Problem A. How Much Memory Your Code Is Using?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	8 seconds
Memory limit:	1024 megabytes

In the C++ language, the values of variables are stored somewhere in the computer memory as zeros and ones. Our program does not need to know the exact location where a variable is stored since one can simply refer to it by its name.

What the program needs to be aware of is the kind of data stored in every variable. Storing a simple integer is not the same as storing a letter or a large floating-point number. Even though they are all represented by zeros and ones, they are not interpreted in the same way, and in many cases, they don't occupy the same amount of memory.

Fundamental data types, implemented directly by the language, represent the basic storage units supported natively by most systems. They can be mainly classified into four types.

Character types: They can represent a single character, such as 'A' or '$'. The most basic type is `char`, which is a one-byte character.

Numerical integer types: They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes. The most basic one is `int`, which is a four-byte integer. `long long` is a larger one which occupies twice as many memories as `int` does. A wider type of integer `__int128` may appear in some new systems, which is a 16-byte integer and rarely useful.

Floating-point types: They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used. `float`, `double` and `long double` correspond to `int`, `long long` and `__int128` where the former types hold the same amount of memory as the latter types used respectively.

Boolean type: The boolean type, known in C++ as `bool`, can only represent one of two states, true or false. It is a one-byte type.

Now you have a code in C++, n lines of which apply several variables and arrays. Also, I have checked it so I can claim that all variables and arrays in this code are global without any conflicts.

As a novice, each line in your code may only apply for one variable or one array; all types of these variables and arrays are mentioned above; any other types unmentioned can not appear in the code. Your task in this problem is to calculate the number of memories in total that the code is using. Output your answer in Kibibyte (1 Kibibyte is 1024 bytes) and round it up to the nearest integer.

Input

The input contains several test cases, and the first line contains a positive integer T indicating the number of test cases which is up to 100.

For each test case, the first line contains a positive integer n, which is up to 1000, indicating the total number of lines to apply (i. e. allocate memory) for variables and arrays in the code. Each of the following n lines may declare a variable in the form

- `type variable_name;`

or declare a new array in the form

- `type array_name[array_size];`

where `type` must be one name of the aforementioned types. All `variable_name` and `array_name` are distinct strings containing only lowercase letters whose lengths are up to 10, and all `array_size` are

positive integers which are up to 10^5. Except for spaces in or after the name of some type, no extra spaces are allowed in the input. In other words, we guarantee that no consecutive spaces appear in the input.

Output

For each test case, output a line containing "`Case #x: y`" (without quotes), where `x` is the test case number starting from 1, and `y` indicates the total number of allocated memories in Kibibyte which is rounded up to the nearest integer.

Example

standard input	standard output
2 8 bool a; char b; int c; long long d; __int128 e; float f; double g; long double h; 1 int a[1000];	Case #1: 1 Case #2: 4

Note

In the second sample case, the memory usage is 4000 bytes, which should be rounded up to 4 Kibibytes.