Problem 0 – Sample Problem

To get you started and see if you can use the programs and do some of the basics for this competition, we want you to do a slightly more advanced version of "Hello World". The difference between this version and your normal version is that you will say hello to multiple specific people instead the whole world.

Input: The input for this problem consists of a header line that tells you how many people you will say hello, to followed by one line for each person that lists their name.

Output: For the output of this problem you should print "Hello " followed by the name of the person you are saying hello to for each name listed.

Sample Input:
4
Mark
Jason
Tyler
Andy

Sample Output:
Hello Mark
Hello Jason
Hello Tyler
Hello Andy

Problem Set

Trinity University ACM
High School Programming Competition
April 9th, 2005

Problem 0 - Shuffler

(Don't all good things start counting as 0?)

Mike is having a night of poker with friends. Just for the occasion Mike went out and purchased a new deck of cards. When the friends get there Mike breaks open the new cards (which are sorted by value and suite) and begins to shuffle. Mike is quite proud of his shuffling abilities and has an uncanny ability to not only cut the deck perfectly in half, but also to get the cards to go perfectly one over the other from alternating halves. Being a bit nervous he shuffles the deck a few more times than he needs to. As he hands the deck across for the cut it slips from his hand and the cards slide face-up across the table. To everyones amazement, especially Mike's, the cards are in perfect sorted order, just as they had started. As everyone stares dumbfounded at the cards, Mike begins to envision a program to help him figure out what is going on.

Typically you shuffle a deck of cards to mix up the cards. However, if you could always shuffle the cards "perfectly", you would find that shuffling too many times can take you right back to the configuration that you started with. By perfectly we mean that the deck is split into two even groups and that the first card down always comes from what had been the bottom half, then the next card comes from the top half and they alternate all the way through. So if your deck started with cards numbered 1 to 8 in order, you would split them into 1-4 and 5-8. Then after one shuffle the cards would be in the following order: 1, 5, 2, 6, 3, 7, 4, 8. How many times you have to shuffle the deck to get back to your original configuration depends on how many cards are in the deck. For this problem you are supposed to write a program to figure out how many times that is.

Input: The input for the problem will consist of a header line telling you how many input sets there will be. Each input set consists of one line with a single number. That number is the number of cards in the deck. It will always be an even number greater than 0 and less than or equal to 1000.

Output: Your output should have one line for each input set. That line will tell how many times a deck of the given size has to be shuffled to get back to the original configuration using the shuffling described above.

Sample Input:

```
3
2
4
52
```

Sample Output:

```
1
2
8
```

Problem 1 – Questionable Calculations

In an effort to make students' lives more difficult, the Trinity Math Department has recently decided to write all of their exams using Reverse Polish Notation. This news came as a shock to student Johnny Hax, who felt the math department was needlessly making students jump through hoops. Never being one to follow the rules, Johnny has decided to program a calculator that will use Reverse Polish Notation to use in his exams. He is fairly sure that the problem can be easily solved using a stack; however, he is at a loss as to how to program it. Ethical considerations aside, Johnny Hax has asked you to help him write his calculator.

Johnny explains that the calculator need only be able to compute addition, subtraction, multiplication or division ('+', '-', '*', '/'). He informs you that in Reverse Polish Notation all the operations are listed after the operands they operate on, thereby eliminating the need for parentheses or order of operations. So for example, "(1+2) * 3" would be written as "1 2 + 3 *" in Reverse Polish Notation. He explains that every time a number is entered, it is "pushed" onto the stack. When the calculator gets to the operations, two numbers are "popped" off the stack, the operation is performed, and the result is pushed back on. At the end of the calculation, the result is the top of the stack. If the expression was well defined then that will be the only item on the stack.

Input: The input will consist of a header line that specifies how many input sets will follow. Each input set will consist of a single line that contains the string of numbers and operators to be calculated. The numbers can be fractional and all operations should be done with doubles.

Output: The output is simply the result of the computation.

Sample Input:

```
3
1 2 + 3 *
3 5 + 7 2 - *
1 2 + 4 * 3 +
```

Sample Output:

```
9.0
40.0
15.0
```

Problem 2 – Maximized Studying

The end of the semester is finally upon you, and with it comes finals. Unfortunately, you have played way too much "World of Warcraft" this semester and done way too little studying. As a result, you now sit facing your locker on a Friday afternoon looking at the stack of books that you need to take home to cram study in the last weekend before finals. While you have incredible faith in your ability to store inordinate amounts of information just long enough for it to matter, you don't have as much faith in your backpack which you are quite certain cannot hold all the books you need to take home.

Thankfully you know exactly how much weight your bag can carry, exactly how much each book weighs, and exactly how much the material in each book will count towards your final grade set. So to make the most of your sad situation, you quickly whip out your laptop and write a program that will determine the largest fraction of your grade you can save by carefully selecting books that won't exceed the carrying capacity of your bag but that will maximize the amount of relevant material you can study.

