Problem 5: Break It Up

Author: Danny Lin, Greenville, South Carolina, United States

Problem Background

Computer programs accept data in an incredibly wide range of formats. You’re familiar with many data formats already: JPEG, PNG, and GIF are all common image formats; documents are often sent in DOCX or PDF format; and ZIP files are used to compress multiple files together to save space. Many programs use their own formats to store and read data. Whatever format is used, it’s very important that the computer program is able to actually *read* that format.

A file format defines how data is stored within the file, and how different portions of data are separated within the file. All of the information pertaining to a single thing (whatever that “thing” may be) is generally referred to as a single “record.” Different details about that record are separated by a “delimiter” - a character or series of characters that are not expected to appear within the data itself. In the event a delimiter character *does* appear within the data, a file format may use quotations (either single (‘) or double (“) quotes) to enclose the data field. Some formats also make use of an escape character to ignore the following character’s normal behavior; this allows the data to include a quotation within a quoted string, use a delimiter within a data field, or include an actual copy of the escape character itself.

For example, if a format uses a pipe character (\|) as a delimiter and a backslash (\\) character as an escape character, the following records would be read as shown:

Record	Field 1	Field 2	Field 3
One|Two|Three	One	Two	Three
“Pi|pe”|Pi|pe	Pi|pe	Pi	pe
Pi\\\|pe|Pi|pe	Pi|pe	Pi	pe
“Quo\\”te”	Quo”te	*Not present*	
Esc\\\\ape	Esc\\ape	*Not present*	

Being able to parse a variety of inputs and make sense of them is critical to solving a number of programming challenges… including this contest!

Problem Description

Your team has been asked to develop a parser for a new “universal format” to be used on all Lockheed Martin programs. Rather than use a fixed delimiter and escape character, the format will specify which characters are to be used for the delimiter and escapes at the beginning of the file. Remember:

- The delimiter character separates different data fields within the same record
- The escape character “disables” the normal behavior of the character immediately behind it.

Additionally, quotes can be used to contain strings which may contain the delimiter character; all text within two non-escaped quotes should be considered to be part of a single field and should not be broken into multiple fields. You will not be given any unmatched quotes, and when quotes appear they will enclose the entire data string; for example, the left value may appear in your data, but the center and right values will not.

`“This is valid.”`	`“This is not.\”`	`Neither “is this.”`

Your program must read in records and, using the provided delimiter and escape characters, parse out the individual data fields for each record. The data you are provided may contain blank fields (which should be printed as <BLANK> to highlight this fact), however all records will have valid formatting.

Sample Input

The first line of your program’s input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing the following information, separated by spaces:
  - A positive integer, **X**, indicating the number of records in the test case
  - A printable, non-whitespace ASCII character representing the delimiter character used throughout this test case
  - A printable, non-whitespace ASCII character representing the escape character used throughout the test case.
- **X** lines, following the format described above, each representing a data record with one or more fields

```
2
3 | \
ABC|123|def
456\|789|hij
“klm|nop|qrs”|000
2 , %
Twelve,Percent,12%%
This%,is%,all%,one%,field
```

Sample Output

For each test case, your program must output the separated data fields, one on each line, in the format “Record X, Field Y: Z”, where:

- X is the number of the record containing the field, where the first record in a test case is record 1
- Y is the number of the field within the record, where the first field in a record is field 1
- Z is the parsed string contained in the data field. If the string is empty (contains no characters), print <BLANK> instead.

```
Record 1, Field 1: ABC
Record 1, Field 2: 123
Record 1, Field 3: def
Record 2, Field 1: 456|789
Record 2, Field 2: hij
Record 3, Field 1: klm|nop|qrs
Record 3, Field 2: 000
Record 1, Field 1: Twelve
Record 1, Field 2: Percent
Record 1, Field 3: 12%
Record 2, Field 1: This,is,all,one,field
```