

**Java program:** Prob01.java

**Input File:** Prob01.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Little Johnny would like help determining the value of the money in his piggy bank. He will provide you with the number of quarters, dimes, pennies, nickels, and half dollars he has in the bank. You need to tell him the total value. And no cheating little Johnny! The input file will contain a list of coins found in the piggy bank and the number of those types of coins. Coins will be in no particular order. The task is to determine the amount of money represented by the group of coins in the bank.

**Program Input**

The file Prob01.in.txt will contain a list of items and their quantities. There will only be one item and quantity pair per line, although duplicate items are allowed on different lines. Items and quantities will be separated by an equal sign with no spaces. There may also be non-coin items in the list that you should ignore. Valid coin names are: HALFDOLLAR, QUARTER, DIME, NICKEL, and PENNY.

**Example Input:**

QUARTER=31

DIME=5

NICKEL=2

DIME=7

HALFDOLLAR=0

PENNY=157

**Program Output**

Your program’s output should display the total value of the money in the bank in standard US currency format, using a dollar sign before the dollar amount and two decimal places for the number of cents.

**Example Output:**

$10.62

**Java program:** Prob02.java

**Input File:** Prob02.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

A neighborhood street has an equal number of houses on both sides of the street. Every house receives exactly one newspaper (either the Times or the Herald). You will be told how many houses on one side of the street subscribe to the Times, and how many houses on the other side of the street subscribe to the Herald. Your task is to figure out which newspaper has the bigger subscribership and how many more subscribers that paper has.

**Program Input**

The file Prob02.in.txt will contain subscriber data. Each line in the file describes a new block, and will contain two numbers separated by a space. The first number is the number of Times subscribers on one side of the street, and the second number is the number of Herald subscribers on the opposite side of the street.

**Example Input:**

22 9

6 14

12 12

**Program Output**

If your program calculates that the number of subscribers is equal for both newspapers, it should output the following message:

Times and Herald have the same number of subscribers

If your program calculates that the number of subscribers is not equal for both newspapers, it should output the following message:

[Newspaper name] has [total difference] more subscribers

Where

[Newspaper name] = "Times" or "Herald" (excluding quotes)

**Example Output:**

Times has 26 more subscribers

Herald has 16 more subscribers

Times and Herald have the same number of subscribers

**Java program:** Prob03.java

**Input File:** Prob03.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Write a program that will take a list of integers and determine whether they are in ascending, descending or random order. Integers can be positive or negative. If you encounter a non-integer value, then the entire line should be considered invalid input.

**Program Input**

The file Prob03.in.txt will contain some number of lines of characters separated by spaces. Each line should be treated as a new input set.

**Example Input:**

5 6 9 13 17 15

1 2 3 4 5

-1 -2 -3 -4 -5

1 2 3 4 =5

**Program Output**

Your program should output one of the following messages:

* The numbers are in random order
* The numbers are in ascending order
* The numbers are in descending order
* The input was invalid

**Example Output:**

The numbers are in random order

The numbers are in ascending order

The numbers are in descending order

The input was invalid

**Java program:** Prob04.java

**Input File:** Prob04.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

You have been asked to decode a secret message. A General Decoder takes a set of input strings. The first string is the English alphabet encoded in the order needed to decode the subsequent strings of numbers.

For example, the sequence:

18-5-1-4 20-8-9-19 13-5-19-19-1-7-5

Can be decoded by applying the alphabet order:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

To read:

READ THIS MESSAGE

Your task is to write a program that will take as inputs a decoding key and an encoded message, and output the decoded message.

**Program Input**

The file Prob04.in.txt will contain the alphabet decoding key on the first line followed by any number of lines of encoded messages. Numbers correspond to the position of the letter they represent in the decoding key, and are separated by either a dash or a space. Dashes are used to separate letters within a word, and spaces denote word boundaries. There will be no numbers less than 1 or greater than 26.

**Example Input:**

AEIOUYBCDFGHJKLMNPQRSTVWXZ

22-24-4 20-4-1-9-21 9-3-23-2-20-11-2-9 3-17 1 24-4-4-9 1-17-9 3

3 22-4-4-14 22-12-2 4-17-2 15-2-21-21 22-20-1-23-2-15-2-9 7-6

1-17-9 22-12-1-22 12-1-21 16-1-9-2 1-15-15 22-12-2 9-3-10-10-2-20-2-17-8-2

**Program Output**

Your program should output the decoded message.

