**《程序设计基础》Project报告**

**（CST11103）**

|  |  |  |  |
| --- | --- | --- | --- |
| **姓 名** | **魏永森** | **学 号** | **20192242** |
| **年级、专业、班级** | **19级计算机科学与技术（卓越）01班** | | |
| **上课时间** | **2021-2022学年第1学期** | | |
| **Project成绩** | **代码成绩**  （70%） | **报告成绩**  （30%） | **综合成绩** |
|  |  |  |
| **评语** |  | | |
| **评价教师** |  | | |

**《程序设计基础》Project任务书**

|  |  |
| --- | --- |
| 名 称 | 分数计算器的设计与实现 |
| 类 型 | □验证性 □设计性 ■综合性 |
| 内 容 | 运用面向对象程序设计思想，基于命令行界面设计并实现一个分数计算器。 |
| 要 求 | （1）按照分数类Franction的声明，实现其所有成员函数、友元及运算符重载。类的声明如下：  class Fraction  {  friend Fraction operator+(const Fraction& frac1, const Fraction& frac2); //重载+运算符  friend Fraction operator-(const Fraction& frac1, const Fraction& frac2); //重载-运算符  friend Fraction operator\*(const Fraction& frac1, const Fraction& frac2); //重载\*运算符  friend Fraction operator/(const Fraction& frac1, const Fraction& frac2); //重载/运算符  friend bool operator==(Fraction frac1, Fraction frac2); //重载==运算符  friend bool operator>(const Fraction& frac1, const Fraction& frac2); //重载>运算符  friend bool operator<(const Fraction& frac1, const Fraction& frac2); //重载<运算符  friend ostream& operator<<(ostream& out, const Fraction& frac); //重载<<运算符  friend istream& operator>>(istream& in, Fraction& frac); //重载>>运算符  friend void sortFraction(参数列表自行设计);//对分数数组排序  public:  Fraction(); //无参造函数  Fraction(int n, int d); //带参造函数  Fraction(const Fraction& f); //复制造函数  void setFraction(int n, int d); //设置分数的分子和分母  int getNumer(); //获取分数的分子  int getDeno(); //获取分数的分母  void RdcFrc(); //当前分数约分  private:  int numer; //分子  int deno; //分母  };   1. 分数计算器的主要功能为：   ① 程序开始时选择想要使用的功能，如图1所示  请选择功能（键入1或者2）：   1. 分数计算 2. 分数排序   —  图1 分数计算器开始界面  ② 如果选择“1.分数计算”则进入计算界面，如图2所示  请输入分数计算式（如1/2+1/3回车），输入#号键返回上一层目录：  —  图2分数计算初始界面  如果输入正确分数计算式给出计算结果，并等待下一次用户输入，如图3所示  请输入分数计算式（如1/2+1/3回车），输入#号键返回上一层目录：  1/4+1/4  =1/2  请输入分数计算式（如1/2+1/3回车），输入#号键返回上一层目录：  —  图3分数计算界面  如果输入错误分数计算式，提示输入错误并等待用户重新输入，如图4所示  请输入分数计算式（如1/2+1/3回车），输入#号键返回上一层目录：  1/4？1/a  分数算式输入错误！  请输入分数计算式（如1/2+1/3回车），输入#号键返回上一层目录：  —  图4分数计算输入错误界面  ③ 如果选择“2.分数排序”则进入分数排序界面，如图5所示  请输入一组分数，用逗号隔开，如需由小到大排序用符号<结尾，由大到小排序用符号>结尾（如1/2,1/4,3/5<回车），输入#号键返回上一层目录：  —  图5 分数排序初始界面  注意：这里分数个数无上限限制，应考虑动态地创建分数的数组序列。  当输入一组正确分数后，输出排序结果，并等待下一次用户输入，如图6所示：  请输入一组分数，用逗号隔开，如需由小到大排序用符号<结尾，由大到小排序用符号>结尾（如1/2,1/4,3/5<回车），输入#号键返回上一层目录：  1/2,1/5,3/7<  1/5 3/7 1/2  请输入一组分数，用逗号隔开，如需由小到大排序用符号<结尾，由大到小排序用符号>结尾（如1/2,1/4,3/5<回车），输入#号键返回上一层目录：  —  图6 分数排序界面  当输入的分数不正确，则提示输入错误，并等待用户重新输入，如图7所示：  请输入一组分数，用逗号隔开，如需由小到大排序用符号<结尾，由大到小排序用符号>结尾（如1/2,1/4,3/5<回车），输入#号键返回上一层目录：  1/2,1/a3/7  输入错误！  请输入一组分数，用逗号隔开，如需由小到大排序用符号<结尾，由大到小排序用符号>结尾（如1/2,1/4,3/5<回车），输入#号键返回上一层目录：  —  图7 分数排序输入错误界面 |
| 任务时间 | 2021年12月6日至2021年12月31日 |

