# ACM International Collegiate Programming Contest 1997/98

Sponsored by IBM

Supported by WICONA International, Wilken GmbH and GPS

# Southwestern European Regional Contest

## Practice Session

## University of Ulm, Germany

### November 22nd, 1997

This problem set should contain five (5) problems on ten (10) numbered pages. Please inform a runner immediately if something is missing from your problem set.

ACM International Collegiate Programming Contest 1997/98
Southwestern European Regional Contest
Practice Session

# Problem A

## IBM minus one

**Source:** `ibm.(c|C|pas)`
**Input:** `ibm.in`

You may have heard of the book '2001 - A Space Odyssey' by Arthur C. Clarke, or the film of the same name by Stanley Kubrick. In it a spaceship is sent from Earth to Saturn. The crew is put into stasis for the long flight, only two men are awake, and the ship is controlled by the intelligent computer HAL. But during the flight HAL is acting more and more strangely, and even starts to kill the crew on board. We won't tell you how the story ends, in case you want to read the book for yourself :-)

After the movie was released and became very popular, there was some discussion as to what the name 'HAL' actually meant. Some thought that it might be an abbreviation for 'Heuristic ALgorithm'. But the most popular explanation is the following: if you replace every letter in the word HAL by its successor in the alphabet, you get . . . IBM.

Perhaps there are even more acronyms related in this strange way! You are to write a program that may help to find this out.

## Input

The input file starts with the integer *n* on a line by itself - this is the number of strings to follow. The following *n* lines each contain one string of at most 50 upper-case letters.

## Output

For each string in the input, first output the number of the string, as shown in the sample output. The print the string that is derived from the input string by replacing every letter by the following letter in the alphabet, and replacing 'Z' by 'A'.

Print a blank line after each string.

## Sample Input

```
2
HAL
SWERC
```

## Sample Output

```
String #1
IBM

String #2
TXFSD
```

ACM International Collegiate Programming Contest 1997/98
Southwestern European Regional Contest
Practice Session

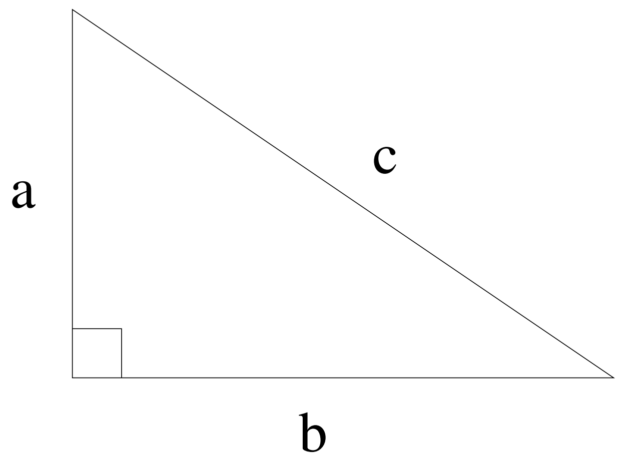
# Problem B

## Geometry made simple

**Source:** `geometry.(c|C|pas)`
**Input:** `geometry.in`

Mathematics can be so easy when you have a computer. Consider the following example. You probably know that in a right-angled triangle, the lengths of the three sides $a$, $b$, $c$ (where $c$ is the longest side, called the hypotenuse) satisfy the relation $a^2 + b^2 = c^2$. This is called Pythagoras' Law.

Here we consider the problem of computing the length of the third side, if two are given.




## Input

The input file contains the descriptions of several triangles. Each description consists of a line containing three integers $a$, $b$ and $c$, giving the lengths of the respective sides of a right-angled triangle. Exactly one of the three numbers is equal to -1 (the 'unknown' side), the others are positive (the 'given' sides).

A description having $a = b = c = 0$ terminates the input.

## Output

For each triangle description in the input, first output the number of the triangle, as shown in the sample output. Then print "`Impossible.`" if there is no right-angled triangle, that has the 'given' side lengths. Otherwise output the length of the 'unknown' side in the format "$s$ = $l$", where $s$ is the name of the unknown side (`a`, `b` or `c`), and $l$ is its length. $l$ must be printed exact to three digits to the right of the decimal point.

Print a blank line after each test case.

