Project 1 Written Report

Introduction

This program was overall an interesting task. Upon receiving the assignment, I thought it looked out of place in a data structures class. While this may still hold *some* truth, I have to say, I learned a lot more than I expected to from this project. In a general discussion, I was happy that certain aspects of the program proved easier than others, because it was a nice break to have one module function perfectly with little effort after spending hours on another part.

Program Structure

I use four classes in this program:

* main – handles interaction between other classes and user, allows for testing.
  + Pulls data in from two different text files in order to test the program.
  + In the first case, testing is performed on the Fraction methods themselves, such as ADD, SUB, DIV, MUL, as well as the rest.
  + The second case pulls in data from my own text file to go through the process of adding different fractions to a list, and then test the outside functions (sum, sort, mean, median, and mode).
    - This case was two-fold, as it tested the outside functions as well as the internal functions of the Fraction class.
* Fraction – the actual fractions are stored in this class. This class also handles all modification of fractions.
  + Stores numerator and denominator values in m\_numerator and m\_denominator.
  + Handles all desired functions internally (excluding sum, sort, mean, etc.)
  + Stores the mixed integer value in m\_mixedN. This is used when printing out fractions. However, this is not used elsewhere as input from the file insists that the mixed integer value is read in.
* LinkedList – A modified version of the LinkedList class created for lab 1 & 2. This linked list is built to handle fractions as Node values.
  + Added functionality: Sort
    - The LinkedList can sort the fractions inside of it. I accomplished this by implementing bubble sort because I knew the data input for this program was small, so an O(n^2) algorithm was not a problem here. However, if this program was meant to deal with lists of much larger sizes, I would have implemented a more efficient algorithm. I mention below that the LinkedList I use here is still only singly list, so that was another factor.
  + Modified functionality: I implemented a deconstructor to manage memory effectively. Also, I modified the print() function to use the Fraction classes print() method.
* Node – A modified version of the Node class created for labs 1 & 2. This node holds a Fraction object instead of an integer object.
  + Modified functionality: The node no longer stores an integer, but a Fraction pointer. The node is still singly linked.

In a general discussion, I took memory management serious in this program. I am a big fan of pointers, and since we’re using C++, I’d be doing myself a disservice if I didn’t use them. You’ll find several delete statements and deconstructors which effectively clean up unused memory. No stack overflows here!

Testing

To begin, I setup a loop to pull all the data in from proj1.data.txt in order to test my methods. I modified the given data file to include several of my own tests, which ensured proper functionality in all cases, including edge cases. For example, I had to make sure correct valid input was always given, which if not, threw an exception. The data files proj1.data.txt & proj1.data2.txt output the following text:

Input: ADD 1/2 + 1/4 = 3/4

Input: ADD 1/2 + -1/4 = 1/4

Input: ADD -1/2 + -1/4 = -3/4

Input: ADD -1/2 + -1/4 = -3/4

Input: XADD 2/3 + 1/2 = 1 1/6

Input: XADD 1/2 + 1/4 = 3/4

Input: SUB 1/2 - 1/4 = 1/4

Input: SUB 1/4 - 1/2 = -1/4

Input: SUB -1/4 - -1/2 = 1/4

Input: MUL 2/3 \* 3/4 = 1/2

Input: MUL 7/24 \* 8/16 = 7/48

Input: DIV 2/3 / 2/1 = 4/3

Input: DIV 3/4 / 1/2 = 3/8

Input: DIV -2/3 / -2/1 = 4/3

Input: XDIV 9/2 / 3/2 = 6 3/4

Input: XDIV 2/3 / 2/9 = 4/27

Input: REC 3/4 inverted is 4/3

Input: RED 80/120 reduced is 2/3

Input: RED 75/75 reduced is 1/1

Input: MIX 100/17 mixed is 5 15/17

Input: UNM 5 3/4 unmixed is 23/4

Input: AMIX 2 1/3 + 3 3/4 = 5 1/12

Input: LESS 2/3 < 3/4 = True.

