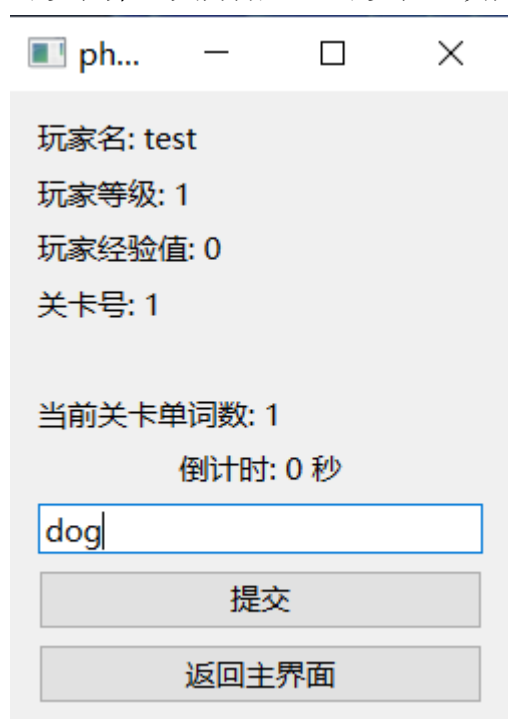
密码错误则无法登录成功



单机游戏用例:使用测试账号进行单机游戏，可以正常开始游戏、答题并取得分数，可以正常出单词，游戏数据保存到数据库，测试通过。



可以从第一关开始，也可以从上次闯关的位置继续



用户ID	用户名	密码	等级	经验值	当前已闯关卡数	最高闯关数	出题数目	单击以
1	Tom	123	27	28	2	8	1	
3	Bob	123	87	10	5	7	10	
5	Bowing	god	13	9	1	9	0	
6	yyy	rk1	91	0	0	10	9	
8	123	123	0	9	1	1	0	
9	test	123	3	9	1	1	0	
(新建)			0	0	0	0	0	

phase4

添加单词

用户名: Bob

等级: 92

经验: 10

当前已出题数量: 10

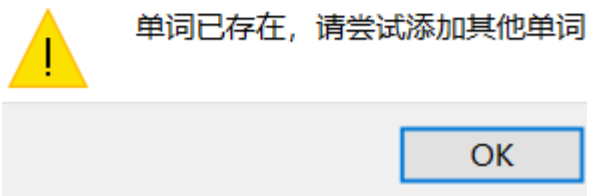
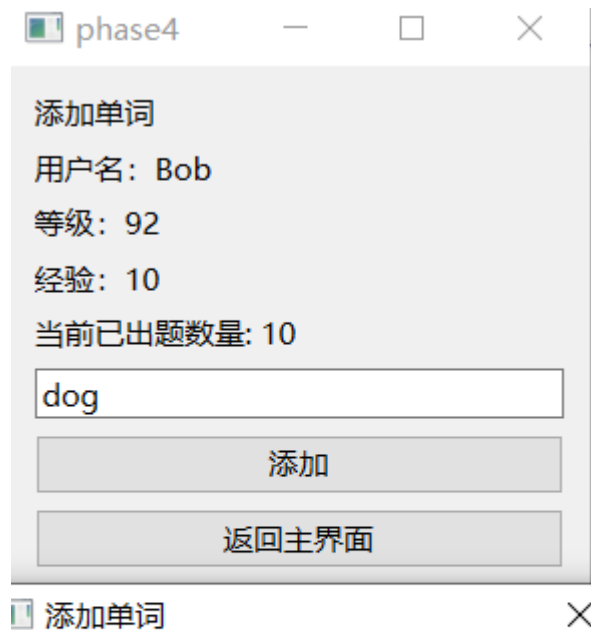
添加