**代码评分标准（满分10分）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **学习态度** | **程序功能** | **程序Bug** | **程序界面** | **异常处理** | **得分** |
| 抄袭、被抄袭 | -- | -- | -- | -- | **0** |
| 自己完成编码  无创新意识 | 没有完成实验要求的所有功能 | -- | -- | -- | **5** |
| 自己完成编码  有一定的创新意识 | 基本完成实验要求的所有功能 | 很多 | 一般 | 无 | **6** |
| 自己完成编码  有一定的创新意识 | 完成实验要求的所有功能 | 较多 | 一般 | 无 | **7** |
| 自己完成编码  有较强的创新意识 | 完成实验要求的所有功能，并有所扩展 | 较少 | 良好 | 有 | **8** |
| 自己完成编码  有较强的创新意识 | 完成实验要求的所有功能，并有所扩展 | 极少 | 友好 | 有 | **10** |

**报告评分标准（满分10分）**

|  |  |  |  |
| --- | --- | --- | --- |
| **文档规范性** | **文档条理性** | **表述准确性** | **得分** |
| 不规范，需求分析、系统设计、系统实现部分不完整。 | 描述缺少逻辑性，没有条理。 | 文字描述不准确。 | **3** |
| 较规范，有需求分析、系统设计、系统实现部分，但实质性描述较少。 | 描述具有一定的逻辑性，条理较清晰。 | 文字描述较准确，但缺少必要的类图、程序流程图等图表说明。 | **5** |
| 规范，有需求分析、系统设计、系统实现部分，描述较充分。 | 描述具有较好的逻辑性，条理清晰 | 文字描述较准确，所描述的类图、程序流程图基本准确、无大错。 | **7** |
| 规范，有需求分析、系统设计、系统实现部分，描述充分。 | 描述具有很好的逻辑性，条理清晰 | 文字描述准确，所描述的类图、程序流程图准确。 | **10** |

目录

[1. 需求分析 1](#_Toc91871262)

[1）数据处理： 1](#_Toc91871263)

[2）数据存储： 1](#_Toc91871264)

[3）功能： 1](#_Toc91871265)

[2. 系统设计 1](#_Toc91871266)

[3. 系统实现 1](#_Toc91871267)

[1）分数类 2](#_Toc91871268)

[2）计算器的运行实现 4](#_Toc91871269)

[4. 总结 6](#_Toc91871270)

[5. 参考文献 7](#_Toc91871271)

[6. 程序的运行过程及结果截图 7](#_Toc91871272)

[1）计算功能 7](#_Toc91871273)

[2）排序功能 7](#_Toc91871274)

[7. 全部源代码的截图 8](#_Toc91871275)

[1） Fraction.h 8](#_Toc91871276)

[2） run.cpp 14](#_Toc91871277)

# 1. 需求分析

拟解决问题：

1）数据处理：

a. 将“数字”+“/”+“数字”这样的组合作为一个分数来进行整体读入，并且还不能影响后续输入的字符或者数字，是具有一定难度的。

2）数据存储：

a. 分数排序过程中需要使用到分数的动态存储，将动态输入的Fraction变量存储到数组中，然后在读取到结束符号的时候对当前数组进行相应方式的排序。

3）功能：

a. 实现界面跳转功能，主界面到计算和排序两个子界面，然后从子界面返回到主界面。

b. 分数计算功能

c. 分数排序功能

d. 退出计算功能

# 2. 系统设计

程序主体分为2个模块：1）分数类Fraction.h 2）运行主函数 run.cpp

目的：把一个多功能的复杂的程序划分为若干个简单的、功能单一的程序模块。

优点：

a. 有利于程序的设计和调试

b. 有利于程序的优化和分工

c. 提高了程序的阅读性和可靠性

d. 使程序的结构层次一目了然

# 3. 系统实现

实现效果与指导书中一致。（运行过程放到6.运行过程及结果截图中）

使用方法如下：

运行之后，会呈现主界面，提示选择分数计算（1）功能还是分数排序（2）功能。退出需要输入‘#’，按其他键会重复显示主界面。

**a.** 输入1进入分数计算界面，按照‘分数’+‘符号’+‘分数’+‘回车’格式输入即可。错误格式计算器会提醒，然后用户可以重新输入。

如输入： 1/2+3/4

按下回车之后会显示结果： =5/4/. (‘+’ 和分数之间可以存在空格)