Input: LESS 2/3 < -3/4 = False.

Input: LESS -2/3 < -3/4 = False.

Input: ADD 1/5 + 2/5 = 3/5

Input: ADD -1/5 + -2/5 = -3/5

Input: ADD -1/5 + -2/5 = -3/5

Input: ADD -1/5 + 2/5 = 1/5

Input: ADD Run-time exception: 0 is not a valid denominator.

Input: XADD 10/1 + 3/4 = 10 3/4

Input: XADD -1/5 + 7/5 = 1 1/5

Input: SUB 1/2 - 3/4 = -1/4

Input: SUB 1/2 - 3/4 = -1/4

Input: SUB -1/10 - 23/43 = -273/430

Input: MUL 0/100 \* 50000/10 = 0/1

Input: MUL 10/1 \* 10/1 = 100/1

Input: DIV Run-time exception: 0 is not a valid denominator.

Input: DIV -2/3 / 2/5 = -4/15

Input: DIV -2/3 / -2/5 = 4/15

Input: XDIV 10/1 / 20/1 = 199 1/1

Input: XDIV 1/2 / 4/3 = 2/3

Input: REC -10/3 inverted is -3/10

Input: REC Run-time exception: 0 is not a valid denominator.

Input: REC 1000/1 inverted is 1/1000

Input: REC -10/1000 inverted is -100/1

Input: RED 10000/500 reduced is 20/1

Input: RED 400/20 reduced is 20/1

Input: RED 300/15 reduced is 20/1

Input: MIX 500/10 mixed is 49 1/1

Input: MIX -90/17 mixed is -5 5/17

Input: MIX 30/7 mixed is 4 2/7

Input: UNM 50 0/10 unmixed is 50/1

Input: UNM -4 12/17 unmixed is -80/17

Input: UNM 4 2/7 unmixed is 30/7

Input: AMIX 100 1/10 + 3 8/10 = 103 9/10

Input: AMIX -10 -3/9 + 11 2/3 = 1 1/3

Input: AMIX Invalid input exception. Sign should be paired with N, not fractions.

Input: LESS -1/10 < -100/90 = False.

Input: LESS 8/7 < 16/15 = False.

Input: LESS 14/3 < 12/5 = False.

Input: D2F -130.20148482 converted is -130 10074241/50000000

Input: D2F 54.209483 converted is 54 209483/1000000

Input: D2F .5 converted is 1/2

Input: D2F -.75 converted is -3/4

Function Testing

Run-time exception: 0 is not a valid denominator.

[1/2, 7/8, 67/89, -6/20, -4/13, -7/9, -14/23, -1/12, -1/12, 14/16]

Operation: SUM

List size: 10

Sum: 4033051/4789980

Operation: SORT

Sorted List:

[-7/9, -14/23, -4/13, -6/20, -1/12, -1/12, 1/2, 67/89, 7/8, 14/16]

Operation: MEAN

List size: 10

Mean (average): 4033051/47899800

Operation: MEDIAN

List size: 10

Median: -1/12

Operation: MODE

List size: 10

Mode(s): [-1/12, 14/16]

The list has been reset!

Run-time exception: 0 is not a valid denominator.

[12/2, 5439/812, -67/389, -6/20, -4/13, -7/9, -14/23, -1/12, -1/12, 14/16, 1/2, 7/8, 231/176, 1/14, 56/93, -90/18, -105/35, 1024/4098, 321/648, 4914/947, 3891/320, -931/439, -90/184, -294/873, 12/4, 342/1, 12/2, 12/2, 12/2]

Operation: SUM

List size: 29

Sum: 313582703/72042816

Operation: SORT

Sorted List:

[-90/18, -105/35, -931/439, -7/9, -14/23, -90/184, -294/873, -4/13, -6/20, -67/389, -1/12, -1/12, 1/14, 1024/4098, 321/648, 1/2, 56/93, 14/16, 7/8, 231/176, 12/4, 4914/947, 12/2, 12/2, 12/2, 12/2, 5439/812, 3891/320, 342/1]

Operation: MEAN

List size: 29

Mean (average): 2076733913/1275716032

Operation: MEDIAN

List size: 29

Median: 321/648

Operation: MODE

List size: 29

Mode(s): [12/2]

The list has been reset!

