8th South African Regional
ACM Collegiate Programming Competition

Sponsored by IBM

Problem D – Blue balloon
Not quite ALE

The ALE (Automatic Link Establishment) protocol is a system designed to allow military radios to agree on a frequency that is to be used for communication. In short, each radio is scanning a number of channels, and it will pause on every channel that it hears some traffic on. If it hears a call (from another radio) to its own address, it will stop on that channel, and alert the operator that a connection has been established.

ALE calls are typically encrypted. Your task will be to implement a decryption routine for such encrypted calls. First, a subset of the ALE protocol will be discussed in the **ALE** and **Scanning ALE** sections. This is followed by a description of the format used to pack the data for transmission in the **ALE data packing** section. Lastly, the **ALE encryption** section defines the encryption algorithm.

ALE
ALE calls consist of two sections: a scanning section, and the calling section. The scanning section precedes the calling section, but it will only be discussed below under the heading **Scanning ALE**. The calling section will now be described.

Every radio has a "self" address which contains only the digits [0-9]. The tricky part is that the addresses are variable length, and therefore they may consist of one, two or three digits, *i.e.* "0", "01", "00", "000", and "001" are all valid, unique addresses. Think of each address as a string of digits, rather than a number.

To establish a call, the caller will transmit a sequence that starts with the callee's address, followed by the caller's. The simplest type of call is one between two single-digit-address radios, which looks like this:

```
TO   1
TO   1
TIS  7
```

This is a call from radio "7" to radio "1". The command "TO" is self-explanatory; the "TIS" command is a contraction of "THIS IS". Note that the callee's address is repeated, *i.e.* it appears twice.

If either the caller or the callee have longer addresses, a more complex sequence is used. Here is an example of a call from radio "7" to radio "13":

```
TO   1
DATA 3
TO   1
DATA 3
TIS  7
```

Note that the callee's address is repeated twice, as in the example above, and that the "DATA" keyword is used to represent the second digit of the address.

When a three-digit address is used, it looks like this:

```
TO   1
DATA 3
REP  5
TO   1
DATA 3
REP  5
TIS  7
```

This is a call from radio "7" to radio "135". Note that the third digit of the address is represented using the "REP" command (REPeat, meaning repeat of the command "DATA", not the digit of the preceding "DATA" command. The "REP" command has its own digit payload.)

The same principle is used for the caller's address, so that when radio "721" calls radio "51", it looks like this:

```
TO   5
DATA 1
TO   5
DATA 1
TIS  7
DATA 2
REP  1
```

To summarise, a typical non-scanning call uses the following syntax:

```
TO <d1> [DATA <d2> [REP <d3>]]
TO <d1> [DATA <d2> [REP <d3>]]
TIS <s1> [DATA <s2> [REP <s3>]]
```

where <d1>-<d3> and <s1>-<s3> represent the digits of the callee and caller's addresses, and [] indicates optional components. Note, each command starts on a new line.

Scanning ALE

There is one additional problem, though. Assume that radio "7" wants to make a call to radio "1". Radio "1" is scanning through N channels, spending a few milliseconds on each to see if there is a call for it on that channel. If radio "7" thus decides to use channel 3 for the call, it must transmit long enough so that radio "1" has time to go through all its channels and still be in time to hear the call from radio "7". To solve this problem, radio "7" will repeat the first command of the call (containing the first digit of the callee's address) a number of times, thus giving radio "1" the opportunity to hear its own address (the first digit thereof). Note that other stations, like radio "15" or radio "123" will also stop to listen, but they will continue scanning once the full address follows, and they can unambiguously determine that the call is not for them. Thus, the simple call listed in the first example above will become, for example,

```
TO   1
TO   1
TO   1
TO   1
TO   1
TIS  7
```

where the number of repetitions of the first "TO" command is arbitrary, but **fewer than 10**. Only the first "TO" command is repeated, so that the second example (a call from "7" to "13") becomes, for example,

```
TO   1
TO   1
TO   1
TO   1
TO   1
TO   1
DATA 3
TO   1
DATA 3
TIS  7
```

The final syntax for a scanning ALE call is thus:

```
TO <d1>*
TO <d1> [DATA <d2> [REP <d3>]]
TO <d1> [DATA <d2> [REP <d3>]]
TIS <s1> [DATA <s2> [REP <s3>]]
```

where "*" means zero of more repetitions of the stated command. The sequence of repetitions of the first "TO" command is called the *scanning section*.