**b.** 输入2进行分数计算界面，按照‘分数’+‘，’+‘分数’+‘，’+…+‘<’（或>结束）+‘回车’。写一行会显示 从小到大 还是 从大到小 排序，并给出排序结果。输入其它格式会报错，然后用户进行下一次排序，重新输入。

例如，输入： 1/2,1/5,3/7<

按下回车之后， 显示结果： 从小到大排序：1/5 3/7 1/2

实现主要分为两步来实现：1.分数类的定义和实现；2.计算器运行的实现。

1）分数类

成员函数都在指导书中给出，实现难度不大。稍微难一点儿的部分是读入（cin）友元函数和排序（sortFraction）友元函数。

a. 读入函数读入的难点是考虑要作为字符串还是数字和字符的组合来实现。

* 刚开始的时候选择了字符串实现方式，具体实现如下：

string s;

    in >> setw(3) >> s;

    if(frac.deno==0){

        cout << "分母不可以为0" << endl;

    }

    if(s[1]!='/' || !isdigit(s[0]) || !isdigit(s[2])){

        cout << "输入不合规范" << endl;

    }

    frac.numer= s[0]-'0';

    frac.deno= s[2]-'0';

    if(s[0]=='#'){  // 在排序功能中代表停止，可以使用分母为0来区别

        frac.numer=0;

        frac.deno=0;

}

return in;

因为作为字符串的话，会将(1/2+3/4)作为一个整体输入。所以要限定每次读入的字符串长度，使用了setw()。这样加减乘除可以实现。

但是在排序的过程中，(1/2,3/7,4/5<)这样的指令也会作为一行字符串输入，很难读取多个数字，而且setw()限制的长度也无法确定，所以这种实现方法不合适。

* 然后就是采用将读入的分数(1/2)作为“数字”+“字符”+“数字”的组合，实现代码如下：

    char c='0';     // 初始化读入字符

    int a=1;        // 读入的分子

    int b=1;            // 读入的分母

    // in >> a >> c >> b;

    if(in>>a){                      // 分部读取，首先读取分子

        frac.numer=a;

    }else{

        cin.clear();cin.get(c);     // 清空缓存区，取出最后读出的字符。

        if(c=='#'){

            frac.deno=20192242;     // 设定特殊值

            return in;              // 检测到输入#则退出

        }

        // cout << c;               // 输出检查

        frac.deno=0;

        return in;

    }

    in >> c;                        // 读取字符

    if(c!='/'){

        cout << "输入出错！" << endl;

        frac.numer=a;

        frac.deno=0;

    }

    if(in>>b){                      // 读取分母

        if(c=='/')

            frac.deno=b;;

    }else{

        cin.clear();cin.get();      // 清空缓存区，取出最后取出的字符（特殊字符，空格或换行）

        frac.deno=0;

    }

    return in;

这里比较难的就是输入缓冲区的问题。因为可能会读入（a/2）这样的错误分数。

但是，我们假设的“数字”+“字符”+“数字”组合会在读到第一个a的时候，因为不是int型数值而导致cin.fail()，影响之后的读入。关于读入问题，将这个组合按顺序分为分子（a）读取出错，分号（/）读取出错，分母（b）读取出错。这里的解决参考了网上深入讲解cin的一篇博客[1]。

首先是分子读取出错。解决方法是需要在第一个数字分子的读入就出现读取类型错误的时候，使用cin.clear()清空缓存区，再使用cin.get(c)将读入的字符（目前在缓存区中）存储到变量c中，就可以查看读入的错误字符的值。如果在读入分子的时候就出错，比如a/2，那么后续的‘/2’也就不需要读入了，所以需要定义一个string s，使用s读取后面已经无效的字符串（包括空格，所以使用getline(cin,s)），这里主要是为了解决‘#’的输入问题。

因为分数计算器中所有的输入都是当作分数来读入的。那么‘#’的输入就比较难。因为无论在计算还是排序功能中，我们都会默认用户首先输入的是一个分数。但是如果用户每次重新开始使用计算和排序功能的时候想要退出，就得输入’#’，这里我们就可以将‘#’这个字符作为输入错误的分子来看。但是因为是在Fraction类中实现的输入重载，所以无法保存变量c的值。我们可以将分数的分母设置为一个特殊的值（这里取20192242）来作为分子读取到‘#’的标记。

接下来是对于分子的分号’/’读取出错，也采用了将分母设置特殊值的办法来表示。我们知道分数的分母不能为0，所以这里将分母设置为0表示分子读入的格式出错。

最后是对于分母b的输入出错，和分子的处理方式一样，读取到不一致的类型，使用cin.clear()和cin.get()清除输入缓冲区，并且取出最后的空格，回车等特殊字符。

b. 排序函数的实现

采用了冒泡排序的方法。使用标记flag来区分顺序排序还是倒序排序。

void sortFraction(Fraction f[],int num,int flag){//对分数数组排序,冒泡排序实现,flag 1和2 分别代表 从小到大 和 从大到小 排序

    int i,j;

    Fraction temp;

    for(i=0;i<num-1;i++) // 冒泡排序

    {

        for(j=1;j<num-i;j++)

