



UIL Computer Science Competition

District 2022

JUDGES PACKET - CONFIDENTIAL

I. Instructions

1. The attached printouts of the judge test data are provided for the reference of the contest director and programming judges. Additional copies may be made if needed for this purpose.
2. This packet must remain CONFIDENTIAL. Additional copies may be made and returned to schools when other confidential contest material is returned.

II. Table of Contents

Number	Name
Problem 1	Adrian
Problem 2	Arusha
Problem 3	Catherine
Problem 4	Diane
Problem 5	Facundo
Problem 6	Haru
Problem 7	Kristina
Problem 8	Lavanya
Problem 9	Manos
Problem 10	Michaela
Problem 11	Pankaj
Problem 12	Shirley

Problem #1
60 Points

1. Adrian

Program Name: Adrian.java

Input File: adrian.dat

Test Input File: None

Test Output to Screen:

JAVA
PYTHON
SWIFT
ADA
C
C++
COBOL
FORTRAN
ALGOL
BASIC
DELPHI
PASCAL
PL1
JAVASCRIPT

Problem #3
60 Points

3. Catherine

Program Name: Catherine.java

Input File: catherine.dat

Test Input File:

```
8
logic texas
purse purse
china cloth
shoes ascot
abcde edcba
chair wreck
bugle bagel
steam teams
```

Test Output to Screen:

```
*****
PURSE
C***h
*s*o*
edCba
*r*c*
B*Gel
teams
```

Problem #4
60 Points

4. Diane

Program Name: Diane.java

Input File: diane.dat

Test Input File:

```
10
12 and 20
30 and 20
22 and 27
1/7 and 2/5
0 and -21
5 1/3 and 7 5/7
10 1/4 and -11 2/3
-1/2 and -3/4
2/3 and -2/3
20 and -1/11
```

Test Output to Screen:

```
32
50
49
19/35
-21
13 1/21
-1 5/12
-1 1/4
0
19 10/11
```

Problem #5
60 Points

5. Facundo

Program Name: Facundo.java

Input File: facundo.dat

Test Input File:

```
10
ABCDEFG 0
ABCDEFG 1
ABCDEFG 2
ABCDEFG 3
ABCDEFGHIJ 0
ABCDEFGHIJ 1
ABCDEFGHIJ 2
ABCDEFGHIJ 3
ABCDEFGHIJKLMNPOQRSTUVWXYZ 10
ABCDE 50
```

Test Output to Screen:

```
ABCDEFG
AEBFCGD
ACEGBDF
ABCDEFG
ABCDEFGHIJ
FAGBHCIDJE
CFIADGJBEH
GCJFBIEAHD
MZLYKXJWIVHUGTFSERDQCPBOAN
AEDCB
```


Problem #7
60 Points

7. Kristina

Program Name: Kristina.java

Input File: kristina.dat

Test Input File: (lines that start with → are continuation of previous line)

```
25
PRE + - * 7 5 - 9 6 -5
POST 7 5 * 9 6 - - -5 +
POST 2 8 3 4 2 * / + -
PRE - 2 + 8 / 3 * 4 2
POST -39 9 + -35 -44 * -100 / -
PRE + -97 - * 47 29 / ^ 3 5 100
POST 2 8 * 3 4 2 / + -
PRE + - * - 7 5 9 6 -5
POST 19 -4 100 10 / * 4 3 ^ - *
POST 100 5 3 ^ -12 -3 / * +
PRE / ^ + 21 -17 3 * - -2 -9 -3
PRE - * 27 41 + / * 28 -20 + -99 47 * 15 29
POST 27 41 * 28 -20 * -99 47 + / 15 29 * + -
POST 83 -55 + 65 21 - 100 7 / -32 -76 - * 11 45 - * / +
PRE + + 83 -55 / - 65 21 * / 100 7 * - -32 -76 - 11 45
POST 18 67 -41 93 -58 28 21 -92 0 67 -55 -5 88 -78 -59 -67 21 44 19 0 44 22 -37
→ 74 -36 + + + + + + + + + + + + + + + + + + + + +
PRE / * / * / * / * / * / * / * / * / * / * / * / * 6 54 -1 -3 7 37 70 24 80 66
→ -13 -53 62 -40 84 -89 -89 78 101 -89 53 -55 -87 -95 93
POST 2 10 ^ 4 2 ^ + 3 5 ^ 5 3 ^ + - 7 3 ^ 6 3 ^ - 8 5 ^ 9 3 ^ - + *
PRE * - + ^ 2 10 ^ 4 2 + ^ 3 5 ^ 5 3 + - ^ 7 3 ^ 6 3 - ^ 8 5 ^ 9 3
POST 1 2 * 3 4 * * 5 6 * * 7 8 * * 9 10 * *
PRE * * * * * 10 9 * 8 7 * 6 5 * 4 3 * 2 1
POST 2 30 ^ 16 2 ^ / 4 3 ^ * 8 5 ^ / 32 2 ^ * 2 10 ^ /
PRE - * 123 -456 / + - -789 234 + 567 -890 - + -345 678 / 901 77
POST 10 11 + 12 + 13 + 14 + 15 + 15 -30 + + -28 14 + + 13 -26 + + -24 12 + + 11
→ -22 + + -10 +
PRE / * * / * -67 -91 * 46 82 + * 44 39 * 71 80 * / * 39 78 * 61 25 * * -16 33 /
→ -98 48 * / * -79 -80 * 10 25 / * -82 -96 * -71 -7
```

