Southeastern European Regional Programming Contest
Bucharest, Romania
October 16, 2010

Problem I
Proof Generator

Input File: I.IN
Output File: ordered output
Program Source File: I.C or I.CPP or I.JAVA

A logical formula has the syntax shown in figure 1(a), where a variable stands for a truth value, the formula **(+ F_1...F_n)** stands for the logical disjunction of the formulae **F_i**, **(* F_1...F_n)** denotes the logical conjunction of the formulae **F_i**, and **~F** is the negation of **F**. If a formula has the particular syntax given in figure 1(b) we say that the formula is in the ACM Normal Form (ACMNF).

```
<formula>::= <variable> | ~<formula> |
             (+<formulae>) | (*<formulae>)
<variable>::= a lower case letter from the English
              alphabet
<formulae>::= <formula> |
              <formula><formulae>
```

a) The general syntax of a formula

```
<ACMNF_formula>::= <term> | (+<terms>)
<term>::= <literal> | (*<literals>)
<terms>::= <term><term> | <term><terms>
<literal>::= <variable> | ~<variable>
<literals>::= <literal><literal> |
              <literal><literals>
<variable>::= a lower case letter from the
              English alphabet
```

b) The ACMNF syntax of a formula

Figure 1. Formula syntax

A formula is converted to ACMNF using the rewriting rules given below, where **F** represents a formula, **s** stands for a non empty sequence of formulae, and **s** and **s'** denote possibly empty sequences of formulae. Applying a rewriting rule **q→r** on a formula **F** means to substitute by **r** a part of **F** that matches the pattern **q**, as in shown figure 2. The conversion terminates when no rewriting rule can be applied. The conversion terminates for any formula, and the result is unique regardless which rules are applied on which parts of the formula and in which order.

1. **~~F→F**
2. **~(*FS)→(+~F~(*S))**
3. **~(+FS)→(*~F~(+S))**
4. **(+F)→F**
5. **(+s(+S)s')→(+sSs')**
6. **(*F)→F**
7. **(*s(*S)s')→(*sSs')**
8. **(*s(+FS)s')→(+(*sFs')(*s(+S)s'))**

```
(+(*(+~(*ab)) (+~a) )c)      —4→
(+(*(+ ~(*ab) )~a)c)         —2→
(+(* (+(+~a~ (*b) ))~a)c)    —6→
(+(* (+(+~a~b)) ~a)c)        —4 or 5→
(+ (*(+~a~b)~a) c)           —8→
(+(+(*~a~a)(* (+~b) ~a))c)   —4→
(+(+(*~a~a)(*~b~a))c)        —5→
(+(*~a~a)(*~b~a)c)
```

Figure 2. Converting a formula to ACMNF

A set of axioms is represented as a list **(V_1 V_2...V_n)** of variables that are true. A variable that is not in the list is false. A proof of a formula **F** according to a set of axioms **A** is a term from the ACMNF of **F** such that the term is true according to **A**. For instance, the terms **(*~a~a)** and **c** are the proofs of the formula **(+(*(+~(*ab))(+~a))c)** according to the axioms **(bc)**.

The problem is to code a proof generator that for a given formula **F**, a set of axioms **A**, and a number **k** outputs the next **k** proofs of **F** in the order in which they appear in the ACMNF of **F**. If the proofs are exhausted, the generator continues from the first proof of **F**. For example, generating the first proof of the formula **(+(*(+~(*ab))(+~a))c)** according to the axioms **(bc)** yields **(*~a~a)**. Generating three more proofs produces **c**, **(*~a~a)**, and **c**. If the ACMNF of a formula contains similar terms, as in the last example in figure 3, these terms are considered distinct.

Input	Output
(+(*(+~(*ab))(+~a))c) (bc) −3 1 1 0 (+~x~y~y) () −2 1 0	c (*~a~a) ~y

Figure 3. Input/output sample

Write a proof generator that reads sets of data from the standard input. The content of a data set is **F A k_1...k_n 0**, **n>0**, where **F** is a formula, **A** is a set of axioms, and **k_1...k_n** are long integers different than **0**. For each **k_i**, **i=1,n**, the program generates the next **$|k_i|$** proofs of **F** and, if **k_i>0**, prints these proofs on the standard output. Each printed proof starts from the beginning of a line and there are no white spaces between the characters of the proof. The generated proofs are not printed if **k_i<0**. White spaces are used freely in the input. A formula has at most **500** characters and a ACMNF term is at most **80** characters long, not counting white spaces. The input data terminate with an end of file, and are correct.