        {

            if((f[j-1]>f[j]) & flag==1) // flag=1标记从小到大排序

            {

                temp=f[j-1];

                f[j-1]=f[j];

                f[j]=temp;

            }

            if((f[j-1]<f[j]) && flag==2) // flag=2标记从大到小排

            {

                temp=f[j-1];

                f[j-1]=f[j];

                f[j]=temp;

            }

        }

    }

}

2）计算器的运行实现

分为开始界面，分数计算和分数排序三部分来实现。

a. 开始界面

每次执行结束需要循环展示出开始界面，所以整体程序使用while(1)不断循环，直到遇到‘#’退出程序。整体程序框架伪代码如下：

    int main(){

while(1){

            if(得到程序终止条件){

                跳出循环;

            }

展示初始界面；选择需要计算还是排序功能。

设定标记变量n；

switch(n){

                     case 1:计算功能实现;

case 2:排序功能实现;

}

return 0;

}

b. 计算功能

因为需要循环读入，所以需要一个while来实现，然后就是判断是否跳出计算的循环，还有计算符号的判断（使用switch）。程序伪代码如下：

while(1){

    char op='0';

    Fraction f1,f2;

    cin >> f1 ;

    if(f1.getDeno()==20192242){

        说明读取到#，退出计算功能；

对之后输入进行处理；

        break;

    }

    cin >> op >> f2;

    // cout << "检查操作数:" << f1 << " " << f2 << endl;

    if(f1.getDeno()==0 || f2.getDeno()==0){

        不合规范，终止本次，开始下次；

对之后输入进行处理；

        continue;

}

判断op是否符合规范；

    switch(op){ // 进行四则运算

        case '+':

        case '-':

        case '\*':

        case '/':

    }

}

c. 排序功能

也是需要一个循环while来实现多次输入排序，每次输入的序列需要重复输入分数，又是一个while，一共有两个while来实现，直到输入了‘#’，跳出两层循环。然后分数的动态输入利用了vector，在输入完毕之后，可以利用vector.size()获取元素数量，然后创建一个Fraaction数组，最后调用sortFraction(Fraction f, int num, int flag)函数。伪代码如下：

flag=0;

while(1){

    if(flag){

        退出排序功能;

        break;

}

vector<Fraction> a;     // 使用vector可以动态添加输入的分数，并且可以很快地得到长度。

    char op;                // 记录输入的分割符号

    Fraction f;

    while(1){ // 再次循环，代表每次序列中的多个分数

        cin >> f;

首先判断是否输入了#，（f.getDeno()==20192242），根据判断break，并设置flag，使其跳出外部排序;

然后判断是否输入出错!

        经过上面的判断才可以添加到动态数组

        cin >> op; // 读取操作符号

        if(op==',')

            continue;

        else if(op=='<'){

            sortFraction(数组,长度,1); // 调用顺序排序

            break;

        }else if(op=='>'){

            sortFraction(数组,长度,2); // 调用倒序排序

            break;

        }else{

            说明输入出错;

            这一行的后续字符串应该被处理掉

            break;

        }

    }

}

# 4. 总结

总体来说结构和实现都不是太难。但是在实现过程中雀食也遇到了一些困难。

其中在我看来最难的部分是分数读取<<的重载，因为不能当成字符串来读取，但是当成数字读取又不会判断格式错误，在经过资料查询之后对cin这个标准函数也有了更深的理解，包括cin.clear(),cin.get(),cin.ignore()的使用。然后#退出功能就可以通过读取分数中将其保存到缓存区来解决。

排序部分的动态读取采用了vector，增加了自己对于stl容器的掌握程度。

但是从这次实验中获得的最大收入我感觉在于对cpp中类的构建与友元函数还有运算符重载**[2]**，因为之前对这方面的使用并不多，了解也不够深入，此次实验正好使自己加强了这方面的锻炼。

