3092 Temporary Variables

The simple programming language **ASSIGN** allows only assignment statements (one statement per line) with the following format. Each assignment statement consists of a Left Hand Side (called LHS) and a Right Hand Side (called RHS) which are separated by the '=' sign with at least one space before and after the '='.

- An LHS can be
  - A (regular) variable name consisting of letters which may be small or capital. Digits are not allowed in variable names.
  - A temporary variable name beginning with an '_t' followed by a number as shown in the examples below.
- A RHS can be **complex**
  - An infix binary expression consisting of two operands (defined below) and one of the following binary operators: '+', '*', '-', '/'. The operator and operands are separated by a space.
  - A prefix unary expression consisting of one operand and one of the following unary operators: '&', '*', '-'. The operator and operand is separated by a space.

  or an RHS can be **simple**
  - A single operand without any operator

An operand can be either a variable name, or a temporary name or an integer constant in decimal format.

You have to write a program to read in a sequence of assignment statements and using the rules explained below until they no longer apply, generate the final output sequence. Note that there will always be only one possible output.

- **Rule 0:** At any point in the derivation, all temporary variables names in the output must be in order. This means that '_t2', for example, should not appear as an LHS before the first occurrence of both '_t0' and '_t1' as an LHS.
- **Rule 1:** If the RHS is complex and if the LHS of an assignment is not a temporary variable, insert a new temporary variable as shown below.

Input

```
value = value * change
newvalue = value
```

=>

Output

```
_t0 = value * change
value = _t0
newvalue = value
```

- **Rule 2:** If the RHS of two statements are identical complex expressions, but the corresponding LHSs are different temporary variables, then eliminate the second temporary variable using the first one as shown below.

Input		Output
`_t1 = a + b`		`_t0 = a + b`
`_t2 = c + _t1`		`_t1 = c + _t0`
`d = _t2`	=>	`d = _t1`
`_t5 = a + b`		`_t0 = a + b`
`_t6 = c + _t5`		`_t1 = c + _t0`
`e = _t6`		`e = _t1`

Note that Rules 0 and 1 have also been used above. For example, the expression '`c + _t1`' in the input becomes '`c +_t0`' in the output and since temporary variable '`_t5`' of the input is same as '`_t0`' if the output, expression '`c + _t5`' in the input also become '`c + _t0`' in the output.

- **Rule 3:** If the RHS of the assignment is a complex binary expression, but contains no variables, then replace RHS expression by its value, and replace the LHS by a new temporary variable as shown below.

Input		Output
`a = 10 * 20`	=>	`_t0 = 200`
		`a = _t0`

Input

The input will contain several test cases. The first line contains a positive integer N giving the number of test cases.

For every test case the first line will a number m giving the number of assignment statements in the input program. This will be followed by m lines, one assignment statement per line, corresponding to the input program.

Output

The output for each test case must be the sequence obtained using the rules explained above as long as any of them is applicable. The outputs for consecutive test cases should be on consecutive lines.

Sample Input

```
2
1

a  =  10 * 20

5

a = a * b
b = * a
c = - a
_t13 = * a
b = & _t13
```

Sample Output

```
_t0 = 200
a = _t0
```

```
_t0 = a * b
a = _t0
_t1 = * a
b = _t1
_t2 = - a
c = _t2
_t1 = * a
_t3 = & _t1
b = _t3 }
```