Ricardo Deodutt
Two Sum Project

There will be 2 codes attached to this Two Sum Project. In the first code, there will be a file named “rng2file.c”. The main objective of this code is to create a .dat file named “BatchOfRNG.dat” and in that file there will be a specified amount of randomly generated numbers. The “rng2file.c” randomly generates numbers between [-1000 to 1000] and then prints the numbers out. I created this code separately from the main Two Sum Project because I felt I could use this code in another project incase I needed it. The annoying parts about creating this file in another file is that, I would need to compile it and run it before I run the main Two Sum Project code in order to keep the numbers random.

With that being said, the other code attached is the main Two Sum Project code. The main perks of my code is, it reads from a file of numbers. In my attached code, it has comments in that that explain most of the code. Essentially my code starts off with initializing the clock() function. The starting clock is then triggered. The clock is needed at the start so it can accurately time the program. I then initialized my “preDeterminedValue”, which will be 0. This is basically the sum (of the two numbers being added) the program will be looking for. Once the program sees that one sum equals to 0, it will trigger a printf function and that will print right next to the equation “The sum met a pre-determined value.”.

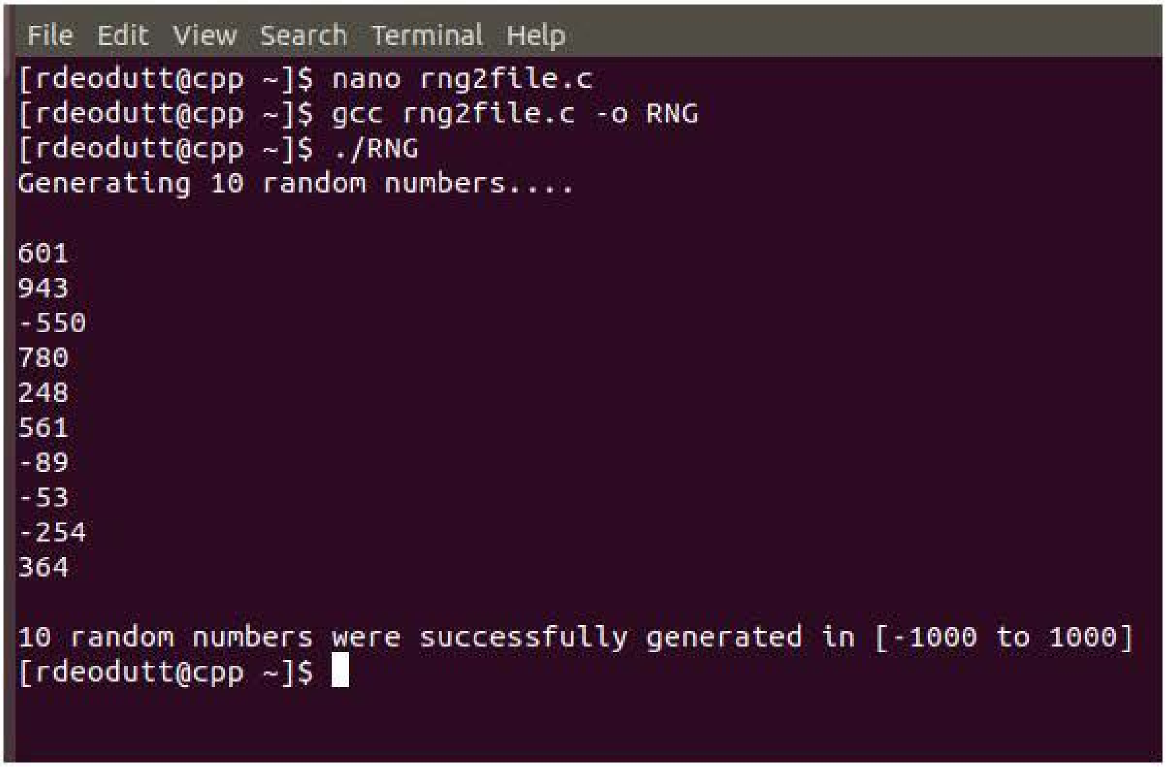
After that, my program will read the file that has a bunch of random numbers. There is a case where if the file cannot be located, it will say “Could not open file.”. The program will read the file of numbers and store them into an array. My code is designed so in the future if I need to change the quantity it would simple. For example, If I wanted to test 1000 numbers I would go to the top of my code and replace “#define LIMIT 10” to “#define LIMIT 1000”. My “rng2file.c” code also uses the same design. Whenever I need to test a different amount of numbers, I just go into the code and change the LIMIT at the top and it changes it for all parts of the code. If this wasn't there it would be time consuming to change the value everywhere in the code. The numbers are stored into the array using fscanf. After it is stored, I felt it was necessary to print out all the numbers that was stored.

After the numbers are printed out onto the console, the program then does the computation and prints out the equation for the user. For example if two numbers was 2 and 3 in the file of numbers, the program will output "2 + 3 = 5" onto the console. In the code, A is the first number and B is the second number being added. The bad part about having the program print out all the computation is that for longer test cases, the program will take longer. The program also checks if the sum of two numbers equal the pre-determined value here. There are some "printf("\n");" statements there for aesthetic purposes. This will make the program look much clean for the users. Once all numbers are ran through, the program stops reading the file and closes it.