**Example Output:**

TWO ROADS DIVERGED IN A WOOD AND I

I TOOK THE ONE LESS TRAVELED BY

AND THAT HAS MADE ALL THE DIFFERENCE

**Java program:** Prob05.java

**Input File:** Prob05.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

With interest rates at historic lows, many people are seeking to refinance their mortgages. But, in order to do that, they have to know how much they owe on their current mortgage. Your task is to write a program that will figure out the remaining principal due on a fixed rate loan. Here are some definitions and formulas pertaining to loans that you might find useful when writing your program:

A fixed-rate loan has equal monthly payments. To calculate the monthly payment M, you can use this formula:

Where:

* P is the original principal of the loan
* J is the monthly interest rate of the loan (J = I / 1200, if I is the yearly interest rate)
* N is the number of months it takes to pay off the loan

Payments are made at the end of each month. To calculate the remaining principal for a given month x, you can use this formula:

Where:

* Px-1 is the remaining principal for the previous month.
* M is the monthly payment amount
* J is the monthly interest rate

**Program Input**

The file Prob05.in.txt will contain loan information, with each new line denoting a new loan. Values on each line will be separated by spaces. The file will be in the following format:

OriginalPrincipal YearlyInterestRate TotalMonths MonthsPaid

The Original principal and yearly interest rate values will be floating point numbers, and the total months and months paid values will be integers.

**Example Input:**

100000 4.75 60 37

150000 6.00 120 18

**Program Output**

Your program must print the remaining principal amount (rounded to the nearest cent) after making the Xth monthly payment.

**Example Output:**

41157.60

132805.23

**Java program:** Prob06.java

**Input File:** Prob06.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

In general, a palindrome is a word, phrase, number, or other sequence of characters that can be read the same way in either direction. The following are all examples of palindromes:

* 1234321
* Never odd or even
* Level
* Too bad – I hid a boot

Your task is to write a program that will check to see if a line of input is a palindrome or not. Any character that is not a letter or a number can be ignored, and case does not matter.

**Program Input**

The file Prob06.in.txt will contain any number of lines of text. Each line should be treated as a new possible palindrome.

**Example Input:**

Test

12321

Lisa Bonet ate no basil

Geese migrate south

**Program Output**

For each line of input, your program should simply state whether or not the line is a palindrome by outputting either "yes" or "no".

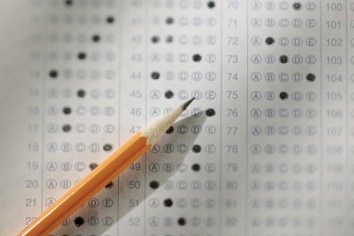
**Example Output:**

no

yes

yes

no

**Java program:** Prob07.java

**Input File:** Prob07.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Scantrons! You know, sheets of paper used to mark your answers on multiple choice tests. You can select A, B, C, D, or E by filling in that little bubble. In this exercise, you will be writing a program to grade students’ scantron papers so the teacher doesn’t have to do it manually.

**Program Input**

The file Prob07.in.txt will contain three sections:

1. The first line of the file will be an integer number telling you how many points each question is worth.
2. The next set of lines will be the key to the test. Blank columns are indicated by an "O", and the bubbled in column is indicated by an equal sign. Columns are separated by a single space. Multiple answers may be bubbled in.
3. The student response section will consist of any number of students’ answers. Each student section will contain the following:
   1. The first line of each student’s section will be in the format "STUDENT #", where the pound sign will be an integer number indicating the student’s ID number. The word student and the ID number will be separated by a single space.
   2. The next set of lines will be the student’s responses to the questions on the test. Questions are answered in order, and only an exact match to the key will receive credit for the question. There is no partial credit.

**Example Input:**

10

O O O O =

O = O O O

O O O O =

= O O O O

= O O O O

O = O O O

O O = O O

O O O = O

O = O O O

O O O O =

STUDENT 12

O = O O =

O = O O O

O O O O =

= O O O O

= O O O O

O = O O O

O O = O O

O O O = O

O = O O O

O O O O =

STUDENT 13

O O O O =

O = O O O

O O O O =

= O O O O

O O O = O

O = O O O

O O = O O

O O O = O

O = O O O

O = O O O

**Program Output**

Your program should output each student’s id number along with their grade in the following format:

STUDENT #: Grade

**Example Output:**

STUDENT 12: 90

STUDENT 13: 80

**Java program:** Prob08.java

**Input File:** Prob08.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Sometimes it’s just nice to take words and make pictures out of them. Your task is to take a phrase and print it out to look like a square or a rectangle.

**Program Input**

The file Prob08.in.txt will contain a list of phrases, one per line. There will be no punctuation or other non-letter characters in each phrase – only words separated by spaces.

