概念题

一、标准输出流(std::cout)和标准错误输出流(std::cerr)两者有什么异同之处？什么时候常用标准输出流？什么时候常用标准错误输出流？

答：

cout 对应于标准输出流，默认情况下是显示器。这是一个被缓冲的输出，可以被重定向，优点是输出文字可以一次输出好多文字，不闪屏。

cerr 对应标准错误流，用于显示错误消息。默认情况下被关联到标准输出流，但它不被缓冲，也就说错误消息可以直接发送到显示器，而无需等到缓冲区或者新的换行符时，才被显示。一般情况下不被重定向，优点是及时输出不被影响。

一般情况下输出信息时用用 cout ，输出错误信息时用 cerr。

二、C 中 scanf 和 printf 有什么不足的地方？C++中 std::cin 和 std::cout 是如何判断输入和输出对象类型的？

答：

scanf 和 printf 缺陷：类型不安全：不是强类型，不利于类型检查，会导致类型相关的运行错误！

std::cin 和 std::cout 的优点：不需要额外指定数据的类型，类型由数据本身决定，避免了类型相关的错误！

C++在进行输入/输出时，首先创建某个 I/O 类的对象，然后，调用该对象类的成员函数进行基于字节流的输入/输出操作。

三、

答：

运行没问题，但是最后文件应该关闭一下

三、什么是程序中的异常？程序中的异常和程序中的错误有什么区别？

程序的错误通常包括：

语法错误：指程序的书写不符合语言的语法规则。这类错误可由编译程序发现。

逻辑错误（或语义错误）：指程序设计不当造成程序没有完成预期的功能。这类错误可通过对程序进行静态分析和动态测试发现。

运行异常：指程序设计对程序运行环境考虑不周而造成的程序运行错误。导致程序运行异常的情况是可以预料的，但它是无法避免的。为了保证程序的鲁棒性（Robustness），必须在程序中对可能的异常进行预见性处理。