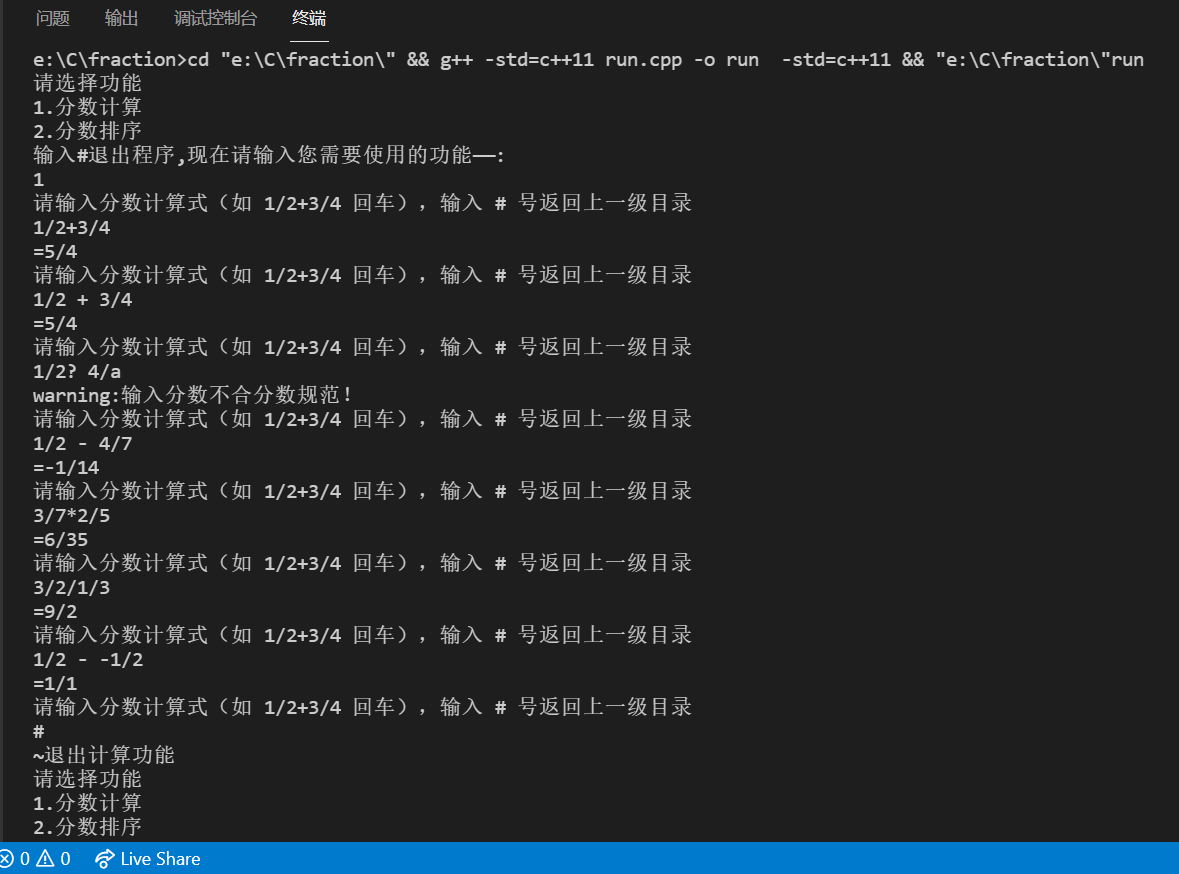
# 5. 参考文献

1. 努力的耿耿. 通过cin＞＞向一个int类型变量中读取一个字符问题[OL]. [(5条消息) 通过cin＞＞向一个int类型变量中读取一个字符问题\_努力的耿耿的博客-CSDN博客](https://blog.csdn.net/weixin_52093215/article/details/121465989).

