Problem D
Edsger Dijkstra

Time limit: 2 seconds
Memory limit: 256 megabytes

Problem Description

The Turing award winner Edsger Wybe Dijkstra is a dutch computer scientist who has an annoyingly confusing name that Asians have trouble pronouncing. In 1960s, He gave the following quote.

> For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all "higher level" programming languages.

You don't want to produce low quality codes, do you? However, your source code contains no loop statements. The only kind of flow control statements in your source code is the `if-goto`. To decrease the density, you have to try and eliminate all the `if-goto` statements in your source code written in a C-like language and replace them with `do-while` loops in the following manner.

- Assume the `if-goto` statement looks like "`if (boolean_expression) goto some_label;`".
- Insert a copy of "`do {`" right after where `some_label` is declared.
- Replace '`if`' by '`} while`'.
- Remove '`goto some_label`'.

For example, the following code

```
int main() {
    int score;
    get_score:
    scanf("%d",&score);
    if (score < 0 || score > 100) goto get_score;
    if (score < 60) goto fail;
    fail:
    puts("you are failed!");
    return 0;
}
```

will be modified into

```
int main() {
int score;
    get_score: do {
    scanf("%d",&score);
    } while (score < 0 || score > 100) ;
    } while (score < 60) ;
    fail: do {
    puts("you are failed!");
    return 0;
}
```

It is not too surprising that the code above cannot be compiled. Here is your new task: given a sequences of statements, please determine whether all `if-goto` statements can be replaced by `do-while` loops without changing the output of your code. For simplicity, you may assume all statements are either in the following two forms.

- Form 1: "`line_`x`: puts("`x`");`" where x is corresponding line number of this statement and "`puts("`x`");`" prints the line number x.
- Form 2: "`if (expr_`x`()) goto line_`y`;`" where x is the corresponding line number of this statement and `line_`y is a valid label in the program. "`expr_`x`()`" will return `true` on first x invocations, and it will return `false` afterward.

Note that the output of the program should be considered as different from the original output if the modification makes the code unable to be compiled.

Input Format

In the first line of input, there will be a single integer T ($T \leq 20$) on a line representing the number of test cases.

Each test case is consists of a sequence of statements. Each statements will be on a single line, which starts on line 1. There can be three kinds of lines: statements in form 1, statements in form 2, and "`END`". "`END`" will indicate the end of a test case. It can only be the last line of a test case, and it should not be considered as a part of the program. There are at most 10000 statements in a single test case (`END`'s are excluded).

Output Format

Print "good" on a single line if replacing all `if-goto` statements with `do-while` will not change the output of the program. Otherwise print "bad". Note: you should output "bad" if the program becomes no longer compilable.

Sample Input

```
3
line_1: puts("1");
if (expr_2()) goto line_1;
END
line_1: puts("1");
line_2: puts("2");
line_3: puts("3");
if (expr_4()) goto line_2;
line_5: puts("5");
if (expr_6()) goto line_2;
END
line_1: puts("1");
if (expr_2()) goto line_5;
line_3: puts("3");
line_4: puts("4");
line_5: puts("5");
END
```

Sample Output

```
good
good
bad
```