PROBLEM :

The problem has 4 user stories for which the candidate has to revert with a well designed,well documented code along with the unit tests. The candidate is expected to complete atleast the first 3 user stories and it would be great if the candidate could get the fourth one in as well. Once we receive the candidate's solutions we would schedule a time for the developer to present his solutions and give us a chance to ask questions about his work.

The Problem description and sample test cases are as follows:

Problem Description

User Story 1

You work for a bank, which has recently purchased an ingenious machine to assist in reading letters and faxes sent in by branch offices. The machine scans the paper documents, and produces a file with a number of entries which each look like this:

```
    _ _   _ _ _ _ _

  | _| _||_||_ |_  ||_||_|

  ||_  _|  | _||_|  ||_| _|
```

Each entry is 4 lines long, and each line has 27 characters. The first 3 lines of each entry contain an account number written using pipes and underscores, and the fourth line is blank. Each account number should have 9 digits, all of which should be in the range 0-9. A normal file contains around 500 entries.

Your first task is to write a program that can take this file and parse it into actual account numbers.

User Story 2

Having done that, you quickly realize that the ingenious machine is not in fact infallible. Sometimes it goes wrong in its scanning. The next step therefore is to validate that the numbers you read are in fact valid account numbers. A valid account number has a valid checksum. This can be calculated as follows:

```
account number:  3  4  5  8  8  2  8  6  5


position names:  d9 d8 d7 d6 d5 d4 d3 d2 d1
```

Checksum calculation: (d1+2*d2+3*d3 +..+9*d9) mod 11 = 0

So now you should also write some code that calculates the checksum for a given number, and identifies if it is a valid account number.

User Story 3

Your boss is keen to see your results. He asks you to write out a file of your findings, one for each input file, in this format:

457508000

664371495 ERR
86110??36 ILL

i.e., the file has one account number per row. If some characters are illegible, they are replaced by a ?. In the case of a wrong checksum, or illegible number, this is noted in a second column indicating status.

User Story 4

It turns out that often when a number comes back as ERR or ILL it is because the scanner has failed to pick up on one pipe or underscore for one of the figures. For example

```
    _ _ _ _ _ _   _

|_||_|| || ||_   |  |  ||_

  | _||_||_||_|  |  |  | _|
```

The 9 could be an 8 if the scanner had missed one |. Or the 0 could be an 8. Or the 1 could be a 7. The 5 could be a 9 or 6. So your next task is to look at numbers that have come back as ERR or ILL, and try to guess what they should be, by adding or removing just one pipe or underscore. If there is only one possible number with a valid checksum, then use that. If there are several options, the status should be AMB. If you still can't work out what it should be, the status should be reported ILL.

Sample Test Cases:

use case 1

```
 _ _ _ _ _ _ _ _ _
| || || || || || || || || |
|_||_||_||_||_||_||_||_||_|
```

=> 000000000

```
                           
  |  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  |  |
```

=> 111111111

```
 _  _  _  _  _  _  _  _  _ 
 _| _| _| _| _| _| _| _| _|
|_ |_ |_ |_ |_ |_ |_ |_ |_ 
```

=> 222222222

```
 _  _  _  _  _  _  _  _  _ 
 _| _| _| _| _| _| _| _| _|
 _| _| _| _| _| _| _| _| _|
```

=> 333333333

```
                           
|_||_||_||_||_||_||_||_||_|
  |  |  |  |  |  |  |  |  |
```

=> 444444444

```
 _  _  _  _  _  _  _  _  _ 
|_ |_ |_ |_ |_ |_ |_ |_ |_ 
 _| _| _| _| _| _| _| _| _|
```

=> 555555555

```
 _  _  _  _  _  _  _  _  _ 
|_ |_ |_ |_ |_ |_ |_ |_ |_ 
|_||_||_||_||_||_||_||_||_|
```

=> 666666666

```
 _  _  _  _  _  _  _  _  _ 
  |  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  |  |
```

=> 777777777

```
 _  _  _  _  _  _  _  _  _ 
|_||_||_||_||_||_||_||_||_|
|_||_||_||_||_||_||_||_||_|
```

=> 888888888

```
 _  _  _  _  _  _  _  _  _ 
|_||_||_||_||_||_||_||_||_|
 _| _| _| _| _| _| _| _| _|
```

=> 999999999

```
    _  _     _  _  _  _  _ 
  | _| _||_||_ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _|
```

=> 123456789

use case 3

```
 _  _  _  _  _  _  _  _    
| || || || || || || ||_   |
|_||_||_||_||_||_||_| _|  |
```

=> 000000051

```
    _  _  _  _  _  _     _ 
|_||_|| || ||_   |  |  | _ 
  | _||_||_||_|  |  |  | _|
```

=> 49006771? ILL

```
    _  _     _  _  _  _  _ 
  | _| _||_| _ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _ 
```

=> 1234?678? ILL

use case 4

```
                           
  |  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  |  |
```

=> 711111111

```
 _  _  _  _  _  _  _  _  _ 
  |  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  |  |
```

=> 777777177

```
 _  _  _  _  _  _  _  _  _ 
 _|| || || || || || || || |
|_ |_||_||_||_||_||_||_||_|
```

=> 200800000

```
 _  _  _  _  _  _  _  _  _ 
 _| _| _| _| _| _| _| _| _|
 _| _| _| _| _| _| _| _| _|
```

=> 333393333

```
 _  _  _  _  _  _  _  _  _ 
|_||_||_||_||_||_||_||_||_|
|_||_||_||_||_||_||_||_||_|
```

=> 888888888 AMB ['888886888', '888888880', '888888988']

```
 _  _  _  _  _  _  _  _  _ 
|_ |_ |_ |_ |_ |_ |_ |_ |_ 
 _| _| _| _| _| _| _| _| _|
```

=> 555555555 AMB ['555655555', '559555555']

```
 _  _  _  _  _  _  _  _  _ 
|_ |_ |_ |_ |_ |_ |_ |_ |_ 
|_||_||_||_||_||_||_||_||_|
```

=> 666666666 AMB ['666566666', '686666666']

```
 _  _  _  _  _  _  _  _  _ 
|_||_||_||_||_||_||_||_||_|
 _| _| _| _| _| _| _| _| _|
```

=> 999999999 AMB ['899999999', '993999999', '999959999']

```
    _  _  _  _  _  _     _ 
|_||_|| || ||_   |  |  ||_ 
```

```
  | _||_||_||_|  |  |  | _|
```

=> 490067715 AMB ['490067115', '490067719', '490867715']

```
    _  _     _  _  _  _  _
 _| _| _||_||_ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _|
```

=> 123456789

```
 _     _  _  _  _  _  _
| || || || || || || ||_   |
|_||_||_||_||_||_||_| _|  |
```

=> 000000051

```
    _  _  _  _  _  _     _
|_||_|| ||_||_   |  |  | _
  | _||_||_||_|  |  |  | _|
```

=> 490867715