Test Output to Screen:

27	28
27	28
-6	78
-6	-33
-15	21615552
1264	21615552
11	3628800
7	3628800
-1976	8192
600	-56084
-3	0
662	20827
662	

Problem #8
60 Points

8. Lavanya

Program Name: Lavanya.java

Input File: lavanya.dat

Test Input File:

```
18
2
3
4
5
9
123456
654321
121212121
1000000001
11
13
13
97531
1999999999
2000000000
2147483645
2147483646
2147483647
```

Test Output to Screen:

```
2 = 2
3 = 3
4 = 2 * 2
5 = 5
9 = 3 * 3
123456 = 2 * 2 * 2 * 2 * 2 * 2 * 3 * 643
654321 = 3 * 218107
121212121 = 83 * 577 * 2531
1000000001 = 7 * 11 * 13 * 19 * 52579
11 = 11
13 = 13
13 = 13
97531 = 7 * 13933
1999999999 = 31 * 64516129
2000000000 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 5 * 5 * 5 * 5 * 5 * 5 * 5 * 5
2147483645 = 5 * 19 * 22605091
2147483646 = 2 * 3 * 3 * 7 * 11 * 31 * 151 * 331
2147483647 = 2147483647
```

Problem #9
60 Points

9. Manos

Program Name: Manos.java

Input File: manos.dat

Test Input File:

```
10
0 1
1 1
1 2
50 50
50 7
100 100
100 99
99 100
99 99
100 0
```

Test Output to Screen:

```
P1
P2
P1
P2
P1
P2
P1
P1
P2
P1
```

Problem #10
60 Points

10. Michaela

Program Name: Michaela.java

Input File: michaela.dat

Test Input File: (indentations are continuations of long lines)

```
11
a
ab
a b
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V X X Y Z
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
to all the judges out there grading this please know i did not make this output
    hard to grade on purpose it just came out this way
please forgive me
if you are not using an autograder i am really really sorry
```

Test Output To Screen: (indentations are continuations of long lines)

```
Message #1: -.....-
Message #2: -.....- -.....-.
Message #3: -.....- -.....- -.....-.
Message #4: -.....- -.....-. -.....-- -.....-. -.....-. -.....-- -.....-- -.....-. -.....-
    .-. -.....-. -.....-- -.....-. -.....-. -.....-- -.....-- -.....-. -.....-
    .-.....- -.....-. -.....-- -.....-. -.....-- -.....-- -.....-. -.....-
Message #5: -.....- -.....-. -.....-- -.....-. -.....-- -.....-- -.....-. -.....-
    .-. -.....-. -.....-- -.....-. -.....-- -.....-- -.....-. -.....-
    .-.....- -.....-. -.....-- -.....-. -.....-- -.....-- -.....-. -.....-
Message #6: -.....- -.....-. -.....-. -.....-. -.....-- -.....-. -.....-. -.....-
    -.....-. -.....-- -.....-. -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....-- -.....-. -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....-- -.....-. -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....--
Message #7: -.....- -.....-. -.....-. -.....-. -.....-- -.....-. -.....-. -.....-
    -.....-. -.....-- -.....-- -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....-- -.....-. -.....-. -.....-- -.....-- -.....-. -.....-
    ..... -.....-- -.....-. -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....--
Message #8: -.-.-.-. -.....-. -.....-- -.....-. -.....-- -.....-- -.....-. -.....-
    .-- -.....-. -.....-- -.....-- -.....-- -.....-- -.....-. -.....-
    -.... -.....-. -.....-- -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....-- -.....-. -.....-- -.....-. -.....-- -.....-. -.....-
    -.....-- -.....-. -.....--
Message #9: -.-.-.-. -.....-- -.....-. -.....-- -.....-- -.....-- -.....-. -.....-
    .... -.....-- -.....-. -.....-- -.....-- -.....-- -.....-. -.....-
    .-.....- -.....-- -.....-. -.....-. -.....-- -.....-- -.....-. -.....-
    -.....-- -.....-- -.....-. -.....-- -.....-- -.....-- -.....-. -.....-
    .... -.....-- -.....-. -.....-- -.....-. -.....-- -.....-. -.....-
    ..... -.....-- -.....-- -.....-- -.....-. -.....-- -.....-. -.....-
```

