Problem F: Serial Numbers

Source file: `serials.{c, cpp, java}`
Input file: `serials.in`

A manufacturer keeps an ordered table of serial numbers by listing in each row of the table a range of serial numbers along with two corresponding pieces of information called the status code and the transfer code. A four-column table stores information about ranges of serial numbers in this order: starting serial number, ending serial number, status code, transfer code. Serial numbers as well as transfer codes are integers from 1 to 2^{31}-1 (2^{31}-1 = 2147483647), and status codes are a single upper-case letter. The table is maintained in increasing order of serial numbers, serial number ranges are never allowed to overlap, and for any given serial number, the table must always accurately represent the most recent data (status code and transfer code) for that serial number.

Let's say that 100,000 serial numbers are created with a status of "A" and a transfer code of "1". An entry for those serial numbers might look like this:

```
1 100000 A 1
```

This is obviously far more efficient than storing 100,000 individual rows all with the same status and transfer codes. The challenge arises when serial numbers within already defined ranges need to be given different status or transfer codes. For example, if serial number 12345 needs to change to status B, the above table would need to become three separate entries:

```
1 12344 A 1
12345 12345 B 1
12346 100000 A 1
```

Now let's change the transfer code of all serial numbers in the range 12000 to 12999 to 2. This gets us:

```
1 11999 A 1
12000 12344 A 2
12345 12345 B 2
12346 12999 A 2
13000 100000 A 1
```

Now change all existing serial numbers from 10000 to 100000 to status C and transfer code 2:

```
1 9999 A 1
10000 100000 C 2
```

Once created a serial number will never be deleted, but it is possible to have ranges of undefined serial numbers between ranges of defined ones. To demonstrate, let's now set all serial numbers from 1000000 to 1999999 to status Z and transfer code 99:

```
1 9999 A 1
10000 100000 C 2
1000000 1999999 Z 99
```

Finally, the table is always maintained with a minimal number of rows, meaning specifically that there will never be two adjacent rows in the table where one would suffice. For example, consider the following serial number table:

```
1 10 A 1
11 20 A 1
21 30 B 1
```

The first two rows could actually be represented by a single row, meaning that the table above does not have a minimal number of rows. The same data represented by a minimal number of rows would look like this:

```
1 20 A 1
21 30 B 1
```

The following table, however, because the first two rows have non-matching transfer codes, already contains the minimal number of rows:

```
1 10 A 1
11 20 A 2
21 30 B 1
```

Similarly, the following table cannot be reduced further because the first two rows do not represent a continuous series of serial numbers:

```
1 10 A 1
12 20 A 1
21 30 B 1
```

Input: Each input case begins with a single line that is a character string naming the test case. This string contains at most 80 characters. The name "END" marks the end of the input. Following this will be 1 to 100 lines of the form "A B S T", where A, B, and T are integers in the range 1 to 2^{31}-1, S is an uppercase letter, and A<=B. These lines are, in the order they are to be applied, the serial number transactions to be recorded, where A is the start of the serial number range, B is the end of the serial number range, S is the status code, and T is the transfer code. The list of serial number transactions is terminated by a line containing only a 0 (zero) character.

Output: For each input case, echo the test case name to the output on a line by itself, followed by the resulting minimal-rows serial number table that results after all serial number transactions have been applied.

Example Input:

```
First Example
1 100000 A 1
12345 12345 B 1
0
And Another
1 100000 A 1
12345 12345 B 1
12000 12999 A 2
12345 12345 B 2
0
Test Case Three
1 100000 A 1
12345 12345 B 1
12000 12999 A 2
12345 12345 B 2
10000 100000 C 2
0
Example Four
1 100000 A 1
12345 12345 B 1
12000 12999 A 2
12345 12345 B 2
10000 100000 C 2
1000000 1999999 Z 99
0
Example 5
1 10 A 1
21 30 B 1
11 20 A 1
0
Example 6
21 30 B 1
1 10 A 1
11 20 A 2
0
Example 7
12 20 A 1
21 30 B 1
1 10 A 1
0
END
```

Example Output:

```
First Example
1 12344 A 1
12345 12345 B 1
12346 100000 A 1
And Another
1 11999 A 1
12000 12344 A 2
12345 12345 B 2
12346 12999 A 2
13000 100000 A 1
Test Case Three
1 9999 A 1
10000 100000 C 2
Example Four
1 9999 A 1
10000 100000 C 2
1000000 1999999 Z 99
Example 5
1 20 A 1
21 30 B 1
Example 6
1 10 A 1
11 20 A 2
21 30 B 1
Example 7
1 10 A 1
12 20 A 1
21 30 B 1
```

Last modified on October 26, 2008 at 2:20 PM.