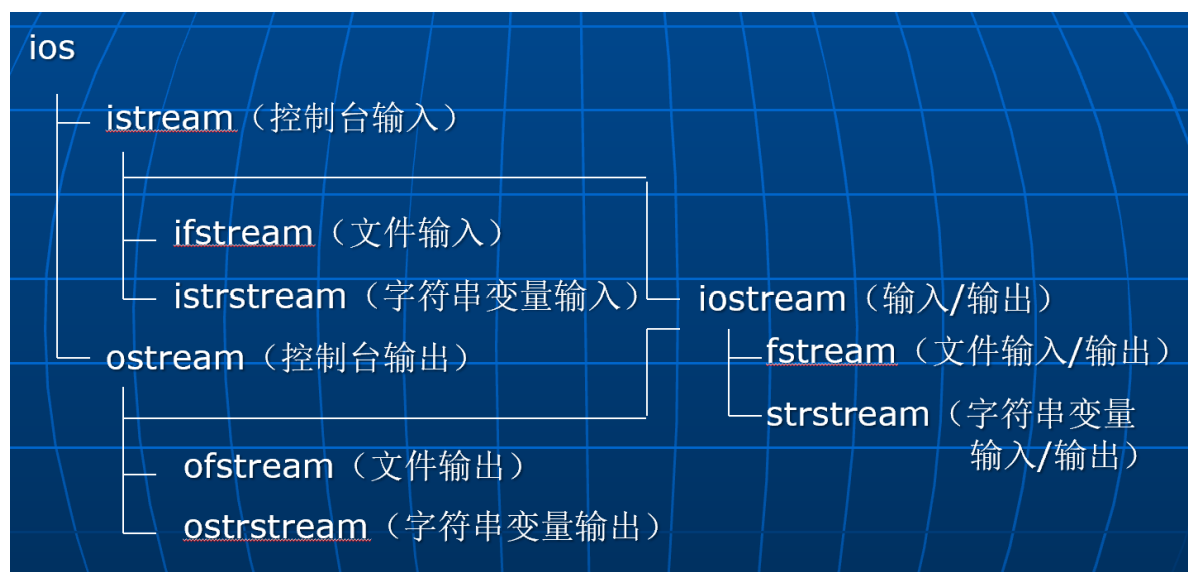


Solution12

191220008 陈南瞳

概念题

1、分析说明C++语言的流类库中为什么要将ios类作为其派生类的虚基类。



从上图中的流类库的基本结构可以看到，`ios` 类是 `istream` 类和 `ostream` 类的基类，而 `iostream` 类通过多重继承 `istream` 类和 `ostream` 类而产生的。如果不将 `ios` 类作为其派生类的虚基类，会导致重复继承 `ios` 类，产生二义性。

2、请简要概述文件缓冲区的作用，并结合其回答，程序中为什么要显式的关闭文件？

当磁盘里存取信息时，我们先把读出（写入）的数据放在缓冲区，计算机再直接从缓冲区中读（写）数据，等缓冲区的数据读完后再去磁盘中读取（或写过程结束），这样就可以减少磁盘的读写次数，由于计算机对磁盘的操作较慢，故应用缓冲区可大大提高文件读写的效率。

因此，需要把文件内存缓冲区的内容写到磁盘文件中。然而在某些情况下（例如上面程序中），即使不显式调用 `close()` 方法，文件的读写操作也能成功执行。因为当文件流对象的生命周期结束时，会自行调用析构函数，此函数内部会先调用 `close()` 方法切断文件流对象与任何文件的关联，最后才销毁它。

但是，在实际进行文件操作的过程中，对于打开的文件，要及时调用 `close()` 方法将其关闭。因为对于已经打开的文件，如果不及时关闭，一旦程序中途出现异常，则很可能会导致之前读写文件的所有操作失效。