返回主界面

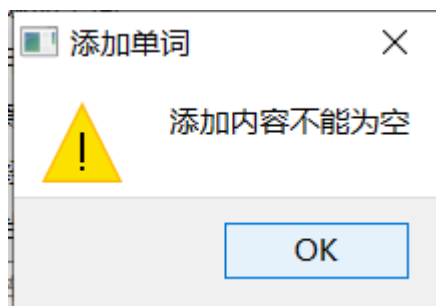
此处用正则表达式，限制输入

只能是 26 个英文字母，可以避免单词库中出现非英文字符

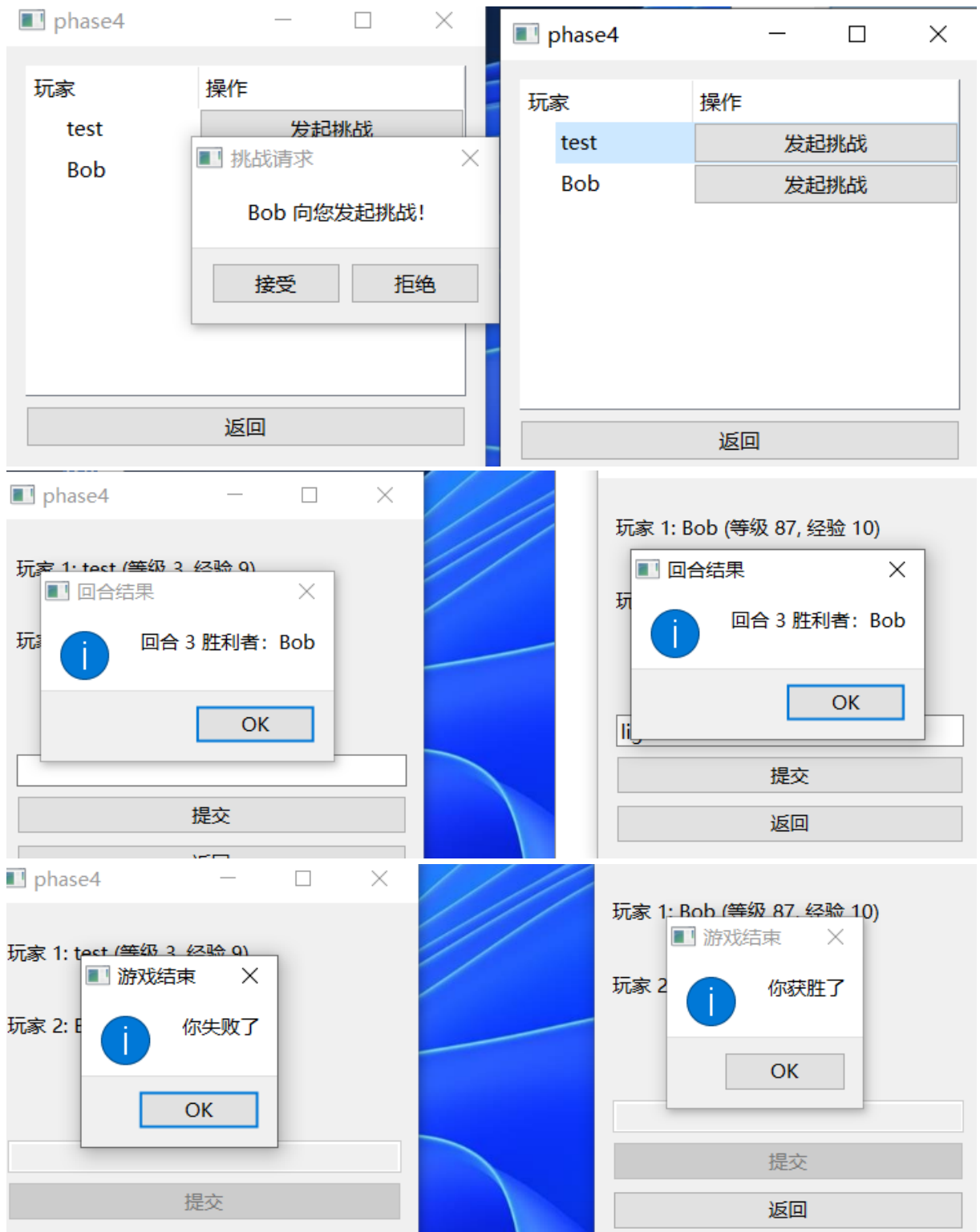
单词库中，若已存在相同的单词，则会提示添加失败



同样, 添加内容也不能为空



在线对战用例: 使用两个测试账号发起和参加在线对战, 可以正常完成一局游戏, 游戏数据保存到数据库, 测试通过。胜利者等级提升 5 级, 失败者等级降低 3 级





排行榜用例:使用测试账号查询排行榜, 排行榜数据正确显示各账号的分数数据, 测试通过。



phase4

排行榜

ID	用户名	等级	经验	当前关卡	最高关卡	出题数
6	yyy	91	0	0	10	9
5	Bowing	13	9	1	9	0
1	Tom	27	28	2	8	1
3	Bob	92	10	5	7	10
8	123	0	9	1	1	0
9	test	0	9	1	1	0

