|  |  |  |  |
| --- | --- | --- | --- |
| **Input 1** | **Input 2** | **Sum (using calculator)** | **Sum (using program)** |
| 1 | 1 | 2 | 2 |
| 20 | 95 | 115 | 115 |
| 2529382928 | 438283832 | 2,967,666,760 | 2967666760 |
| 9999999999 | 1 | 10,000,000,000 | 10000000000 |
| 9223372036854775808 | 9223372036854775808 | 18,446,744,073,709,551,616 | 18446744073709551616 |
| 9223372036854775808 | 486127654835486515383218192 | 486,127,664,058,858,552,237,994,000 | 486127664058858552237994000 |
| 1486515347865138654821865 | 348621896435183186 | 1,486,515,696,487,035,090,005,051 | 1486515696487035090005051 |

**Part 1 (Problem 3): Addition of Very Large Numbers**

* The program that I have created can successfully find the sum of large numbers that cannot be generally stored in the native data types.
* This can be seen in the table above as the sum of input 1 and input 2 are the same if calculated using a calculator or if calculated using the program largeSum.cpp.

**Part 2 (Problem 3): Sorting Patient Records by Specific Category**

|  |  |
| --- | --- |
| **Sorted Category** | **Table** |
| Age |  |
| ID |  |
| Sex |  |
| Last Name |  |
| First Name |  |

* The program successfully sorts all categories accordingly.
* However, for 2D array categories that are ordered alphabetically, it only orders accordingly based on the first character in the string.
* The program does not look at any sequential characters to determine which word is alphabetically first.

**Part 2 (Problem 4): Tiebreaker with First Name Sorting**

|  |  |
| --- | --- |
| **Sorted Category** | **Table** |
| Age |  |
| ID |  |
| Sex |  |
| Last Name |  |
| First Name |  |

* The difference between Part 2 Problem 3 and Part 2 Problem 4 is seen when age and sex are sorted.
* For example, when sorting age lowest to greatest there were some values that were found multiple times such as 15 and 22.
* In Problem 3, there was no tiebreaker to sort values that were equal in the chosen category.
* However, in Problem 4 a tiebreaker was chosen. The tiebreaker was the persons first name, which ultimately organizes the code better.
* The code could be made more robust by adding a tiebreaker using a different category. This is because when sorting by first name, there are two people with the name Michael and thus we cannot use our first name tiebreaker.