TAYLOR
HIGH SCHOOL
PROGRAMMING CONTEST

NOVICE PROBLEM SET
NOVEMBER 2007

Problem 2.1 **Battle Cry** (page 1 of 2)

General Statement: By the beard of Zeus! Hades has conquered Mount Olympus, pillaging the homes of lesser deities and ravaging the gorgeous community. Hades and his hellish zombie army scared away many of the gods with their shrill, bone-chilling battle cry, “By the beard of Zeus!” However, some deities were dozing away the docile afternoon, and failed to detect the distant battle-cries and impending chaos. Hades then captured these unfortunate gods and impressed them into service, expanding his militaristic might and threatening Zeus’s domain. To expel the invaders, Zeus started amassing an army, the likes of which had never been seen before. Humans, gods, fairies, nymphs, animals, mystical beasts – all were sought after and recruited by Zeus. Even Hercules agreed to join the cause and rally with Zeus’s army.

Immediately Zeus implemented a harsh training regimen to prepare for a battle of epic proportions – Hades’ Waterloo. For hours and hours, for weeks and weeks, Zeus and his minions worked doggedly to prepare for one final showdown. Bows, swords, spears, shields, and arrows – all had to be stockpiled in preparation for total war.

On the eve of battle, Zeus and his army celebrated their last night. How many would be standing at the end of the next day, nobody knew. Ambrosia abounded, spirits splashed, food flowed, joy prevailed. Men, gods, beasts, and animals all partook in the festivities. As the night died down, every warrior went home, well aware of the gravity of the next day.

As the sun peaked over the angelic, serene, snow-capped Mount Olympus, Zeus gathered his army outside of Olympia’s gates. Battle plans were iterated and recursed. The battle’s execution needed to be flawless. Any error would result in the extermination and subjugation of Zeus’s army, and the capitulation of Mount Olympus to Hades. Zeus – astride a gallant phoenix with brilliant feathers and terribly vicious talons, and prepared for the momentous battle – rallied his troops with one last, formidable battle cry.

Houses shook, trees rustled, animals fled, and the earth trembled. To this day, the shrill and intrepid cry echoes in the rocks of Mount Olympus. Hades, awakened by a deep, visceral fear, knew that Judgment Day had arrived. Immediately, he gathered his forces and mounted a quick defense. Hades and his forces waited, as the regimented sound of marching footsteps drew ever nearer. Suddenly, a barrage of arrows – a harbinger of Zeus’s forces – rained upon Hades’ army. It had begun. The men braced themselves for the impending onslaught. A torrent of blood would flow that day. Hades rallied his men, and, in one final endeavor, charged the battle field, shouting their distinctive battle cry…

Problem 2.1 **Battle Cry** (page 2 of 2)

Output the battle cry of Hades' army.

Input: No input for this problem

Name of Data File: nov21.dat (No input; the file will be blank)

Time Allocation: 1 second

Output: Your program should produce exactly the following sample output.

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

Assumptions: Do not output any leading or trailing spaces.

Sample Input: No input for this problem

Sample Output:

```
By the beard of Zeus!
```

Problem 2.2 **The Right Way to Gamble** (page 1 of 1)

General Statement: Zeus and Hades were gambling with a die in a heavenly garden full of lilacs when Hades lost for the 666th time in a row and set fire to the last deck of Hello Kitty cards—they had run out of gambling mechanisms! As Hades sheepishly glanced about himself, thinking that there would be one more deck left, he failed to find any. He almost broke down into tears under Zeus' grin (his pride kept his stoic facial expression intact) because the rule stated that once all the cards were gone, the debtor had to pay up. What prevented Hades from crying like a little girl without her Barbie was that he realized you could divinely intervene and save him 20 billion mesos (roughly 100,000 USD).
If you say, "One, Two, Three, Jango!" Zeus will be hypnotized and Hades will be able to get away without paying up and instead, he can use the money to buy Persephone a 500th anniversary present!

Input: No input for this problem

Name of Data File: nov22.dat (No input; the file will be blank)

Time Allocation: 1 second

Output: Your program should produce exactly the following sample output.

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

Assumptions: Do not output any leading or trailing spaces.

Sample Input: No input for this problem

Sample Output:

```
One, Two, Three, Jango!
```

Problem 2.3 **The Deathless Beasts** (page 1 of 1)

General Statement: Zahhak, the dark summoner, has conjured an army in order to take over the Persian lands. To defend his kingdom, the Persian King has ordered his most fearless and adamant fighter, Rostam, to command the Persian army. After leading the army for many days through rough terrain, Rostam finally confronts Zahhak and his summoned two headed salamander warriors. Initially thinking he could easily defeat the enemy, Rostam charges with his men into battle. After many hours, the battle seems to be almost over; however, Zahhak with a swift twist of his hand resummons his whole army, one even mightier than his previous. Finally realizing Zahhak's power, Rostam calls for a wizard who can stop him from casting more spells. Kaveh the Bloodless Magician comes to Rostam's help with his spellbook. He finds the three spells that must be cast in order to silence Zahhak; however Kaveh is mute and therefore needs you to print out the spells for him.