**Example Input:**

TIME FLIES LIKE AN ARROW

Squares are fun

**Program Output**

Your program should print out each phrase’s square or rectangle with a blank line in between each to separate the phrases. Use the following rules for determining how to print your squares and rectangles:

* To prevent the end of the phrase from touching the beginning, you should insert a space at the end of each phrase if there is not one there already. Add the minimum possible additional spaces as necessary to fill out your square or rectangle.
* All spaces in the phrase (and added at the end) should be represented in your output by periods.
* Arrange the resulting text in a rectangle/square starting at the upper left corner and proceeding in a clockwise direction. In the event that your phrase forms a rectangle instead of a square, the width of the rectangle must be one greater than the height.

**Example Output:**

TIME.FLI

. E

. S

W .

O L

R I

RA.NA.EK

Squar

. e

n s

u .

f.era

**Java program:** Prob09.java

**Input File:** Prob09.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

You are looking for words that are spelled almost exactly the same but have one letter different (e.g., air and fir). The two words must be the same length (e.g., man and many would not be a match, but man and men would be). The letters in both words must be in the same sequence (e.g., miles and smite would not be a match, but smile and smite would be).

**Program Input**

The file Prob09.in.txt will contain lines of input with a list of words on each line. Words will be separated by spaces. Capitalization should be ignored when comparing if the words are one letter different or not, but should be retained for alphabetization purposes. In the event that there are duplicate words, the first instance of the word should be kept and the rest of the instances discarded.

**Example Input:**

band bar bard bend dent fir forest forestay forestry lame land lime man many mile miles smile smite Air

**Program Output**

Your program’s output should consist of a list of word pairs where the two words meet the above conditions. The pairs of words should be in alphabetical order (i.e., the words within a matching pair should be alphabetized, and then the whole list should be alphabetized). A word can appear more than once in the output list if it matches more than one other word. Each word pair should be printed on a separate line.

**Example Output:**

Air fir

band bard

band bend

band land

forestay forestry

lame lime

smile smite

**Java program:** Prob10.java

**Input File:** Prob10.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

A factorial is the product of a given positive integer (n) multiplied by all lesser positive integers and is represented as n! For example:

4! = 4 x 3 x 2 x 1 = 24

Your task is to write a program which will calculate factorials for numbers from 1 to 50.

**Program Input**

The file Prob10.in.txt will contain a list of numbers, one per line. All numbers will be integers between 1 and 50, inclusive. There will be no non-integer input.

**Example Input:**

5

10

15

**Program Output**

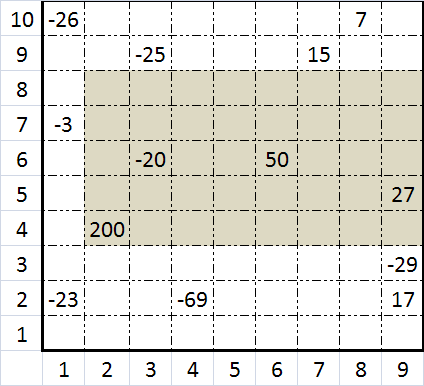
Your program should output the values of the factorials for the numbers encountered in the input file in the same order, one per line.

**Example Output:**

120

3628800

1307674368000

**Java program:** Prob11.java

**Input File:** Prob11.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

A farmer wants to buy a rectangular plot of land, but some parts of the land are worth more than others. The land has been partitioned into a grid, with the value of each grid section given in thousands of dollars (see figure). Grid locations with no value have a value of $0.

Your task is to write a program that will find the most valuable rectangular plot of land that is the largest in size (represented by the shaded area in the figure).

**Program Input**

The file Prob11.in.txt will contain data about the land. Numbers in the file will be separated by commas. The file will contain two sections:

1. The first line will give the total size of the available land for purchase. The number of columns will be listed first, and the number of rows will be listed second.
2. The rest of the file will contain land value data in the following format:

column,row,value

The value will be an integer, representing the value of the land in thousands of dollars.

The "origin" will be at the bottom left of the land, so for a plot of land with C columns and R rows:

* Point 1,1 will be the grid section at the bottom left of the land
* Point C,R will be the grid section at the top right of the land

**Example Input:**

9,10

1,2,-23

4,2,-69

9,2,17

9,3,-29

2,4,200

9,5,27

3,6,-20

6,6,50

1,7,-3

3,9,-25

7,9,15

1,10,-26

8,10,7

**Program Output**

Your output should contain the following three lines:

1. The location of the bottom left corner of the land the farmer should purchase.
2. The location of the top right corner of the land the farmer should purchase.
3. The total value of the plot of land that the farmer should purchase, with a dollar sign before the value and a lower case k after value.

