习题一

有时，出版商发现不被实际单词分心地评价布局和格式设计是很有用的。为此，他们有时用这样的方法来排版示例页面：将所有原始的字母用随机的字母来替换。结果文本有原始的空格和标点结构，但单词采用了这种方法的设计后不再表达任何的含义。这种替换文本的出版项目的方法是greek，大概是在古老的谚语“It 's all Greek to me”后，它才被应用在莎士比亚的《凯撒大帝》中的一行文字中。

编写一个程序，从一个输入文件中读取字符，进行适当的随机替换后，将它们显示在控制台上。你的程序应该将输入中的每个大写字母用随机的大写字母替换，每个小写字母用随机的小写字母替换。非字母表字符应该保持不变。例如，假设输入文件 Troilus.txt包含下面的来自莎士比亚的《特洛伊罗斯克瑞西达》（Tnoilus and Cressida）的文本：

```
Troilus.txt:
Ay, Greek; and that shall be divulged well In characters as red as Mars his heart
Inflamed with Venus:
```

```
// 实现思路
//  r_text += 'A' + offset;
// 亮点:
//     getline(in_file, line); //对ifstream进行读取。
string greek(const string& text) {
    string r_text;
    for (auto ch : text) {
        if (isalpha(ch)) {
            int offset = randomInteger(0, 25);
            assert(offset ≥ 0 && offset < 26);
            if (isupper(ch)) {
                r_text += 'A' + offset;
            } else {
                assert(islower(ch));
                r_text += 'a' + offset;
            }
        } else {
            r_text += ch;
        }
    }
    return r_text;
}
int main() {
    string file_name;
    cout << "Input file: ";
    cin >> file_name;

    string line;
    ifstream in_file(file_name);
    getline(in_file, line); //偷懒了，由于源文件只有一行。

    cout << greek(line);

```

```
    return 0;
}
```

习题二（写不出来）但是要学会分解任务，将问题分解成去除单行注释与去除多行注释。

尽管注释对人类读者来说很重要，但编译器会简单地忽略它们。如果你正在编写一个编译器，因此需要能够识别并消除出现在源文件中的注释。

编写一个函数：

```
void removeComments(istream & is, ostream & os);
```

除了出现在C++注释中的字符，该函数将来自输入流is中的字符复制到输出流os中。你实现的程序应该能识别两种注释公约：

- 任何以 `/*` 开始并以 `*/` 结束的文本，可能其中有很多行。
- 任何以 `//` 开始的文本，扩展到行的结尾。

> 真正的C++编译器需要检测确保这些字符不包含在引用字符串内，但是忽略该细节你应该感到很舒服，问题是它十分狡猾。

```
void copyStream(istream &is, ostream &os) {
    char ch;
    while (is.get(ch)) {
        os.put(ch);
    }
}
```

```
#include <iostream>
#include <fstream>
#include <string>

// 去除单行注释
std::string removeSingleLineComments(const std::string& line) {
    std::string result;
    bool inComment = false;

    for (size_t i = 0; i < line.length(); ++i) {
        if (!inComment && line[i] == '/') {
            if (i + 1 < line.length() && line[i + 1] == '/') {
                break; // 找到单行注释，停止处理当前行
            }
        }

        if (!inComment) {
            result += line[i];
        }

        if (line[i] == '"') {
            inComment = !inComment; // 如果遇到引号，切换注释状态
```

```
        }
    }

    return result;
}

// 去除多行注释
std::string removeMultiLineComments(const std::string& line, bool& inComment) {
    std::string result;
    size_t pos = line.find("/*");

    while (pos != std::string::npos) {
        size_t endPos = line.find("*/", pos + 2);
        if (endPos == std::string::npos) {
            inComment = true;
            result += line.substr(0, pos); // 截取多行注释开始前的部分
            return result;
        } else {
            result += line.substr(0, pos); // 截取多行注释开始前的部分
            line.substr(endPos + 2); // 截取多行注释结束后的部分
            pos = line.find("/*", endPos + 2);
        }
    }

    return line;
}

// 去除注释
void removeComments(std::istream& is, std::ostream& os) {
    std::string line;
    bool inComment = false;

    while (std::getline(is, line)) {
        if (!inComment) {
            line = removeSingleLineComments(line);
            line = removeMultiLineComments(line, inComment);
        } else {
            line = removeMultiLineComments(line, inComment);
        }

        os << line << std::endl;
    }
}

int main() {
    std::ifstream inputFile("input.cpp");
    std::ofstream outputFile("output.cpp");

    if (inputFile.is_open() && outputFile.is_open()) {
        removeComments(inputFile, outputFile);
        inputFile.close();
        outputFile.close();
```

```
    } else {
        std::cerr << "Failed to open files." << std::endl;
    }

    return 0;
}
```

习题三

使用函数stringToInteger和integerToString作为一个模型，编写实现函数stringToReal和realToString 所需的必要代码。

```
int stringToInteger(string str) {
    istringstream stream(str);
    int value;
    stream >> value >> ws;
    if (stream.fail() || !stream.eof()) {
        error("stringToIntegar: Illegal integer format");
    }
    return value;
}

string integerToString(int n) {
    ostringstream stream;
    stream << n;
    return stream.str();
}
```

