Problem G: When

Source: `when.{c,cpp,java}`
Input: `when.in`
Output: `when.out`

"When" is an event driven language for machine control. It only has three statements: Set, Print, and the compound When clause. The (case insensitive) grammar is as follows

```
PROGRAM := WHEN | PROGRAM WHEN
WHEN := 'when ' EXPRESSION EOL STATEMENTS 'end when' EOL
STATEMENTS := STATEMENT | STATEMENTS STATEMENT
STATEMENT := PRINT | SET
PRINT := 'print '  EXPRESSION_LIST EOL
SET := 'set ' ASSIGNMENT_LIST EOL
EXPRESSION_LIST := EXPRESSION | EXPRESSION_LIST ',' EXPRESSION
ASSIGNMENT_LIST := ASSIGNMENT | ASSIGNMENT_LIST ',' ASSIGNMENT
ASSIGNMENT := VARIABLE '=' EXPRESSION
EXPRESSION := '(' EXPRESSION OP EXPRESSION ')' | VARIABLE
                | NUMBER
OP :=  '<' | '+' | '-' | 'and' | 'or' | 'xor'
VARIABLE := '$' NOT_DOLLAR_STRING '$'
NUMBER := DIGIT | NUMBER DIGIT
DIGIT := '0' | .. | '9'
NOT_DOLLAR_STRING := any sequence of printing characters
                     (including blanks) that does not
                     contain a $ symbol.
```

In the above, any string enclosed in single quotes are to be treated literally. EOL is the end of line.

In words, a program consists of a list of when blocks, which themselves contain set and print statements. Case is ignored for key words and variable names. Spaces are allowed before or after any literal except inside a number. Spaces are allowed in variable names, and each non-empty sequence of spaces is treated as a single underscore, so the following refer to the same variable

```
$Remote Switch#1$
$Remote_Switch#1$
$Remote   switch#1$
```

All variable and literal values are integers between -1000000000 and 1000000000, inclusively. All variables are global and initially zero. The when programs you will be tested on will never have an **`EXPRESSION`** that evaluates to a value outside of this range. The logical operators evaluate to 0 for false and 1 for true, and treat any nonzero value as true.

Running the When program amounts to executing all the active when clauses until none are active. More specifically, the active list of when clauses is initially empty, then the following steps are repeated:

- In the order they appear in the program, the conditions of all when clauses that are not currently active are evaluated. If true, the clause is added to the end of the active list, with its first statement marked as "ready". Each active when clause has one "ready" statement.
- If the active list is empty after this step, the program terminates.
- The "ready" statement from the "current" when clause (initially the first clause in the active list) is executed.
- The statement marked as "ready" is advanced, removing the when clause from the active list if this is the last statement in the "current" when clause.
- The when clause marked as "current" is advanced, cycling to the beginning of the active list if the end is reached.

In other words, inactive when conditions are evaluated to determine if these clauses are added to the active list. Then one statement (set or print) is executed from the current active when clause. If this is the last statement in that clause, it is removed from the active list. One the next iteration, one statement is executed from the next active when clause, etc.

A set statement executes all the assignments concurrently, so that

```
set $x$=$y$,$y$=$x$
```

swaps the values of x and y. The same variable cannot appear twice on the left hand part of the same set statement (so set x=1,x=2 is illegal).

A print statement evaluates and prints the given expressions in the output, separated by commas and followed by a new line. So

```
print 1,(2+3)
```

results in the line

```
1,5
```

in the output.

Input

The input consists of a single syntactically correct program. You may assume that the program will not execute more than 100000 set statements and 100000 print statements.

Output

Print the output produced by executing the given program.

Sample Input

```
When ($Mr. Bill$<5)
   Set $mr._bill$=($mr.  bill$+1),$Y$=($Y$+10)
End When
When ($mr. Bill$<10)
  Set $MR. BILL$=($mr. bill$+1)
  Print $mr. bill$,$Y$
End When
```

Sample Output

```
3,20
6,40
7,40
8,40
9,40
10,40
```