UIL – Computer Science Programming Packet – District - 2022

```

.....
-.-.-.- -.-.-.- ..... -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
.. -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.-
Message #10: -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
----- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
Message #11: -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
..- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-
-.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.- -.-.-.-

```

Problem #11
60 Points

11. Pankaj

Program Name: Pankaj.java

Input File: pankaj.dat

Test Input File: (indentations are continuations of long lines)

```
11
A,B,C,D,E,F,G,H,I,J,K,L
A<->E,A<->F,A<->G,A<->H,A<->I,A<->J,A<->K,A<->L,B<->E,B<->F,B<->G,B<->H,B<->I,B<->J,B<->K,B<->L,C<->E,C<->F,C<->G,C<->H,C<->I,C<->J,C<->K,C<->L,D<->E,D<->F,D<->G,D<->H,D<->I,D<->J,D<->K,D<->L
A,B,C,D,E,F,G,H,I,J,K,L
A<->E,A<->F,A<->G,A<->H,A<->I,A<->J,A<->K,A<->L,B<->E,B<->F,B<->G,B<->H,B<->I,B<->J,B<->K,B<->L,C<->E,C<->F,C<->G,C<->H,C<->I,C<->J,C<->K,C<->L,D<->E,D<->F,D<->G,D<->H,D<->I,D<->J,D<->K,D<->L,G<->H
A,B,C,D,E,F,G,H
A<->B,A<->D,A<->E,B<->F,B<->C,C<->G,C<->D,D<->H,H<->E,E<->F,F<->G
A,B,C,D,E,F,G,H
A<->B,A<->D,A<->E,C<->F,B<->C,C<->G,C<->D,D<->H,H<->E,E<->F,F<->G
A,B,C,D,E,F,G
A<->F,B<->F,C<->E,C<->G,D<->G
A,B,C,D,E,F,G
A<->F,B<->F,C<->E,C<->G,D<->G,A<->E
A,B,C,D,E,F,G
A<->F,B<->F,C<->E,C<->G,D<->G,A<->D
A,B,C,D,E,F,G
A<->F,B<->F,C<->E,C<->G,D<->G,A<->D,B<->G
Alex,Pete,Bob,Quinn,Cade,Reid,Dave
Alex<->Pete,Alex<->Quinn,Alex<->Reid,Bob<->Pete,Bob<->Quinn,Bob<->Reid,Cade<->Pete,Cade<->Quinn,Cade<->Reid,Dave<->Pete,Dave<->Quinn,Dave<->Reid
Alex,Pete,Bob,Quinn,Cade,Reid,Dave
Alex<->Pete,Alex<->Quinn,Alex<->Reid,Bob<->Pete,Bob<->Quinn,Bob<->Reid,Cade<->Pete,Cade<->Quinn,Cade<->Reid,Dave<->Pete,Dave<->Quinn,Dave<->Reid,Alex<->Dave
Alex,Pete,Bob,Quinn,Cade,Reid,Dave
Alex<->Dave
```

Test Output to Screen:

```
Test case 1: possible
Test case 2: impossible
Test case 3: possible
Test case 4: impossible
Test case 5: possible
Test case 6: possible
Test case 7: possible
Test case 8: impossible
Test case 9: possible
Test case 10: impossible
Test case 11: possible
```

Problem #12
60 Points

12. Shirley

Program Name: Shirley.java

Input File: shirley.dat

Test Input File:

```

9
6 7
1859 739 0 1101 1569 312 635
1069 1789 164 678 568 1282 1633
1399 1780 56 0 183 234 1835
0 0 973 1155 1157 0 490
1750 1523 1714 388 670 0 1256
1926 1342 1342 1397 689 0 965
3 5
1865 201 1164 1265 248
164 1933 971 1030 166
1828 1624 1654 1136 1693
2 2
1234 0
0 1999
10 10
973 739 339 1187 0 56 734 369 468 213
1356 853 1345 0 1043 1688 1625 925 1792 1698
1772 188 688 860 0 841 706 208 1332 1861
1460 1186 1221 220 469 1843 1375 1005 1677 1301
401 1433 437 839 182 462 85 431 309 50
1015 164 1114 227 1811 962 1764 1376 498 190
1786 1264 1968 611 239 1635 539 1821 475 1151
1648 94 1362 163 1388 674 396 1067 618 440
743 420 1807 893 182 1209 896 1756 1901 1308
1537 1961 1153 196 329 1007 1401 734 1392 737
10 10
858 1570 1540 1414 1966 250 1353 418 651 1912
1691 0 0 0 0 0 0 0 0 931
487 0 1569 250 249 249 0 1665 0 1245
1153 0 1178 176 1686 660 183 250 0 250
1284 0 777 233 1175 1867 134 824 0 301
1128 0 250 237 250 1644 156 1278 0 805
1542 0 1195 167 1491 1865 249 833 0 1814
561 0 1141 0 249 249 0 1666 0 318
1423 0 0 0 0 0 0 0 0 801
822 1766 1264 250 1288 1837 1263 1440 654 418
10 10
1408 1042 1962 1444 1249 458 1137 685 1918 702
287 439 1112 906 1818 1161 979 615 365 447
1297 325 1905 1800 804 1291 1110 547 250 923
1308 844 1106 1148 295 1286 1793 788 888 1043
1428 1091 791 268 1899 966 1901 546 263 1518
1769 1736 1319 1277 606 554 350 251 411 1928
1986 1531 1441 1545 646 637 1038 1675 317 838
275 1077 1598 622 844 316 1950 1537 531 256
487 543 1597 327 1137 1807 253 1197 1327 1972
1770 1690 822 1068 379 250 1027 1032 1803 801
15 15
47 1375 382 23 1719 1051 1534 783 159 654 13 737 516 156 1051
1913 851 1760 100 1880 1527 278 1295 1793 1581 1022 1937 219 1934 477
555 1882 1102 1063 644 1359 1127 261 486 1398 500 217 1939 1917 743
469 1919 791 1408 1457 1153 1847 224 47 889 1370 394 583 353 1506
33 1223 712 764 608 976 1851 1516 1528 1347 1347 1197 1297 178 1037