2. Stephen Prata. C++ Primer Plus[M]. 北京:人民邮电出版社 2017：380-408.

# 6. 程序的运行过程及结果截图

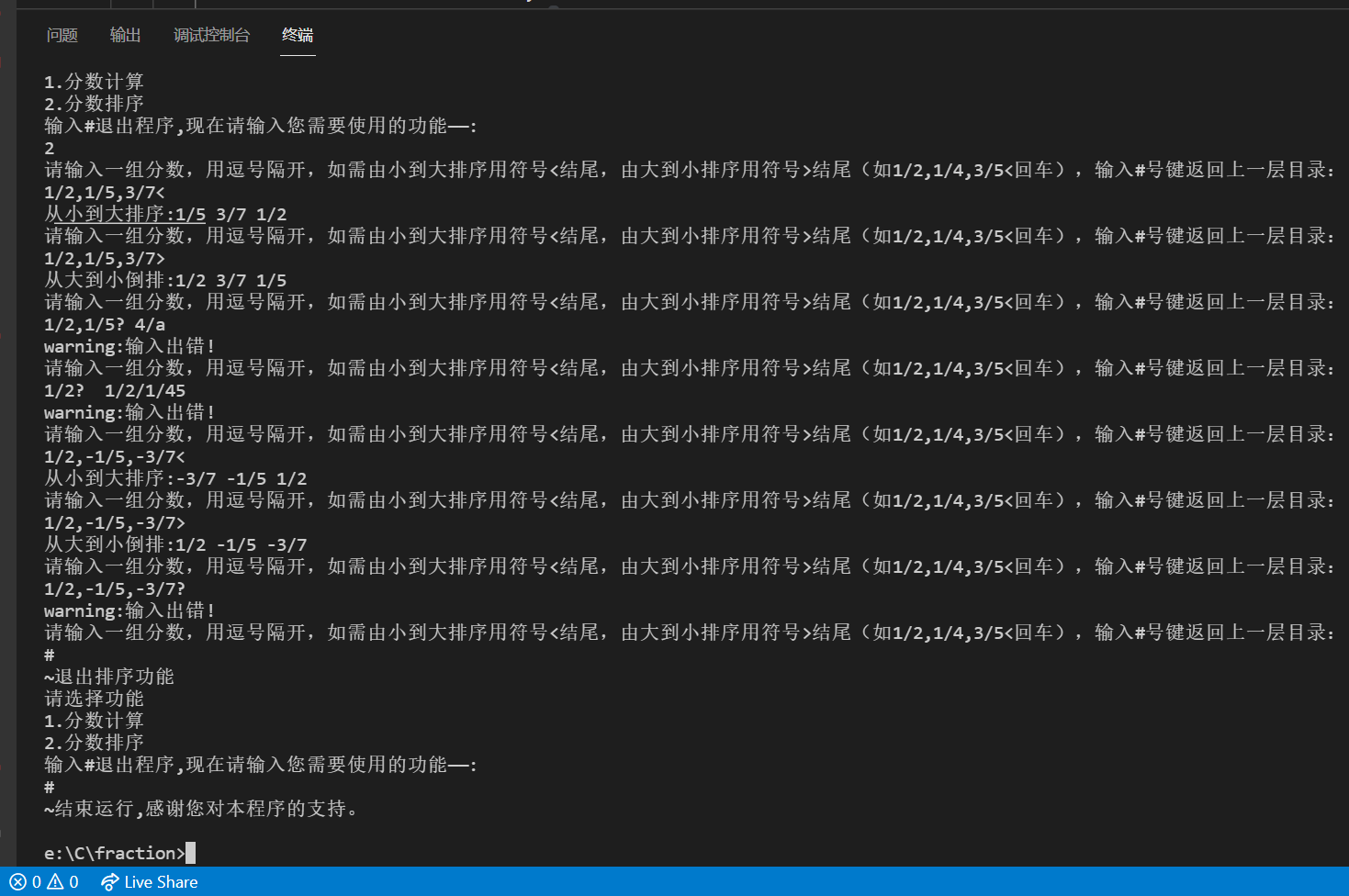
1）计算功能



可以从截图中看出，程序运行首先进入主界面，然后选择1会进入分数计算界面。如果输入分数算式合理，那么将会正常进行+-\*÷计算。但是如果输入格式有问题，那么将会提示warning（不影响下一次的输入），然后等待用户重新输入。支持负数的计算。

最后输入‘#’可以退出分数计算功能，并给出提示，之后返回主界面。

2）排序功能



从图中可以看出，选择2进入分数排序功能。

之后按照提示的分数排序格式输入并回车，那么会正确显示，排序方式和排序结果。但是如果格式不正确，那么就会提示输入出错（不影响下一次的输入），然后会结束本次输入，开始下一次，并让用户重新输入。

支持负数的排序，按下‘#’退出排序功能，再次按下‘#’退出整个程序。

# 7. 全部源代码的截图

1） Fraction.h

#include <bits/stdc++.h>

using namespace std;

class Fraction {

    private:

        int numer; //分子

        int deno; //分母

    public:

        Fraction(); //无参造函数

        Fraction(int n, int d); //带参造函数

        Fraction(const Fraction& f); //复制造函数

        void setFraction(int n, int d); //设置分数的分子和分母

        int getNumer(); //获取分数的分子

        int getDeno(); //获取分数的分母

        void RdcFrc(); //当前分数约分

        friend Fraction operator+(const Fraction& frac1, const Fraction& frac2); //重载+运算符

        friend Fraction operator-(const Fraction& frac1, const Fraction& frac2); //重载-运算符

        friend Fraction operator\*(const Fraction& frac1, const Fraction& frac2); //重载\*运算符

        friend Fraction operator/(const Fraction& frac1, const Fraction& frac2); //重载/运算符

        friend bool operator==(Fraction frac1, Fraction frac2); //重载==运算符

        friend bool operator>(const Fraction& frac1, const Fraction& frac2); //重载>运算符

        friend bool operator<(const Fraction& frac1, const Fraction& frac2); //重载<运算符

        friend  ostream& operator<<(ostream& out, const Fraction& frac); //重载<<运算符

        friend  istream& operator>>(istream& in, Fraction& frac); //重载>>运算符

        friend void sortFraction(Fraction f[],int num,int flag);//对分数数组排序

};

Fraction::Fraction(){

    numer=1;

    deno=1;

}

Fraction::Fraction(int n,int d){

    numer=n;

    deno=d;

}

Fraction::Fraction(const Fraction& f){

    numer=f.numer;

    deno=f.deno;

}

void Fraction::setFraction(int n,int d){ // 设置数值

    numer=n;

    deno=d;

}

int Fraction::getNumer(){ //获取分数的分子

    return numer;

}

int Fraction::getDeno(){ //获取分数的分母

    return deno;