[1/1, 6/2, 5/92, -9/32, -34/89]

Operation: SUM

List size: 5

Sum: 222129/65504

Operation: SORT

Sorted List:

[-34/89, -9/32, 5/92, 1/1, 6/2]

Operation: MEAN

List size: 5

Mean (average): 222129/327520

Operation: MEDIAN

List size: 5

Median: 5/92

Operation: MODE

List size: 5

Mode(s): [-34/89, -9/32, 5/92, 1/1, 6/2]

The list has been reset!

[1/1, 2/1, 3/1, 1/1, 2/1, 3/1, 1/1]

Operation: SUM

List size: 7

Sum: 13/1

Operation: SORT

Sorted List:

[1/1, 1/1, 1/1, 2/1, 2/1, 3/1, 3/1]

Operation: MEAN

List size: 7

Mean (average): 13/7

Operation: MEDIAN

List size: 7

Median: 2/1

Operation: MODE

List size: 7

Mode(s): [1/1]

The list has been reset!

[1/1, 2/1, 3/1, 4/1, 1/1, 2/1]

Operation: SUM

List size: 6

Sum: 13/1

Operation: SORT

Sorted List:

[1/1, 1/1, 2/1, 2/1, 3/1, 4/1]

Operation: MEAN

List size: 6

Mean (average): 13/6

Operation: MEDIAN

List size: 6

Median: 2/1

Operation: MODE

List size: 6

Mode(s): [1/1, 2/1]

The list has been reset!

[1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, -1/2, -1/3, -1/4, -1/5, -1/6, -1/7, -1/8, -1/9, -1/10]

Operation: SUM

List size: 18

Sum: 0/1

Operation: SORT

Sorted List:

[-1/2, -1/3, -1/4, -1/5, -1/6, -1/7, -1/8, -1/9, -1/10, 1/10, 1/9, 1/8, 1/7, 1/6, 1/5, 1/4, 1/3, 1/2]

Operation: MEAN

List size: 18

Mean (average): 0/1

Operation: MEDIAN

List size: 18

Median: 0/1

Operation: MODE

List size: 18

Mode(s): [-1/2, -1/3, -1/4, -1/5, -1/6, -1/7, -1/8, -1/9, -1/10, 1/10, 1/9, 1/8, 1/7, 1/6, 1/5, 1/4, 1/3, 1/2]

The list has been reset!

[56/103]

Operation: SUM

List size: 1

Sum: 56/103

Operation: SORT

Sorted List:

[56/103]

Operation: MEAN

List size: 1

Mean (average): 56/103

Operation: MEDIAN

List size: 1

Median: 56/103

Operation: MODE

List size: 1

Mode(s): [56/103]

The list has been reset!

[1/2, 3/4]

Operation: SUM

List size: 2

Sum: 5/4

Operation: SORT

Sorted List:

[1/2, 3/4]

Operation: MEAN

List size: 2

Mean (average): 5/8

Operation: MEDIAN

List size: 2

Median: 5/8

Operation: MODE

List size: 2

Mode(s): [1/2, 3/4]

The list has been reset!

