# TAYLOR
# HIGH SCHOOL
# PROGRAMMING CONTEST

ADVANCED PROBLEM SET

NOVEMBER 2007

**Problem 2.1** **The Dead Center of the Battle** (page 1 of 1)

General Statement: Under siege by the Greek army, the Trojans are desperately trying to defend their city from invasion. After each battle (which lasts many hours), the two sides count their dead. The Trojans use these numbers to determine how many new warriors to recruit for the next day's battle. However, because there are so many dead, the reports from different sources are inconsistent. Lacking formal mathematics education, the Trojan military leader is unable to find the average. Instead, he takes the three numbers given to him by his three generals and finds the median—that is, the number in between the other two.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line of input. Each line contains three integers *a b c*, each separated by a single space.

Name of Data File: adv21.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a single integer *d*, corresponding to the median of *a*, *b*, and *c*. Note that the value of *d* will always be one of either *a*, *b* or *c*.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The values for *a b c* each will be between 0 and 999999999, inclusive.
The values for *a b* and *c* are not exclusive; *a b* and *c* may contain duplicate values.
All input will be valid.

Sample Input:

```
5
6 6 7
1582 1234 1337
6153 6214 10252
8 8 8
70 70 69
```

Sample Output:

```
6
1337
6214
8
70
```

**Problem 2.2** **Aim Isum** (page 1 of 1)

General Statement: Isum, the Sumerian divine messenger, has just received a message from Nergal, Lord of the Underworld. With the message came instructions to deliver it to Enlil, the chief deity, as quickly as possible. However, as Isum is embarrassed to admit, while he is by far the quickest of the gods, his sense of direction is certainly lacking. He would truly appreciate it if, based on his current heading, you could, with the quickest turn possible, point him in Enlil's direction. In return, he'll reward you with fabulous riches (or so he says).

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a pair of uppercase strings *a b*, separated by exactly one space and representing two cardinal directions from the following list: `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, and `NW`. In the order listed, each pair of consecutive cardinal directions is separated by 45 degrees, and the pair between `NW` and `N` is also separated by 45 degrees.

Name of Data File: adv22.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a single integer *d*, representing the smaller degree difference between the two cardinal directions *a* and *b*. If *a* and *b* are in opposite directions, output `180`; if *a* and *b* are the same direction, output `0`. Note that the order of input does not affect the output (that is, an input of *a b* should give the same output as an input of *b a*.)

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The cardinal directions *a* and *b* may be the same.
The value for *d* will be between 0 and 180, inclusive.
All input will be valid.

Sample Input:

```
3
NW S
NE NE
E W
```

Sample Output:

```
135
0
180
```

**Problem 2.3** **The Trojan Horse** (page 1 of 1)

General Statement: In the notorious Trojan War, the leaders of the Greek army conceived an ingenious plan—to build an enormous horse filled with the best Greek warriors. The Trojan horse is to be offered as a gift of surrender to the people of Troy and brought into its impenetrable walls. Once inside, the hidden soldiers are to sneak out and open the gates for the others waiting outside; together they will set fire to the city and conquer Troy. In order to construct the incredibly large wooden animal, the Greek architects must calculate the number of boards of a given size for each length of the horse.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains two decimal numbers *a b*, separated by exactly one space. Each of these numbers *a* and *b* will be of the format *x.yyyyy*, namely, an integer portion followed by a period `.` and exactly five decimal places.

Name of Data File: adv23.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a decimal number *c*, representing the quotient of *a* divided by *b*, rounded to two decimal places.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The values for *a* and *c* each will be between 0.00000 and 999.99999, inclusive.
The value for *b* will be between 0.00001 and 999.99999, inclusive.
All input will be valid.

Sample Input:

```
3
896.27092 72.34242
8.00000 8.00000
0.00000 3.23429
```

Sample Output:

```
12.39
1.00
0.00
```

## Problem 2.4 Close Enough (page 1 of 1)

General Statement: Before the first Olympic Games of Greece, many aspiring men trained on the track to prepare for the big race. Because of their lack of technology, the captains were forced to manually estimate the final position of each contestant on the track. There were hundreds of contestants per trial; perfectly calculating each runner's position was anything but quick, so the captains formed a technique to round each runner's final position on the track to the nearest half lap.

Rounding to the nearest half involves some special rules. Like rounding to the whole number, a number that lies exactly half-way between the two nearest marks rounds up. So, 2.25 would round up to 2.5, and 3.75 would round up to 4.0, while 2.24 would round down to 2.0 and 3.7499 would round down to 3.5.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a decimal (base-10) number *d*.