The clock() is then triggered again to record the time stamp of the end. I then calculated the difference between the start and end of the clock and divided it with "POST_CLOCKS_PER_SEC". I don't need to initialize this because at the top of the program, "#include <time.h>" defines it for me. Once the difference of time is calculated, the program ends printing out the number of entries and execution time. The number of entries the the amount of numbers in the file. The execution time says "%2.15lf". This basically means the program will display up to 15 decimal places. This program was pretty fun to create because I finally got to learn C. Even though I knew C++, I felt it was easier to write in C then C++. This program was also a great assignment because it was not that hard but it was challenging. I thought it was going to be a simple but I had to try in this program. I also felt it was easier to write this on a windows computer using Dev C++ and then FileZilla'ing the program over to the server. It was much easier on my end.

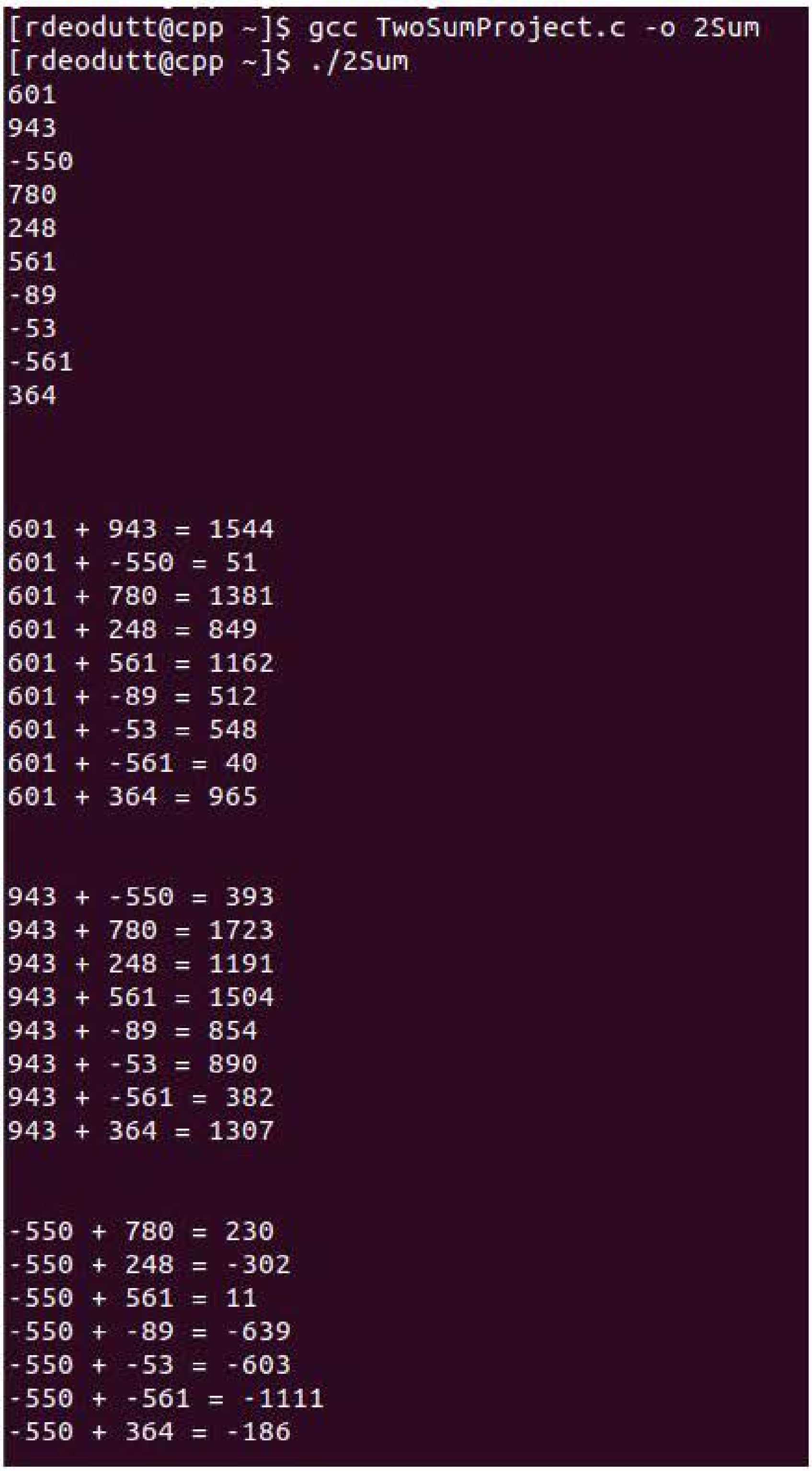
Test Cases

Sample of how rng2file.c works. This is how 10 numbers randomly generated looks like.

