Subtraction Calculator Report

My implementation of a subtraction calculator uses a Number class to store the int array that will hold the numbers, and the length of the number. The program begins by asking the user to enter two positive numbers, which are taken in as strings (eliminating the need to press enter after every digit for the user).

Input Validation

Each time the user enters a number, the program validates their input by calling upon 3 methods that check to make sure that the number isn't negative, that the number isn't too large (more than 50 digits), and that the input only involves valid digits. If the number fails any of these validation tests, the program will provide an appropriately informative error message and ask the user to input a new number. This number validation and error message process will continue until the user enters a number approved by all three methods.

Inserting String Number to Int Array

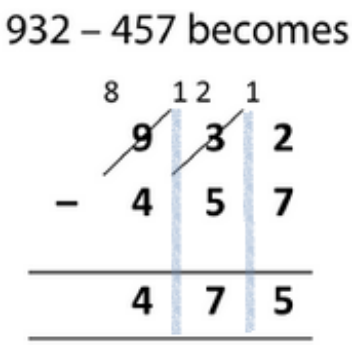
Once numbers are confirmed valid, the program will call methods to find and set the length of the number in a Number object, and to insert the number into the Number object's int array. It does this by subtracting the character '0' from each character in the number string (accessible in a string via the same method for accessing elements in a character array) and inserting the result into the int array. This results in the correct int values being inserted because the system uses the ASCII codes of characters (e.g. 49 for '1') in situations where a character is treated like a number. Subtracting '0' from a numerical character therefore produces the correctly corresponding range of digits 0 - 9 because they all have ASCII values that are also arranged together in an ascending range of ASCII codes, 48 – 57. For example, '3' – '0' is the same as 51 – 48 when replaced with their ASCII values, which equals 3.

Subtract Method Parameters

Once both Number objects are ready, the program calls a method to subtract one number from the other using long subtraction (see **Figure 1.** for an explanation). The simplest way to subtract very large numbers using long division where the second number is larger (i.e. the result will be negative) is to subtract the smaller number from the larger, and make the result negative. Considering this, the subtraction method takes in the two numbers as a "smaller" and a "larger" number, with a boolean variable to keep track of whether the larger number was originally the second input or not (i.e. the number to be subtracted). The program will therefore first call a method to check to see which number is larger, and then will call the subtract method with the numbers being placed into the "larger" and "smaller" parameters accordingly, and the Boolean parameter true for the second number being larger, or false that it wasn't.

Figure 1. Long subtraction:

Long subtraction is where two numbers are lined up, first number over the top of the second, to form columns of digits of matching unit value (see for an illustration of these columns). These matched digits then simply become smaller subtractions where the bottom (second number) digit is subtracted from the top (first number) digit, and the resulting digit written below them in the same unit-value column. If the bottom digit is larger than the top digit in the column the answer would be negative, so 1 is taken (i.e. "borrowed") from the adjacent higher-unit-value column's top digit, and added as 10 to the current column's top digit. This makes it larger than the current bottom digit and therefore allowing the easy subtraction of the bottom/second digit from the top.



Subtraction – "Lining up" Digits by Unit Value

Since all numbers are placed into arrays starting at index 0, numbers of different lengths do not line up according to their unit value (e.g. the digits at index 0 in each array won't be of the same unit value). Considering this, the subtract method first calculates any difference in length between the numbers, and subtracts this difference from any indexes used to reference the digits in the smaller number each time, to ensure the digit is of matching unit value with the larger number.

Subtraction – Borrowing and Subtracting

For each set of matched digits, the subtract method (beginning at the smallest unit value pair) goes through subtracting the smaller number's digit from the larger's, borrowing from higher unit value digits when necessary by subtracting one from the digit being borrowed from and adding 10 to the digit borrowing. Each answer is inserted into an answer array at the same index as the one used in the larger number for the given unit value (this results in leading 0s in answers that are smaller in length than the larger number used in the subtraction). Once all digit pairs have been subtracted, any digits left in the larger number without a matching digit to subtract are then simply inserted into the array in their appropriate indexes for their given unit values.

Negative Answers and Display

Once the answer is all inserted into the array, the program checks the Boolean parameter to see if the second number was originally the larger one, and if so, converts the first digit in the answer number to be negative, and returns the answer number. The main program will then display the answer to the user, and allow them the option to try doing another subtraction calculation, or to exit the program.

Testing

I employed a testing strategy that involved methodically going through and testing the function of each method written by entering input that tests the different paths or possible results for the given method, and checking that the program dealt with the input correctly by viewing the output on screen, or displaying the variables I am interested in seeing if they would not normally be displayed otherwise. For example, while testing a function that is intended to check whether a number is negative I would enter a positive number as input and then a negative number the next time. In each case, I would then make sure that the program correctly detected the sign of the number and reacted accordingly. Test results are detailed below including the inputs and outputs. All test yielded the expected result.

getNumber() – isNegative()

Input: *-123456789*

Result: *Sorry, that is a negative number! Please enter a positive number:*

getNumber() – isValidSize()

Input: *12345678901234567890123456789012345678901*
(51 digits – 1 out of bounds)

Result: *Sorry, the number you entered is too large! Please enter a number that is 50 digits or less:*

getNumber() – isValidNumber()

Input: *1234567890-*

Result: *Sorry, the number you entered contains invalid characters! Please take care as you type and try again:*

getNumber()

Input: 3 invalid numbers (e.g. the three above) consecutively

Result: Continues giving error messages and asking for a new number while they're not valid

getNumber()

Input: *12345678901234567890123456789012345678901234567890*
(50 digits – max within bounds)

Result: Number is accepted and the program moves on

getNumber()

Input: *123456*
(6 digits – much smaller number than max)

Result: Number is accepted and the program moves on

getNumber()

Input: *1*
(1 digit – smallest acceptable)

Result: Number is accepted and the program moves on

setWholeNumber()

Input: *1234567898765434567*

Result: Print Number object details to test/view contents:
Number Length: 19
Number: 1234567898765434567

firstIsLarger()

Input: First number: *1*
Second number: *10*

Result: Cout message depending on if the method returns true or false:
Second number found to be larger

firstIsLarger()

Input: First number: *10*
Second number: *1*

Result: Cout message depending on if the method returns true or false:
First number found to be larger

subtract2Nums()

Input: Situation: First number larger, different lengths (Example in brief)
First Number: *2398340958039458624408359834521849852983 58*
Second number: *939542309853120721934217021372984729812*

Result: *Answer: 2388945534940927417189017664308120005685 46* (Correct Answer)

subtract2Nums()

Input: Situation: Second number larger, different lengths (opposite of previous)
First Number: *939542309853120721934217021372984729812*
Second number: *2398340958039458624408359834521849852983 58*

Result: *Answer: -2388945534940927417189017664308120005685 46* (Correct Answer)

subtract2Nums()

Input: Situation: Exact same number, same lengths
First Number: *1234567890*
Second number: *1234567890*

Result: *Answer: 0* (Correct Answer)

main()

Input: After finishing a subtraction the user is asked to input if they want to continue (y/n)?
y

Result: Continues on and asks user to input two new numbers for another subtraction.

main()

Input: After finishing a subtraction the user is asked to input if they want to continue (y/n)?
n

Result: Exits the application