《面向对象程序设计实践（C++）》综合实验

**单词消除游戏系统设计与开发**

**【程序设计思路】**

（第一版&&第二版）

**数据结构**：

本程序总共设计了四个类：

User 用户类 Player 闯关者类 Qmaker 出题者类 Game 游戏类

其中User是基类，Player和Qmaker继承User类。Game类是整个游戏机制内容

class User // 用户

{

protected:

string Name; //用户名

string PassWord; //密码

int Level; //等级

public:

void SetName(string name);

void SetPassWord(string password);

void SetLevel(int level);

int GetLevel();

string GetName();

string GetPassWord();

bool operator ==(const string& name);

User(int level = 1);

};

class Player : public User //闯关者

{

private:

int Exp; //经验值

int Chapter; //游戏关卡

public:

void SetChapter(int chapter );

int GetChapter();

void SetExp(int exp);

int GetExp();

Player(int exp = 0, int chapter = 1);

};

class Qmaker : public User //出题者

{

private:

int QNumber; //出题数

public:

void SetQNumber(int qnumber);

int GetQNumber();

Qmaker(int qnumber=0);

};

class Game

{

public:

int Register(); //注册

int Login(); //登录

void PlayGame(Player& player); //玩游戏

void MakeQusetion(Qmaker& qmaker); //出题

void Query(); //查询功能

bool GetJudegUser();

private:

bool JudgeUser; //1代表出题者，0代表创关者

int RadomWord(int chapter); //根据关卡随机选择单词

int ExpectWordLen(); //随机产生期望单词长度

void LegalInput(int& ExpectInput, int down, int up);//数字合法化输入

void LegalInput(bool& ExpectInput);

};

除此之外还有三个全局变量：

vector<Word> Vocabulary; //词汇表

vector<Player> PlayerList; //闯关者列表

vector<Qmaker> QmakerList; //出题者列表

void GetVacabulary(); //从本地文件读入词汇表

void UpdateVacabulary(); //更新本地词汇表文件

void GetPlayerList();//从本地文件读入闯关者列表

void UpdatePlayerList(); //更新本地闯关者列表文件

void GetQmakerList();//从本地文件读入出题者列表

void UpdateQmakerList();//更新本地出题者列表文件

【**文件存储形式**】

采用文件存储，将词汇表，闯关者信息，出题信息分别记录在Vocabualry.txt、Player.txt、Qmaker.txt三个文本文件中。每次程序启动将本地文件数据读入程序容器中，程序结束前将程序容器数据写入本地文件。

【**程序结构主要逻辑**】

**Game类**提供程序执行的主要功能。包括注册、登录、闯关者游戏、出题者出题、用户查询功能等。该类中有一个数据成员**JudgeUser**来判断当前用户的属性（闯关者或者出题者）。在main函数中对Game提供的方法构造用户交互界面。由于前两版都是非最终版，因此用户交互界面比较粗糙，只供开发测试人员测试功能使用。

【**游戏核心设计策略**】

1. **登录注册策略**

本游戏在开始用户可以选择注册用户或者直接登录。注册时需要选择注册出题者或者闯关者，输入用户名未被注册过且两次密码输入相同后，注册成功。登录需要选择登录出题者或者闯关者，在输入用户名密码，通过数据库检验后，成功登录。用户只有在登录后，才能进行闯关或者出题。

1. **闯关关卡难度策略**

本游戏闯关共有10个关卡，随着关卡序号的提高，闯关的难度（单词的长度）也不断增加。单词产生策略是：将词汇表按照长度从小到大顺序排列，第一关从词汇表的前10%的单词里随机取出一个，第二关从序号在10%到20%的单词中随机取出一个，以此类推。其中最短单词长度为1，最长单词长度为14。按照这种随机策略，使得同一关卡的单词长度比较灵活(不是固定长度)，但是又在一定的范围内。随着关卡序号的增加，单词难度可以很流畅地递增（而不是某个关卡固定单词长度）。

1. **出题者出题策略**

本游戏为了满足“**出题者添加单词不应影响闯关关卡难度**”而设置了一种特殊的出题策略。首先考虑出题者添加单词较少情况下，由于单词表十分大（3600以上），新添加一两个单词并不会影响闯关者难度。而我根据初始单词库各长度单词比例约束出题者按照一定概率出相应长度的单词。这样在时间足够长，出题者出题数非常大的情况下，根据大数定理，单词表各长度单词所占比例应基本保持不变，这样就保证了各个关卡难度不发生大的变化。

1. **闯关者升级策略**

在闯关者在不同难度的关卡下获得不同的经验者（关卡\*10 第一关10经验 第十关100经验），并根据经验值升级。每一级需要升级的经验值为 **6\*level²+50** ，即每次升级所需经验值与当前等级呈二次关系。这样可以保证随着等级的提升，升级所需要的经验越来越多。其中的比例系数是多次调整出来的，保证前期不会一关升好多级，后期不会好多关也不会升一级。

1. **出题者升级策略**

由于出题者出题难度较大，不像闯关者在高关卡可以获得更多经验。因此我设计出题者是每出5个符合要求的新单词，那么出题者就可以升一级。

1. **查询系统策略**

本系统可以使用4个功能：**查询闯关者、查询出题者、闯关者排行榜、出题者排行榜。**其中前两个是根据用户输入的名字检索相应的闯关者或出题者，若检索到则把其除密码外所有信息打印出。闯关者排行按照经验值从高到低排序，出题者排行榜按照出题数从多到少排序。