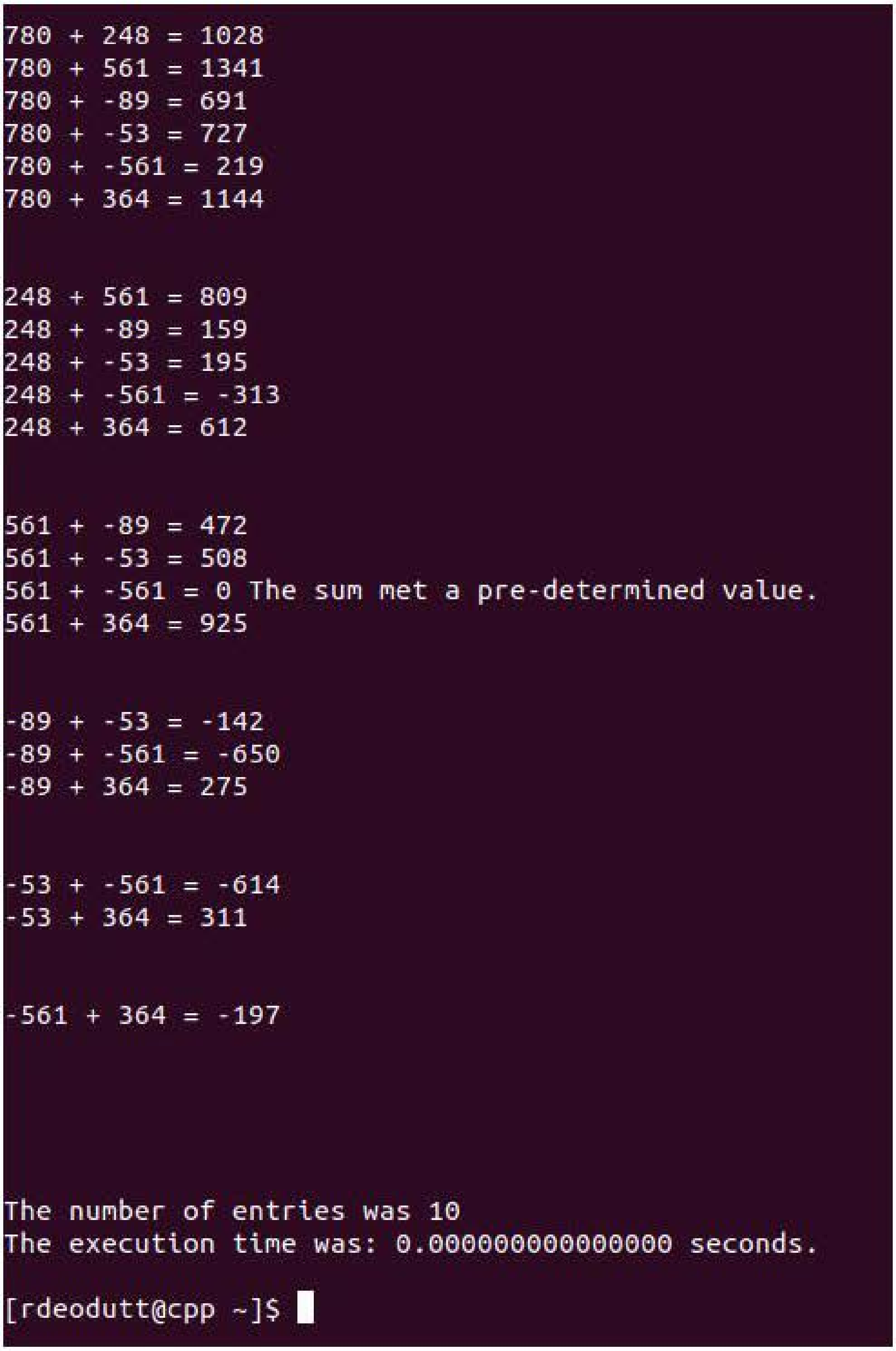


```
[rdeodutt@cpp ~]$ nano rng2file.c
[rdeodutt@cpp ~]$ gcc rng2file.c -o RNG
[rdeodutt@cpp ~]$ ./RNG
Generating 10 random numbers....

601
943
-550
780
248
561
-89
-53
-254
364

10 random numbers were successfully generated in [-1000 to 1000]
[rdeodutt@cpp ~]$
```

