A math or programming approach to the solution?
The puzzle about placing the minimal possible number of "+" and "-" operations between the digits of the string "9 8 7 6 5 4 3 2 1" is a nice programming exercise which can demonstrate the power of recursion.

First of all, this is not a math problem, this is purely a programming task. As long as you are asked to find “some” solution, this can be considered a math puzzle, because you can try to use logic to combine numbers in such a way that you get the desired result. But as soon you are asked for the “optimal” solution (minimal possible number of operations), you set logic aside because the problem becomes a purely programming one. In this particular case, you have to check all 3^8 = 6561 possibilities and select the only one that satisfies your conditions (if any).

From the programming point of view, this problem is pretty simple. You have 8 potential positions where you can place some operation (or just merge two digits), make some configuration of those operations, calculate the result, compare the result with the requested result, and check all possible configurations, which are not many in this particular case.

Two programming approaches
Let's consider two basic approaches to this class of problems. We see that what we have to do is to move along the string of digits, consecutively inserting one of the possible operations, modifying our string. When we reach the end of the string, we have to interpret it, calculate the result and check to see if it is equal to the desired one.

This consecutive motion along the string can be implemented by using two principally different approaches. The simple one is to hardcode the whole sequence using nested loops. This is easy to understand, but extremely clumsy and inelegant. In this particular case, this program would have 8 nested loops. Imagine having to solve the same problem with 100 digits -- you would have to hardcode 99 nested loops. This is definitely not the ideal way of solving this problem.

The second way exploits the idea of recursion*. A recursive function is a function which evokes itself at a certain step – a very useful strategy when you have to perform several identical functional steps with different conditions. In our case, we have to perform an insertion of a certain operation into different positions of the string.

For our puzzle, the block diagram of such a recursive function would be as simple as this:
1. pick up an operation from the list;
2. insert the operation into the current position of the string;
3. if the position is terminal, calculate the result and compare with 100, -- otherwise evoke this function for currently modified string and incremented position.

Starting run for the initial string from the initial position. That's it.

The beauty of this approach is that you use the same code no matter how long the initial string is, for different sets of possible operations, different desired results, etc. – the same very short code as opposed to the clumsy nested loops approach, where you would have to write a new program for every new configuration. The longer the string, the longer the code.

The drawback of the recursive approach is that it is harder to understand. Paradoxically, the shorter and more concise your code, the more difficult it may be to understand. I remember myself as a student (back in the Paleolithic era. We had stone computers and had regular battles between dorms) when i spent a night trying to figure out what a short 4-line piece of code was actually doing. Usually, the difficulty of such things is due to their non-linearity (recursion is a non-linear thing) and the high level of abstraction encapsulated in the code.

Also, I cannot but mention some elegant extension of the clumsy nested loops approach. Some would call that a "God level", but in fact it's called "Meta level", or "Meta language". This is when you speak about your language using the means of your language. An example of this from our common life is, say, when you are learning Japanese and at a certain level you become able to read a book about Japanese grammar written in Japanese. In programming, of course, your language must be powerful enough to provide you with such a possibility.

For our puzzle I can write a short piece of code that will write me a specific nested loops program for any input conditions of this puzzle, a kind of code generator. Once again, I have to write not a code that solves the problem but a code that will itself write me a code that solves the problem. An analogy in molecular biology would be the DNA molecule. DNA is not a code that solves the problem, it is actually the code that generates another code (RNA), which in turn solves the problem.

In conclusion, I have to note that, in real practice we usually have to solve not one specific problem, but investigate whole classes of similar problems. This is called generalization. For example, given our input conditions as a "9 8 7 6 5 4 3 2 1" string, set of operations "+" and "-", we can try to figure out how the number of possible solutions depends on the required result. There is some non-trivial dynamics. We can try to explore a problem with variable number of digits in the input string -- in this case the nested loops approach is inapplicable, you must use recursion. Usually, the higher the level of generalization, the higher the level of abstraction you will have to use to solve this class of problems.

And, before the curtain falls, for those who still think that this is not a programming problem, try to find out how many possible solutions there are for the same string of "9 8 7 6 5 4 3 2 1", a set of possible operations "+ - * / ^" (plus, minus, multiply, divide, power), which gives you 100 as a result. There are only 6^8 = 1679616 cases to check.

Answer: 348.

By: Andrey Mezentsev, Birkeland Centre for Space Science