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
```

Problem D
Edsger Dijkstra

Time limit: 2 seconds
Memory limit: 256 megabytes

Problem Description

圖靈獎得主艾茲赫爾・韋伯・戴克斯特拉 (Edsger Wybe Dijkstra) 是荷蘭計算機科學家，他的名字的發音總是讓許多亞洲人感到困擾。在 1960 年代，他給出了以下的評論：

> 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.

你不想產出低品質的程式碼，對吧？然而你發現你的程式碼裡面一個迴圈敘述都沒有。唯一的流程控制程式碼，就只有`if-goto`。為了降低`goto` 的密度，你得試著消除掉一份類 C 程式碼裡所有的`if-goto` 敘述，用`do-while` 迴圈以下面的方式取代：

- 假定`if-goto` 看起來像是 "`if (boolean_expression) goto some_label;`".
- 在`some_label` 宣告處後方插入一份 "`do {`"。
- 用 '`} while`' 取代 '`if`'。
- 移除 '`goto some_label`'。

舉例來說，就是將以下程式碼

```
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;
}
```

轉換為下面這樣：

```
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;
}
```

以上的程式碼無法通過編譯，並不是一個令人意外的結果。你的新任務是：給定一連串的敘述，請判斷是否所有的`if-goto` 敘述，可以用`do-while` 迴圈，以前述的方式取代，並且不影響到程式的輸出。為了簡單起見，你可以假定程式碼中的敘述只有下列兩種格式：

- 格式一：“`line_`x`: puts("`x`");`” 其中 x 是此敘述對應的行號且 “`puts("`x`");`” 會印出行號 x。
- 格式二：“`if (expr_`x`()) goto line_`y`;`” 其中 x 是該敘述對應的行號，`line_`y 是程式中合法的標籤 (Label)。“`expr_`x`()`” 在前 x 次呼叫時，將會回傳 `true`，之後會回傳 `false`。

請留意如果程式碼無法通過編譯，將被視為與原先的程式碼輸出不同。

Input Format

輸入資料的第一行有一個整數 T ($T \leq 20$) 代表有多少組測試資料。每一組測試資料由一連串的敘述所構成，每行有一個敘述。而構成測試資料的敘述有三種：格式一、格式二以及`END`。`END` 代表該組測試資料結束，只會出現在每筆測試資料的最後一行，不用把它看作程式碼的一部分。單一筆測試資料，至多有 10000 個敘述 (不含`END`)。

Output Format

如果將所有的`if-goto` 敘述換成`do-while` 迴圈不會改變輸出的話，請印出 “good”，反之印出 “bad”。如果程式碼將無法編譯，請輸出 “bad”。

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
```