**Example Output:**

2,4

9,8

$257k

**Java program:** Prob12.java

**Input File:** Prob12.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

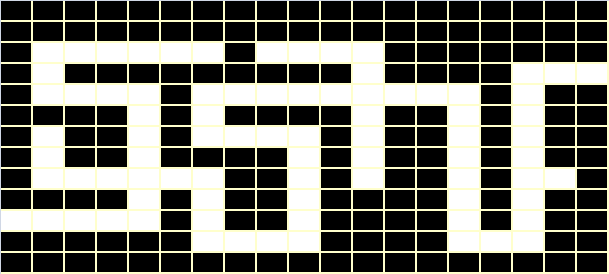
To help your team navigate through the dense jungle, you will be presented with the layout of the terrain as captured by an aerial photo. Your job is to write a program to map out the one and only route to pass completely through the jungle.

**Program Input**

The file Prob12.in.txt will contain a representation of the aerial photo of the jungle. Each line represents a new row of data. An X represents impassible jungle, and an O represents a navigable path.

**Example Input:**

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXX

XOOOOOOXOOOOXXXXXXX

XOXXXXXXXXXOXXXXOOO

XOOOOXOOOOOOOOOXOXX

XXXXOXOXXXXOXXOXOXX

XOXXOXOOOOXOXXOXOXX

XOXXOXXXXOXOXXOXOXX

XOOOOOOXXOXOXXOXOOX

XXXXOXOXXOXXXXOXOXX

OOOOOXOXXOXXXXOXOXX

XXXXXXOOOOXXXXOOOXX

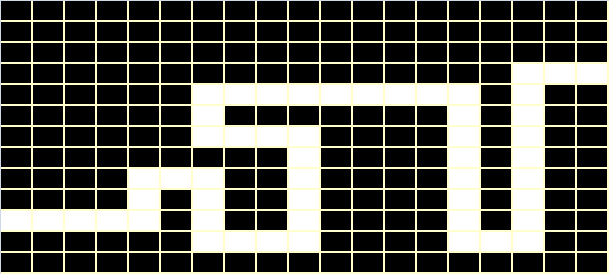
XXXXXXXXXXXXXXXXXXX

**Program Output**

Your program should output a representation of the jungle with all paths that do not belong to the maze solution removed and replaced with an X. That way, if your team were to navigate through the jungle you would know which way to go at each intersection. There will be no loops on the correct path, and any orphaned paths (i.e., paths that do not connect to the maze solution) should also be removed. Only horizontal and vertical movement is permitted.

**Example Output:**

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXOOO

XXXXXXOOOOOOOOOXOXX

XXXXXXOXXXXXXXOXOXX

XXXXXXOOOOXXXXOXOXX

XXXXXXXXXOXXXXOXOXX

XXXXOOOXXOXXXXOXOXX

XXXXOXOXXOXXXXOXOXX

OOOOOXOXXOXXXXOXOXX

XXXXXXOOOOXXXXOOOXX

XXXXXXXXXXXXXXXXXXX

**Java program:** Prob13.java

**Input File:** Prob13.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Grading tests is hard – just ask your teachers! Your task is to write a program that will read in an answer key and student responses, and grade the students’ responses based on the key.

**Program Input**

The file Prob13.in.txt will contain two sections:

1. **The answer key**

Each line of the answer key section will be in the following format:

ProblemNumber Points Operand AnswerList

Here are some notes on the answer key:

* ProblemNumber will be an integer
* Points will be an integer representing how many points to add or subtract from a student if their answer matches the criteria given by the operand and the answer list
* Operand will be one of the following three strings and shall invoke the following scoring rules:
  + EQ: This stands for equals. The student’s answer must exactly match one of the answers in the answer list to get credit for this answer. For example, if the key said EQ "250.0" and the student’s answer was "250.00", the student would not receive points.
  + IN: This stands for includes. At least one of the answers in the answer list must be contained in the student’s answer for the student to get credit. For example, if the key said IN "250.0" and the student’s answer was "250.00", the student would receive points.
  + EX: This stands for excludes. None of the answers in the answer list can be contained in the student’s answer for the student to get credit. For example, if the key said EX "250.0" and the student’s answer was "250.00", the student would not receive points.
* AnswerList will be any number of quoted text strings separated by commas. Text strings will not contain quotation marks.
* A line containing problem number 999 will mark the end of the key. Problem 999 may or may not have an answer.