Input: No input for this problem

Name of Data File: nov23.dat (No input; the file will be blank)

Time Allocation: 1 second

Output: Output the three spells cast by Kaveh the Bloodless, each on a new line. Your program should produce exactly the following sample output.

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

Assumptions: Do not output any leading or trailing spaces.

Sample Input: No input for this problem

Sample Output:

```
Ava Kadab\ra
Kuba\balara
Bee\tara lumanos
```

Problem 2.4 **Ending an Endless Sleep** (page 1 of 2)

General Statement: Up in the heavens, the great Greek God Zeus practiced his accuracy with shooting his lightning bolts. However, one day, Zeus made a mistake while practicing and completely missed his target. The thunderbolts rained down and caused great destruction on the Greeks; his aim was so bad that the lightning destroyed the water and food supplies of the Greeks completely. Quickly, Zeus put the whole Greek civilization to sleep using his mysterious and fabulous powers. He decided to keep it this way until he could undo the damage he had caused. Unfortunately, Zeus completely forgot how to wake the Greeks up.

The Gods quickly hold a meeting to fix this problem. They come to the conclusion that speaking of what the Greeks have lost will break the spell of eternal slumber. As the Gods send out their messengers to try this method, Zeus researches the cure. He finds that the key words of breaking the spell have to be very general. If the message contains the key word "food" or "drink," Zeus will not have to worry about waking those people, but those that don't hear the words will still sleep. Zeus will have to search within the messengers' messages for these words to determine if he needs to wake them himself or not.

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*, representing the sentence in which the words `drink` and `food` will be searched.

Name of Data File: nov24.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 lowercase word, either `awake` or `asleep`. Output `awake` if *s* contains any instances of the words `drink` or `food`, in any order. Otherwise, output `asleep`.

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

Assumptions: The value for *n* will not exceed 1000.
If the word `drink` or `food` appears in the sentence, it will not appear as part of another word.
Punctuation may immediately follow an occurrence of the word `drink` or `food`.
All input will be valid.

Problem 2.4 **Ending an Endless Sleep**

Sample Input:

```
5
The Greeks enjoy food.
The Egyptians did not drink from the Nile.
I do not drink much food.
I do, however, love cookies.
Isn't water the best drink?
```

Sample Output:

```
awake
awake
awake
asleep
awake
```

Problem 2.5 **Who Wants Leftovers?** (page 1 of 1)

General Statement: !Xu, the sky god of the !Kung people in Southern Africa, is in a bind. The wet season is nearly here, but !Xu has a major problem he needs to resolve: every year, he distributes supernatural powers to new shamans of his people's tribes. He gives each of the new shamans an equal number of powers. Unfortunately, he's no god of mathematics, and sorely needs your help. !Xu knows how many people he has to distribute his limited number of powers among, but he needs your help in determining how many he'll have left over after distributing the powers evenly.

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 integers *a b*, separated by exactly one space. Integer *a* represents the dividend; integer *b* represents the divisor.

Name of Data File: nov25.dat

Time Allocation: 1 second

Output: Your program should produce *n* lines of output (one for each data collection). Each line of output should contain a single integer *r*, representing the remainder when *a* is divided by *b*. Note that *r* should always be less than *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 values for *a* and *b* will be between 1 and 1000, inclusive.
All input will be valid.

Sample Input:

```
5
5 2
9 9
7 28
28 7
17 3
```

Sample Output:

```
1
0
7
0
2
```

Problem 2.6 **Troubled Thoth** (page 1 of 1)

General Statement: After a long day of taking extensive notes of the happenings of the Egyptian underworld, Thoth carelessly left his scrolls of papyrus out in the open. Wanting to take revenge for his murder, Seth, the god of destruction, decided to cause some trouble. Thoth's system of note-taking is unique: he takes excessive notes and then records numbers that indicate where the essential bits of the message are. Seth decided that he would change these numbers, forcing Thoth to wade through all of his copious notes to find the important parts again. Seth, luckily, did not randomize the numbers, but simply used a different organizational system. Seth's numbers correspond to the number of letters in the important message, starting from the end of entire message. Help Thoth reorganize his notes using Seth's system.

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 lines of input. The first line of each data collection contains a single integer *m* indicating the number of characters to be outputted. The second line of each data collection contains a string *s* indicating the string from which the last *m* characters will be outputted.

Name of Data File: nov26.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 *o*, consisting of the last *m* characters of string *s*.

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 *s* will not exceed 100 characters.
The value for *m* will not exceed the length of string *s*.
All input will be valid.

Sample Input:

```
3
5
Apples baby charlie delta
3
The ambrosia much flavor has
4
Alas have many elephant sit gold
```

Sample Output:

```
delta
has
gold
```

Problem 5.1 **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: nov51.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 5.2 **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: nov52.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 5.3 **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: nov53.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 5.4 **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: nov54.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 5.5 **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: nov55.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 5.6 **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: nov56.dat

Time Allocation: 1 second

Problem 5.6 **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 9.1 **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: nov91.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 9.2 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: nov92.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 9.3 **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: nov93.dat

Time Allocation: 1 second

Problem 9.3 **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 9.4 **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: nov94.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 9.4 **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 9.5 **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: nov95.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 9.6 **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: nov96.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 9.6 **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
```