Problem B
Quickscope

Semantic analysis is one of the intermediate steps a compiler takes when compiling a program. In this problem, you will be implementing some of the checks done by a compiler during semantic analysis.

When a variable is declared, a compiler has to check that there is no other variable already declared with the same name, in the same block. A *block* is the list of statements enclosed within a pair of braces {}.

Blocks can contain other blocks, and you can declare 2 variables with the same name if one declaration is in the outer block and another is in the inner block.

For example, the first code block is valid, while the second one is not.

```
{
    int a;
    int b;
    {
        float a;
    }
}

{
    int a;
    int b;
    float a; // Multiple declaration
}
```

At various points in the code, the compiler needs to check the data type of a variable. To do that, it first checks among the variables declared in the current block. If the variable was declared in the current block, the compiler can determine its type, and the algorithm terminates. If the variable is not found in the current block, it goes up one level to the parent block and recursively repeats the process, until either a match is found or the compiler reaches the outermost block. If no match is found even in the outermost block, that means the variable has not been declared.

The input file represents a simplified program code. Your task is to:

1. Check that all variable declarations are valid.
2. Print the type of a variable when queried.

The following sections will describe this in more detail.

Input

The first line contains an integer, $1 \leq N \leq 200\,000$, the number of lines in the program.

The next N lines can be of the following 4 types:

1. { - Represents entering a new block of code.
2. } - Represents exiting the current block of code.
3. **DECLARE <identifier> <type>** - Represents the declaration of a variable with name **<identifier>** and data type **<type>**.
4. **TYPEOF <identifier>** - Represents a query to print the type of the variable with name **<identifier>**.

All variable identifiers and types are strings consisting of between 1 and 6 lowercase letters. **Note**: The types can be any string and are not restricted to the common data types in programming languages.

It is guaranteed that the braces are properly matched.

Output

For each **DECLARE** line, if the declaration is valid, don't print anything. If there already exists a variable declared with the same name in the same block, print the string **"MULTIPLE DECLARATION"** and exit the program **without printing anything else**.

For each **TYPEOF** line, print the data type of the variable followed by a newline. Follow the rules described above in the case of nested blocks. If the variable has not been declared, print the string **"UNDECLARED"** followed by a newline.

Example

The following code snippet written in C syntax represents sample input 4. The **TYPEOF** queries along with the expected outputs are indicated with comments.

```
int apple;
double banana;
{
    float apple;
    char orange;
    // TYPEOF apple -> float
    // TYPEOF banana -> double
    // TYPEOF orange -> char
}
// TYPEOF apple -> int
// TYPEOF banana -> double
// TYPEOF orange -> UNDECLARED
char orange;
// TYPEOF orange -> char
char banana; // MULTIPLE DECLARATION
// TYPEOF banana -> STATEMENT NOT REACHED
```

Subtasks

1. (30 **Points**): $N \leq 1\,000$. There are no statements of type 1 or 2 in the input. That is, there are no nested blocks.
2. (30 **Points**): $N \leq 200\,000$. There are no statements of type 1 or 2 in the input. That is, there are no nested blocks.
3. (30 **Points**): $N \leq 200\,000$. The level of nesting of blocks is at most 10 levels deep.
4. (10 **Points**): No additional constraint.

Note: Sample Inputs 4 - 6 correspond to Subtask 3. You can ignore them if you are only attempting Subtask 1 or 2.

Warning

The input files are large. Please use fast I/O methods.

Sample Input 1

```
8
TYPEOF a
TYPEOF b
DECLARE a int
TYPEOF a
TYPEOF b
DECLARE b float
TYPEOF a
TYPEOF b
```

Sample Output 1

```
UNDECLARED
UNDECLARED
int
UNDECLARED
int
float
```

Sample Input 2

```
5
DECLARE a int
DECLARE a float
TYPEOF a
DECLARE a double
TYPEOF a
```

Sample Output 2

```
MULTIPLE DECLARATION
```

Sample Input 3

```
6
DECLARE aaaa bbbb
DECLARE cccc dddd
TYPEOF aaaa
TYPEOF bbbb
TYPEOF cccc
TYPEOF dddd
```

Sample Output 3

```
bbbb
UNDECLARED
dddd
UNDECLARED
```

Sample Input 4

```
16
DECLARE apple int
DECLARE banana double
{
DECLARE apple float
DECLARE orange char
TYPEOF apple
TYPEOF banana
TYPEOF orange
}
TYPEOF apple
TYPEOF banana
TYPEOF orange
DECLARE orange char
TYPEOF orange
DECLARE banana char
TYPEOF banana
```

Sample Output 4

```
float
double
char
int
double
UNDECLARED
char
MULTIPLE DECLARATION
```

Sample Input 5

```
18
{
DECLARE x a
{
DECLARE x b
{
{
DECLARE x c
{
TYPEOF x
}
TYPEOF x
}
}
TYPEOF x
}
TYPEOF x
}
TYPEOF x
```

Sample Output 5

```
c
c
b
a
UNDECLARED
```

Sample Input 6

```
12
{
DECLARE x abc
TYPEOF x
}
{
TYPEOF x
DECLARE x abc
DECLARE y abc
TYPEOF x
TYPEOF y
DECLARE y abc
}
```

Sample Output 6

```
abc
UNDECLARED
abc
abc
MULTIPLE DECLARATION
```