```

UIL – Computer Science Programming Packet – District - 2022

```

302 739 1468 199 504 327 386 1488 754 1594 860 159 1964 910 133
762 1651 229 1767 445 671 1875 351 123 664 1005 1445 1076 1121 1
912 1702 1821 524 33 1468 1293 967 859 1056 1234 1985 642 229 65
762 2000 219 1004 184 128 1006 984 1698 1822 1529 971 1037 1213 377
1095 236 1641 1164 1395 54 829 1595 1277 948 171 1319 1870 954 1544
384 4 96 801 1478 725 1778 253 775 320 1672 1682 1878 830 557
1871 941 348 1013 1081 835 172 1931 162 730 1486 1981 25 730 1028
586 1164 1445 1576 1343 744 1769 644 1493 794 1336 620 4 1559 1430
39 396 818 1824 629 1085 1460 1612 279 1425 569 733 1863 1322 1517
465 1604 953 100 325 681 970 718 1212 316 835 356 5 963 1774
10 9
1905 801 0 0 1414 0 0 735 649
1046 1004 0 1132 0 1220 0 297 395
0 0 1209 0 1488 0 537 0 1499
335 1233 0 698 1488 738 0 0 388
812 0 1488 1441 1441 1727 529 0 1638
398 0 0 1732 1390 703 0 462 0
908 1336 691 0 365 0 161 807 0
0 0 1997 1509 0 1468 1656 791 0
1686 1507 0 1234 1505 0 0 1480 756
456 1382 0 208 248 920 0 1834 1495
7 5
1664 0 1525 0 1140
0 1691 0 1071 0
1864 0 0 0 1841
0 0 2000 0 0
1015 0 0 0 1312
0 1999 0 2000 0
565 1015 0 1259 723

```

Test Output to Screen:

```

Case #1:
16026 31.0
12324 28.6
8635 14.3
^^^^^^^^^^^^^^
Case #2:
14298 66.7
NONE
NONE
^^^^^^^^^^^^^^
Case #3:
NONE
NONE
NONE
^^^^^^^^^^^^^^
Case #4:
53239 49.0
36371 32.0
NONE
^^^^^^^^^^^^^^

```

```

Case #5:
38668 36.0
10638 8.0
6516 6.0
^^^^^^^^^^^^^^
Case #6:
103706 100.0
NONE
NONE
^^^^^^^^^^^^^^
Case #7:
211509 84.0
NONE
NONE
^^^^^^^^^^^^^^
Case #8:
15228 14.4
11958 12.2
10749 10.0
^^^^^^^^^^^^^^
Case #9:
3982 8.6
3579 8.6
NONE
^^^^^^^^^^^^^^

```



Computer Science Competition District 2022 Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
2. All problems have a value of 60 points.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.
5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Adrian
Problem 2	Arusha
Problem 3	Catherine
Problem 4	Diane
Problem 5	Facundo
Problem 6	Haru
Problem 7	Kristina
Problem 8	Lavanya
Problem 9	Manos
Problem 10	Michaela
Problem 11	Pankaj
Problem 12	Shirley

1. Adrian

Program Name: Adrian.java

Input File: None

Adrian is always amused when he sees the classic “Hello World” program! He wonders how many different programming languages have used that program as the first program for students learning the language. From his research he has come up with an initial list of programming languages as shown below. Believe it or not, he even found a COBOL example!

Input: None

Output: Display the following exactly as shown below.

Sample input: None

Sample output:

JAVA
PYTHON
SWIFT
ADA
C
C++
COBOL
FORTRAN
ALGOL
BASIC
DELPHI
PASCAL
PL1
JAVASCRIPT

2. Arusha

Program Name: Arusha.java

Input File: arusha.dat

Given a string s , the right rotation of s is the last letter of s , followed by every other letter of s . The left rotation is the first letter of s , preceded by every other letter of s .