四、什么时候需要对异常采用就地处理策略？什么时候需要采用异地处理策略？

（1）如果该该异常影响整个程序，或者是小范围影响，则需要就地处理。

（2）如果该异常影响调用者，则需要进行异地处理。

六、假设有一个从异常基类派生来的异常类层次结构，则应按什么样的顺序放置 catch 块？并说明理由。

答：应按从子孙到祖先的顺序排列 catch 语句块

编程题

一、题目描述

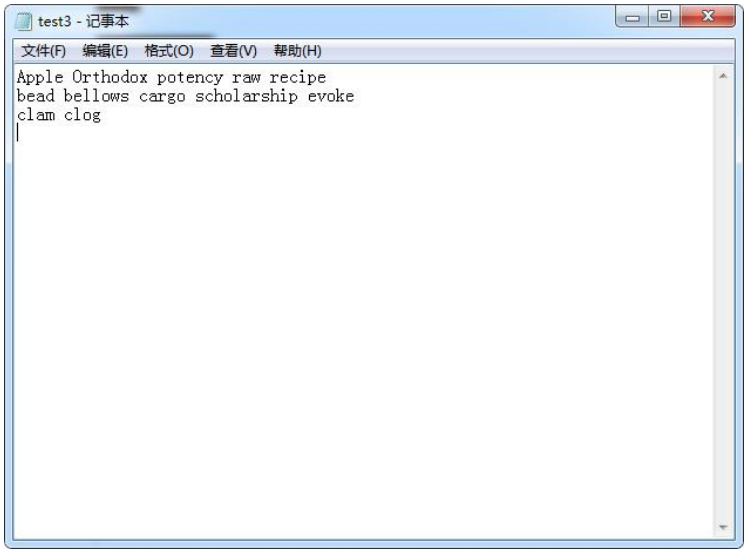
编写一个程序，要求将两个文本文件的内容作为输入，创建一个新文本文件进行输出。该程序将两个输入文件中对应的行拼接起来，并用空格分隔，然后将结果写到输出文件中。如果两个输入文件行数不一致，则将较长文件的余下行直接复制到输出文件中。

```
#include<iostream>
#include <fstream>
#include<string>
#include <vector>
using namespace std;


void read_file(string name,vector<string>&s)
{
    ifstream file;
    file.open(name, ios::in);

    if (!file.is_open())
    {
        throw name;
    }
    else
    {
        while (!file.eof())
        {
            string str;
            getline(file, str);
            s.push_back(str);
        }
        file.close();
    }
}

void out_put(string name,const vector<string>&s)
```

```
{
    ofstream file;
    file.open(name, ios::out);
    if (!file.is_open())
    {
        throw name;
    }
    else
    {
        for (size_t i = 0; i < s.size(); i++)
            file << s[i]<<endl;
    }

}

int main()
{
    vector<string> s1, s2;
    try
    {
        read_file("test1.txt",s1);
    }
    catch (string i)
    {
        cerr << "文件"<<i<<"打开失败" << endl;
    }
    try
    {
        read_file("test2.txt", s2);
    }
    catch (string i)
    {
        cerr << "文件" << i << "打开失败" << endl;
    }

    //开始处理并保存
    if (s1.size()>=s2.size())
    {
        for (size_t k2 = 0; k2 < s2.size(); k2++)
        {
            s1[k2] = s1[k2] + " " + s2[k2];
        }
```

```
        try
        {
            out_put("test3.txt",s1);
        }
        catch (string i)
        {
            cerr << "文件" << i << "打开失败" << endl;
        }
    }
    else
    {
        for (size_t k1 = 0; k1 < s1.size(); k1++)
        {
            s2[k1] = s2[k1] + " " + s1[k1];
        }
        try
        {
            out_put("test3.txt", s2);
        }
        catch (string i)
        {
            cerr << "文件" << i << "打开失败" << endl;
        }
    }

    return 0;
}
```

效果验证：



二、题目描述

二、题目描述

给定下面的User类

```
class User
{
    char uID[11];    // 用户ID
    char uPwd[16];   // 用户密码
  public:
    ...
    void createUsers(const char *, const char *);  //创建用户
    void varifyUsers(); //验证用户
};
```

User类中用户的ID是不超过10位的字符串，并以_或者大写字母开头；用户密码是大于等于6位且小于16位的包含大小字母的字符串。用户的信息以二进制格式存储在本地，文件名为`user.dat`。

现有如下要求：

1. 自定义一个异常类`MyException`，其中有一个私有成员变量`msg`和一个公有函数`what()`用来输出`msg`信息；
2. 创建用户包括三个操作：验证用户ID合法性、验证用户密码合法性、存储用户信息。

   2.1 验证用户ID，如果用户ID不合法，抛出相应的异常；

   2.2 验证用户密码，如果用户密码不合法，抛出相应的异常；

   2.3 存储用户信息，在用户信息合法的情况下，将用户信息存入文件`user.dat`中，并在控制台给出提示信息。
3. 验证用户包括两个操作：查找用户ID、验证用户密码。

   3.1 查找用户ID，即在`user.dat`中能否找到相应的用户ID，如果不能找到，抛出相应的异常；

3. 验证用户包括两个操作：查找用户ID、验证用户密码。

   3.1 查找用户ID，即在`user.dat`中能否找到相应的用户ID，如果不能找到，抛出相应的异常；

   3.2 验证用户密码，如果用户ID对应的密码不正确，抛出相应的异常，如果密码正确，在控制台给出提示信息。

---

4. 创建和验证用户可能涉及文件的打开和关闭操作，如果文件打开时发生错误，抛出相应的异常。

说明：

1. 以上所有抛出的异常都要求能够在程序内处理并在控制台呈现出异常信息；
2. 在实际过程中可能还有其他出现异常的情况，尽可能完善你的程序；

根据上述要求，自行编写代码和测试用例。