Input: The input for your program will consist of a header line telling you how many input sets there are. Each input set will consist of the following. The first line has two integers for how many pounds your bag can carry and how many books you have in your locker. There will never be more than 10 books. That line is followed by one line of input for each book where each line gives the book title (single word) followed by the weight of the book in pounds and the fraction of your grades it will contribute to. All numbers are non-negative integers.

Output: The output of your program will consist of one number for each input set that tells the maximum fraction of your grade you can actually carry home with you.

Sample Input:

```
2
10 3
History 7 30
Math 4 40
English 6 30
10 2
CliffNotes 5 99
OfficialText 8 1
```

Sample Output:

```
70
99
```

Problem 3 – Some Assembly Required

Keller found a great game online this last week. Or at least the description sounds great. The only problem is that it was written in CSIR assembly and Keller doesn't have a machine built on the CSIR architecture, nor can he find an emulator for it. So Keller has decided to write his own assembly interpreter for the CSIR assembly language, which happens to be fairly simple. Your job is to help him get started on this.

For this problem you will just do the basic mathematical operation of ADD, SUB, MULT, and DIV. The machine has 4 integer registers for storing values and have a very simple instruction format.

OP destination source1 source2

In this, OP is one of the four operations listed above (all in caps), the destination is a register, and the sources are either registers or numeric literals. A register is always denoted with a $ followed by the number of the register (zero referenced so the number is in the range 0-3). A number can be any positive or negative integer in the range of a Java int. The operator and operands are space separated. So the instruction "ADD $3 $2 1" will add 1 to the value in register 2 (the third register) and store that in register 3 (the fourth register). You can think of this line as being something like $3=$2+1.

By convention, all the registers begin with a value of 0 stored in them. You will write code that will read in a series of instructions, then you must print out the values of the four registers on one line. Arithmetic is done with standard integer arithmetic so division truncates. You will never be given an input that performs a division by zero.

Input: The input will consist of one single "program". The first line of the input tells you how many assembly lines there are in that program. That will be followed by the proper number of lines using the format described above. The last line will say EXIT.

Output: The output will be a single line with 4 numbers on it. Those numbers are the values of the register in order.

Sample Input:

```
ADD $0 $0 3
SUB $1 7 $0
MULT $2 $0 $1
DIV $3 $2 6
EXIT
```

Sample Output:

```
3 4 12 2
```

Problem 4 – Words Findable?

Pat is a publisher of children's puzzle books. One of the standard puzzle styles in Pat's books is the word find. Recently one of the content creators has gotten into the bad habit of giving lists of words that contain words that aren't actually present in the word find. He does good work generally so Pat doesn't want to have him stop making puzzles, but one can't be publishing word finds that list words that can't be found. People get really frustrated with stuff like that.

Pat has asked you to write a program to help with this. The program should take a word find rectangle and a list of words and print out only the words that are actually present in the word find. The rectangle is simply a grid of characters and words can appear long any straight line in the puzzle and run in any of eight directions (horizontally, vertically, and diagonally going forward or backward).

Input: The input for this problem will consist of a single word find puzzle and a list of words. The input begins with a line that has two numbers which represent the number of row and columns in the word find. Both numbers will not be greater than 20. That is followed by a number of lines of strings. Each string has the same number of characters as the word find has columns and the number of strings is equal to the number of rows in the word find. After that is a line with a number telling you the number of words you are looking for, followed by a list of words, one on each line. All letters in the word find and in the words will be lower case.

Output: The output for this problem is a list of words, giving only the words from the list that were actually in the word find.

Sample Input:

```
4 4
have
evod
fleo
kidc
6
code
have
dove
fun
do
trinity
```

Sample Output:

```
code
have
dove
do
```

Problem 5 – One Dollar Words

Becky has just opened a new photo store for developing pictures, but is unsure of how much to charge her customers. She remembers from her childhood that a picture is worth a thousand words and given the expense of opening her store, she could really use the income. Words could be priced as the sum of each of their letters' position in the alphabet (`a = 1¢, z = 26¢`). As a result she decides that every time a customer comes to pick up their pictures, she will have a conversation with them to determine the price she will charge. It isn't long before she realizes that no one is willing to pay for every word their picture is worth, so she marks down pictures to only cost the words priced at exactly $1.00.

Unfortunately, Becky is not good at adding things up in her head, so she asks you to write a program which will calculate the price of a word. The program should take a word, add all the letters of the word together with the assigned value as given above, and report whether or not the word equals $1.

Input: The input for this problem should read in a sentence (without punctuation) and then perform the necessary operations on each word of the sentence. The input might contain capitals. You should ignore that and only consider position in the alphabet so 'a' and 'A' are both 1¢.