For example, with the string “ABCDE”, the right rotation is “EABCD”, and the left rotation is “BCDEA”.

Given a starting string and a sequence of rotations, Arusha wants to know the resulting string. Write a program to help Arusha calculate this.

Input:

The first line of input will contain a single integer T , the number of test cases to follow ($1 \leq T \leq 10$)

Each test case will consist of two strings S and I , denoting the string to manipulate, and the instruction sequence.

$1 \leq |S| \leq 100$

$1 \leq |I| \leq 100$

I will consist of only characters ‘L’ and ‘R’, denoting a left and right rotation instruction.

Output:

For each test case on its own line, output the resultant string after completing all rotations.

Sample input:

```
3
ABCDE L
ABCDE R
AB LLL
```

Sample output:

```
BCDEA
EABCD
BA
```

3. Catherine

Program Name: Catherine.java

Input File: catherine.dat

Catherine, like the rest of the word, is intrigued by the game of Wordle. She needs a program that will allow her to compare a target word and an attempted guess. Write a program that inputs first, the target word, then the guess. Compare the two words and provide the indicated output.

Note: Each word will be a 5-letter word consisting of lower-case letters only. For this program, neither word will contain any repeated letters. That is each word will contain five unique lower-case letters.

The program should compare the two words letter-by-letter.

- (1) If the letter in position N of the guess matches the letter in position N of the target, the upper-case form of the letter will be printed in position N of the output.
- (2) If the letter in position N of the guess is not anywhere in the target, an asterisk will be printed in position N of the output.
- (3) If the letter in position N of the guess is in the target, but not in position N, the lower-case form of that letter in guess will be printed in position N of the output.

Input: The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data consist of two five-character strings each consisting only of lower-case letters. One space will separate those words.

Output: Each output will be five characters consisting of uppercase letters, lowercase letters, and asterisks.

Sample input:

```
5
logic texas
purse purse
china cloth
shore ascot
abcde edcba
```

Sample output:

```
*****
PURSE
C***h
*s*o*
edCba
```

4. Diane

Program Name: Diane.java

Input File: diane.dat

Diane simply wants to add two values that may be integers, proper fractions, or mixed numbers.

Write a program to take two values and find the sum in simplified format.

Input: The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data consist of two numeric values A and B, separated by the lower-case word "and". One space separates value A from "and". Another space separates "and" from value B.

If A or B are proper fractions, the format will be numerator, slash ("/"), then denominator with no spaces.

If A or B are mixed numbers, the format will be integer followed by one space then the proper fraction as mentioned above.

If a proper fraction is negative, the negative will be in the numerator. If a mixed number is negative, the negative will be attached to the integer.

All values will be in the range [-1000,1000].

Output: Each output will be either an integer, a fraction, or a mixed number representing the sum of the two values.

If the sum is an integer, it will be written in integer form.

If the sum is a proper fraction, it shall be written in lowest terms in numerator/denominator form with no spaces separating the numerator from the "/", and no spaces separating the "/" from the denominator.

Output should be simplified and in lowest terms.

Sample input:

```
7
12 and 20
30 and 20
22 and 27
1/7 and 2/5
0 and -21
5 1/3 and 7 5/7
10 1/4 and -11 2/3
```

Sample output:

```
32
50
49
19/35
-21
13 1/21
-1 5/12
```

5. Facundo

Program Name: Facundo.java

Input File: facundo.dat

Facundo needs a program that will emulate what in his eyes is a perfect shuffle. Your job is to create this program.

The deck of cards will be represented by a string of uppercase letters. The first letter in the string is the top of the deck.

First, the deck will be split into an upper deck and a lower deck. The upper deck will consist of the half of the deck containing the top card. The top card will be the top card of the upper deck. The lower deck will consist of the rest of the cards with the bottom card of the original deck being the bottom card on the lower deck.

If there is an even number of cards, the upper and lower decks will have the same number of cards. If there is an odd number of cards, the upper deck will have one more card than the lower deck.

Then, the decks will be shuffled. The bottom card of the upper deck always goes first to the bottom position in the shuffled deck. The bottom card of the lower deck will go on top of that card. The upper and lower decks continue to alternate being placed on the stack until there are no more cards.

This process is executed as many times as is indicated by the data. For ABCDEFG 3, the cards are shuffled three times.

Input: The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data will consist of a string of uppercase letters followed by an integer in the range of [0,100]. The string will be of length in the range of [2,26].

Output: Each output will be a string of uppercase letters.