**最终服务器客户端版**

**【与前两版的主要区别】**

第三版采用服务器客户端模式，词汇表以及玩家表在服务器存储，用户使用客户端进行游戏。客户端和服务器采用socket通信，来获取服务器端的数据。服务器支持多线程并发，可以同时处理多个客户端的数据请求。除此之外，我还增添了“网上双人匹配对战”功能，用户可以在线匹配其他玩家进行联机对战。

【**程序结构逻辑**】

由于在第三版中，词汇表和用户表客户端不可直接获取，因此程序逻辑与前两版有所不同。对于**客户端**，它保留前两版程序的所有功能，是用户进行游戏的一方。在与服务器建立连接后，若遇到一些需要词汇表和用户表数据的操作时，客户端会向服务器发送特定的**请求**，服务器在接收到客户端的请求后，经过处理后会给予客户端相应的**回应**。（数据**请求失败**标志或者**请求成功**标志+**数据**）。服务器采取多线程，为每一个连接的客户端单独分配一个线程，并发处理客户端的数据请求并回应。

客户端

服务器

客户端

三次握手建立连接

三次握手建立连接

请求

请求

回应

回应

【游戏核心设计策略】（相较于前两版新增）

1. **C/S通信策略**

本游戏为了实现“在线游戏”“网络对战”等模式，将游戏分割为服务器和客户端。用户通过客户端进行注册、登录、游戏、出题、查询等操作。采用的socket通信机制是基于TCP协议的面向连接的通信，确保了实时性和安全性。为了便于在同一台计算机上演示，我采取了双方采用同一本地IP，在相同端口上通信。

1. **消息格式设计策略**

由于Socket网络通信只能传输字节流，因此采取合适发送格式十分重要。我采用消息格式是**标志位**（1个字节）+**数据**（n个字节），**标志位**是一些特定的字符。对于客户端是英文字母（a-n），每个字符代表设定好的固定数据请求。而对于服务器端，标志位只有‘1’和‘0’两种，‘1’代表请求成功，返回所需数据，‘0’代表请求失败。对于**数据**部分，我是将各种类型的数据（姓名 经验 等级等）**统一转换成为字符串型**，使用专门编写的**merge**函数将它们拼接成用#隔开的一条字符串，同样在接收方可以用编写的**split**函数将一条字符串拆分成多条字符串。为了方便实现，我在转化中增添了一级缓冲形式：string容器，将要**发送的数据存入vector<string>**中，统一送往merge函数中合并，并通过send函数发送。

1. **匹配对战模式设计策略**

玩家选择在线匹配对战模式时，系统会将另外一个请求在线游戏的玩家匹配在一起，并同时给他们发送一个相同的单词。如果优先答对的玩家可以获得大量的经验。由于服务器各个线程之间是独立的，要想实现两个客户端（线程）匹配在一起，需要使用**全局变量建立连接**。我使用的3个全局变量：**ClientNumber** 玩家数、**OnlineWord** 在线游戏单词、**Finish** 是否已有玩家完成。ClinetNumber用于控制两个线程同时发送单词、OnlineWord用于两个线程发送相同的单词、Finsh用于判断两个客户端回答的先后。

1. **匹配对战模式奖励策略**

为了鼓励玩家多多使用双人匹配对战模式（便于减少玩家匹配时间），故采取以下奖励策略：为了增加游戏的竞技性，我采取的是抢答模式：玩家抢先作答且答对，将获得120经验（远高于单机作战）；玩家落后对面答对，将获得50经验；答错不得经验。这就意味着，即使对方答错了，但是比你先回答，也算你落后回答。

【**程序存在的缺陷/还需改进**】

1. 在多客户端登录时，同一个账户可以同时被多个客户端登录，甚至可以出现一个同一个账号匹配对战的情况。应该对用户账号加以限制，只允许登录一次。
2. 匹配对战模式，如果只有一对客户端对战，不会出现问题。如果多对客户端进行匹配对战，可能会出现误判的情况（因为只设置一个回答标志位）。把判断回答顺序的先后设在客户端，通过客户端发来的玩家花费时间来判断先后应该可以解决这个问题。
3. 对于共享全局变量没有加上互斥锁，可能会造成多个线程访问同一数据产生冲突的情况。对于多线程加上互斥锁可以解决这个问题。

【**实验收获**】

在这个综合实验中，我学到了很多C++高级的功能。其中最好用的一个就是**容器**，使用容器可以很方便的处理数据，而不用被数据的类型和格式所困惑。其次我学到了软件项目的构建方法，如何设计合适的类。C++的对象的使用让我体会到了**代码复用**的方便之处。然后是标准类库和库函数真的是好用，用别人写好的代码实现自己的功能真的很方便。

除此之外，我还学会了使用VS2019这种大型的IDE。之前一直写些小的源代码，只是用decpp或者vscode就可以满足了。但是到了写稍微大些的项目时，才知道VS2019这种IDE的强大之处，各种方便的功能可以大大提高开发者的效率。