Name of Data File: adv24.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a decimal (base-10) number *r*, representing the number *d* rounded to the nearest half. Use the above mentioned rules for rounding. The output should be of the form either `#.0` or `#.5`, depending on which half is nearer.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The value for *d* will never be negative.
*d* will always contain the decimal point **.** followed by at least one digit.
All input will be valid.

Sample Input:

```
5
5.4
1.245
7.75
0.0
0.7499999
```

Sample Output:

```
5.5
1.0
8.0
0.0
0.5
```

**Problem 2.5** **Reinventing the Sphere** (page 1 of 1)

General Statement: One day, after conducting his daily sacrifices to the Egyptian sun god Amun-Ra, Pharaoh Amenhotep III was swept away by fervor to imitate the Sun God. Secretly believing that the sun was a sphere, Amenhotep ordered the construction of a spherical burial tomb, in contrast to the pyramidal ones of his predecessors. Concerned that he would run out of building material, he asked for an estimate of the volume of various sizes of his spherical tomb. Since the ancient Egyptians had not yet been introduced to algebra, he offered a hefty prize for anyone who can provide him with an estimate of his tomb's volume.

The volume of a sphere is given by the formula

$$v = 4/3 \pi r^3$$

where v is the volume and r is the radius of the sphere. Although the ancient Egyptians understood the concept of π (pi), they did not have the precision of modern-day computers (3.141592653589793). The most accurate approximation for π would have been 22/7. In the spirit of the ancient Egyptians, use this approximation for π in your calculations.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a single integer *r*, representing the radius of the sphere.

Name of Data File: adv25.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain one integer *v*, representing the calculated volume of the sphere rounded to the nearest integer.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The value for *r* will be between 1 and 100, inclusive.
The value for *v* will not exceed 999999999.
All input will be valid.

Sample Input:

```
3
1
3
7
```

Sample Output:

```
4
113
1437
```

**Problem 2.6** **No Lying!** (page 1 of 2)

General Statement: Baldr, the Norse god of innocence and peace, is convinced that Loki, god of mischief, is behind the attempt on his life, but he has no evidence. However, from his knowledge of recent technology, he has constructed a simple lie detector which he is sure will prove him right. Help Baldr determine if Loki is really to blame.

Lie detectors work by measuring the skin's electrical resistance: when a person is lying, more perspiration results in a decreased skin resistance. Given the highly variable nature of these measurements, however, three initial readings will be taken to calibrate the measuring device (and to account for different levels of skin resistance for different people). Also, the more readings taken, the more accurate the average (arithmetic mean) of the readings will be to the actual skin resistance. The range of acceptable variation from the mean is calculated by 24 divided by the total number of readings taken up to that point, including the three initial calibration readings.

For example, in the first case of sample data, the readings would proceed as follows:
Initial readings: `5 5 5` yield a mean of `5.00`

| Reading Time Frame # `(i)` | Value `(v)` | Mean before reading `(b)` | Mean after reading | Acceptable Range `(24/i)` | Threshold `(b-24/i)` |
|---|---|---|---|---|---|
| 4 | 5.3 | 5.00 | 5.075 | 6.0 | -1.0 |
| 5 | 5.3 | 5.075 | 5.12 | 4.8 | 0.275 |
| 6 | 5.3 | 5.12 | 5.15 | 4.0 | 1.12 |
| 7 | 4.7 | 5.15 | 5.086 | 3.43 | 1.72 |
| 8 | 4.7 | 5.086 | 5.0375 | 3.0 | 2.09 |
| 9 | 4.7 | 5.0375 | 5.00 | 2.67 | 2.37 |
| 10 | 0.1 | 5.00 | LIE DETECTED: 0.1<2.37 | | |

The lie was detected at time frame 10, because the detected value was less than the acceptable threshold (`v<b-24/i`).

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a series of readings. The first line of each data collection contains a number *m* representing the number of readings that follow. The next *m* lines each contain a single decimal number *v*, representing a reading of the instrument in a single time frame.

Name of Data File: adv26.dat

Time Allocation: 1 second

**Problem 2.6** **No Lying!** (page 2 of 2)

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain an integer *t* representing the time frame in which a lie was first detected. Output `-1` if the readings in the data collection do not indicate that a lie was detected.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The value for *m* will be between 4 and 1000, inclusive.
The values for *v* will not be less than 0 or greater than 10000.
There may be more readings in a data collection after a lie is first detected; these later readings should be completely ignored.
The calibration process starts anew for each different data collection.
All input will be valid.