10 Numbers



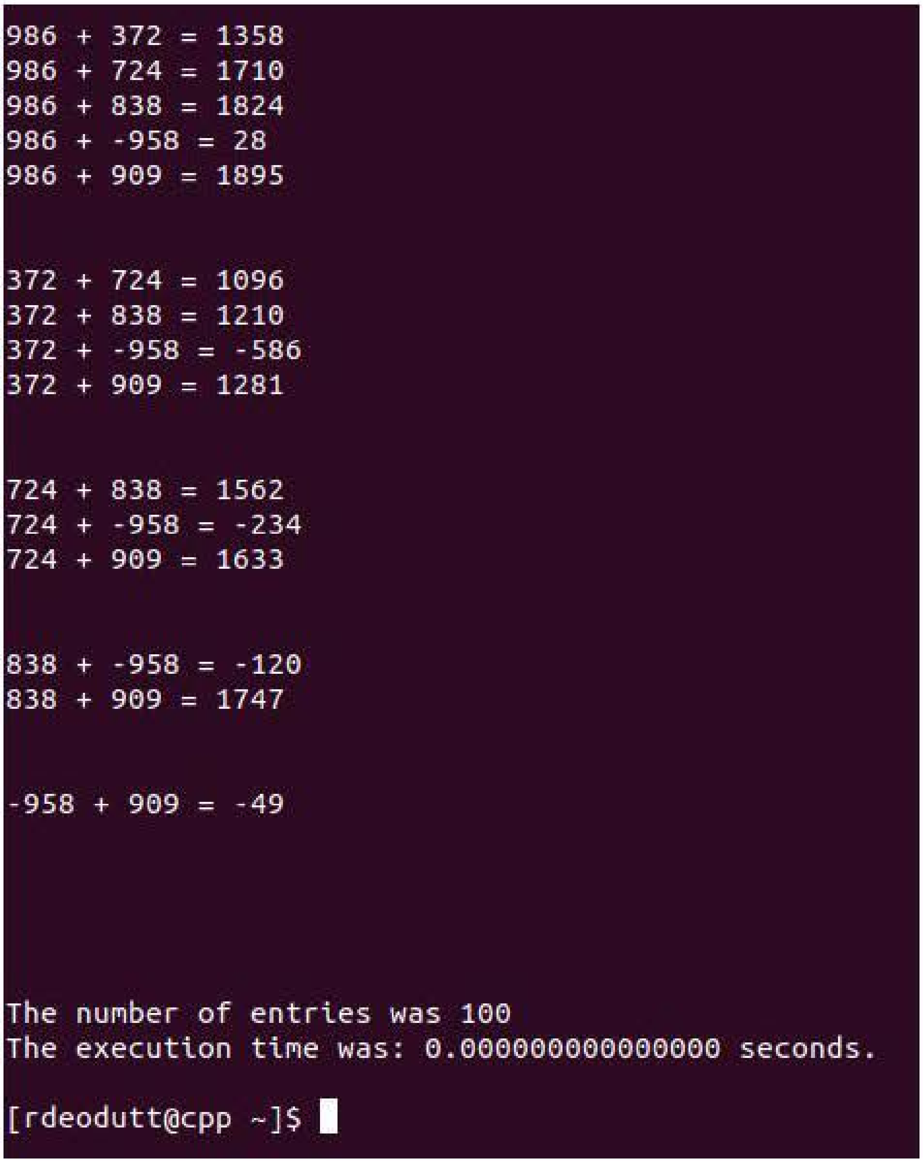
```
[rdeodutt@cpp ~]$ gcc TwoSumProject.c -o 2Sum
[rdeodutt@cpp ~]$ ./2Sum
601
943
-550
780
248
561
-89
-53
-561
364



601 + 943 = 1544
601 + -550 = 51
601 + 780 = 1381
601 + 248 = 849
601 + 561 = 1162
601 + -89 = 512
601 + -53 = 548
601 + -561 = 40
601 + 364 = 965


943 + -550 = 393
943 + 780 = 1723
943 + 248 = 1191
943 + 561 = 1504
943 + -89 = 854
943 + -53 = 890
943 + -561 = 382
943 + 364 = 1307


-550 + 780 = 230
-550 + 248 = -302
-550 + 561 = 11
-550 + -89 = -639
-550 + -53 = -603
-550 + -561 = -1111
-550 + 364 = -186
```



```
780 + 248 = 1028
780 + 561 = 1341
780 + -89 = 691
780 + -53 = 727
780 + -561 = 219
780 + 364 = 1144


248 + 561 = 809
248 + -89 = 159
248 + -53 = 195
248 + -561 = -313
248 + 364 = 612


561 + -89 = 472
561 + -53 = 508
561 + -561 = 0 The sum met a pre-determined value.
561 + 364 = 925


-89 + -53 = -142
-89 + -561 = -650
-89 + 364 = 275


-53 + -561 = -614
-53 + 364 = 311


-561 + 364 = -197





The number of entries was 10
The execution time was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
```