编程题

1、在第九次作业中，我们实现了复数矩阵的运算，本次作业要求增加一些输入输出接口。

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

using namespace std;

class Complex
{
private:
    double real;
    double image;
public:
    Complex(double _real = 0.0, double _image = 0.0) :real{ _real }, image{
_image } {}
    Complex(const Complex& complex) :real{ complex.real }, image{ complex.image
} {}
    bool operator == (const Complex& complex) {
        return real == complex.real && image == complex.image;
    }
    bool operator != (const Complex& complex) {
        return real != complex.real || image != complex.image;
    }
    Complex operator + (const Complex& complex) {
        return Complex(real + complex.real, image + complex.image);
    }

    Complex operator - (const Complex& complex) {
        return Complex(real - complex.real, image - complex.image);
    }

    Complex operator * (const Complex& complex) {
        double _real = real * complex.real - image * complex.image;
        double _image = image * complex.real + real * complex.image;
        return Complex(_real, _image);
    }

    Complex operator / (const Complex& complex) {
        double _real = (real * complex.real + image * complex.image) /
(complex.real * complex.real + complex.image * complex.image);
        double _image = (image * complex.real - real * complex.image) /
(complex.real * complex.real + complex.image * complex.image);
        return Complex(_real, _image);
    }
    Complex& operator = (const Complex& complex)
    {
        if (this != &complex)
        {
            real = complex.real;
            image = complex.image;
        }
        return *this;
    }
};
```

```

    }
    Complex& operator += (const Complex& complex) {
        real += complex.real;
        image += complex.image;
        return *this;
    }

    friend istream& operator >> (istream& in, Complex& complex);
    friend ostream& operator << (ostream& out, Complex& complex);
};

istream& operator >> (istream& in, Complex& complex) {
    string str;
    in >> str;
    string::size_type pos = str.find("+");
    complex.real = stod(str.substr(0, pos));
    complex.image = stod(str.substr(pos + 1));
    return in;
}

ostream& operator << (ostream& out, Complex& complex)
{
    out << complex.real;
    if (complex.image >= 0)
    {
        out << "+" << complex.image << "i";
    }
    else
    {
        out << complex.image << "i";
    }
    return out;
}

template <class Type>
class Matrix
{
private:
    Type** p_data; //表示矩阵数据
    int row, col; //表示矩阵的行数和列数
public:
    Matrix(); //构造函数
    Matrix(int r, int c); //构造函数
    Matrix(const Matrix<Type>& m); //拷贝构造函数
    ~Matrix(); //析构函数
    Type*& operator[] (int i); //重载[], 对于Matrix对象m, 能够通过m[i][j]访问第i+1行、
    第j+1列元素
    Matrix<Type>& operator = (const Matrix<Type>& m); //重载=, 实现矩阵整体赋值, 若
    行/列不等, 归还空间并重新分配
    bool operator == (const Matrix<Type>& m) const; //重载==, 判断矩阵是否相等
    Matrix<Type> operator + (const Matrix<Type>& m) const; //重载+, 完成矩阵加法, 可
    假设两矩阵满足加法条件(两矩阵行、列分别相等)
    Matrix<Type> operator * (const Matrix<Type>& m) const; //重载*, 完成矩阵乘法, 可
    假设两矩阵满足乘法条件(this.col = m.row)

    friend istream& operator >> <Type>(istream& input, Matrix<Type>& m); //矩阵的
    输入, 第一行为 行数n 列数m, 紧接着有n行m列的矩阵元素

```

```

        friend ostream& operator << <Type>(ostream& output, const Matrix<Type>& m);
//矩阵的输出, 输出首先为 行数 列数 回车 矩阵内容
};

template <class Type>
Matrix<Type>::Matrix()
{
    ;
}

template <class Type>
Matrix<Type>::Matrix(int r, int c)
{
    row = r;
    col = c;
    p_data = new Type * [row];
    for (int i = 0; i < row; i++)
    {
        p_data[i] = new Type[col];
        for (int j = 0; j < col; j++)
            p_data[i][j] = 0;
    }
}

template <class Type>
Matrix<Type>::Matrix(const Matrix<Type>& m)
{
    row = m.row;
    col = m.col;
    p_data = new Type * [row];
    for (int i = 0; i < row; i++)
    {
        p_data[i] = new Type[col];
        for (int j = 0; j < col; j++)
        {
            p_data[i][j] = m.p_data[i][j];
        }
    }
}

template <class Type>
Matrix<Type>::~~Matrix()
{
    for (int i = 0; i < row; i++)
        delete[] p_data[i];
    delete[] p_data;
}

template <class Type>
Type*& Matrix<Type>::operator[] (int i)
{
    return p_data[i];
}

template <class Type>
Matrix<Type>& Matrix<Type>::operator = (const Matrix<Type>& m)
{
    if (row != m.row || col != m.col)

```