Sample Input:

```
2
10
5
5
5
5.3
5.3
5.3
4.7
4.7
4.7
0.1
4
17
17
17
1809
```

Sample Output:

```
10
-1
```

**Problem 5.1** **Scheduling Nightmare** (page 1 of 1)

General Statement: Mannah Hontana, deity of preteen girls in Pop culture, scheduled to perform a concert annually on November 10th, in honor of Dalt Wisney's becoming an FBI informant to report on Hollywood subversives in 1940. Sadly, due to prior commitments, such as filming her hit TV show on the Chisney Dannel, recording albums, and presenting at award shows around the world, chances are scarce. Thus, Mannah can only perform on Nov. 10th on the same day of the week as her original concert. Mannah's young, devoted fan club members are desperate to know when this next year is! Unfortunately, they can't read calendars; in exchange for an honorary membership, find the year of her next show.

Leap years are calculated by the following formula: every year evenly divisible by 4 is a leap year, except that every year evenly divisible by 100 is not a leap year. On top of this, however, every year evenly divisible by 400 is a leap year. So, the years 1996, 2000, and 2004 all would be leap year, but the years 1998, 2100, and 2501 are not leap years. A leap year has an extra day in the month of February.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains four-digit integer *a*, representing a year.

Name of Data File: adv51.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a single four-digit integer *b*, representing the next year (after year *a*) in which November 10 will fall on the same day of the week as it did in year *a*.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The values for *a* and *b* will be between 1582 and 9999, inclusive.
All input will be valid and will be based on the modern (Gregorian) calendar.

Sample Input:

```
3
2000
2002
2004
```

Sample Output:

```
2006
2013
2010
```

**Problem 5.2** **Odyssean Encryption** (page 1 of 1)

General Statement: Odysseus is stranded on an uncharted island in the middle of the Mediterranean Sea. Worse yet, he cannot escape because the Evil Naga, with its über-pointy trident, is guarding the island. Odysseus tried asking for help from Athena, but the Evil Naga simply intercepted all his messages and stopped them from reaching Athena. Odysseus finally devised a simple way to encode his message by removing the overlapping portions between pairs of words to generate a single word. Thinking his message was gibberish, the Evil Naga allowed his message to pass. However, even Athena can't understand his message! Help her decode Odysseus' message so she can aid his escape.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a pair of strings. The first line of each data collection will contain string *a*. The second line of each data collection will contain string *b*. Both strings *a* and *b* will contain only the lowercase characters between `a` and `z`, inclusive.

Name of Data File: adv52.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a single string *s* representing the joining of non-overlapping portions of the two strings *a* and *b*. The overlap is determined by the largest value of *m* such that the last *m* characters of string *a* exactly match the first *m* characters of string *b*.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The lengths for strings *a b* and *s* each will not exceed 100.
All input will be valid.

Sample Input:

```
3
helpers
ersme
escapeade
ade
ody
sseus
```

Sample Output:

```
helpme
escape
odysseus
```

**Problem 5.3** **O! It's Big!** (page 1 of 2)

General Statement: The ZIA, Zeus Intelligence Agency, has picked up a leak that Hades is about to unleash havoc upon all the major cities of the world. Wasting no time, the agency calls in their special agent Hermes, the messenger between the gods and the humans. Hermes' task is to warn all the cities of the imminent danger; therefore he has to find the most effective way to travel to all the major cities. Knowing only the Big-O and the results of one previous test, help him figure out how long each method of travel will take him.

Note: Big-O is a computer science concept that describes how a method's running time is related to its size. It is customary to assign the variable n to the initial size. Then, given a Big-O of $n^2$, doubling the input size (2n) would multiply the running time by four ($2^2=4$), while tripling the input size (3n) would multiply the running time by nine ($3^2=9$). A Big-O of $n^3$ would mean that doubling the input size (2n) would multiply the running time by eight ($2^3=8$), while tripling the input size (3n) would multiply the running time by twenty-seven ($3^3=27$). A similar logic can be used for other Big-O classes and other new input sizes. Note that a Big-O of n is equivalent to saying a Big-O of $n^1$, that is, a linear relationship. For example, in the first case of sample input, increasing the size (distance) from 5 to 100 (a 20-fold increase) results in an increase in time from 10 to 200 (a 20-fold increase with Big-O of n). With a Big-O of $n^2$, however, the increase in time is from 10 to 4000 (a 400-fold increase in time, as calculated from $20^2=400$).

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a string *s* followed by three integers *a b c*, each separated by exactly one space. String s is the Big-O of each method of travel, containing either a single lowercase character `n` (representing a Big-O of n) or the characters `n^#`, where `#` represents a single digit between 2 and 9 inclusive. Integer *a* represents the distance he traveled on his test. Integer *b* represents the time it took to travel distance *a*. And integer *c* represents the distance he will have to travel to visit all the cities.