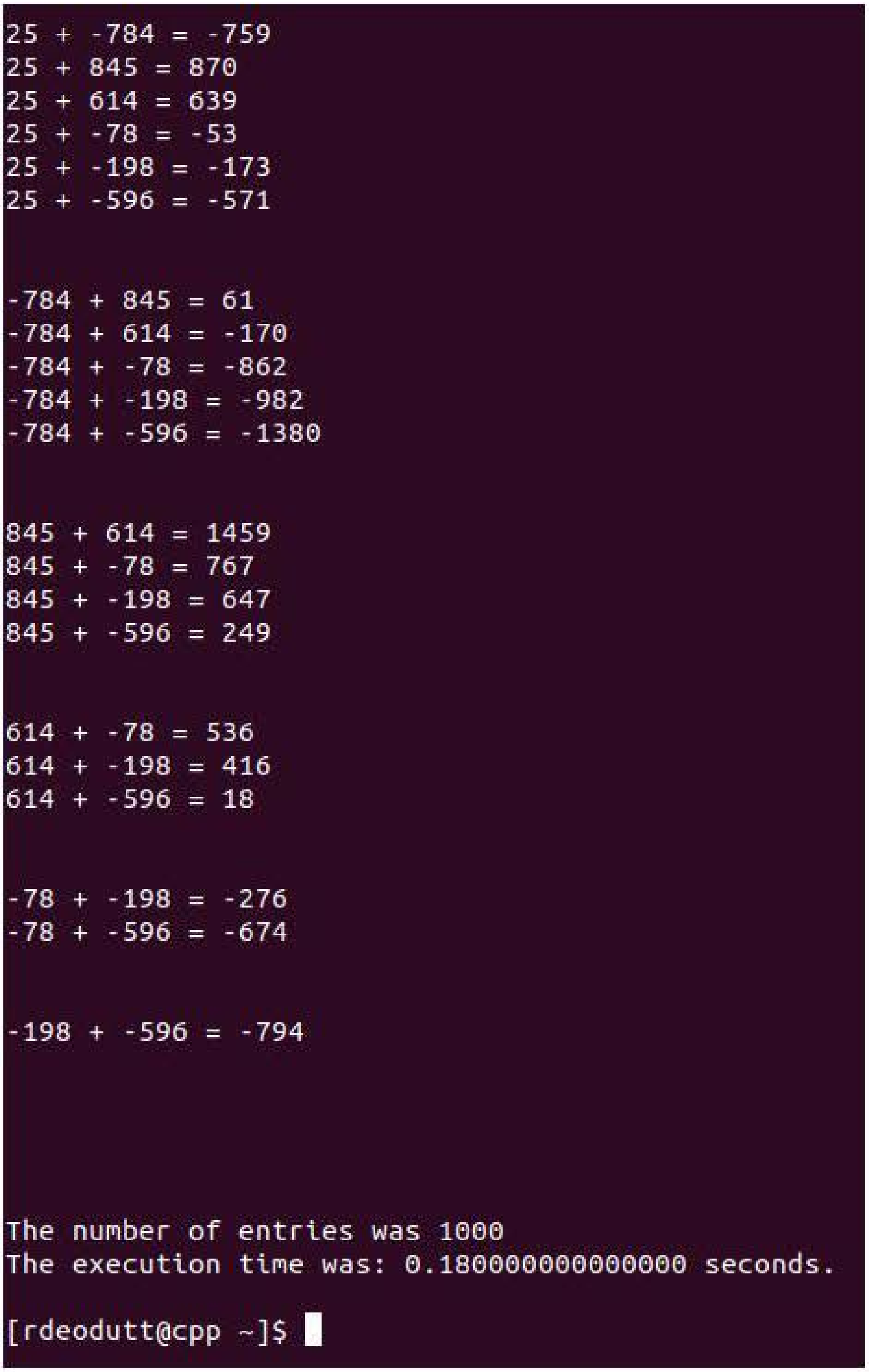
This is the only time I will be showing the entire console that the program printouts. After 10 numbers it gets pretty chaotic to show all the numbers. After this example I will just show execution time. The reason why this execution time says 0.0000… is because 10 numbers is not a large amount of numbers for this program. Also the server runs these codes fast. Once the numbers start to increase, the execution time will start to show but that will not happen until around 5000 numbers. In this example, some of the random numbers being added equals to the pre-determined value and the program prints out a message next to each of those equations.

100 Numbers



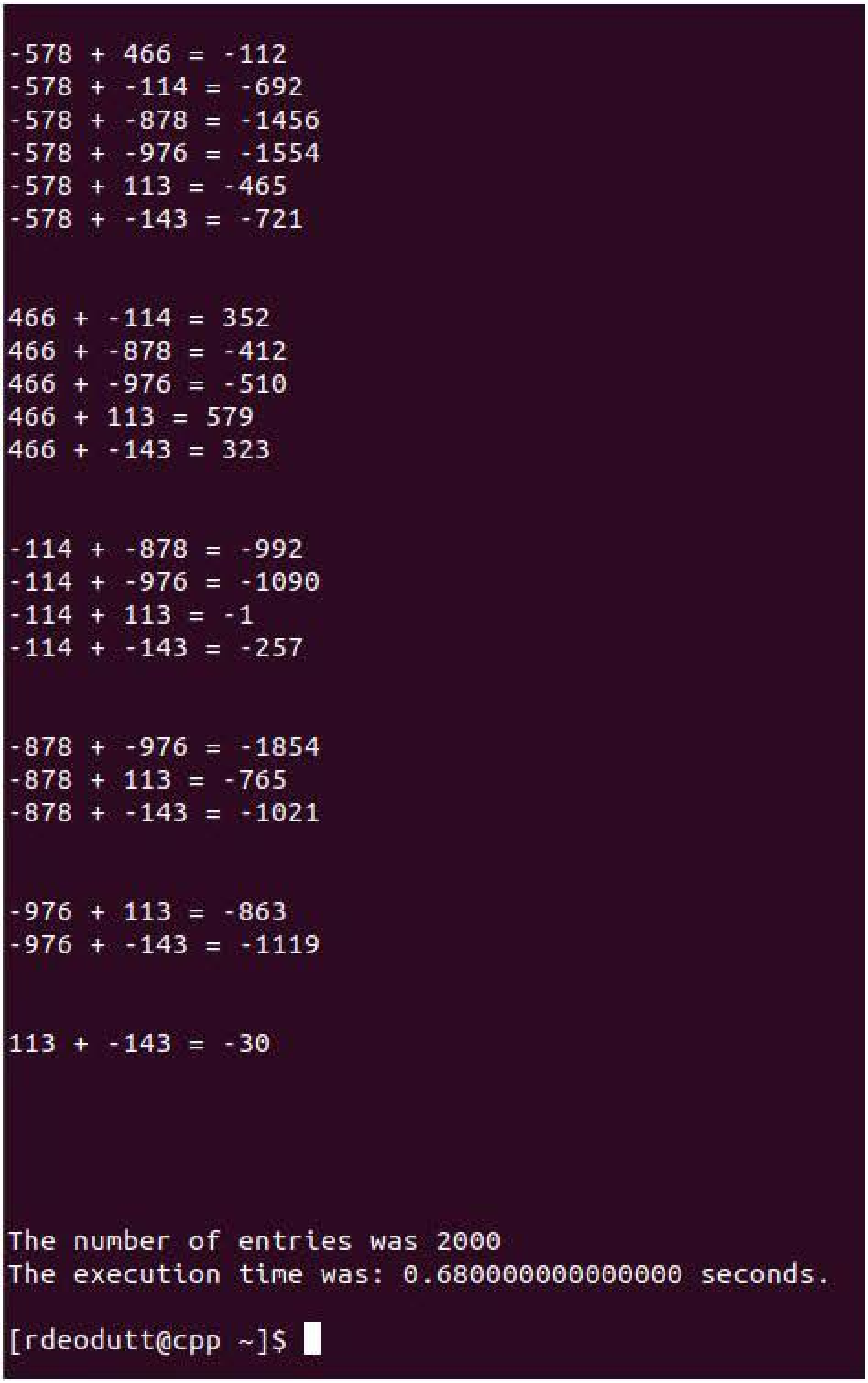
```
986 + 372 = 1358
986 + 724 = 1710
986 + 838 = 1824
986 + -958 = 28
986 + 909 = 1895


372 + 724 = 1096
372 + 838 = 1210
372 + -958 = -586
372 + 909 = 1281


724 + 838 = 1562
724 + -958 = -234
724 + 909 = 1633


838 + -958 = -120
838 + 909 = 1747


-958 + 909 = -49





The number of entries was 100
The execution time was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
```

