

Интерпретация

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Для любого достаточно мощного оружия и лабораторий, его обслуживающих, требуется программное обеспечение. Поскольку в эру Оппенгеймера компьютеры только зарождались, это утверждение еще не было правдой. Но это не означает, что программирования тогда вообще не существовало — любой повседневный алгоритм можно представить в виде псевдокода.

В лаборатории Оппенгеймера на стене висел псевдокод, выводящий какие-то характеристики ядерной бомбы. Псевдокод устроен довольно просто: он состоит только из блоков, которые мы будем обозначать как `<block>`, и работает только с целыми числами, поэтому Оппенгеймер при необходимости запускает его у себя в уме.

В этом псевдокоде:

1. `<block>` — либо примитивное действие, либо цикл;
2. Цикл имеет вид

```
for <var> = <value1>...<value2> {  
    <block>  
    <block>  
    ...  
}
```

Такой цикл проходит переменной `<var>` по всем значениям от `<value1>` до `<value2>` *включительно*, и для каждой выполняет указанную в своем теле последовательность блоков. Переменная `<var>` при этом может использоваться только внутри этого цикла и только с правой части от знака `'='`.

3. `<var>` — имя переменной;
4. `<value>` — либо имя переменной, либо целочисленная константа;
5. Примитивная операция может быть
 - либо `«print(<var>)»` — напечатать значение переменной,
 - либо `«read(<var>)»` — записать в переменную значение со ввода,
 - либо присвоение значения в переменную в формате `«<var> = <expression>»`
6. `<expression>` — либо имя переменной, либо целочисленная константа, либо арифметическое выражение;
7. Арифметические выражения имеют вид `«<value1> [+ -] <value2>»`, то есть либо сумма, либо разность двух величин.

Для этого псевдокода верно следующее:

- Счетчики циклов уникальные и не используются вне соответствующих циклов.
- Значения счетчиков циклов никогда не изменяются внутри цикла (не стоят слева от знака `'='` и не считываются с помощью `read`).
- Область видимости переменных — глобальная: если какая-то переменная создается, она после этого доступна до конца работы программы.
- Если верхняя граница цикла меньше нижней, цикл не совершает итерации вообще.

Вы — молодой лаборант, не разбирающийся в программировании (действительно, его пока почти что нет). Основные проблемы в понимании у вас вызывают вложенные циклы. Вы не понимаете вложенность, поэтому хотите перевести этот псевдокод на другой язык, чтобы иметь возможность помогать Роберту Оппенгеймеру в работе.

Другой язык поддерживает все те же примитивные операции (ввод, вывод и присвоение значений в переменную), и также имеет глобальную область видимости переменных, но в нем абсолютно нет скобок и возможности делать вложенные циклы. Вместо этого:

- Для вызова `print` или `read` надо написать соответствующую команду *заглавными буквами* и ее аргумент через пробел.
- Для записи значения в переменную надо написать «`<var> GETS <expression>`».

А также в языке есть две специальные команды:

- Команда `MACRO`. Если вы напишете «`MACRO <macro_name>:`», то в следующих строках с отступом в 4 пробела по одной команде на строке можно перечислить последовательность команд (кроме еще одной команды `MACRO`). Имена макро могут состоять только из маленьких латинских букв от 'a' до 'z' и иметь длину от 1 до 10.
- Команда `REPEAT`. Если вы напишете «`REPEAT <macro_name> <value>`», то последовательность команд, ассоциированная с соответствующим макро (который должен быть объявлен выше), будет последовательно выполнена `<value>` раз. Если передать отрицательное количество повторений, макро не будет вызван ни разу.

Переведите код из обычного псевдокода в более понятный вам, без вложенностей. И кто знает, может быть именно вы под руководством Роберта Оппенгеймера совершите очередной прорыв в ядерной физике!

Формат входных данных

В первой строке ввода дано целое число n — количество строк в псевдокоде ($1 \leq n \leq 1000$). Следующие n строк содержат сам псевдокод.

Псевдокод строго отформатирован согласно грамматике из условия, включая пробелы и отступы. Изначально отступ равен 0. Каждый следующий вложенный цикл имеет отступ на 4 пробела больше, чем предыдущий. Каждый цикл содержит символ '{' в конце строки с объявлением и заканчивается строкой, содержащей только символ '}' на нужном отступе.

Имена переменных — строки из маленьких латинских букв от 'a' до 'z' и цифр от '0' до '9' длины от 1 до 10. Имя переменной не может быть равно «for», «print» или «read» и не может начинаться с цифры. Все целочисленные константы неотрицательны и не превосходят 2000.

Если левая граница цикла больше правой, цикл совершает 0 итераций.

Гарантируется, что псевдокод не обращается к переменным, в которые до этого не было записано значение. Также гарантируется, что все используемые в объявлении цикла переменные имеют уникальные имена, не совпадающие с встречавшимися ранее переменными, используются только внутри соответствующего цикла, и не находятся слева от знака '=' или в аргументе операции `read`. **Никакая переменная в исходном псевдокоде не начинается на букву 'x'.**

Формат выходных данных

В первой строке выведите целое число $m \leq 5 \cdot n$ — количество строк в вашем переведенном коде. В следующих m строках выведите код в анти-вложенном псевдокоде в формате, описанном в условии.

Ваш код должен в точности повторять алгоритм, описанный псевдокодом из входных данных. **Не обязательно** добиваться того, чтобы все переменные имели те же имена и итоговые значения — достаточно, чтобы программы выводили одно и то же при условии одинакового ввода.

Все имена переменных в вашем коде должны состоять также из маленьких латинских букв и иметь длину от 1 до 10. Целочисленные константы в вашем коде должны быть неотрицательными и не превосходить 10000.

Примеры

стандартный ввод	стандартный вывод
<pre> 6 n = 1 read(k) for i = 0...k { n = n + k } print(n) </pre>	<pre> 9 n GETS 1 READ k MACRO increase: n GETS n + k iters GETS k + 1 REPEAT increase iters PRINT n </pre>
<pre> 14 read(somevalue) read(morevalue) for i = 0...10 { for j = 1...somevalue { print(j) } smthidk = i wut = 42 for k = 1...morevalue { smthidk = smthidk + k wut = smthidk + smthidk } print(wut) } </pre>	<pre> 23 MACRO printer: PRINT j j GETS j + 1 MACRO summator: smthidk GETS smthidk + k wut GETS smthidk + smthidk k GETS k + 1 MACRO outer: j GETS 1 REPEAT printer somevalue smthidk GETS i k GETS 1 wut GETS 42 REPEAT summator morevalue PRINT wut i GETS i + 1 READ somevalue READ morevalue i GETS 0 REPEAT outer 11 </pre>