1. **The student responses**

Each line of the student answer section will be in the following format:

StudentId ProblemNumber Answer

Here are some notes on the student answers:

* StudentId will be a three digit number, and may contain leading zeroes. Each student has a unique Id number.
* ProblemNumber will be an integer
* Answer will be a quoted text string which is the student's answer to that question. Only one text string will occur for each answer. Text strings will not contain quotation marks.

**Example Input:**

1 7 IN "36 IN","3 F"

2 5 EQ "250","250.0"

2 2 EX "N/A"

3 7 IN "ADAMS"

3 -7 IN "Q"

999

001 1 "36 INCHES"

001 2 "N/A"

001 3 "QUEEN ELIZABETH"

002 1 "33 FURLONGS"

002 2 "250 or 395"

002 3 "JOHN QUINCY ADAMS"

003 1 "3 FEET or 36 INCH"

003 2 "250.0"

003 3 "EDIE ADAMS OR JAMES MADISON"

**Program Output**

Your program should display the points earned by each student for each question, as well as displaying each student’s total points earned.

Grading rules:

* Questions may have more than one line in the key, and a student’s answer may match multiple key lines.
* Some key lines will be worth negative points. In the event that a student has negative points for a particular question, they should receive a score of 0 for that question.
* Each key line should only be counted once per student when awarding points.

For individual question output, use the following format:

STUDENT StudentId ProblemNumber PointsAwarded

For student total output, use the following format:

STUDENT StudentId TOTAL TotalPoints

**Example Output:**

STUDENT 001 1 7

STUDENT 001 2 0

STUDENT 001 3 0

STUDENT 001 TOTAL 7

STUDENT 002 1 7

STUDENT 002 2 2

STUDENT 002 3 0

STUDENT 002 TOTAL 9

STUDENT 003 1 7

STUDENT 003 2 7

STUDENT 003 3 7

STUDENT 003 TOTAL 21

**Java program:** Prob14.java

**Input File:** Prob14.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

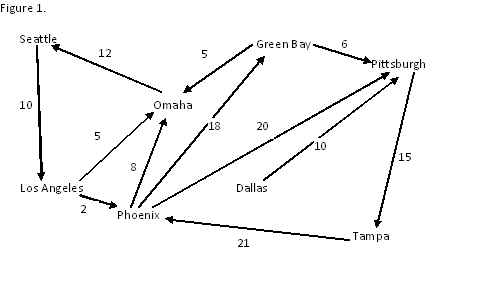
With the cost of gas at an all time high, it is useful to know how to get places quickly. A certain airline uses a series of one-way flights to get its customers from place to place. Your task is to write a program that will take route information and figure out the cheapest way to get a passenger from point A to point B.

**Program Input**

The file Prob14.in.txt will contain two sections:

1. The first line will contain the departure and arrival cities separated by a semicolon.
2. The rest of the lines will contain route data. Each route will have a departure city, an arrival city, and the cost of traveling that route all separated by semicolons. All routes are one way, but it is possible to have two routes between two cities by having two different route information lines. All costs will be integers.

**Example Input:**



Phoenix;Tampa

Seattle;Los Angeles;10

Los Angeles;Omaha;5

Los Angeles;Phoenix;2

Omaha;Seattle;12

Phoenix;Omaha;8

Phoenix;Green Bay;18

Phoenix;Pittsburgh;20

Green Bay;Omaha;5

Green Bay;Pittsburgh;6

Dallas;Pittsburgh;10

Pittsburgh;Tampa;15

Tampa;Phoenix;21

**Program Output**

Your program should output the flight path and also give the total cost in the following format:

DepartureCity->Intermediate->Cities->Destination=TotalCost

**Example Output:**

Phoenix->Pittsburgh->Tampa=35

**Java program:** Prob15.java

**Input File:** Prob15.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Part numbers are a big deal to any company, and the pattern of a part number can contain a lot of imbedded information. Your task will be to write a program that will take a list of part numbers as input and output the list of parts that conforms to a list of 15 part number patterns that you have been given below so these parts can be set aside for further processing.

Part number pattern rules:

* Characters that are outside a pair of parenthesis ( ) are considered to be static and cannot change, meaning that the input part number must contain the exact string in the pattern.
  + Example: The pattern ANABC-123( )-B-(A,B,C) would match part number ANABC-1234-B-A but not ANABC-1224-B-A.
* Anything inside a pair of parenthesis ( ) is considered to be dynamic and must conform to the pattern in the parenthesis. Patterns separated by commas are considered to be a list of values or range of values that are valid for that pattern.
  + Example: The pattern ANABC-123(A,B,C)-C would match only the part numbers ANABC-123A-C, ANABC-123B-C, and ANABC-123C-C.
* Empty parenthesis (nothing inside) indicates a wild-card, meaning that any length of character(s) may exist or not.
  + Example: The pattern ANABC( )-C would match both ANABCGOOD123-C and ANABC-C.

