Advanced Programming* Homework *IX* LaTeX

* Teacher: Shujian Huang. TA: Yiyu Zhang

1st 张逸凯 171840708

Department of Computer Science and Technology

Nanjing University

zykhelloha@gmail.com

张逸凯 171840708

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？

解: `cout` 经过缓冲后输出，默认情况下是显示器, 是一个被缓冲的输出，是标准输出，并且可以新定向; 而 `cerr` 不经过缓冲而直接输出，一般用于迅速输出出错信息，是标准错误，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.

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？

解: scanf和printf的不足: 传入的类型不安全：不是强类型，不利于类型检查，会导致类型相关的运行错误, 当格式串描述与数据不一致时会导致运行时刻的错误. C++ 中std::cin和std::cout是通过函数重载来判断各种数据类型的, cout的本质是对象, 而操作符实际是cout对象的一个成员函数，即cout实际是cout.operator <<(类型), 利用函数重载cout可实现各种输入数据的自动匹配.

三

```
int main() {
    fstream file;
    file.open("test1.txt", ios::in);
    while(!file.fail()) {
        string str;
        file >> str;
        cout << str << endl;
    }
    file.close();
    file.open("test2.txt", ios::in);
    while(!file.fail()) {
        string str;
        file >> str;
        cout << str << endl;
    }
}
```

阅读上述代码，请思考上述代码是否存在问题，并说明理由。

解:

首先查看C++ reference(见本题末尾)

file.fail()是判断文件是否打开成功用的, 不能作为文件是否读到结尾的判断, 认真查看文档之后, 判断failbit, badbit有没有在在stream中被set. 这意味着大多数情况是流的完整性被破坏时, 所以file.fail()为true是文件已经读空了, 又多运行了一次, 流破坏之后跳出, **导致输出多了一个换行**.

改变file.fail()的顺序即可:

```
int main() {
    fstream file;
    file.open("test1.txt", ios::in);
    string str;
    file >> str;
    while(!file.fail()) {
        cout << str << endl;
        file >> str;
    }
    file.close();
}
```

> http://www.cplusplus.com/reference/ios/ios/fail/
>
> Check whether either failbit or badbit is set
>
> Returns `true` if either (or both) the failbit or the badbit *error state flags* is set for the stream.
>
> At least one of these flags is set when an error occurs during an input operation.
>
> failbit is generally set by an operation when the error is related to the internal logic of the operation itself; further operations on the stream may be possible. While badbit is generally set when the error involves the loss of integrity of the stream, which is likely to persist even if a different operation is attempted on the stream. badbit can be checked independently by calling member function bad:

iostate value (member constants)	indicates	functions to check state flags				
good()	eof()	fail()	bad()	rdstate()		
goodbit	No errors (zero value iostate)	`true`	`false`	`false`	`false`	goodbit
eofbit	End-of-File reached on input operation	`false`	`true`	`false`	`false`	eofbit
failbit	Logical error on i/o operation	`false`	`false`	`true`	`false`	failbit
badbit	Read/writing error on i/o operation	`false`	`false`	`true`	`true`	badbit

> eofbit, failbit and badbit are member constants with implementation-defined values that can be combined (as if with the bitwise OR operator).
>
> goodbit is zero, indicating that none of the other bits is set.
>
> Reaching the *End-of-File* sets the eofbit. But note that operations that reach the *End-of-File* may also set the failbit if this makes them fail (thus setting both eofbit and failbit).

四、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？

解: 异常是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。程序中的错误通常包括语法错误和逻辑错误, 语法错误可由编译程序发现, 逻辑错误在程序运行环境正常的情况下也可能发生. 但是异常往往是程序设计者对程序运行环境的一些特殊情况考虑不足所造成的, 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.

五、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？

解: 就地处理策略: 在发现错误的地方处理异常, 可以使用`abort()`立即终止程序的执行，不作任何的善后处理工作。或者`exit()`, 在终止程序的运行前，会做关闭被程序打开的文件、调用全局对象和存储类的局部对象的析构函数.

异地处理策略: 发现异常时，在发现地（如在被调用的函数中）有时不知道如何处理这个异常，或者不能很好地处理这个异常，要由程序的其它地方（如函数的调用者）来处理.

就地处理策略比较适用于一些不可恢复的异常(除非程序重新设定环境重启), 而异地处理比较适用于可以通过与用户交互恢复的异常.

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。

解:

类内针对类可能发生的错误创建异常类, 来便于能获取到最准确表达异常的异常类, **应按从派生类到基类的顺序排列**catch**语句块**, 这样`throw`的异常可以一层一层被判断是否捕获, 这样可以有分层的结构保证可以确定是什么异常, 精确捕获. C++标准异常处理类就是这样做的.

> 注意在C++里所有的异常类都继承自exception基类，exception类下的logic_error和runtime_error又是两个比较大类，包含有多个自类，它们分表代表逻辑类错误和运行时错误...

