**GLUT ACM Practice Summary**

**班级:2016级计算机6班 2017年11月6日**

**姓名:杨易 学号:3162052051639**

**实验题目:POJ：分数加减法**

This summary was auto generated by python, you could found it at [**https://github.com/cthulhujk/glut\_acm\_practice**](https://github.com/cthulhujk/glut_acm_practice).

**1.Lab Topic**

* POJ：分数加减法

编写一个C程序，实现两个分数的加减法

**2.Requirement Analysis**

输入包含多行数据。每行数据是一个字符串，格式是"a/boc/d"。 。其中a, b, c, d是一个0-9的整数。o是运算符"+"或者"-"。数据以EOF结束

**3.APIs Design**

///////////////////////////////////////////////////////////////////////

/// Public Interface: Note that these interfaces were even not actually

/// APIs which could be called by other functions.

///////////////////////////////////////////////////////////////////////

// @summary 计算 a op b

// @param int a

// @param int b

// @return int 返回a op b

int doComputing(int a, int b, char op);

// @summary 给出两个数，寻找这两个数的最大公因数

// @param int a 数A

// @param int b 数B

// @return int A和B的最大公因数

int findGreatestCommonFactor(int a, int b);

// @summary 将分数转化为以最大公因数为底的分数形式

// @param int a 分子

// @param int b 分母

// @param int greatestCommonFactor 最大公因数

// @return pair<int,int> 以最大公因数为底的分数

std::pair<int, int> transformFraction(int a, int b, int greastestCommonFactor);

// @summary 返回a/b化简后的记过

// @param int 分子

// @param int 分母

// @return pair<int,int> 化简后的a/b

std::pair<int,int> simplifyFraction(int a, int b);

**4.Detail Design**

循环读取用户输入，直到EOF停止。首先使用parseNumberFrom()函数解析字符串，然后使用transformFraction()转化为pair对表示分数，最后simplifyFraction()化简分数计算后的最终结果。该题主要复杂度（代码）体现在parseNUmberFrom()

///////////////////////////////////////////////////////////////////////

/// Pseudocode implementations: implementation details of program

/// described by pseudocode. Its syntax was familiar to Python.

///////////////////////////////////////////////////////////////////////

while !cin.eof():

numbers[] = parseNumberFrom(UserInput)

fraction1 = transform(numbers1,numbers2)

fraction2 = transform(numbers3,numbers4)

resultFraction = fraction1 ['+'|'-'] fraction2

if(resultFraction<0){

print '-'.result.'\n'

}else if(resultFraction>0){

print result.'\n'

}else{

print '0'.'\n'

}

**5.Building and running**

I have A LOT free time to do uesless work. Obviously this project can easily use g++/clang compiled for all, but I still write a cmake rules to make it more engineered and modernization, even in previous you could see manual makefile. Notice that this project was licensed under [MIT License](https://github.com/cthulhujk/glut_acm_practice/blob/master/public/LICENSE), which means you can do almost everything you want with it. Anyway, I just want to make it more fun :-)

Prerequisite:

* unix make
* g++ version >= 4.8.4
* git
* cmake >= 3.5.0

For linux users:

$ git clone https://github.com/NagaseMinato/glut\_acm\_practice.git

$ cd glut\_acm\_practice/build

$ cmake .. -G"Unix Makefiles"

$ make -j8

$ ../bin/poj\_3979\_30percentAC # try to run a built program

$ make clean

There are more details about each programs in testsuites, you can run it to see more if you had been installed valgrindtoolchain:

$ sudo apt-get install valgrind

$ make test # run testsuite

For windows users:

> git clone https://github.com/NagaseMinato/glut\_acm\_practice.git

> cd glut\_acm\_practice/build

> cmake .. -G"Visual Studio 15 2017"

Then double-click glut\_acm\_practice.sln, select a sub project and build it, enjoy it!

**6.Test Output**

编写一个C程序，实现两个分数的加减法

**7.Appendix**

* All programs had been passed my own test cases, you can add/modify/delete testcases into corresponding test/\*.data file. For me, I eliminated all warnings and memory leaking using [valgrind](http://valgrind.org/)
* For more information about how to run these test, see # 5.Usage section

Test project /home/racaljk/Desktop/glut\_acm\_practice/build

Start 1: test\_running\_\_01pack\_v1.0

1/16 Test #1: test\_running\_\_01pack\_v1.0 ............ Passed 0.01 sec

Start 2: test\_running\_\_8queen

2/16 Test #2: test\_running\_\_8queen ................. Passed 0.00 sec

Start 3: test\_running\_\_poj3299\_WA

3/16 Test #3: test\_running\_\_poj3299\_WA ............. Passed 0.00 sec

Start 4: test\_running\_\_poj\_1007\_40percentAC

4/16 Test #4: test\_running\_\_poj\_1007\_40percentAC ... Passed 0.01 sec

Start 5: test\_running\_\_poj\_1590\_33percentAC

5/16 Test #5: test\_running\_\_poj\_1590\_33percentAC ... Passed 0.00 sec

Start 6: test\_running\_\_poj\_2386\_49percentAC

6/16 Test #6: test\_running\_\_poj\_2386\_49percentAC ... Passed 0.00 sec

Start 7: test\_running\_\_poj\_2538\_47percentAC

7/16 Test #7: test\_running\_\_poj\_2538\_47percentAC ... Passed 0.00 sec

Start 8: test\_running\_\_poj\_3278\_31percentAC

8/16 Test #8: test\_running\_\_poj\_3278\_31percentAC ... Passed 0.00 sec

Start 9: test\_running\_\_poj\_3979\_30percentAC

9/16 Test #9: test\_running\_\_poj\_3979\_30percentAC ...\*\*\*Failed 0.13 sec

Start 10: test\_running\_\_solution1

10/16 Test #10: test\_running\_\_solution1 .............. Passed 0.00 sec

Start 11: test\_running\_\_solution2

11/16 Test #11: test\_running\_\_solution2 .............. Passed 0.00 sec

Start 12: test\_running\_\_solution3

12/16 Test #12: test\_running\_\_solution3 .............. Passed 0.00 sec

Start 13: test\_running\_\_solution4

13/16 Test #13: test\_running\_\_solution4 .............. Passed 0.00 sec

Start 14: test\_running\_\_solution5

14/16 Test #14: test\_running\_\_solution5 .............. Passed 0.00 sec

Start 15: test\_running\_\_solution6

15/16 Test #15: test\_running\_\_solution6 .............. Passed 0.00 sec

Start 16: test\_running\_\_solution8

16/16 Test #16: test\_running\_\_solution8 .............. Passed 0.00 sec

94% tests passed, 1 tests failed out of 16

Total Test time (real) = 0.19 sec