The following is the list of part number patterns to match against:

CW3101A( )-( )(P,S)

CW426( )-( )

CW427-( )C( )

CW500A4-(3 THRU 8)

CW507B1032R(8 THRU 32 BY 2)

CW3085-(001 THRU 050,102 THRU 178,201 THRU 284)

CWCG20Z-(M,N,P,Q,R,S)(101 THRU 999)B

CWDPX2-( )(P,S)( )(P,S)33-00( )

CWT02(E,P)18-(11,32)(P,S)( )

CW12326(E,G,J,K,L,M,N,P,R,T)(00375 THRU 20000)(A,S)

CW15232C(02,04,06,08,3 THRU 6)(-,H)(3 THRU 16,18 THRU 48 BY 2)

CW15263-(02,04,06,08,3,4,5,6)(-,H)(3 THRU 16,18 THRU 48 BY 2,59)

CW20001(C,P)(H,X,Y)(2,3,4,5,6,8,9,10,12,14,16,17)-(0100 THRU 7200)

CW102-2-(6 THRU 10,12 THRU 50 BY 2)-(6 THRU 10,12 THRU 50 BY 2)

CW8602-( )B( )PNSPM26

**Program Input**

The file Prob15.in.txt will contain a list of part numbers to be checked against the patterns.

**Example Input:**

CW3101AABC-123-P

CW426-3

CW427-C0DEWARSCR0CKS

CW500A4-7

CW507B1032R8

CW3085-001

CWCG20Z-M101B

CWDPX2-PS33-00

CWT02E18-11PS10

CW12326E00375A

CW15232C02-16

CW15263-02-3

CW20001CH2-0100

CW102-2-6-6

CW8602-( )B( )PNSPM26

ThisWillNotMatch

**Program Output**

Your program should output only the part numbers in the list that conform to any of the patterns given above. Your output should be in the same order that the input file was in. Part numbers that do not match any pattern should be ignored.

**Example Output:**

CW3101AABC-123-P

CW426-3

CW427-C0DEWARSCR0CKS

CW500A4-7

CW507B1032R8

CW3085-001

CWCG20Z-M101B

CWDPX2-PS33-00

CWT02E18-11PS10

CW12326E00375A

CW15232C02-16

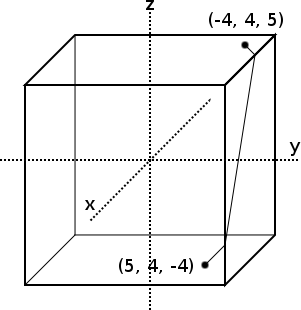
CW15263-02-3

CW20001CH2-0100

CW102-2-6-6

CW8602-( )B( )PNSPM26

**Java program:** Prob16.java

**Input File:** Prob16.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Dr. Bizarro has been tasked with developing a planetary rover prototype to explore the surface of an implausible set of cube-shaped planetoids discovered in the far reaches of our solar system. He wants you to develop a program that determines the shortest distance to travel between two points on the surface of one of these planetoids – whose faces are also implausibly flat – so that vital statistics such as the amount of power required to travel between the two points can be determined.

You will be provided with the length of an edge on the planetoid (in km) as well as the location of each point on the surface in a Cartesian format (x,y,z). The origin point (0,0,0) of the Cartesian system will be set as the center of each world’s mass and all edges of the world will be parallel to one of the system’s axes.

Remember, the distance between two points on a Cartesian plane can be determined using the following formula:

**Program Input**

The file Prob16.in.txt will contain planetoid data in groups of three lines:

1. The first line of each group will be the length of the planetoid’s sides.
2. The second line of each group will be the location of the starting point in Cartesian format.
3. The third line of each group will be the location of the ending point in Cartesian format.

**Example Input:**

10

5 4 -4

-4 4 5

200

1 1 -100

-35 -100 -100

30

-15.0 15.0 15

15 -15.0 -15

**Program Output**

Your program should calculate the distance between the two points, and print it out to 4 decimal places using standard rounding rules.

**Example Output:**

14.1421

107.2241

67.0820

**Java program:** Prob17.java

2

1

3

4

5

7

6

8

9

10

12

11

13

14

15

17

16

18

19

20

22

21

23

24

25

**Input File:** Prob17.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

A puzzle is played on a 5x5 grid of 25 squares. The squares are numbered as shown in the diagram. You start in the square numbered 1 (upper left hand corner). You can travel up, down, right, or left (not diagonally) to another square containing a wall in common with the square you are in. One square contains the treasure (your goal).