## Sample Input

```
3 4 -1
-1 2 7
5 -1 3
0 0 0
```

## Sample Output

```
Triangle #1
c = 5.000

Triangle #2
a = 6.708

Triangle #3
Impossible.
```

## Implementation Hints

In order to print a variable exact to three digits to the right of the decimal point, you can use the following statements:

**In C/C++:** `printf("%.3f",x)`, where `x` is a double variable.

**In Pascal:** `write(x:0:3)`, where `x` is a real variable.

ACM International Collegiate Programming Contest 1997/98
Southwestern European Regional Contest
Practice Session

# Problem C

## Carbon Dating

**Source:** `carbon.(c|C|pas)`
**Input:** `carbon.in`

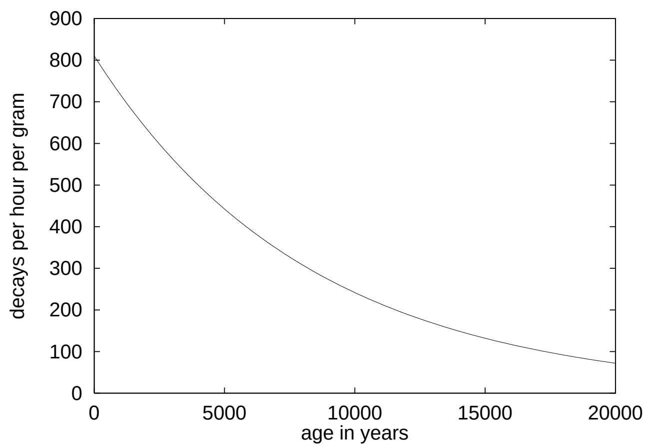
Until the second half of the 20th century, determining the actual age of an archaeological find had been more or less a matter of educated guessing. Comparing finds to previous, already dated, ones and evaluation of the surroundings of a find were the best available techniques.

But nowadays, there is a much more reliable method: carbon dating. Carbon dating works as follows: Naturally occurring carbon is a mixture of stable isotopes (mostly $^{12}C$) and the unstable, radioactive, isotope $^{14}C$. The ratio between the two is almost constant in living organisms: $^{14}C$ slowly decays, but at the same time, the radiation of the sun produces the same amount in the upper atmosphere, which is taken in by the organisms.

But when, for example, a tree is felled and made to wood, it does not receive any new $^{14}C$, and the amount present in the wood becomes less and less due to radioactive decay. In this problem, you are to write a program that uses information about the amount of remaining $^{14}C$ to determine the approximate age of a sample. The following facts must be used:

- The amount of $^{14}C$ present in a sample halves every 5730 years (this is called the half-life of $^{14}C$).
- The rate of decay (measured in decays per hour per gram of carbon) is proportional to the amount of $^{14}C$ left in the sample.
- In living organisms (age zero), there are 810 decays per hour per gram of carbon.

So, for example, if we measure in a sample of one gram of carbon 405 decays per hour, we know that it is approximately 5730 years old.



## Input

The input file contains the measurements taken of several samples we want to date. Every line contains two positive integers, $w$ and $d$. $w$ is the amount of carbon in the sample, measured in grams, and $d$ is the number of decays measured over one hour.

The input is terminated by a test case starting with $w = d = 0$.

## Output

For each sample description in the input, first output the number of the sample, as shown in the sample output. The print the approximate age in the format

`The approximate age is` $x$ `years.`

If the age is less than 10,000 years, $x$ should be rounded to the closest multiple of 100 years (rounding up in case of a tie). If the age is more then 10,000 years, round it to the closest multiple of 1000 years (again rounding up in case of a tie).

Print a blank line after each sample.

## Sample Input[1]

```
1 405
5 175
0 0
```

## Sample Output

```
Sample #1
The approximate age is 5700 years.

Sample #2
The approximate age is 26000 years.
```

[1] More true than ever!

ACM International Collegiate Programming Contest 1997/98
Southwestern European Regional Contest
Practice Session

# Problem D
## URLs

**Source:** `url.(c|C|pas)`
**Input:** `url.in`

In the early nineties, the World Wide Web (WWW) was invented. Nowadays, most people think that the WWW simply consists of all the pretty (or not so pretty) HTML-pages that you can read with your WWW browser. But back then, one of the main intentions behind the design of the WWW was to unify several existing communication protocols.