Name of Data File: adv53.dat

Time Allocation: 1 second

**Problem 5.3** **O! It's Big!**

Output: Your program should produce n lines of output (one for each data collection). Each line should contain a single integer *d* representing the time it will take Hermes to travel distance *c* based on the Big-O of his travel method.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The value for *a b c* each will not exceed 1000.
The value for *d* will not exceed 999999999.
Integer *c* will always be evenly divisible by *a*.
All input will be valid.

Sample Input:

```
3
n 5 10 100
n^2 5 10 100
n^5 5 7 20
```

Sample Output:

```
200
4000
7168
```

**Problem 5.4** **Grappling with the Greeks** (page 1 of 2)

General Statement: The mighty Mauryan empire has just lost a battle to Seleucus, one of Alexander the Great's generals. Indra, the mighty king of the Hindu gods is angered at their loss. To assure that the Indians never lose another battle, he charters the Mauryans to create a battle formation to defeat the Greeks. The Indians work diligently to create a perfect formation, but cannot seem to create a fitting one. The desperate Indian king, Chandragupta the Great, turns to you for help. You are praying to Saraswathi, the goddess of wisdom, for inspiration, when an idea suddenly strikes you! You decide to use the Greek's phalanx against Seleucus; however you decide to make it smaller, sleeker, more deadly and efficient than its Greek counterpart. Smiling to yourself, you begin to work.

The original Greek phalanx was a 7-column by 4-row (7 × 4) grid of warriors. You, a brilliant military genius, understood that a 5 × 3 phalanx would be far more effective in battle due to its enhanced mobility. After making this change you realize you should remove the last warrior in each row, so you are left with a 4 × 3 Phalanx. Now that you have done this, you re-arrange the modified Phalanx in alphabetical order so the people can be easily identified by officers and field commanders. In order to secure your position on the battle field, you need to output the finalized phalanx formation.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a phalanx. Each data collection will contain 3 lines, each containing 5 strings separated by a single space. These strings, in order, represent the names of people in the phalanx.

Name of Data File: adv54.dat

Time Allocation: 1 second

Output: Your program should produce 3*n* lines of output (three for each data collection). The output for each data collection should consist of 3 lines of output, each line containing the names of the 4 people remaining in each row of the phalanx, after applying the above transformations. Separate each name by exactly one space.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 100.
Each name will be a single uppercase character followed by one or more lowercase characters.
All input will be valid.

**Problem 5.4** **Grappling with the Greeks**

Sample Input:

```
1
Lakshman Ajay Achintya Tarun Maneesh
Pankaj Anand Hemachandra Ulysses Oberoi
Parmesh Ekanga Ekalinga Tej Sankalp
```

Sample Output:

```
Achintya Ajay Anand Ekalinga
Ekanga Hemachandra Lakshman Pankaj
Parmesh Tarun Tej Ulysses
```

**Problem 5.5** **A Hexing Conversion** (page 1 of 1)

General Statement: Make-make, creator of the Rapa Nui of Easter Island, is in economic turmoil! The Moari, who inhabited the island before the Rapa Nui, utterly devastated the island by cutting down all the trees to serve as rollers for transporting their giant head statues. Now, the Rapa Nui are desperately in need of lumber. However, the closest source of lumber is Mesoamerica, and their cultures use an entirely different currency system. Make-make has taken it upon himself to conduct trade with Cochimetl, the Aztec god of commerce, in order to relieve his people. However, in order to conduct this trade, Make-make needs your help converting his people's hexadecimal-based currency into the Aztec's octal system. Beware, the scale of their trading can be huge!

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a hexadecimal (base 16) string *h*, which will contain only the characters `0123456789ABCDEF` and represents a nonnegative number to be converted into octal (base 8). No spaces will be present in the input line. The string *h* will not begin with the character `0`; no leading zeroes will appear in the input.

Name of Data File: adv55.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a single string *o* containing only the characters `01234567`, representing the octal (base 8) equivalent of string *h*. The string *o* should not begin with the character `0`; no leading zeroes should appear in the output.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The length of string *h* will be between 1 and 500 characters, inclusive.
The length of string *o* will be between 1 and 1000 characters, inclusive.
All input will be valid.

Sample Input:

```
3
DECADE
1337ACE
10000
```

Sample Output:

```
67545336
114675316
200000
```

**Problem 5.6** **The Octagonal Market** (page 1 of 2)

General Statement: The Roman king has decided to redesign the layout of this town market to accommodate the many new shops arriving to the area that have no room to conduct business. This size of the circular area allowed for the market suggested that an octagonal layout would be most efficient. The king assigned the local city planner to arrange the shops along the new octagonal path, but the only knowledge he received of each shop was one random character.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a string *s*, indicating the input string to be outputted in octagon format. Preserve all punctuation, casing, and whitespace.