```

{
    this->~Matrix();
    p_data = new Type * [m.row];
    for (int i = 0; i < m.row; i++)
    {
        p_data[i] = new Type[m.col];
        for (int j = 0; j < m.col; j++)
            p_data[i][j] = 0;
    }
    row = m.row;
    col = m.col;
}
for (int i = 0; i < row; i++)
{
    for (int j = 0; j < col; j++)
    {
        p_data[i][j] = m.p_data[i][j];
    }
}
return *this;
}

template <class Type>
bool Matrix<Type>::operator == (const Matrix<Type>& m) const
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (p_data[i][j] != m.p_data[i][j])
                return false;
        }
    }
    return true;
}

template <class Type>
Matrix<Type> Matrix<Type>::operator + (const Matrix<Type>& m) const
{
    Matrix<Type> matrix(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            matrix.p_data[i][j] = p_data[i][j] + m.p_data[i][j];
        }
    }
    return matrix;
}

template <class Type>
Matrix<Type> Matrix<Type>::operator * (const Matrix<Type>& m) const
{
    Matrix<Type> matrix(row, m.col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < m.col; j++)
        {

```

```

        for (int k = 0; k < col; k++)
        {
            matrix.p_data[i][j] += p_data[i][k] * m.p_data[k][j];
        }
    }
}
return matrix;
}

template <class Type>
istream& operator >> (istream& input, Matrix<Type>& m)
{
    input >> m.row >> m.col;
    m.p_data = new Type * [m.row];
    for (int i = 0; i < m.row; i++)
    {
        m.p_data[i] = new Type[m.col];
        for (int j = 0; j < m.col; j++)
            m.p_data[i][j] = 0;
    }
    for (int i = 0; i < m.row; i++)
    {
        for (int j = 0; j < m.col; j++)
        {
            input >> m.p_data[i][j];
        }
    }
    return input;
}

template <class Type>
ostream& operator << (ostream& output, const Matrix<Type>& m)
{
    output << m.row << ' ' << m.col << endl;
    for (int i = 0; i < m.row; i++)
    {
        for (int j = 0; j < m.col; j++)
        {
            output << m.p_data[i][j] << ' ';
        }
        output << endl;
    }
    return output;
}

```

测试代码:

```

int main()
{
    Complex C;
    cin >> C;
    cout << endl << "输出C:" << endl << C << endl << endl;
    Matrix<Complex> m1, m2, m3;
    cin >> m1;
    cout << endl << "输出m1:" << endl << m1;
}

```

```

        ifstream in_file("C:\\南\\大二下\\高级程序设计\\作业\\12\\matrix_test.txt",
ios::in);
        if (!in_file)
            exit(-1);
        in_file >> m2;
        cout << endl << "输出m2:" << endl << m2 << endl;
        in_file.close();
        m3 = m1 * m2;
        cout << "输出m3=m1*m2:" << endl << m3;
        return 0;
    }

```

测试结果:

```

2+3
输出C:
2+3i
2 2
1+2 2+3
3+2 1+1
输出m1:
2 2
1+2i 2+3i
3+2i 1+1i
输出m2:
2 3
1+1i 2+1i 3+1i
2+3i 4+2i 5+3i
输出m3=m1*m2:
2 3
-6+15i 2+21i 2+28i
0+10i 6+13i 9+17i

```

2、给定一个文件，内容为1-100数字的拼接，首先你需要生成这样的一个文件，然后读取文件，将以0结尾的数字输出出来。