}

void Fraction::RdcFrc(){ //获取约分

    if(deno==0 || numer==0){

        return ;

    }

    else{

        int temp=0;    // 遍历的除数大小

        if(numer>deno){

            temp=sqrt(numer);

        }else{

            temp=sqrt(deno);

        }

        for(int i=2;i<=temp+1;i++){     // 每次都从2开始遍历

            if(numer%i==0 && deno%i==0){

                numer/=i;

                deno/=i;

                i=1;

            }

        }

        if(numer%deno==0){  // 如果两者是倍数关系

            numer/=deno;

            deno=1;

        }else if(deno%numer==0){

            deno/=numer;

            numer=1;

        }

    }

}

ostream& operator<<(ostream& out, const Fraction& frac){ //重载<<运算符

    int flag=0;

    if(frac.numer\*frac.deno>=0){

        flag=1;

    }else{

        flag=0;

    }

    if(flag){

        out << frac.numer << "/" << frac.deno;

    }else{

        out << "-" << abs(frac.numer) << "/" << abs(frac.deno);

    }

    return out;

}

istream& operator>>(istream& in, Fraction& frac){ //重载>>运算符

    // string s;

    // in >> setw(3) >> s;

    // if(frac.deno==0){

    //     cout << "分母不可以为0" << endl;

    // }

    // if(s[1]!='/' || !isdigit(s[0]) || !isdigit(s[2])){

    //     cout << "输入不合规范" << endl;

    // }

    // frac.numer= s[0]-'0';

    // frac.deno= s[2]-'0';

    // if(s[0]=='#'){  // 在排序功能中代表停止，可以使用分母为0来区别

    //     frac.numer=0;

    //     frac.deno=0;

    // }

    char c='0';     // 初始化读入字符

    int a=1;        // 读入的分子

    int b=1;            // 读入的分母

    // in >> a >> c >> b;

    if(in>>a){                      // 分部读取，首先读取分子

        frac.numer=a;

    }else{

        cin.clear();cin.get(c);     // 清空缓存区，取出最后读出的字符。

        if(c=='#'){

            frac.deno=20192242;     // 设定特殊值

            return in;              // 检测到输入#则退出

        }

        // cout << c;               // 输出检查

        frac.deno=0;

        return in;

    }

    in >> c;                        // 读取字符

    if(c!='/'){

        cout << "输入出错！" << endl;

        frac.numer=a;

        frac.deno=0;

    }

    if(in>>b){                      // 读取分母

        if(c=='/')

            frac.deno=b;;

    }else{

        cin.clear();cin.get();      // 清空缓存区，取出最后取出的字符（特殊字符，空格或换行）

        frac.deno=0;

    }

    return in;

}

Fraction operator+(const Fraction& frac1, const Fraction& frac2){ //重载+运算符

    Fraction ans;

    // 利用通分计算

    ans.deno=frac1.deno \* frac2.deno;

    ans.numer= frac1.numer\*frac2.deno + frac2.numer\*frac1.deno;

    ans.RdcFrc();

    return ans;

}

Fraction operator-(const Fraction& frac1, const Fraction& frac2){ //重载-运算符

    Fraction ans;

    // 利用通分计算

    ans.deno=frac1.deno \* frac2.deno;

    ans.numer= frac1.numer\*frac2.deno - frac2.numer\*frac1.deno;

    ans.RdcFrc();

    return ans;

}

Fraction operator\*(const Fraction& frac1, const Fraction& frac2){ //重载\*运算符

    Fraction ans;

    ans.numer=frac1.numer\*frac2.numer;

    ans.deno=frac1.deno\*frac2.deno;

    ans.RdcFrc();

    return ans;

}

Fraction operator/(const Fraction& frac1, const Fraction& frac2){ //重载/运算符

    Fraction ans;

    ans.numer=frac1.numer\*frac2.deno;

    ans.deno=frac1.deno\*frac2.numer;

    ans.RdcFrc();

    return ans;

}

bool operator==(Fraction frac1, Fraction frac2){ //重载==运算符

    return (frac1-frac2).numer==0;

}

bool operator>(const Fraction& frac1, const Fraction& frac2){ //重载>运算符

    return (float(frac1.numer) / frac1.deno) > (float(frac2.numer) /frac2.deno);

}

bool operator<(const Fraction& frac1, const Fraction& frac2){ //重载<运算符

        return (float(frac1.numer) / frac1.deno) < (float(frac2.numer) / frac2.deno);

}

void sortFraction(Fraction f[],int num,int flag){//对分数数组排序,冒泡排序实现,flag 1和2 分别代表 从小到大 和 从大到小 排序

    int i,j;

    Fraction temp;

    for(i=0;i<num-1;i++)

    {

        for(j=1;j<num-i;j++)