1,000 Numbers



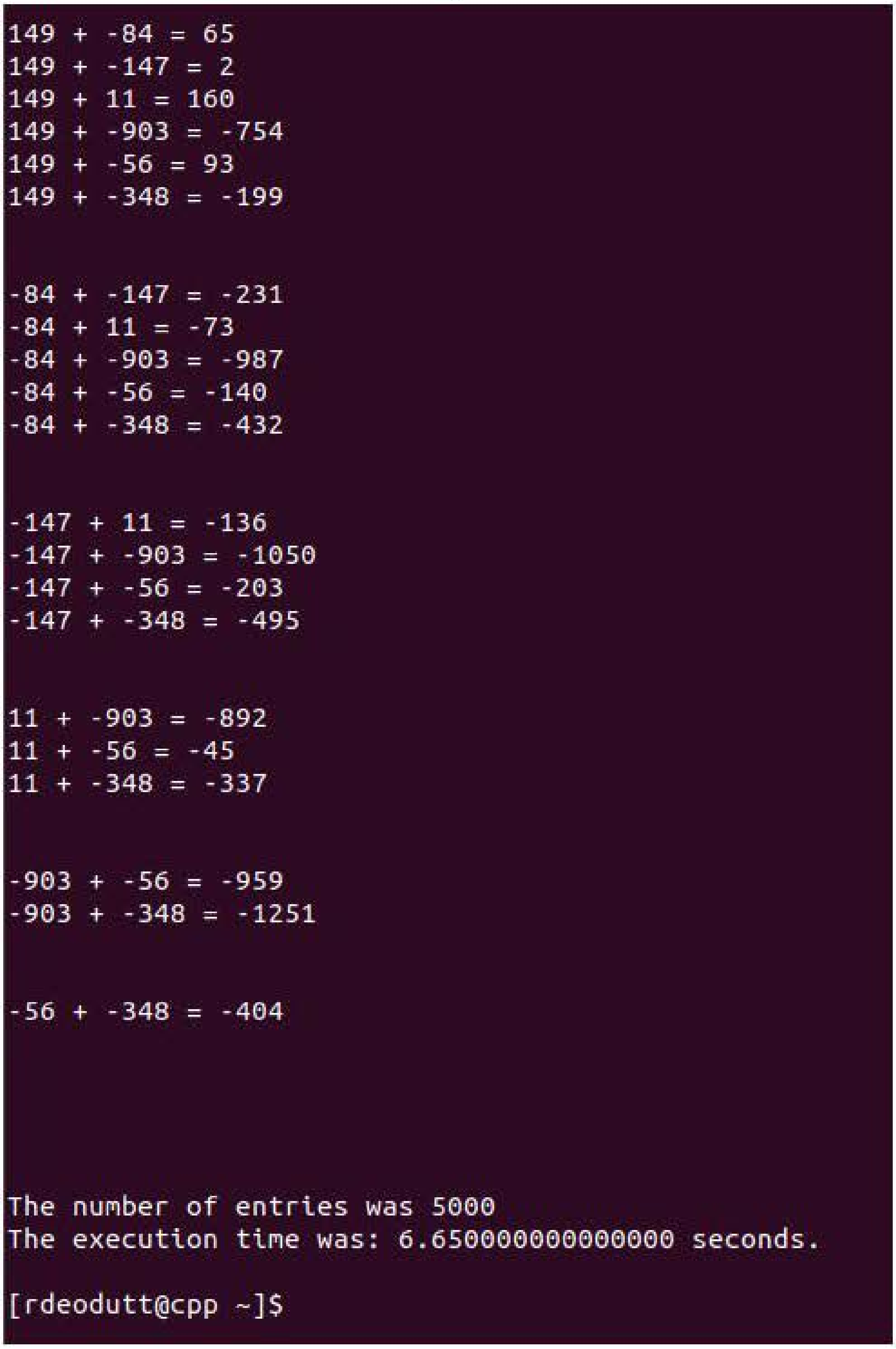
```
25 + -784 = -759
25 + 845 = 870
25 + 614 = 639
25 + -78 = -53
25 + -198 = -173
25 + -596 = -571


-784 + 845 = 61
-784 + 614 = -170
-784 + -78 = -862
-784 + -198 = -982
-784 + -596 = -1380


845 + 614 = 1459
845 + -78 = 767
845 + -198 = 647
845 + -596 = 249


614 + -78 = 536
614 + -198 = 416
614 + -596 = 18


-78 + -198 = -276
-78 + -596 = -674


-198 + -596 = -794





The number of entries was 1000
The execution time was: 0.180000000000000 seconds.

[rdeodutt@cpp ~]$
```