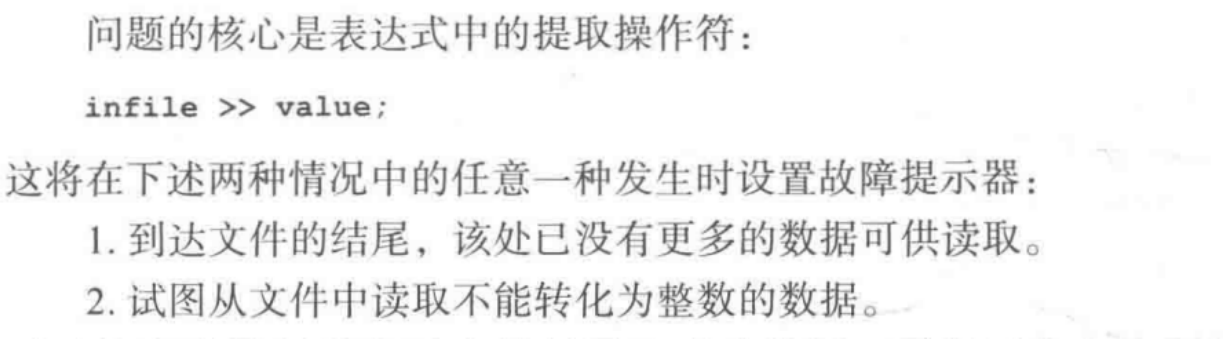
```
double stringToReal(string str) {
    istringstream stream(str);
    double value;
    stream >> value >> ws;
    if (stream.fail() || !stream.eof()) {
        error("stringToIntegar: Illegal Double format");
    }
    return value;
}

string realToString(doulbe n) {
    ostringstream stream;
    stream << n;
    return stream.str();
}
```

习题四

尽管提取操作符使得编写一个简单地从控制台读取输入数据的测试程序变得简单，**但在实际中它并没有广泛采用。**

`>>` **操作符的主要问题是它几乎不提供任何支持检测用户输入是否有效的功能。**众所周知，用户在向计算机中输入数据时是很草率的。他们会造成一些“笔误”，或者更糟糕的是，他们根本没有理解程序真正想要什么输入。设计良好的程序会检测用户的输入以确保它形式正确，并且在程序中是有意义的。



在我们的实现中，`stream >> value >> ws;` 被用于如下的情景，prompt...: my_input [换行符]

`getline()` 会将输入确认为 `[1.可能存在的空格] my_real_input [2.可能存在的空格\换行符]`

在其中，第一个 `>>` 提取运算符会忽略 `[1]` ,而ws将会忽略 `[2]` 。

在此之外的场景，如输入 `my_real_input1 my_real_input2` 或者"1t" ，会因为无法触发 `EOF` 而导致循环无法正确结束。进而，我们提供给用户一个重新输入的机会。也就是说，getInteger总会读取到一个整数，并且总是一个整数(而不是多个)。

> "1t"在提取运算符会被宽容处理，指读取1作为结果。

通过实现函数getReal和getLine未完成simpio.h接口的实现。

> 有个bug，如果输入的后面紧接一个[换行符]，读取会失败?

```
/*
 * Function: getInteger
 * Usage: int n = getInteger(prompt);
 * ----------------------------------
 * Reads a complete line from <code>cin</code> and scans it as an
 * integer. If the scan succeeds, the integer value is returned. If
 * the argument is not a legal integer or if extraneous characters
 * (other than whitespace) appear in the string, the user is given
 * a chance to reenter the value. If supplied, the optional
 * <code>prompt</code> string is printed before reading the value.
 */


int getInteger(std::string prompt = "") {
```

```
    int value;
    string line;
    while (true) {
        cout << prompt;
        getline(cin, line);
        istringstream stream(line);

        if (stream >> value) {
            if (stream.eof()) break;        // e.g., 123
            else if (stream >> ws) break;   // e.g., 123 followed by space(s)...
            else                            // e.g., 123 followed by non-space character(s)
                println("getIntegar: a single valid integer");
        }
        else {
            println("getIntegar: Illegal integer format. Try again.");
        }
    }
    return value;
}

double getReal(std::string prompt = "") {
    double value;
    string line;
    while (true) {
        cout << prompt;
        getline(cin, line);
        istringstream stream(line);

        if (stream >> value) {
            if (stream.eof()) break;        // e.g., 123.
            else if (stream >> ws) break;   // e.g., 123. followed by space(s)...
            else                            // e.g., 123. followed by non-space character(s)
                println("getIntegar: a single valid Real");
        }
        else {
            println("getIntegar: Illegal Real format. Try again.");
        }

    }
    return value;
}

string getLine(std::string prompt = "") {
    cout << prompt;
    string line;
    getline(cin, line);
    return line;
}
```

```cpp
void showStateBit(std::istream& is) {
    cout << (is.good() ? "GOOD " : "____ ");
    cout << (is.fail() ? "FAIL " : "____ ");
    cout << (is.eof() ?  "END "  : "____ ");
    cout << (is.bad() ?  "BAD"   : "____");
    cout << endl;
}
```