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 <bits/stdc++.h>

using namespace std;

vector<string> res;

int main() {
    FILE *fp = fopen("./test1.txt", "r");
    int cnt = 0;
    if (fp == NULL) {
        cerr << "File 1 open failed!\n" << endl;
        exit(0);
    }

    while (!feof(fp)) {
        char buf[1010] = "";
```

```
        fgets(buf, 1000, fp);
        res.push_back(buf);

        if (!res[cnt].empty())
            res[cnt][res[cnt].size() - 1] = ' ';

        ++cnt;
    }

    fclose(fp);

    if ((fp = fopen("./test2.txt", "r")) == NULL) {
        cerr << "File 2 open failed!\n" << endl;
        exit(0);
    }

    cnt = 0;
    int file1Size = res.size();
    while (!feof(fp)) {
        char buf[1010] = "";
        fgets(buf, 1000, fp);
        if (cnt < file1Size) {
            res[cnt] += buf;
        }
        else {
            res.push_back(buf);
        }
        ++cnt;
    }
    res[cnt - 1] += "\n";

    fclose(fp);


    if ((fp = fopen("./result.txt", "w")) == NULL) {
        cerr << "File 3 open failed!\n" << endl;
        exit(0);
    }

    for (int i = 0; i < res.size(); ++i) {
        // cout << res[i];
        fputs(res[i].c_str(), fp);
    }

    fclose(fp);

    return 0;
}
```

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； 3.2 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；

```
#include <bits/stdc++.h>
using namespace std;

class MyException : public exception {
    MyException(string arr) {
        for (int i = 0; i < arr.size(); ++i)
            msg[i] = arr[i];
        msg[arr.size()] = '\0';
    }
    const char* what() const throw () {
        return msg;
    }
private:
    char msg[1010];
};

class User {
    char uID[11]; // 用户ID
    char uPwd[16]; // 用户密码

    void writeToBFile(const string& id, const string& pwd) {
        ofstream outFile("user.dat", ios::out | ios::app | ios::binary);
        string wri = id + "|" + pwd;
        outFile.write(wri.c_str(), sizeof(uID) + sizeof(uPwd) + 1);
    }

```

```
27      bool isInBFile(const string& idS, const string pwdS) {
28          ifstream inFile("user.dat", ios::in | ios::binary);
29
30          // 读取文件失败.
31          if (!inFile) {
32              throw MyException("File user.dat read failed");
33          }
34
35          char des[1010] = "";
36          int readSize = sizeof(uID) + sizeof(uPwd) + 1;
37          while (inFile.read(des, readSize)) { //一直读到文件结束
38              string tmp(des);
39              int gap = tmp.find('|');
40
41              if (idS == tmp.substr(0, gap) && pwdS != tmp.substr(gap + 1,
    tmp.size() - 1 - gap)) {
42                  throw MyException("ERROR: Wrong password!");
43              }
44
45              // 上面 throw 完了还会往下执行吗.
46              else if (idS == tmp.substr(0, gap) && pwdS == tmp.substr(gap +
    1, tmp.size() - 1 - gap)) {
47                  cout << "User authentication successful!" << endl;
48                  return true;
49              }
50
51              /*isFind = true;
52              for (int i = 0; i < gap; ++i) {
53                  if (id[i] != tmp[i]) {
54                      isFind = false;
55                      break;
56                  }
57              }
58              if (isFind == false)
59                  continue;
60              for (int i = gap + 1; i < tmp.size(); ++i) {
61                  if (pwd[i - gap - 1] != tmp[i]) {
62                      isFind = false;
63                      break;
64                  }
65              }
66              if (isFind == false)
67                  continue;
68
69              return true;*/
70          }
71          throw MyException("ERROR: Can't find such USER!");
72
73          return false;
74      }
75
76  public:
77      void createUsers(const char* cid, const char* cpwd) {
78          string id(cid), pwd(cpwd);
79          do {
80              id.clear(), pwd.clear();
81              cout << "(SIGN IN) Please input your ID:";
82              cin >> id;
```

```
            cout << "(SIGN IN) Please input your password:";
            cin >> pwd;
        } while (id.find('|') != string::npos || pwd.find('|') != string::npos);

        writeToBFile(id, pwd);
    }
    void varifyUsers() {
        string id, pwd;
        cout << "(LOG IN) Please input your ID:";
        cin >> id;
        cout << "(LOG IN) Please input your password:";
        cin >> pwd;
        isInBFile(id, pwd);
    }

    void test() {
        writeToBFile("111", "abcd123");
        writeToBFile("121", "abcd124");

        writeToBFile("181", "sk12");
        char t1[110] = "111", t2[110] = "sldfd122";

        cout << isInBFile(t1, t2) << endl;
    }
};


int main() {
    try {
        User u;

        char id[1010], pwd[1010];
        u.createUsers(id, pwd);

        u.varifyUsers();
    }
    catch (MyException& e) {
        std::cout << e.what() << std::endl;
    }

    return 0;
}
```

抛出的异常会在调用者处处理, 上层调用者知道如何处理异常. 用户先注册, 注册完即可登录, 在登录过程中会验证是否密码正确, 是否存在用户.