Output: The output for this problem should be a list of words whose value is equal to $1.

Sample Input:

```
The old shipyard was smitten with asbestos
```

Sample Output:

```
shipyard
smitten
asbestos
```

Problem 6 – Afternoon Off

School just let out for the day and you have arrived home with a stack of homework. The problem is that your house is simply full of different diversions that tend to take you away from your studies. When you get home you have 6 hours before you have to go to sleep. The question is whether you manage your time well. For this problem we will tell you how many hours of homework you come home with and how many hours of entertainment you partake in and you will tell us whether you did a good job or not.

Input: The input starts with a line telling you how many data sets there are. Each data set contains one line which has two numbers on it. The first number is the number of hours of homework you need do and the second number is how many hours you actually spend doing things other than homework that evening.

Output: For each input set you will output one line. If you can't fit all the homework in your 6 hour evening along with your other activities then you should print "Would you like fries with that?". However, if your free activities were limited enough so you can also get all your homework done, then print "All work and no play leads to a high paying job."

Sample Input:

```
3
4 1
3 3
3 5
```

Sample Output:

```
All work and no play leads to a high paying job.
All work and no play leads to a high paying job.
Would you like fries with that?
```

Problem 7 – Possible Palindrome?

As you are inevitably well aware, a palindrome is a string where when all the spaces and punctuation is removed, it is the same forward and backward, ignoring case. A simple example that everyone seems to use is the word “radar”. For this problem you aren't going to tell us whether a string is a palindrome, instead you will tell us whether the characters in the string could be arranged to form a palindrome. So the string “darar” would qualify because it could be rearranged to be “radar”.

Input: There will be multiple data sets. The first line of the input will have the number of data sets. Each data set will have one line that is a string you need to check to see if it could be turned into a palindrome. It could have spaces or punctuation which you should ignore.

Output: For each data set you will output one line that says either “Could be a palindrome.” or “Can't be a palindrome.”

Sample Input:

```
3
darar
This is a test. Tea ist siths!
This won't work.
```

Sample Output:

```
Could be a palindrome.
Could be a palindrome.
Can't be a palindrome.
```

Problem 8 – strfry Words

Back to our friend Pat, the children's book publisher. Pat is having problems with another content creator. This time, it is the woman who writes the word scrambles. You know, the ones where you are given letters out of order and you have to figure out what word they spell. Well, this time the solutions that Pat is being provided with don't always work with the clues that are given in the puzzle. Now, one would think that the content creator could simply put in the word and run strfry (a standard C library function to generate permutations of a string) on it to get the strings to put in the puzzle. However, this content creator likes to move letters to make more complex patterns and sometimes forgets to swap them with other letters. What Pat wants from you is a program that can take the scrambled work and the solution word and tell whether or not you can make the solution from the scrambled word so the puzzles can easily be checked.

Input: The input will consist of multiple data sets and the first line of input tells you how many data sets there are. Each data set will be one line long and will have two words on it, the scrambled word and the provided solution word. The words will only contain lowercase letters.

Output: For each data set you will print out one line of text. That line will either be "Properly fried." or "Puzzle overcooked."

Sample Input:

```
3
zradil lizard
nipkpox knoppix
java coffee
```

Sample Output:

```
Properly fried.
Properly fried.
Puzzle overcooked.
```

Problem 9 – No Place Like Home

Graphy the gopher has gotten lost, and has forgotten how to get home, or even if he can. What you need to do is write a program that will bring hope back to poor lost Graphy, and tell him if there is a path from his current gopher hole back to his home.

You are given the number of holes other than Graphy's home; there are no more than 5 other holes. Graphy always starts at hole 0 and his home is always hole 5. Each hole has one or more tunnels leading to other holes. The successive lines list the holes each hole leads to. Some tunnels are too steep for Graphy to go both ways. If a tunnel can be traveled in either direction, that tunnel will appear in the list of both of the rooms that it connects. For example, if the tunnel from hole 0 goes to hole 1 and back then the line for hole 0 will have a 1 on it and the line for hole 1 will have a 0 on it. It is also possible for a tunnel to loop back around to the same hole. The holes are numbered starting with 0 and ending at one less than the total number of holes (5 is never represented).

Input:
Input begins with the number of different data sets that you need to process. This is followed by the inputs for the data sets. Each data set has the following format.
<Number of Holes>
<Hole 0 Destination 1> <Hole 0 Destination 2>
<Hole 1 Destination 1> <Hole 1 Destination 2>
…
<Hole n-1 Destination 1> <Hole n-1 Destination 2>

Output: Output is a simple YES or NO for each data set depending on if he can arrive home.

Sample Input:
```
2
3
1 2
2
0 2 5
2
0
5
```

Sample Output:
```
YES
NO
```