        {

            if((f[j-1]>f[j]) & flag==1) // flag=1标记从小到大排序

            {

                temp=f[j-1];

                f[j-1]=f[j];

                f[j]=temp;

            }

            if((f[j-1]<f[j]) && flag==2) // flag=2标记从大到小排

            {

                temp=f[j-1];

                f[j-1]=f[j];

                f[j]=temp;

            }

        }

    }

}

2） run.cpp

#include "Fraction.h"

int main(){

    int exit=0; //标记退出计算器

    while(1){

            if(exit){

                cout << "~结束运行,感谢您对本程序的支持。" << endl;

                break;

            }

            cout << "请选择功能" << endl;

            cout << "1.分数计算" << endl;

            cout << "2.分数排序" << endl;

            cout << "输入其他值退出程序,现在请输入您需要使用的功能——:";

            int n=0; // 标记选择的功能

            cin >> n;

            int flag=0; // 在排序功能需要跳出时，标记为1

            switch(n){

                case 1:

                        while(1){

                            cout << "请输入分数计算式（如 1/2+3/4 回车），输入 # 号返回上一级目录" << endl;

                            char op='0';

                            Fraction f1,f2;

                            cin >> f1 ;

                            if(f1.getDeno()==20192242){

                                cout << "~退出计算功能" << endl;

                                string s;

                                getline(cin,s); // 对该行后续输入进行处理，使其不影响下次输入

                                break;

                            }

                            cin >> op >> f2;

                            // cout << "检查操作数:" << f1 << " " << f2 << endl;

                            if(f1.getDeno()==0 || f2.getDeno()==0){

                                cout << "warning:输入分数不合分数规范！" << endl;

                                string s;

                                getline(cin,s); // 对该行后续输入进行处理，使其不影响下次输入

                                continue;

                            }

                            int func=0;     // 看op是否属于加减乘除四种情况

                            switch(op){

                                case '+':

                                        cout << "=" << f1+f2 << endl;

                                        func=1;

                                        break;

                                case '-':

                                        cout << "=" << f1-f2 << endl;

                                        func=1;

                                        break;

                                case '\*':

                                        cout << "=" << f1\*f2 << endl;

                                        func=1;

                                        break;

                                case '/':

                                        cout << "=" << f1/f2 << endl;

                                        func=1;

                                        break;

                            }

                            if(!func ){     // 如果func值未改变，说明op输入不合格

                                cout << "warning:分数计算符号输入错误！" << endl;

                            }

                        }

                        break;

                case 2:

                        flag=0;

                        while(1){

                            if(flag){

                                cout << "~退出排序功能" << endl;

                                break;

                            }

                            cout << "请输入一组分数，用逗号隔开，如需由小到大排序用符号<结尾，由大到小排序用符号>结尾（如1/2,1/4,3/5<回车），输入#号键返回上一层目录：" << endl;

                            vector<Fraction> a;     // 使用vector可以动态添加输入的分数，并且可以很快地得到长度。

                            char op;                // 记录输入的分割符号

                            Fraction f;

                            while(1){

                                cin >> f;

                                if(f.getDeno()==20192242){

                                    flag=1;     // 说明输入了#，跳出

                                    break;

                                }

                                if(f.getDeno()==0){

                                    cout << "warning:输入出错!本次输入结束。" << endl;

                                    string s;

                                    getline(cin,s); // 这一行的后续数字应该被处理掉

                                    break;

                                }

                                a.push\_back(f);

                                cin >> op;

                                if(op==',')

                                    continue;

                                else if(op=='<'){

                                    cout << "从小到大排序:";

                                    int len=a.size();           // 得到fraction个数，建立数组

                                    Fraction b[len];            // 将vector转化为fractor数组，然后就可以调用自己在类中实现的排序方法。

                                    for(int i=0;i<len;i++){

                                        b[i]=a[i];

                                    }

                                    sortFraction(b,len,1);      // 调用排序函数，flag设置为1，顺序排序

                                    for(int i=0;i<len;i++){     // 输出结果

                                        cout << b[i] << " ";

                                    }

                                    cout << endl;

                                    break;

                                }else if(op=='>'){

                                    cout << "从大到小倒排:";

                                    int len=a.size();

                                    Fraction b[len];

                                    for(int i=0;i<len;i++){

                                        b[i]=a[i];

                                    }

                                    sortFraction(b,len,2);      // 调用排序函数，flag设置为2，倒序排序

                                    for(int i=0;i<len;i++){

                                        cout << b[i] << " ";

                                    }

                                    cout << endl;

                                    break;

                                }else{

                                    cout << "输入出错!" << endl;

                                    string s;

                                    getline(cin,s); // 这一行的后续数字应该被处理掉

                                    break;

                                }

                            }

                        }

                        break;

                default:

                        exit=1;

                        break;

            }

        }

    return 0;

}