```

#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ofstream out_file("C:\\南\\大二下\\高级程序设计\\作业\\12\\zero_search.txt",
ios::out);
    if (!out_file)
        exit(-1);
    for (int i = 1; i <= 100; i++)
        out_file << i;
}

```

```

out_file.close();
ifstream in_file("C:\\南\\大二下\\高级程序设计\\作业\\12\\zero_search.txt",
ios::in);
if (!in_file)
    exit(-1);
char ch;
int len = 2;
char num[4];
num[3] = '\0';
while(!in_file.eof())
{
    in_file >> ch;
    if (ch == '0')
    {
        in_file >> ch;
        in_file.seekg(-3, ios::cur);
        if (ch == '0')
            len = 3;
        in_file.read(num, len);
        cout << num << endl;
        if (len == 3)
            break;
    }
}
in_file.close();
return 0;
}

```

测试结果：

zero_search - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707

第 1 行, 第 1 列 100% Windows (CRLF) UTF-8


```
10
20
30
40
50
60
70
80
90
100
```

3、某课程G需要将考试成绩录进系统并进行相应的保存，需要实现一个简单的成绩表管理系统。

一条成绩的记录包括学号，姓名，性别和分数，例如“1 小明 男 88”。

1. 我们需要从 **键盘** 录入成绩的记录信息，然后保存到文件 a 当中。
2. 从文件 a 中读出成绩信息。
3. 输出成绩前3名的学生信息，输出到新文件 b 中。
4. 对于男女生各计算平均分，再将成绩在平均分以下的记录输出到一个新文件 c 中。
5. 增加几条新的补考记录，对分数按 90% 折算录进系统。
6. 输出数据到文件 a 中，完成表格的更新。

要求：设计成绩表为一个类，通过重载 << 和 >> 来实现成绩表的输入/输出。

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;

int n;

class Grade
{
public:
    string id;
    string name;
    string sex;
    double score;

    friend istream& operator >> (istream& in, Grade& grade);
    friend ostream& operator << (ostream& out, const Grade& grade);
};

istream& operator >> (istream& in, Grade& grade)
{
    in>>grade.id >> grade.name >> grade.sex >> grade.score;
    return in;
}

ostream& operator << (ostream& out, const Grade& grade)
```

```

{
    out << grade.id << ' ' << grade.name << ' ' << grade.sex << ' ' <<
grade.score;
    return out;
}

class GradeList
{
public:
    vector<Grade>grade_list;

    friend istream& operator >> (istream& in, GradeList& gradelist);
    friend ostream& operator << (ostream& out, const GradeList& gradelist);
};

istream& operator >> (istream& in, GradeList& gradelist)
{
    Grade grade;
    for (int i = 0; i < n; i++)
    {
        in >> grade;
        gradelist.grade_list.push_back(grade);
    }
    return in;
}

ostream& operator << (ostream& out, const GradeList& gradelist)
{
    for (int i = 0; i < gradelist.grade_list.size(); i++)
        out << gradelist.grade_list[i] << endl;
    return out;
}

int main()
{
    // 1.我们需要从键盘录入成绩的记录信息，然后保存到文件 a 当中。
    cout << "请输入学生人数和需要录入的成绩记录信息：" << endl;
    cin >> n;
    GradeList gradelist;
    cin >> gradelist;
    ofstream out_filea("C:\\南\\大二下\\高级程序设计\\作业\\12\\a.txt", ios::out);
    if (!out_filea)
        exit(-1);
    out_filea << gradelist;
    out_filea.close();

    // 2.从文件 a 中读出成绩信息。
    ifstream in_filea("C:\\南\\大二下\\高级程序设计\\作业\\12\\a.txt", ios::in);
    if (!in_filea)
        exit(-1);
    GradeList gradelist2;
    in_filea >> gradelist2;

    // 3.输出成绩前3名的学生信息，输出到新文件 b 中。
    sort(gradelist2.grade_list.begin(), gradelist2.grade_list.end(), [](Grade&
grade1, Grade& grade2) {return grade1.score > grade2.score; });
    ofstream out_fileb("C:\\南\\大二下\\高级程序设计\\作业\\12\\b.txt", ios::out);
    if (!out_fileb)