Sample input:

```
8
ABCDEFG 0
ABCDEFG 1
ABCDEFG 2
ABCDEFG 3
ABCDEFGHIJ 0
ABCDEFGHIJ 1
ABCDEFGHIJ 2
ABCDEFGHIJ 3
```

Sample output:

```
ABCDEFG
AEBFCGD
ACEGBDF
ABCDEFG
ABCDEFGHIJ
FAGBHCIDJE
CFIADGJBEH
GCJFBIEAHD
```

6. Haru

Program Name: Haru.java

Input File: haru.dat

Haru and his friend are playing a game called Spikebikes. In this game, each player programs their bike with a set of directions. The bikes will then follow these movement instructions, leaving spikes behind them. Bikes will crash if they run into a spike or another bike. Given each bike's starting location and instruction set, determine the outcome of the game!

Input:

The first line of input will consist of a single integer T ($1 \leq T \leq 10$), the number of test cases to follow.

Each test case will begin with a single integer N , denoting the length of the instruction sequences.

The next two lines will be of the format $X \ Y \ S$ and will denote the starting location and instruction sequences of players one and two, respectively. X and Y will be integers ($0 \leq X, Y \leq 100$) and S will be a string of length N containing characters $\{U, D, L, R\}$ denoting movement in the $\{+Y, -Y, -X, +X\}$ directions, respectively.

Coordinates are a standard X - Y system, and locations less than zero and greater than 100 contain spikes and will crash a bike.

No test case will contain a head on collision at a spike location.

Output:

For each test case, report one of the following outcomes:

- 1) DRAW (if no bike crashes)
- 2) DOUBLE SPIKE (if both bikes hit a spike at the same time)
- 3) HEAD ON (if the bikes crash into each other)
- 4) P1 WIN (if player 2 runs into a spike first)
- 5) P2 WIN (if player 1 runs into a spike first)

Sample input:

```
5
1
5 6 R
6 6 L
3
3 3 RRR
5 4 DDD
1
0 0 U
1 1 L
2
0 0 UR
1 1 DL
2
0 0 UU
1 1 UU
```

Sample output:

```
HEAD ON
P2 WIN
HEAD ON
DOUBLE SPIKE
DRAW
```

7. Kristina

Program Name: Kristina.java

Input File: kristina.dat

Kristina has been studying prefix and postfix notation for arithmetic expressions and has asked you to create a combination prefix/postfix calculator program that will check her manual evaluation of expressions of those forms.

Prefix notation places operators before their operands and operations have to be delayed until two operands are available. Consider the prefix example: `+ - * 7 5 - 9 6 -5`

```
* 7 5 is 7 * 5 = 35
- 9 6 is 9 - 6 = 3
- 35 3 is 35 - 3 = 32
+ 32 -5 is 32 + -5 = 27
```

Postfix places operators after their operands which means the needed operands are available as soon as an operator is encountered. Consider the postfix example: `7 5 * 9 6 - - -5 +`

```
7 5 * is 7 * 5 = 35
9 6 - is 9 - 6 = 3
35 3 - is 35 - 3 = 32
32 -5 + is 32 + -5 = 27
```

Kristina wants to work with only integers and wants to use `^` as an exponent operator. For example, `2 ^ 5` is 32.

Can you create the program for Kristina?

Input: First line of data file contains a positive integer T , the number of test cases that follow with $1 \leq T \leq 25$. The following T lines will start with either “PRE” for a prefix expression or “POST” for a postfix expression. Those will be followed by a single space and a properly formed expression of the indicated notation. The combination of operators (`^`, `*`, `/`, `+`, and `-`) and integers, N , will be separated by single spaces with $-100 \leq N \leq 100$. However, exponent and other operations are guaranteed to not result in values that exceed a standard Java integer. Expressions are also guaranteed to not result in division by 0. Length of lines will not exceed 200 characters.

Output: For each test case, display the resulting integer value.

Sample input:

```
6
PRE + - * 7 5 - 9 6 -5
POST 7 5 * 9 6 - - -5 +
POST 2 8 3 4 2 * / + -
PRE - 2 + 8 / 3 * 4 2
POST -39 9 + -35 -44 * -100 / -
PRE + + -97 - * 47 29 / ^ 3 5 100 ^ 2 10
```

Sample output:

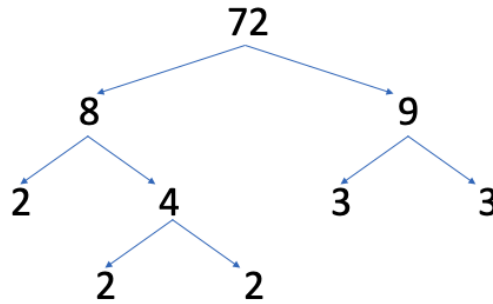
```
27
27
-6
-6
-15
2288
```

8. Lavanya

Program Name: Lavanya.java

Input File: lavanya.dat

Lavanya just learned how to take the prime factorization of a number in her Algebra class. Recall, prime factorization is a way of expressing a number as a product of its prime factors. Remember, a prime number is a number that is divisible by 1 and itself only. For example, the prime factorization of 72 is: $2 * 2 * 2 * 3 * 3$. A visual representation of this is:



In order to double check her homework problems, Lavanya has decided to write a program to tell her the prime factorization of any whole number. Can you help her write such a program?