[-1/23, 3/4, 78/14]

Operation: SUM

List size: 3

Sum: 4043/644

Operation: SORT

Sorted List:

[-1/23, 3/4, 78/14]

Operation: MEAN

List size: 3

Mean (average): 4043/1932

Operation: MEDIAN

List size: 3

Median: 3/4

Operation: MODE

List size: 3

Mode(s): [-1/23, 3/4, 78/14]

Looking at the Output

If we look at this output, we can see that valid input is checked rigorously, and signed fractions are handled accordingly. In order to guarantee the output was credible, I printed out the list each time before performing operations on it. View each function’s discussion below:

* SUM: After checking each of the five different tests it’s clear that the sum operation is fully functional. The given fraction is even more exact than my TI-84 can calculate, as the TI-84 stores the result as a double, which essentially loses the ability to recall a fraction if the fraction was complex enough.
* SORT: The sort method is easy to check, and on the six times it was tested with different lists, all lists were sorted correctly, as shown. I also made sure to account for the edge cases of small list sizes.
* MEAN: Testing the mean function was probably the easiest of the bunch, I simply performed the same process as summing the numbers, and then divided the summed fraction by a fraction that was equal to the size of the list, which produced the average. Again, I checked all six of my tests with my calculator and the means were correct every time.
* MEDIAN: This function was also easy to test. I had already written the sort function, so all I had to do was take the middle of the list (or average the middle two values). There’s not much to discuss here, other than the fact that upon inspect, lists with odd numbered sizes give a fraction in the middle of the list, and lists with even numbered sizes give the average of the two middle values, which is expected.
  + Update: I talk about this in problems encountered, but it’s worth mentioning here that I failed to test this for odd numbered lists adequately, so I added several more test cases with odd sized lists in order to completely test this function among others, which allowed me to fix a bug in the earlier version of the project.
* MODE: The mode function took the longest to code because of an error with my Node class. I had forgotten to make a new pointer to the fraction upon initialization, so when I would delete fractions, the fraction inside of the node would also be deleted, which was not wanted. I tested this by adding repeat values to the lists on several occasions. You can see that on several occasions I had two or three modes (or n nodes, if no repeat elements) in the list. This was reported and confirmed upon inspection of the list.
  + I also used the test cases provided on the assignment sheet to test my function (for example, the list 1 2 3 1 2 3 1 should have mode of 1.)

Problems Encountered

As with any other project, I encountered several problems. The most notable, which I described above, involved my Node class constructor. When initializing a Node, the input parameter was a pointer to a Fraction object. However, I was simply storing the value of the pointer in the Node, instead of creating a new Fraction object. I noticed this problem only in the MODE function, as I deleted a list of stored fractions if a larger sized mode was found. However, when I found a larger node, the list was compromised because of this.

Next, I had a slight error in my MEDIAN function, which is the reason I have resubmitted this project. My median function looked for the middle of the list by dividing the list size by two. The method then stored the value at the middle (size / 2), and the one to the right of it. If the list size was even, the second value was used. If not, only the first was used. However, this was incorrect because the way I calculated this was returning the value to the left of the middle when list sizes were odd. I was able to fix this by switching the two value retrievals around. With this new strategy, I would grab the (size / 2) + 1 index on odd numbered lists, and the (size / 2) and (size / 2) + 1 indexes on even numbered lists, which I would then average. I also added three more tests cases in order to check this. Including this, I also didn’t account for lists of size 1, so the result was some strange number with a denominator of 0. I was able to fix this by adding an outside check inside this method.

In a general discussion, I wouldn’t say I encountered many other “problems” as much as I encountered things I didn’t know how to do. While a good part of this program was busy work, I also spent a few hours simply reading up on concepts and strategies so I could write my own implementation.

In terms of the future, I would like to handle the interaction between the program and data files in a more elegant manner. Right now, my code seems bulky in this section of main.cpp and while it may not be as inefficient as I think it is, it could be improved.

Conclusion

I believe that the testing I have performed here demonstrates full functionality of the program. I have included a README text file with instructions on how to perform you own tests on the outside functions, and if you have any problems, please feel free to contact me by email: [jayofferdahl@ku.edu](mailto:jayofferdahl@ku.edu), or by cell phone: (573) – 673 – 5212.