Name of Data File: adv56.dat

Time Allocation: 1 second

Output: Your program should produce an indefinite number of lines of output (multiple lines for each data collection). Left-justify all output.

Start on the top edge, at the upper left corner. Print the sentence character by character clockwise, following the border of a regular octagon. Your octagon should be the smallest regular octagon that will contain all the characters of the sentence without overlap. If there are fewer characters in the input string than character spots available in the smallest regular octagon, output an asterisk `*` for each available character spot after the end of the input. Each character that joins two sides counts toward the number of characters for both sides.

Trailing empty spaces on each line will not be judged. However, the spaces at the front of each line and between the first and last characters in each line are necessary for alignment, and those spaces must be exact. So, using `#` to denote a space, the first line of the sample output can be outputted as `#DE` or `#DE#`, but the leading space cannot be missing, as in `DE` or `DE#`. Likewise, the second line of the third sample output can be either `#####*#######i#####` or `#####*#######i`, but the leading and internal spaces must be exact. Of course, any spaces that appear in the input string *s* must also be exact.

No empty lines should be outputted between the outputs for consecutive data collections.

The output is to be formatted exactly like that for the sample output given below.

**Problem 5.6** **The Octagonal Market** (page 2 of 2)

Assumptions: The value for *n* will not exceed 100.
The length of string *s* will be between 1 and 80 characters, inclusive.
All input will be valid.

Sample Input:

```
3
DESIGN
Octagons
The efficiency of using an octagon is high.
```

Sample Output:

```
 DE
*  S
*  I
 NG
 Oc
s  t
n  a
 og
      The eff
     *       i
    *         c
   *           i
  *             e
 *               n
.                 c
h                 y
g
i                 o
h                 f

s                 u
 i               s
                i
   n           n
    o         g
     g
      atco na
```

**Problem 9.1** **Reverse Polish Notation**

General Statement: One day while working on your math homework, you decided to take a break and stroll through a nearby forest, calculator still in hand. While wandering through the forest, you encounter Baba Jaga, the Slavic witch of the forest and master of magic. She is angry that you have invaded her forest, and casts a spell on you. You run away and are relieved that the spell seems to have done nothing to you. However, when you start working on your math homework again, you are shocked to find that your calculator is not working properly. It is now operating in Reverse Polish Notation, and you must figure out how to use it to finish your assignment.

Reverse Polish Notation (abbreviated RPN) is a form of postfix—that is, operators appear after the operands are inputted. RPN uses the concept of a stack, in which numbers are pushed onto the stack and removed in a “last in, first out” manner. There are two modes in RPN: number input mode, and calculation mode. Here is a summary of what certain keystrokes do, in the two different modes:

| **Keystroke** | **Number input mode** | **Calculation mode** |
|---|---|---|
| A digit 0 through 9 (digit *d*) | Appends the digit *d* to the end of the number in memory. | Switches to number input mode, starts a new number in memory with the digit *d* as its only digit. |
| An operator `+-*/^` (operator op) | Switches to calculation mode. Pops the last two numbers off the stack into *y* and *x* respectively, and then computes *y* op *x*. Pushes this computed value onto the stack. | Pops the last two numbers off the stack into *y* and *x* respectively, and then computes *y* op *x*. Pushes this computed value onto the stack. |
| The `ENTER` key | Switches to calculation mode. Pushes the number in memory onto the stack. | Pops the last number on the stack and pushes it onto the stack twice (duplicates the last number). |

**Problem 9.1** **Reverse Polish Notation**

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a list of calculator keystrokes. The first line of each data collection contains an integer *m* that represents the number of keystrokes in the data collection. The next *m* lines will each contain exactly one of the following:

- A single digit `0123456789`, representing a keystroke of that digit
- A single-character operation `+-*/^`, corresponding to the following operations in RPN respectively: addition, subtraction, multiplication, integer (truncate) division, and exponentiation
- A single all-capitalized word `ENTER`, signifying either the end of the input of the previous number (if it follows a digit) or the duplication of the last number in the stack (if it follows an operation or another `ENTER`)

Name of Data File: adv91.dat

Time Allocation: 1 second

Output: Your program should produce an indefinite number of lines of output (one or more lines for each data collection).

Output the integers in the calculator's memory stack, beginning with the lowest layer of the stack. That is, output the numbers beginning with the earlier numbers inputted or calculated.