```

```

        exit(-1);
    out_fileb << gradelist2.grade_list[0] << endl;
    out_fileb << gradelist2.grade_list[1] << endl;
    out_fileb << gradelist2.grade_list[2] << endl;

    // 4.对于男女生各计算平均分, 再将成绩在平均分以下的记录输出到一个新文件 c 中。
    double average_male = 0, average_female = 0;
    GradeList gradelist_male, gradelist_female;
    copy_if(gradelist.grade_list.begin(), gradelist.grade_list.end(),
    back_inserter(gradelist_male.grade_list), [](Grade& grade) {return grade.sex ==
    "男"; });
    copy_if(gradelist.grade_list.begin(), gradelist.grade_list.end(),
    back_inserter(gradelist_female.grade_list), [](Grade& grade) {return grade.sex
    == "女"; });
    average_male = accumulate(gradelist_male.grade_list.begin(),
    gradelist_male.grade_list.end(), 0.0, [](double a, Grade& grade) {return a +
    grade.score; }) / gradelist_male.grade_list.size();
    average_female = accumulate(gradelist_female.grade_list.begin(),
    gradelist_female.grade_list.end(), 0.0, [](double a, Grade& grade) {return a +
    grade.score; }) / gradelist_female.grade_list.size();
    GradeList gradelist_male_below, gradelist_female_below;
    copy_if(gradelist_male.grade_list.begin(), gradelist_male.grade_list.end(),
    back_inserter(gradelist_male_below.grade_list), [average_male](Grade& grade)
    {return grade.score < average_male; });
    copy_if(gradelist_female.grade_list.begin(),
    gradelist_female.grade_list.end(),
    back_inserter(gradelist_female_below.grade_list), [average_female](Grade& grade)
    {return grade.score < average_female; });
    ofstream out_filec("c:\\南\\大二下\\高级程序设计\\作业\\12\\c.txt", ios::out);
    if (!out_filec)
        exit(-1);
    out_filec << gradelist_male_below << gradelist_female_below;
    out_filec.close();

    // 5.增加几条新的补考记录, 对分数按 90% 折算录进系统。
    cout << "请输入补考学生人数和需要录入的补考成绩记录信息: " << endl;
    int n_rest;
    cin >> n_rest;
    Grade grade_rest;
    for (int i = 0; i < n_rest; i++)
    {
        cin >> grade_rest;
        grade_rest.score *= 0.9;
        gradelist.grade_list.push_back(grade_rest);
    }

    // 6.输出数据到文件 a 中, 完成表格的更新。
    ofstream out_filea2("c:\\南\\大二下\\高级程序设计\\作业\\12\\a.txt", ios::out);
    if (!out_filea2)
        exit(-1);
    out_filea2 << gradelist;
    out_filea2.close();

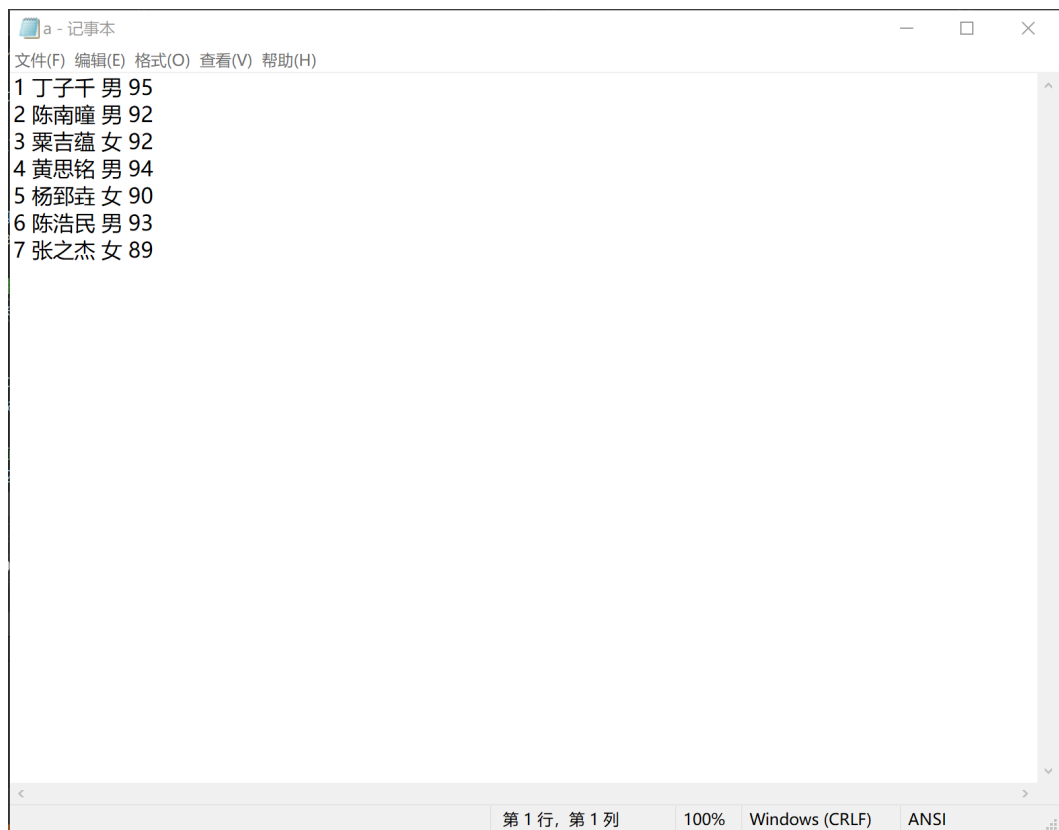
    return 0;
}

