--- Day 3: Mull It Over ---

"Our computers are having issues, so I have no idea if we have any Chief Historians in stock! You're welcome to check the warehouse, though," says the mildly flustered shopkeeper at the North Pole Toboggan Rental Shop. The Historians head out to take a look.

The shopkeeper turns to you. "Any chance you can see why our computers are having issues again?"

The computer appears to be trying to run a program, but its memory (your puzzle input) is corrupted. All of the instructions have been jumbled up!

It seems like the goal of the program is just to multiply some numbers. It does that with instructions like `mul(X,Y)`, where `X` and `Y` are each 1-3 digit numbers. For instance, `mul(44,46)` multiplies `44` by `46` to get a result of `2024`. Similarly, `mul(123,4)` would multiply `123` by `4`.

However, because the program's memory has been corrupted, there are also many invalid characters that should be ignored, even if they look like part of a `mul` instruction. Sequences like `mul(4*`, `mul(6,9!`, `?(12,34)`, or `mul ( 2 , 4 )` do nothing.

For example, consider the following section of corrupted memory:

```
xmul(2,4)%&mul[3,7]!@^do_not_mul(5,5)+mul(32,64]then(mul(11,8)mul(8,5))
```

Only the four highlighted sections are real `mul` instructions. Adding up the result of each instruction produces `161` (`2*4 + 5*5 + 11*8 + 8*5`).

Scan the corrupted memory for uncorrupted `mul` instructions. What do you get if you add up all of the results of the multiplications?

Your puzzle answer was `166905464`.

--- Part Two ---

As you scan through the corrupted memory, you notice that some of the conditional statements are also still intact. If you handle some of the uncorrupted conditional statements in the program, you might be able to get an even more accurate result.

There are two new instructions you'll need to handle:

- The `do()` instruction enables future `mul` instructions.
- The `don't()` instruction disables future `mul` instructions.

Only the most recent `do()` or `don't()` instruction applies. At the beginning of the program, `mul` instructions are enabled.

For example:

```
xmul(2,4)&mul[3,7]!^don't()_mul(5,5)+mul(32,64](mul(11,8)undo()?mul(8,5))
```

This corrupted memory is similar to the example from before, but this time the `mul(5,5)` and `mul(11,8)` instructions are disabled because there is a `don't()` instruction before them. The other `mul` instructions function normally, including the one at the end that gets re-enabled by a `do()` instruction.

This time, the sum of the results is `48` (`2*4 + 8*5`).

Handle the new instructions; what do you get if you add up all of the results of just the enabled multiplications?

Your puzzle answer was `72948684`.

Both parts of this puzzle are complete! They provide two gold stars: **

At this point, you should return to your Advent calendar and try another puzzle.

If you still want to see it, you can get your puzzle input.

You can also [Share] this puzzle.