Some of the other squares contain blocks. A block fills the entire square and prevents you from traveling through that square unless you push the block out of the way. You can push a block to another square as long as there is an empty square on the opposite side of the block. You cannot push a block on top of the treasure. You cannot push a block diagonally or outside the grid.

A move consists of pushing a block from one square to an adjacent square. A move is recorded as a pair of numbers representing the square a block is moving from and a square that block is moving to, separated by a hyphen. For example, if you pushed a block from square 13 to square 8, you would record the move as 13-8. The object of the puzzle is to reach the square containing the treasure in the fewest number of moves. Movement from square to square without pushing any blocks is free.

**Program Input**

The file Prob17.in.txt will contain lines of lists of 25 integers separated by spaces representing the contents of the 25 numbered squares on the board. A "0" indicates that the square is empty as you start the puzzle. A "1" indicates that the square contains a block. A "2" indicates that the square contains the treasure. Square number 1 will always contain a zero at the start because that is your starting position, and only one square will contain a "2". Each new line is a new puzzle.

**Example Input:**

0 0 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 2 0 0 0 1 1

0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 2 0

Example 2

Example 1

T

T

**Program Output**

Your program should print the shortest series of moves that opens a path to the treasure for each puzzle. In the event that the treasure is already accessible at the beginning of the puzzle, your program should print out the phrase "No moves necessary". In the event that the treasure cannot be reached, your program should print out the phrase "No solution".

**Example Output:**

3-4 9-10

15-20 20-25 19-18

**Java program:** Prob18.java

**Input File:** Prob18.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Companies spend a lot of money on integrated circuit testing to make sure that their manufacturing processes are as good as possible. There are many things that can go wrong with making a circuit – too much metal at a given location, not enough vertical distance between crossing wires, poorly soldiered contacts…the list goes on and on. Fortunately, many of these defects can be effectively modeled as something called a "stuck-at fault". This means that a given node in the circuit is assumed to be stuck at a given value. Your task is to write a program that will read a circuit model and determine the correct output for a given set of inputs either with or without stuck-at faults applied to the circuit.

The only valid values for a node in the circuit are 0 and 1. To implement your program, you will need to know about the different logic gates that can be used to construct circuits:

|  |  |  |
| --- | --- | --- |
| **Gate Type** | **Description** | **Symbol** |
| PI (Primary Input) | The PI gate is a placeholder for the primary input signals. The inputs to your program will set the values of the PI gates. | N/A |
| AND | The AND gate outputs a 1 if and only if all its inputs are a 1. Otherwise, it outputs a 0. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-AND_ANSI.svg.png |
| NAND | The NAND is the negation of the AND gate. It outputs a 0 if and only if all its inputs are a 1. Otherwise it outputs a 1. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-NAND_ANSI.svg.png |
| OR | The OR gate outputs a 1 if any of its inputs are 1. If all inputs are 0, it outputs a 0. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-OR_ANSI.svg.png |
| NOR | The NOR gate is the negation of the OR gate. It outputs a 0 if any of its inputs are 1. If all inputs are 0, it outputs a 1. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-NOR_ANSI.svg.png |
| XOR | The XOR gate outputs a 1 if it has an odd number of inputs that are 1. Otherwise, it outputs a 0. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-XOR_ANSI.svg.png |
| XNOR | The XNOR gate is the negation of the XOR gate. It outputs a 0 if it has an odd number of inputs that are 1. Otherwise, it outputs a 1. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-XNOR_ANSI.svg.png |
| NOT | The NOT gate outputs a 1 if its input is a 0. Otherwise, it outputs a 1. | C:\Users\trinkmr\Desktop\CodeQuest\gates\100px-NOT_ANSI.svg.png |
| PO (Primary Output) | The PO gate is a placeholder for your program’s output. You will be printing out the value of the PO gates. | N/A |

PI, PO, and NOT gates may only contain a single input. Other gate types can have an unbounded number of inputs.