Input: Input will begin with an integer T, the number of test cases. T will be in the range of [1,50]. The following T lines will each contain a single integer I. I will be in the range of [2,2147483647]

Output: For each integer I, you are to output: “I = primefactor1 * primefactor2 * ... * primefactorN”. There is one space between all numbers and operators. NOTE: in the output format, the prime factors are in ascending order, ie, smallest to largest. Your output must match this ascending order.

Sample input:

```

10
72
13
23
100
512
27
28
1000
1001
42
    
```

Sample output:

```

72 = 2 * 2 * 2 * 3 * 3
13 = 13
23 = 23
100 = 2 * 2 * 5 * 5
512 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2
27 = 3 * 3 * 3
28 = 2 * 2 * 7
1000 = 2 * 2 * 2 * 5 * 5 * 5
1001 = 7 * 11 * 13
42 = 2 * 3 * 7
    
```


9. Manos

Program Name: Manos.java

Input File: manos.dat

Manos is playing a 2-player game called Two Towers. In this game, there are two stacks of blocks. During each turn, a player can take as many blocks as they wish from a single stack but must take at least one. The player who removes the last block wins the game! Given the number of blocks in each stack and assuming each player plays optimally, determine which player will win.

Input:

The first line of input will consist of a single integer T, the number of test cases to follow. ($1 \leq T \leq 10$)

The next T lines will be of the form A B, denoting the height of the towers for each test case. ($0 \leq A, B \leq 100$)

There will be at least one block.

Output:

For each test case on a separate line, print P1 if the first player to go will win, and P2 if the second player will win.

Sample input:

```
3
0 1
1 1
1 2
```

Sample output:

```
P1
P2
P1
```

10. Michaela

Program Name: Michaela.java

Input File: michaela.dat

Michaela and her friends have created a new way to communicate with each other that utilizes a series of dots (.) and dashes (-) to make sentences or phrases. They've done this so that they can communicate with each other without their nosy teacher, Miss Dolores Umbridge, from eavesdropping on their conversations. They've utilized the ASCII values of the capital letters 'A'-'Z' as well as the ASCII value for the ' ' (space) to create the representations as seen below. Now that their language is created, they need your help in writing a program that can translate an English phrase or sentence into their new language. Can you help them write such a program?

Character	ASCII Value	Representation
(space)	32	-.....
A	65	-.....-
B	66	-.....-
C	67	-.....--
D	68	-.....--
E	69	-.....--
F	70	-.....--
G	71	-.....--
H	72	-.....--
I	73	-.....--
J	74	-.....--
K	75	-.....--
L	76	-.....--
M	77	-.....--

Character	ASCII Value	Representation
N	78	-.....-
O	79	-.....--
P	80	-.....--
Q	81	-.....--
R	82	-.....--
S	83	-.....--
T	84	-.....--
U	85	-.....--
V	86	-.....--
W	87	-.....--
X	88	-.....--
Y	89	-.....--
Z	90	-.....--

Input: Input will begin with an integer M, the number of messages to convert. M will be in range of [1,20]. The following M lines will contain a single message made up of one or more words. Valid messages will be made up of both upper and lower case letters and spaces only. NOTE: even though input can be lower case, all the output will match that of the table above, i.e. upper case letters only.

Continues next page...

UIL – Computer Science Programming Packet – District - 2022

Michaela continued...

Output: For each message you are to output: “Message #: series_of_dots_or_dashes series_of_dots_or_dashes ...”. There should be a space between each series_of_dots_or_dashes to denote the beginning and end of a letter. Each message is to be output on its own, individual line.

Sample input:

```
6
Hello World
MORSE CODE
UiL cOmPuTeR sCiEnCe
catch the hp reference
SHE WILL NEVER FIGURE THIS OUT
the quick brown fox jumps over the lazy dog
```

Sample output:

```
Message #1: -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
Message #2: -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
Message #3: -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
Message #4: -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
Message #5: -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
Message #6: -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
.-.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.- -.-.-.-.-
```

11. Pankaj

Program Name: Pankaj.java

Input File: pankaj.dat

Pankaj will be working for the Ultimate Intelligent Learners Summer camp in the summer of 2022. This camp is geared towards getting the campers to make new friends within their own community and the surrounding areas. This means there is the potential for a camper to know zero, one, or more than one of the other campers attending any given session.

The camp counselors have asked Pankaj to take a list of campers as well as a list of known relationships, and determine if the campers can be broken into two distinct groups such that no two campers in a group have a relationship with anyone else in the group. Remember, the focus of this camp is to get the campers to make new friends, and how can you make new friends with people you already know?

Can you help Pankaj write a program to determine whether it is possible or impossible to break a group of campers up into two distinct groups such that no two campers know anyone else in their group?