Then (and even now), information on the Internet was available via a multitude of channels: FTP, HTTP, E-Mail, News, Gopher[2], and many more. Thanks to the WWW, all these services can now be uniformly addressed via *URLs* (Uniform Resource Locators). The syntax of URLs is defined in the Internet standard RFC 1738. For our problem, we consider a simplified version of the syntax, which is as follows:

$$< \mathit{protocol} > \ \texttt{"://"} \ < \mathit{host} > \ \texttt{[} \ \texttt{":"} \ < \mathit{port} > \ \texttt{]} \ \texttt{[} \ \texttt{"/"} \ < \mathit{path} > \ \texttt{]}$$

The square brackets `[]` mean that the enclosed string is optional and may or may not appear. Examples of URLs are the following:

- `http://www.informatik.uni-ulm.de/acm`
- `ftp://acm.baylor.edu:1234/pub/staff/mr-p`
- `gopher://veryold.edu`

More specifically,

$< \mathit{protocol} >$ is always one of `http`, `ftp` or `gopher`.

$< \mathit{host} >$ is a string consisting of alphabetic (`a-z`, `A-Z`) or numeric (`0-9`) characters and points (`.`).

$< \mathit{port} >$ is a positive integer, smaller than 65536.

$< \mathit{path} >$ is a string that contains no spaces.

You are to write a program that parses an URL into its components.

## Input

The input starts with a line containing a single integer $n$, the number of URLs in the input file. The following $n$ lines contain one URL each, in the format described above. The URLs will consist of at most 60 characters each.

[2] The ancestor of today's WWW. Now nearly extinct, but quite important when the WWW was invented.

## Output

For each URL in the input first print the number of the URL, as shown in the sample output. Then print four lines, stating the protocol, host, port and path specified by the URL. If the port and/or path are not given in the URL, print the string `<default>` instead. Adhere to the format shown in the sample output.

Print a blank line after each test case.

## Sample Input

```
3
ftp://acm.baylor.edu:1234/pub/staff/mr-p
http://www.informatik.uni-ulm.de/acm
gopher://veryold.edu
```

## Sample Output

```
URL #1
Protocol = ftp
Host     = acm.baylor.edu
Port     = 1234
Path     = pub/staff/mr-p

URL #2
Protocol = http
Host     = www.informatik.uni-ulm.de
Port     = <default>
Path     = acm

URL #3
Protocol = gopher
Host     = veryold.edu
Port     = <default>
Path     = <default>
```

# Problem E

## Definite Values

**Source:** `definite.(c|C|pas)`
**Input:** `definite.in`

A common error in programming is to use variables that have not been initialized before. For example, in C and C++, all variables have an indefinite value after declaration – their value can be anything. Thus, the following program

```
main()
{
  int x;
  printf("%d\n",x);
}
```

could print any number. But even in languages such as Pascal, where all values are initialized to zero, it is useful to give variables definite values before using them, to avoid side effects when your code portion is placed into a different context.

Generally, the problem of deciding for a given program whether all variables have been assigned values before they are read out, is undecidable. But if you, as in this problem, consider only a sequence of assignments, the problem becomes solvable.

## Input

The input file contains several program parts. Each part starts with a number $n$ on a line by itself, the number of lines in the program part. The following $n$ lines contain each an assignment of the form "$variable_1$ = $variable_2$", where the $variable_i$'s are lower-case letters.

The input is terminated by a test case starting with $n = 0$.

## Output

Assume that before the execution of the given program part, variable `a` has some definite value, while all other variables are undefined. You have to print the names of the variables which have a definite value after the execution of the program part. More specifically, format your output as follows.

For each program part in the input, first print the number of the program, as shown in the sample output. Then print a line containing the names of the variables which have a definite value after the execution of the given program part. Print them in alphabetically sorted order, and leave one blank after each variable name. If none of the variables has a definite value after the execution of the program part, print the word "`none`".

Print a blank line after each test case.

## Sample Input

```
4
b = a
c = d
d = b
e = f
1
a = b
0
```

## Sample Output

```
Program #1
a b d

Program #2
none
```