**Program Input**

The file Prob18.in.txt will contain two sections:

1. The circuit model section will contain lines in the following format:

NodeNumber GateType InputList

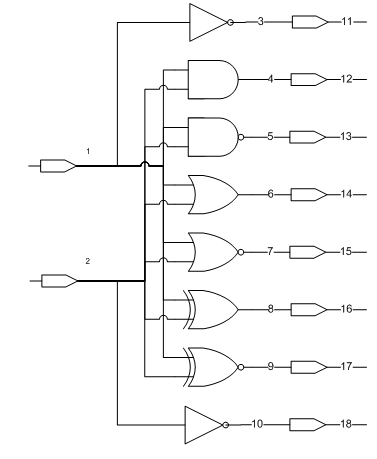
Where NodeNumber is the unique id of the node, GateType is one of the gate types described in the table above, and InputList is a list of node numbers that act as the inputs to the current gate being described. All values will be separated by spaces. A line with only the word "INPUTS" marks the end of the circuit model section.

1. The circuit input section will contain an ordered list of values for the PI gates. The values for the PI gates should be applied in the order that the PI gates were encountered in the input section, not necessarily in numerical node order.

If there are stuck-at fault values to be applied to the circuit, the following repeating pattern will follow the PI inputs on the same line, once for each stuck-at fault:

NODE NodeNumber SA Value

Where NodeNumber is the node that has the fault and Value is the value that the node is stuck at.

**Example Input:**

1 PI

2 PI

3 NOT 1

4 AND 1 2

5 NAND 1 2

6 OR 1 2

7 NOR 1 2

8 XOR 1 2

9 XNOR 1 2

10 NOT 2

11 PO 3

12 PO 4

13 PO 5

14 PO 6

15 PO 7

16 PO 8

17 PO 9

18 PO 10

INPUTS

0 0

0 1

1 0

1 1

1 1 NODE 3 SA 1 NODE 4 SA 0

**Program Output**

Your program should print the values of the PO gates in the order that they were encountered in the circuit model section. Print each PO on its own line, and insert a blank line between each set of outputs to denote the running of each input set. PO values should be printed as follows:

PO NodeNum = Value

**Example Output:**

PO 11 = 1

PO 12 = 0

PO 13 = 1

PO 14 = 0

PO 15 = 1

PO 16 = 0

PO 17 = 1

PO 18 = 1

PO 11 = 1

PO 12 = 0

PO 13 = 1

PO 14 = 1

PO 15 = 0

PO 16 = 1

PO 17 = 0

PO 18 = 0

PO 11 = 0

PO 12 = 0

PO 13 = 1

PO 14 = 1

PO 15 = 0

PO 16 = 1

PO 17 = 0

PO 18 = 1

PO 11 = 0

PO 12 = 1

PO 13 = 0

PO 14 = 1

PO 15 = 0

PO 16 = 0

PO 17 = 1

PO 18 = 0

PO 11 = 1

PO 12 = 0

PO 13 = 0

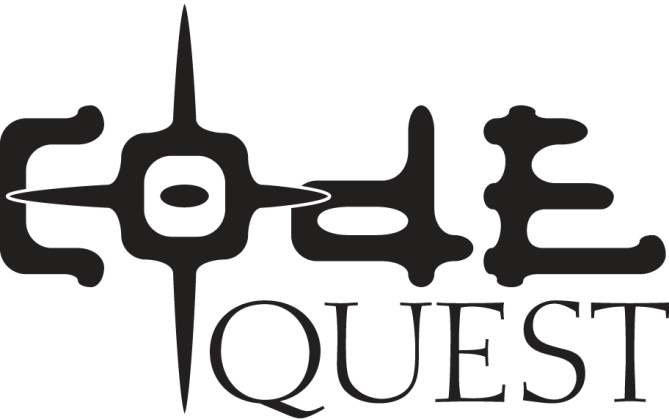
PO 14 = 1

PO 15 = 0

PO 16 = 0

PO 17 = 1

PO 18 = 0



**2012**

Problem Packet

|  |  |
| --- | --- |
| **Problem** | **Point Value** |
| Problem 1: Piggy Bank | 2 |
| Problem 2: Who Sells More Newspapers? | 3 |
| Problem 3: Numerical Order | 3 |
| Problem 4: Hidden Message | 4 |
| Problem 5: Loan Amortization | 5 |
| Problem 6: Palindromes | 6 |
| Problem 7: Grade the Scantron | 6 |
| Problem 8: Around the Block | 7 |
| Problem 9: Almost the Same | 8 |
| Problem 10: Factorials | 9 |
| Problem 11: Let’s Buy Land | 10 |
| Problem 12: Crack the Maze | 11 |
| Problem 13: Score the Test | 12 |
| Problem 14: Shortest Flight Path | 13 |
| Problem 15: Pattern Part Play | 25 |
| Problem 16: Bizarro World Rover | 35 |
| Problem 17: Pushing Blocks | 30 |
| Problem 18: Stuck-At Faults | 25 |
| **Total Possible Points** | **214** |