No empty lines should be outputted between the outputs for consecutive data collections.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The values for *n* and *m* each will not exceed 1000.
All numbers on the stack at any given time will not exceed 999999999.
All operations will be called only when two or more numbers are on the stack.
The calculator starts in number input mode with empty memory.
The first keystroke of each data collection will always be a digit to start number input.
The word `ENTER` will not be the first line of input.
All data collections will end with an `ENTER` or an operation; no data collections will end in the middle of number input.
All input will be valid.

**Problem 9.1** **Reverse Polish Notation**

Sample Input:

```
3
9
3
4
ENTER
7
ENTER
ENTER
*
*
ENTER
4
5
ENTER
3
/
24
6
ENTER
2
ENTER
3
ENTER
6
ENTER
9
ENTER
+
-
8
ENTER
*
ENTER
ENTER
/
/
8
ENTER
+
*
-
```

Sample Output:

```
1666
1666
1
182
```

**Problem 9.2** **Special Sorting** (page 1 of 2)

General Statement: After killing Baldur, Loki goes into hiding and reviews his life and all the mischievous deeds he did. After looking at them all, he decides to assign his own code to sort them from Exceedingly Evil, to Slightly Silly.

As he is going about his business with sorting, he is captured and forced to face Eternal Punishment until Ragnarok comes, and all of his lists get out of order, and he summons you, a random being, to help him. Since in today's technology people are lazy and can write elegant programs to do things for them, you must do the same here.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains two sets of strings. The first line of each data collection contains an integer *m* specifying the number of elements in each set. The next *m* lines will each contain two strings *a* and *b*, each separated by a single space and each containing no whitespace internally. All strings *a* belong to set *A*, and all strings *b* belong to set *B*.

The strings in set *A* are to be sorted by the following method:

- The sum of the ASCII values of the characters in the string (strings with lower sums are placed before those with higher sums)
- The length of the string (shorter strings are placed before longer strings)
- The value of the `compareTo` method in the `String` class (`String x` is placed before `String y` if `x.compareTo(y)<0`)

Regardless of its final position in the sorted set *A*, string *a* still corresponds to the string *b* that was initially inputted on the same line as that string *a*.

Strings *a* and *b* will consist only of alphanumeric characters.

Name of Data File: adv92.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a string *s*, composed of the concatenated strings *b* in the order of the sorted strings in set *A*. No spaces should be outputted.

The output is to be formatted exactly like that for the sample output given below.

**Problem 9.2** **Special Sorting**

Assumptions:

The value for *n* and *m* each will not exceed 100.
The lengths of strings *a* and *b* each will not exceed 100 characters.
Within each data collection, no string *a* will be duplicated.
All input will be valid.

Sample Input:

```
2
3
Anne or
Ann Ta
Anmf yl
2
IIII ks
AaAA roc
```

Sample Output:

```
Taylor
rocks
```

## Problem 9.3 Determinate That Determinant (page 1 of 2)

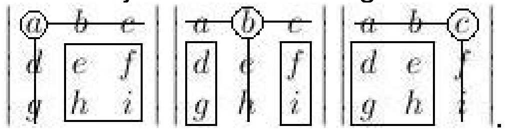
General Statement: Odin is convinced that the Valkyries are being lazy and have stopped bringing the slain Norse warriors to Valhalla. However, he does not know how to prove it. Odin has only the data of the number of warriors brought to Valhalla by the Valkyries, which he has entered into matrices. Unfortunately, he does not know how to find the determinant, which, if found, would prove the Valkyries' sloth. Odin needs your help to find the determinant and catch the lazy Valkyries. To assist you in your task, Odin obtained the following information about matrices and determinants from his preeminent mathematician:

A matrix can be considered a table of numbers. The determinant is a special operation for matrices with equal numbers of rows and columns. The determinant of a 2 × 2 matrix is defined to be

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

That is, the determinant is the product *a*×*d* minus the product *b*×*c*, where $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ refers to the determinant of the 2 × 2 matrix formed by the numbers *a*, *b*, *c*, and *d*. Notice that the determinant is a number.

Determinants of any matrix of size *m* × *m*, with *m*>2, are calculated by the following process. Start with the top-left corner and move across the top row. For each element, cross out the elements in the matrix in the same column and row and join the remaining elements together, as in



The remaining elements form a matrix of size *m-1* × *m-1*. Multiply the determinant of this smaller matrix with the element in the top row of the original matrix. The determinant of the original matrix is formed by adding and subtracting these products, alternating operations for each term and beginning with subtraction between the first two terms.