In a valid scanning ALE call, only certain commands may follow others. Here is a list of valid combinations:

- TO may be followed by TO, DATA, or TIS
- TIS may be followed by DATA
- DATA may be followed by TO, REP, or TIS
- REP may be followed by TO, or TIS

ALE data packing
The individual commands can be packed into bytes using the following format:

- The commands are represented using 4-bit values, which can be packed into the upper 4 bits of a byte. This effectively assigns hexadecimal values as follows: TO = 0x00, TIS = 0x10, DATA = 0x20, and REP = 0x30
- An address digit is represented as a 4-bit value, thus 0 = 0, 1 = 1, ... , 9 = 9. These values are stored in the lower 4 bits of a byte. Note that all other 4-bit values (0x0a through 0x0f) are invalid.

The packed commands are thus formed by adding (or bit-wise OR-ing) the command value to the digit value. The upper two bits of the resulting byte are always zero. This means that TO 1 is encoded as 0x01, and REP 7 as 0x37, and so on.

ALE encryption
Now things get more interesting: this is a military protocol, hence the data must be encrypted. The original ALE protocol uses 56-bit encryption keys; our simplified algorithm will use a simple look-up table.

Key #	**Key phase number**									
	0	1	2	3	4	5	6	7	8	9
1	0x2a	0x15	0x2a	0x22	0x15	0x11	0x1a	0x2d	0x25	0x1e
2	0x17	0x2e	0x15	0x17	0x0f	0x28	0x2d	0x1e	0x3b	0x2f
3	0x11	0x2c	0x13	0x0c	0x23	0x28	0x3f	0x16	0x07	0x32

This table lists three encryption keys, $C(s)_n$. For a given call, any one of the three keys (s = 1, 2 or 3) is selected. Communications are encrypted as follows: a call command sequence (as defined above) is packed into bytes (of which the upper two bits are always set to zero) as described above. Call this sequence p_n, the plaintext sequence. The encrypted sequence is then obtained by performing a bit-wise XOR with the corresponding word from the encryption key, to obtain the encrypted word, thus:

$e_n = C(s)_{(n \% 10)} \wedge p_n$

where "%" denotes the modulo operator, and "^" the bit-wise XOR operator. The value n % 10 is called the *phase* of the key sequence. n is the byte-index of the word in the message or encrypted message.

Example: The sequence TO 1, TO 1, TIS 7, in packed form, is 0x01, 0x01, 0x17. Encrypting it with key 2 yields 0x16, 0x2f, 0x02.

Things are a bit more complicated when the scanning section must be encrypted. Radio "1" can arrive on the channel (that radio "7" is calling on) at any time. Thus, if radio "7" just blindly encrypted the entire call, it would be very difficult for radio "1" to determine the correct encryption phase, since radio "1" effectively misses the first *k* words of the call, and it does not know what the value of *k* is. The solution is to alternate between key phase 0 and key phase 1 for the scanning part of the call. This reduces the search space for radio "1" considerably, making the problem tractable. In tabular form, here is an example of an encrypted call (using key 2, as example) with a scanning section:

Call word	**Packed words**	**Key phase**	**Encrypted words**
TO 1	0x01	1	0x2f
TO 1	0x01	0	0x16
TO 1	0x01	1	0x2f
TO 1	0x01	0	0x16
DATA 8	0x28	1	0x06
TO 1	0x01	2	0x14
DATA 8	0x28	3	0x3f
TIS 7	0x17	4	0x18
DATA 5	0x25	5	0x0d
REP 3	0x33	6	0x1e

Note that the key phase is always aligned so that the last word of the scanning section ends on phase 1, so that the first command that forms part of the full address (the 4th command in the example above) always starts on phase 0.

Your task is to implement a decryption algorithm for this simplified ALE encryption scheme. As input, you will receive a sequence of encrypted bytes representing encrypted ALE words. The calls may include an optional scanning section (of no more than 10 words), followed by the call itself. The call will be encrypted as described

above, using the encryption table provided above. You have to determine the correct key number to use, and you have to recover the key phase using only the encrypted data.

Note that your encrypted sequence will always contain words with a key phase of 0, 1 and 2. Longer messages will use higher phase numbers. A scanning section may or may not be present at the start of the encrypted sequence.

All input test data is guaranteed to have a unique solution. You may also assume that the reception of the words was flawless, *i.e.* there were no transmission errors.

Input

Your input will consist of a number of records. Each record is a sequence of integers in an ASCII-hexadecimal representation, separated by spaces. Each sequence is terminated by a 0xff value.

Output

For each input record, your output should be the decrypted ALE call sequence, as shown below. Please format your text as shown below: the first column should be 4 characters wide, left-justified, followed by a single-space column, followed by the digit column. After each input record has been processed, a newline character must be printed to the output.

Sample Input

```
0x10 0x2d 0x04 0x2b 0xff
0x2f 0x16 0x2f 0x16 0x2f 0x02 0x32 0x3c 0xff
```

Sample Output

```
TO   1
TO   1
TIS  7
DATA 7

TO   1
TO   1
TO   1
TO   1
TO   1
TIS  7
DATA 5
REP  3
```