2,000 Numbers



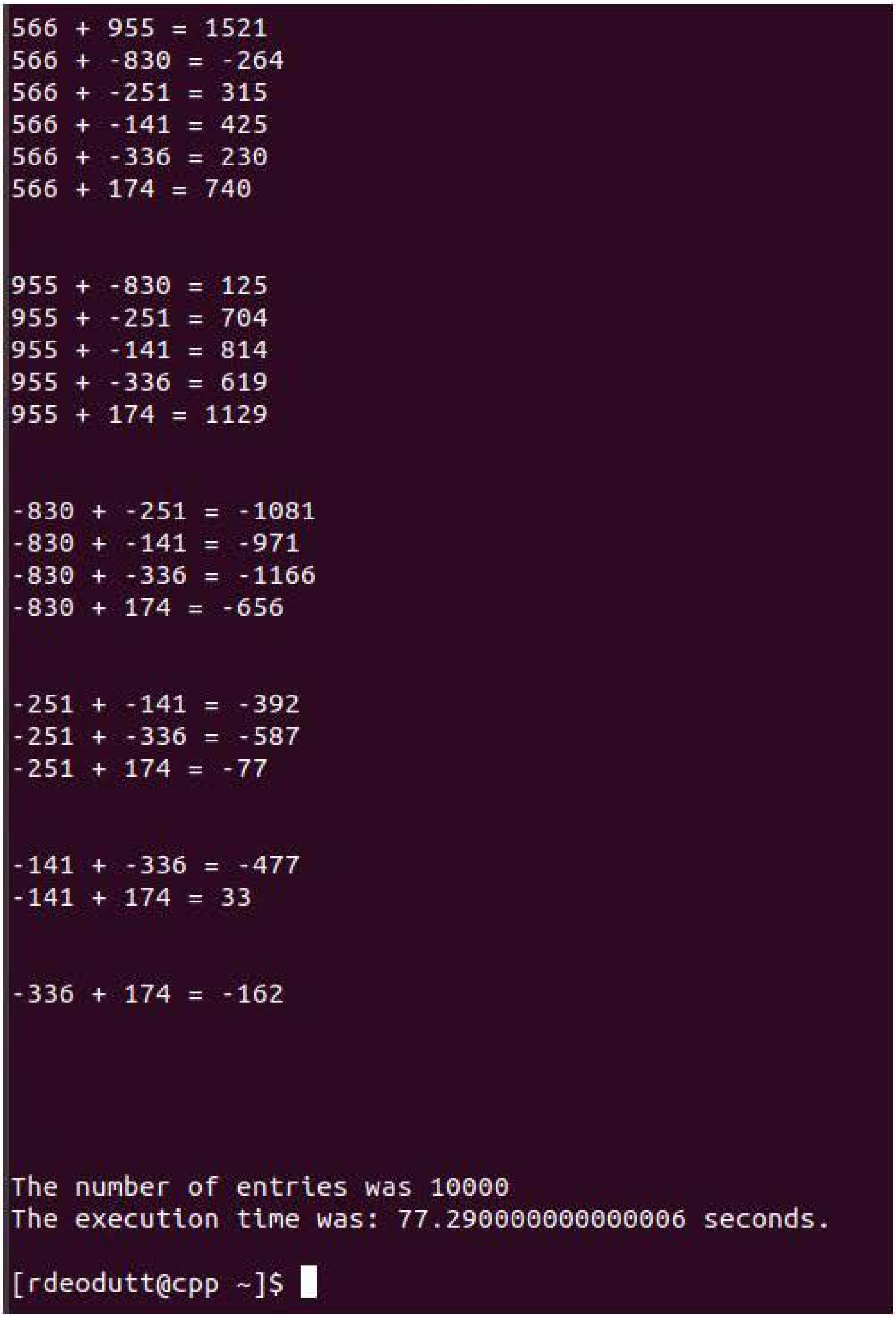
```
-578 + 466 = -112
-578 + -114 = -692
-578 + -878 = -1456
-578 + -976 = -1554
-578 + 113 = -465
-578 + -143 = -721


466 + -114 = 352
466 + -878 = -412
466 + -976 = -510
466 + 113 = 579
466 + -143 = 323


-114 + -878 = -992
-114 + -976 = -1090
-114 + 113 = -1
-114 + -143 = -257


-878 + -976 = -1854
-878 + 113 = -765
-878 + -143 = -1021


-976 + 113 = -863
-976 + -143 = -1119


113 + -143 = -30




The number of entries was 2000
The execution time was: 0.680000000000000 seconds.

[rdeodutt@cpp ~]$
```