Thus, the determinant of a 3 × 3 matrix can be calculated as follows:

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$
$$= (aei - afh) - (bdi - bfg) + (cdh - ceg)$$

Likewise, the determinant of a 4 × 4 matrix can be calculated as follows:

$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{vmatrix} = a\begin{vmatrix} f & g & h \\ j & k & l \\ n & o & p \end{vmatrix} - b\begin{vmatrix} e & g & h \\ i & k & l \\ m & o & p \end{vmatrix} + c\begin{vmatrix} e & f & h \\ i & j & l \\ m & n & p \end{vmatrix} - d\begin{vmatrix} e & f & g \\ i & j & k \\ m & n & o \end{vmatrix}$$

Determinants of larger matrices are calculated in a similar manner.

**Problem 9.3** **Determinate That Determinant**

Notice that if any of the elements on the top row is 0, then the associated determinant does not need to be calculated—0 times any number is still 0.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a matrix. The first line of each data collection contains an integer *m* indicating the dimensions of the matrix. The next *m* lines will each contain *m* integers $a_1$ $a_2$ $a_3$ … $a_m$ each separated by exactly one space. The integers inputted will be inserted into a blank $m \times m$ matrix from left to right, with each line of input corresponding to a row in the matrix.

Name of Data File: adv93.dat

Time Allocation: 3 seconds

Output: Your program should produce *n* lines of output (one for each data collection). Each line of output should consist of an integer *d* representing the determinant of the matrix in the corresponding data collection.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 100.
The value for *m* will be between 2 and 10, inclusive.
The value for each element $a_i$ will be between -999 and 999, inclusive.
The value for *d* will be between -999999999 and 999999999, inclusive.
All input will be valid.

Sample Input:

```
3
4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
2
87 78
87 78
5
1 3 4 2 1
999 999 999 999 999
5 4 7 2 1
1 2 3 4 5
1 2 3 4 4
```

Sample Output:

```
1
0
19980
```

**Problem 9.4** **The Sphinx** (page 1 of 2)

General Statement: For the past 200 years, the Sphinx has asked the same riddle, and after having devoured the thousands of passersby who failed to answer her question, she is incredibly bored. To relieve her monotony, the Sphinx decides to try asking something a bit easier—a series of Boolean expressions. Since each expression has only one solution—true or false—she figures that passersby will have a fairly good chance of guessing the solutions, even if they do not actually know them.

The next day, Oedipus passes by the Sphinx on the road to Thebes. Being of peculiar intellect, he manages to answer the Sphinx's initial Boolean expressions instantly. Suddenly feeling a resurgence of blood-thirst and fearing the loss of her once-impeccable reputation, the Sphinx decides to stump Oedipus by using all four Boolean operators, along with parentheses and order of operations. Help save Oedipus from certain death by evaluating the Sphinx's puzzling Boolean expressions.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection lies on a single line. Each line contains a Boolean expression of unspecified length. The uppercase characters `T` and `F` will be used in the Boolean expression to represent the terms true and false, respectively. The order of operations is as follows:

`()` Evaluate the expression in parentheses first. Parentheses may be nested; evaluate from the most inner set of parentheses first.
`!` Evaluate the term immediately following the NOT operator according to the rules `!T=F` and `!F=T`
`&` Evaluate the terms immediately preceding and following the AND operator according to the rules `T&T=T`, `T&F=F`, `F&T=F`, and `F&F=F`
`^` Evaluate the terms immediately preceding and following the XOR operator according to the rules `T^T=F`, `T^F=T`, `F^T=T`, and `F^F=F`
`|` Evaluate the terms immediately preceding and following the OR operator according to the rules `T|T=T`, `T|F=T`, `F|T=T`, and `F|F=F`

No characters beside `TF()!&^|` will appear in the lines of input.

Name of Data File: adv94.dat

Time Allocation: 1 second

**Problem 9.4** **The Sphinx** (page 2 of 2)

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain one uppercase character, either `T` or `F`, indicating the result of evaluating the corresponding Boolean expression.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for *n* will not exceed 1000.
The `!` operator will immediately precede only the characters `(` `T` or `F`.
The length of the inputted expression will not exceed 500 characters.
The input may or may not have an operator.
All input will be valid.

Sample Input:

```
5
T&F|F&F|T&(F|!F)
F&T
!F
!T^F
((!T))
```

Sample Output:

```
T
F
T
F
F
```

**Problem 9.5** **It's Dodgebolt Time!** (page 1 of 2)

General Statement: Zeus is angry that Hermes stands around and flaunts his golden shoes all day. To "motivate" Hermes to move, Zeus decides to throw lightning bolts at him. Forced to dodge the bolts, Hermes finds his shoes render him nearly equal in speed to the lightning bolts. However, in order to conserve energy while dodging the lightning bolts, he must use the least number of left/right moves.