```

测试结果：

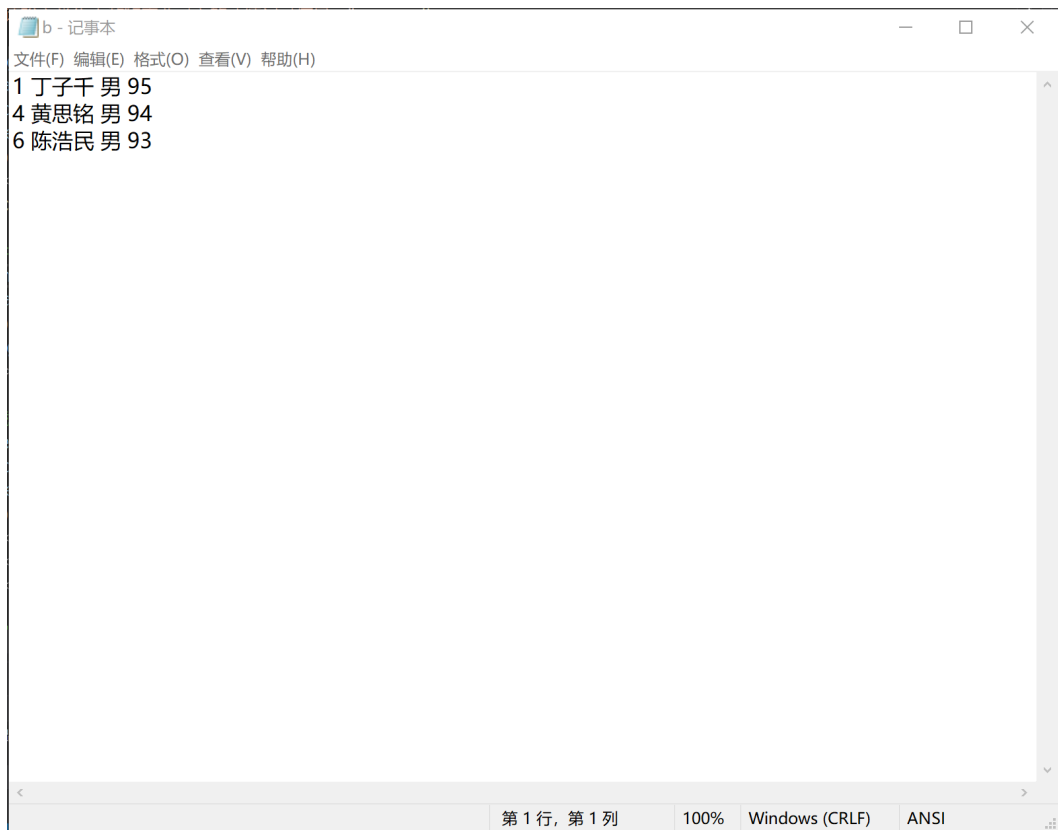
```
请输入学生人数和需要录入的成绩记录信息：
7
1 丁子千 男 95
2 陈南瞳 男 92
3 粟吉蕴 女 92
4 黄思铭 男 94
5 杨郢垚 女 90
6 陈浩民 男 93
7 张之杰 女 89
请输入补考学生人数和需要录入的补考成绩记录信息：
2
8 李华坤 男 60
9 叶洪宇 女 61
```