排序方式: 按最高关卡排序

返回



负载测试用例:使用 4 个客户端同时连接服务器进行测试,服务器运行稳定,各客户端可以正常使用功能,测试通过。

4.3.4 部署和发布

客户端和服务端程序已通过测试,打包为 `exe` 可执行文件。但是因为没有使用云服务器,目前只能在本地网络环境下测试运行。

如果部署至公网,需要进行如下工作:

购买云服务器:购买 CPU、内存配置适宜的云服务器作为项目服务器。

环境配置:在云服务器上配置数据库环境、QT 运行环境等,确保客户端和服务端程序可以正常运行。

发布程序:将测试通过的客户端和服务端 `exe` 程序,以及相关文件部署至云服务器。

数据迁移:如在本地使用测试数据,需要将测试数据迁移到云服务器数据库中。

上线测试:部署完成后进行简单的上线测试,确认各功能正常后即可开放使用。

维护与迭代:上线后需监控系统运行情况,修复 bug 或问题,并根据需求推出 system 的新版本。

目前系统只在本地网络环境下进行测试,如果要面向公网提供服务,需要购买云服务器作为托管环境,完成环境配置、程序部署、数据迁移等工作,然后再进行上线测试,

确保系统运行稳定后即可开放使用，并持续进行监控和维护。

4.4 项目总结

4.4.1 项目收获

通过本项目的设计与实现，我对游戏开发、C++语言、数据库和网络编程有了更深入的认识和理解。项目整体采用 C++和 Qt 开发，使用数据库实现数据持久化，服务端实现网络通信和游戏逻辑，客户端实现界面展示与交互，实现一个简单的单词消除游戏软件，包括用户系统、游戏系统和排行榜系统。

在项目开发过程中，我熟练掌握了 Qt 框架，特别是界面设计、事件处理、网络通信等基本技能。同时，运用了 C++面向对象编程思想，设计了较为合理的类的继承和组合关系，理解了软件设计与开发的主要过程。对于数据库的设计与操作也有了较深的认识，能够设计简单的数据库来存储项目数据。

通过本项目，我加深了对软件开发的理解，提高了开发实际软件项目的能力。后续，我将继续学习更多高级技术，如多线程、加密算法等，不断提高自己的编程技能与水平。

4.4.2 项目展望

本项目开发的是一个简单的单词消除小游戏，功能还不算太丰富。随着技能的提高，未来可在该项目中加入更多功能以完善改进该软件，提高用户体验。例如：

添加更多玩法，如时间限制模式，增加游戏难度和挑战性。

扩充更丰富的游戏界面，通过动画、音效等提高游戏的趣味性。

增加 social 功能，玩家可以添加好友，发起多人在线对战，实现排行榜分享等。

结合更高级的技术，如程序优化、加密传输、可扩展性设计等。

支持移动端，开发出手机客户端，玩家可以随时随地开始一局游戏。

后台管理系统，方便管理添加词库、更新资源、维护用户等。

与其他 API 或平台对接，如微信分享接口，提供更丰富的社交功能。

综上，本项目还有较大的改进与发展空间，将身上的技能与理论知识运用到实践项目中，不断学习与总结，提高自己的软件设计与开发能力，开发出更优秀的软件产品。这也是我学习软件工程的动力与目标。

第五章 总结

通过本课程的学习与项目实践，我掌握了 C++面向对象程序设计的基础理论知识和应用技能。第二章从类与对象、继承与派生到运算符重载等多个方面介绍了相关理论，并通过实例分析和代码实现加深理解，奠定了必要的基础。

第三章至第五章以 C++开发的一个简单单词消除游戏项目为例，从需求分析、架构设计到功能实现、测试与部署等全流程展开项目实践。在项目设计时运用了第二章所学知识，设计出基本合理的类与模块结构。项目实现过程中运用 C++语言与 QT 框架完成图形界面展示、游戏逻辑与服务端网络通信等功能开发，并使用 Access 数据库实现数据的持久化存储。项目测试与部署时进行了测试用例设计与执行，找到并修复了程序缺陷，对项目进行了初步部署，理解了将程序部署至服务器的基本要求与流程。

通过理论学习与项目实践，我对 C++语言与面向对象程序设计的认识更加深入，掌握了软件设计与开发的基本方法与技能。但是我的经验还是比较欠缺，在后续的学习与开发中，需要解决的问题还有待提高的方面较多。我将继续深入学习与研究，提高项目开发的能力与水平。

综上，本课程的学习与项目实践通过扎实的理论知识学习与代码实现，提高了我使用 C++进行软件开发的能力。同时也提出了自己不足之处，指明了后续需要继续努力的方向。本课程对我软件工程专业的学习与实践起到了很好的引导作用，收获颇丰。