Input: The input will consist of an integer I , the number of test cases. I will be in the range of $[1,20]$. For each test case, input will consist of two lines. Line 1 will contain the name of all the campers attending a given session. Names will be limited to first names only, and only one word first names. It will be guaranteed that no two campers have the same exact first name. Names in the list will be separated by a comma “,” and there will be no spaces in the list. Line 2 will consist of the known relationships between the campers. The relationships will be given in the form: “camper1<->camper2” this means that camper1 knows camper2 and camper2 knows camper1. A relationship in this problem is always two-way, each camper will know the other camper if a relationship is present. Relationships will be separated by a comma “,” and there will be no spaces in the list. There will be at least one relationship present between two campers, but potentially more.

Output: For each test case, you are to output “Test Case #: possible” if the campers can be broken into two distinct groups such that no two campers know each other or “Test Case#: impossible” if the campers can not be broken into two such groups.

Sample input:

```
5
Alex,Brent,Chris,Dave,Eric,Fred
Alex<->Chris,Alex<->Eric,Brent<->Dave,Brent<->Fred,Chris<->Eric,Dave<->Fred
Alex,Brent,Chris,Dave,Eric,Fred
Alex<->Chris,Alex<->Eric
Alex,Brent,Chris,Dave,Eric,Fred
Alex<->Brent,Brent<->Chris,Chris<->Dave,Dave<->Eric,Eric<->Fred,Fred<->Alex
Alex,Brent,Chris,Dave,Eric,Fred
Alex<->Brent,Alex<->Chris,Alex<->Dave,Alex<->Eric,Alex<->Fred
Alex,Brent,Chris,Dave,Eric,Fred
Alex<->Brent,Alex<->Chris,Alex<->Dave,Alex<->Eric,Alex<->Fred,Brent<->Chris
```

Sample output:

```
Test case 1: impossible
Test case 2: possible
Test case 3: possible
Test case 4: possible
Test case 5: impossible
```

12. Shirley

Program Name: Shirley.java

Input File: shirley.dat

After the reports about shortages of Christmas trees this past year, Shirley thought it might be interesting to analyze aerial data from Christmas tree farms. Current generation mapping drones have amazing resolution capabilities and artificial intelligence (AI) processing of photographs have become astounding!

Shirley found some data but needs help from the UIL programming team to reduce the data into something she can work with. The data she was able to obtain consists of grids of 1-acre (4,840 square yards) square plots with an estimate of the number of trees in that plot. A typical tree farm can vary in their density but seem to hover around 1500 trees per acre. Annual harvesting is usually limited to about 12.5% of the trees to support regeneration on an 8-year cycle with annual harvesting for Christmas. Shirley is an aspiring environmentalist not a computer scientist studying programming and does not view a grid as a zero-base array like most programmers.

Grid Columns ↘ Grid ↓ Rows	1	2	3	4	5	6	7
1	1859	739	0	1101	1569	312	635
2	1069	1789	164	678	568	1282	1633
3	1399	1780	56	0	183	234	1835
4	0	0	973	1155	1157	0	490
5	1750	1523	1714	388	670	0	1256
6	1926	1342	1342	1397	689	0	965

A cluster is 2 or more adjacent plots that each contain 250 or more estimated trees. Adjacent cells are those that have a common edge, not a just a corner. However, a cluster can wrap around a plot or group of plots that contain low numbers. For each grid, Shirley would like a list of the 3 largest clusters in the grid with the total estimated number of trees in the cluster and the percentage of the grid it covers. For the 6 x 7 grid above, the cluster that includes the plot (1, 1) contains 8635 trees and covers 6 plots which is 14.3% of the total grid.

Let's help Shirley with her analysis of Christmas tree data!

Input: The first line is a positive integer $1 \leq T \leq 10$, the number of test cases in the data file. That will then be followed by T sets of data. For each dataset, the first line will contain 2 integers: the number of rows (R) and the number of columns (C) with $2 \leq R, C \leq 15$. The dataset continues with R rows, each containing C integers (N) to populate the individual array cells with $0 \leq N \leq 2000$. All items are whitespace delimited.

Output: For each test case, the first line contains a case number, formatted as shown in sample. The next three lines contain the tree counts and grid percentages, separated by one space, for the largest three clusters from largest count to smallest count. If there are less than three clusters, the later lines will display NONE instead of a count and percentage. The final line for each test case will contain 12 carats "^^^^^^^^^^^^". There are no blank lines.

Sample input:

```
2
6 7
1859 739 0 1101 1569 312 635
1069 1789 164 678 568 1282 1633
1399 1780 56 0 183 234 1835
0 0 973 1155 1157 0 490
1750 1523 1714 388 670 0 1256
1926 1342 1342 1397 689 0 965
3 5
1865 201 1164 1265 248
164 1933 971 1030 166
1828 1624 1654 1136 1693
```

Sample output:

```
Case #1:
16026 31.0
12324 28.6
8635 14.3
^^^^^^^^^^^^
Case #2:
14298 66.7
NONE
NONE
^^^^^^^^^^^^
```