5,000 Numbers



```
149 + -84 = 65
149 + -147 = 2
149 + 11 = 160
149 + -903 = -754
149 + -56 = 93
149 + -348 = -199


-84 + -147 = -231
-84 + 11 = -73
-84 + -903 = -987
-84 + -56 = -140
-84 + -348 = -432


-147 + 11 = -136
-147 + -903 = -1050
-147 + -56 = -203
-147 + -348 = -495


11 + -903 = -892
11 + -56 = -45
11 + -348 = -337


-903 + -56 = -959
-903 + -348 = -1251


-56 + -348 = -404





The number of entries was 5000
The execution time was: 6.650000000000000 seconds.

[rdeodutt@cpp ~]$
```

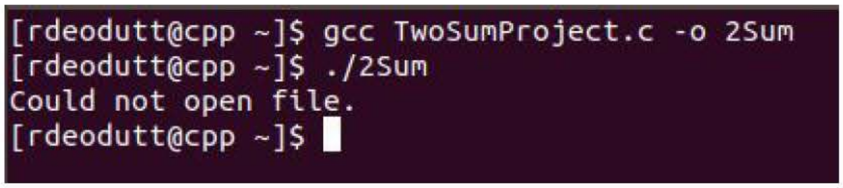
10,000 Numbers



```
566 + 955 = 1521
566 + -830 = -264
566 + -251 = 315
566 + -141 = 425
566 + -336 = 230
566 + 174 = 740


955 + -830 = 125
955 + -251 = 704
955 + -141 = 814
955 + -336 = 619
955 + 174 = 1129


-830 + -251 = -1081
-830 + -141 = -971
-830 + -336 = -1166
-830 + 174 = -656


-251 + -141 = -392
-251 + -336 = -587
-251 + 174 = -77


-141 + -336 = -477
-141 + 174 = 33


-336 + 174 = -162





The number of entries was 10000
The execution time was: 77.29000000000006 seconds.

[rdeodutt@cpp ~]$
```

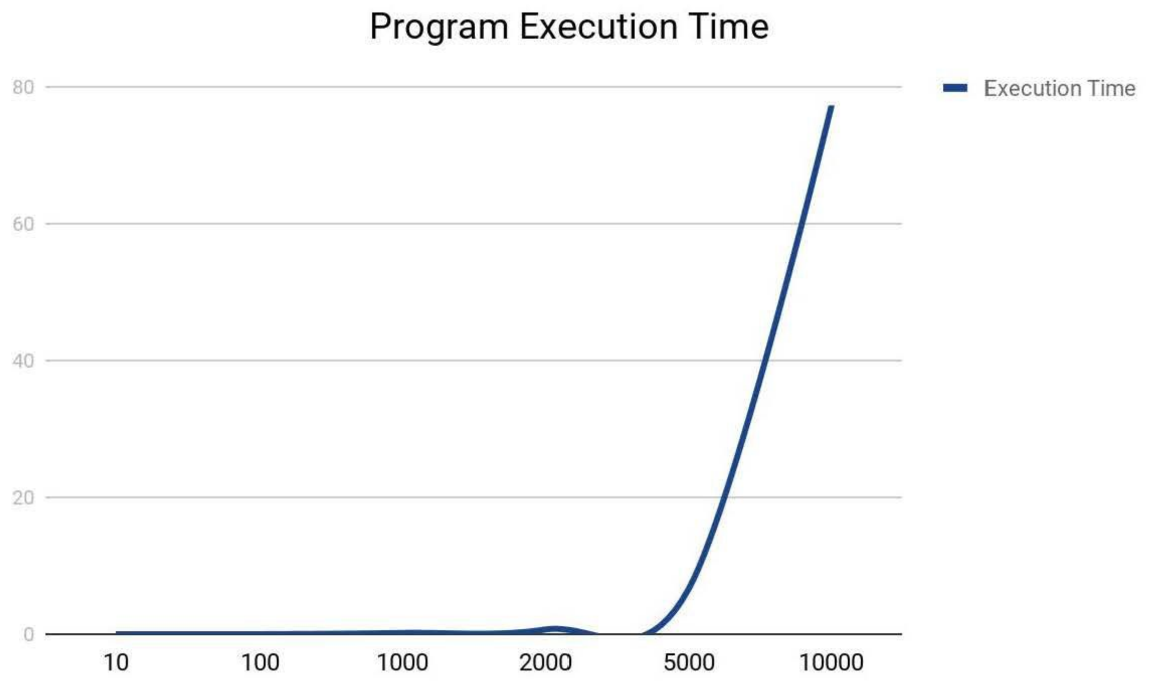
The reason why 10,000 numbers look like it took a small amount of time, is because the program finishes fast on the server and it was taking time printing out all the equations onto the screen for the user. It took approximately 3-5 minutes for the code to finish running on my end.

File Path not found. (or File name is wrong in code).



```
[rdeodutt@cpp ~]$ gcc TwoSumProject.c -o 2Sum
[rdeodutt@cpp ~]$ ./2Sum
Could not open file.
[rdeodutt@cpp ~]$ 
```

Those were all the test cases that I could think of while doing this write up report. Please look at the attached codes. There will be “rng2file.c” and “TwoSumProject.c”. Below is also a line graph to show how long the program took based off amount of numbers.


Program Execution Time
Execution Time