Here are the rules:

1. Hermes can always choose not to move left or right.
2. Bolts fall from top to bottom.
3. Hermes can move left or right at most one grid unit per turn.
4. Each turn, Hermes chooses to move or stay put before the bolts fall. Thus, he cannot move into a bolt on the left or right, even if that space would be vacated when the bolt falls in the same turn.
5. If a bolt hits Hermes, he dies.
6. Hermes lives if he navigates successfully from the starting point to any location on the final row of the grid.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a grid of bolts. The first line of each data collection contains two integers *y x* separated by a single space. The integers represent the height and length of the maze. The next *y* lines will each contain *x* characters, with each character representing a grid unit.

There will be exactly one starting position, denoted by the uppercase `S`, in the *y*th line of grid input for each data collection.
There may be any number of ending positions, or there may not be any ending positions. A valid ending position is any location without bolts on the first line of grid input for each data collection.
Grid units without bolts are denoted by a period `.`.
Grid units with bolts are denoted by an asterisk `*`.

Name of Data File: adv95.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain one integer *m*, indicating the minimum number of left/right moves necessary to win the game.

It may be impossible for Hermes to live; in that case, output `-1`.

The output is to be formatted exactly like that for the sample output given below.

**Problem 9.5** **It's Dodgebolt Time!**

Assumptions:

The value for *n* will not exceed 100.
The value for *y* will be between 2 and 99 inclusive.
The value for *x* will be between 1 and 99 inclusive.
The value for *m* will not exceed 999999999.
All input will be valid.

Sample Input:

```
3
5 5
.....
**.*.
...**
.**..
..S..
2 1
.
S
13 10
****..**..
..........
***.*.**..
..........
***.****..
..........
**.****.**
..........
**.******.
..........
..........
..........
......S...
```

Sample Output:

```
-1
0
6
```

**Problem 9.6** **Cretan Confusion** (page 1 of 2)

General Statement: One of King Minos' architects came up with the brilliant idea to build a multi-storied labyrinth. If a person (or monster) could get lost for a lifetime inside a one-story labyrinth, think how much better a multi-storied labyrinth would be! However, the architect made the changes to the labyrinth overnight, without telling King Minos. Much to Minos' dismay, the next morning, he became disoriented and entered the maze, mistaking it for his magnificent house.

Minos discovered that there were 6 cardinal directions he could go: left, right, up, down, in, and out. He could not move diagonally, but could achieve the same effect by moving in these cardinal directions. Because of the hasty construction, some of the walls had crumbled, potentially blocking the entrance from which Minos entered. Direct Minos to the quickest exit out of the labyrinth…if there is one!

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection contains a labyrinth. The first line of each data collection contains three integers *z y x* separated by a single space. The integers represent the height, length, and width of the labyrinth. The next *z* sets of *y* lines will each contain *x* characters, with each character representing a grid unit of the labyrinth.

There will be exactly one starting position, denoted by the uppercase `S`.
There will be exactly one ending position, denoted by the uppercase `E`.
Valid locations are denoted by a period `.`.
Walls (invalid locations) are denoted by an asterisk `*`. Minos cannot move into, through, or over a wall in any dimension.
The starting and ending positions are, obviously, valid locations.

Name of Data File: adv96.dat

Time Allocation: 3 seconds

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a string *m*, indicating the path of minimum length in the six cardinal directions to reach the exit.

The following uppercase characters should be used in the output:

| | |
|---|---|
| `L` | Represents a move to the left, in the *x* dimension |
| `R` | Represents a move to the right, in the *x* dimension |
| `U` | Represents a move up, in the *y* dimension |
| `D` | Represents a move down, in the *y* dimension |
| `I` | Represents a move "into" the labyrinth (from one level into another one that is inputted after it), in the *z* dimension |
| `O` | Represents a move "out of" the labyrinth (from one level into another one that is inputted before it), in the *z* dimension |

**Problem 9.6** **Cretan Confusion**

No other characters, including spaces, should appear in the output.

The labyrinth may or may not have a solution; that is, there may be no path of valid moves from start to end. In that case, output in all capital letters `NONE` (no punctuation or spaces).

The output is to be formatted exactly like that for the sample output given below.

Assumptions:

The values for *n* will not exceed 100.
The values for *x y* and *z* each will not exceed 10.
The length of string *m* will not exceed 999 characters.
Each data collection will have exactly one correct output—either the single correct shortest path or the word `NONE`.
All input will be valid.

Sample Input:

```
2
3 4 3
***
*S*
*.*
...
.*.
*..
***
***
...
.*.
...
..E
3 4 3
***
*S*
*.*
...
.*.
*.*
***
***
...
.*.
...
..E
```

Sample Output:

```
IRIDD
NONE
```