a.txt:

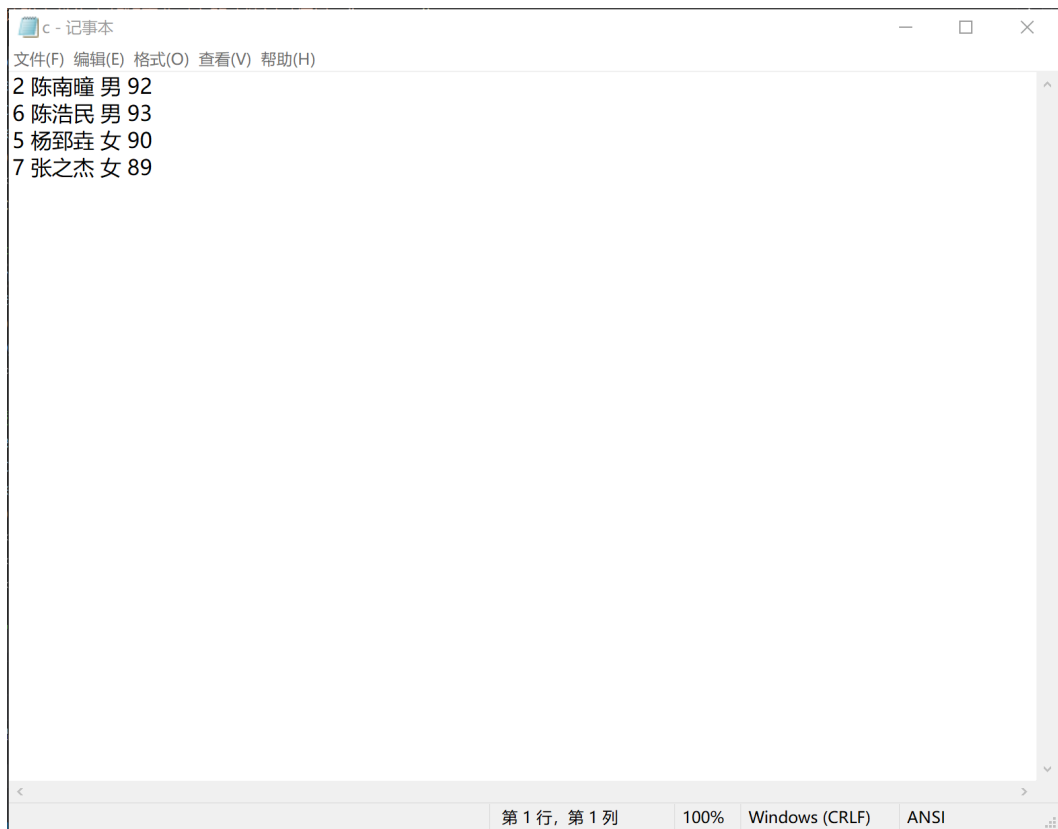


```
a - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1 丁子千 男 95
2 陈南瞳 男 92
3 粟吉蕴 女 92
4 黄思铭 男 94
5 杨郢垚 女 90
6 陈浩民 男 93
7 张之杰 女 89
第 1 行, 第 1 列 100% Windows (CRLF) ANSI
```

b.txt:



c.txt:



a.txt:

