

1 - Advanced - Out of Order Orders

Problem Statement

Frieda's *First Come, First Served FroYo* prides itself on serving customers in the exact order that they place their orders. Unfortunately, frozen yogurt machines are constantly in need of repair, so sometimes a customer's order is delayed and they have to wait while others who placed their order later receive their FroYo first. Frieda especially doesn't like it when an earlier order is picked up after one placed later.

Since this is a very popular location, Frieda has installed three ordering kiosks. Each kiosk assigns an order number that begins with the ID of the kiosk. For example, orders placed at kiosk 1 will be assigned order numbers in the range of 100 through 199, kiosk 2 would range from 200 through 299, and kiosk 3 would range from 300 through 399.

Each order is assigned a number that is a positive integer, where each order number is always bigger than the last order issued. For example, if a customer places an order on kiosk 1 and receives order number 142, the very next order placed may be number 145 (their order system is a little glitchy).

Frieda uses a tracking system to determine when customers receive their food. The tracking system records a list of integers representing order numbers, which are listed in the order that the customers received their FroYo. Write a program to determine how many customers ordered before another customer at the same kiosk, but received their orders later.

Input

Your program should take the following input:

- Six order numbers as integers, where each number is between 100 and 399

You may assume that these are the only six orders received and that all customers receive their food. The orders may come from one, two, or three kiosks

Output

Your program should output the number of customer who receive their food directly after a larger order number from the same kiosk as a positive integer.

Example

Input	Output
Order Numbers: 172 168 215 170 171 216	Orders out of order: 3
Order Numbers: 328 105 267 269 108 266	Orders out of order: 1

```
1 """
2 Model Solution
3 Advanced Problem 1: Out of Order
4 """
5
6 # Read input
7 orders = [int(order) for order in input('Please enter
      the all 6 order numbers, separated by a space: ').
      split(' ')]
8 kiosk1 = [order for order in orders if order >= 100
      and order < 200]
9 kiosk2 = [order for order in orders if order >= 200
      and order < 300]
10 kiosk3 = [order for order in orders if order >= 300
      and order < 400]
11
12 # Store number of orders out of order
13 out_of_order = 0
14
15 # Store initial order number
16 max_num = kiosk1[0]
17
18 for order in kiosk1:
19     # If this order is less than the previous one
20     if order < max_num:
21         # Increase out of order count
22         out_of_order += 1
23     # Check if this order is larger
24     if order > max_num:
25         max_num = order
26
27 if len(kiosk2) > 0:
28     # Store initial order number
29     last = kiosk2[0]
30
31     for order in kiosk2:
32         # If this order is less than the previous one
33         if order < max_num:
34             # Increase out of order count
35             out_of_order += 1
36         # Check if this order is larger
```

```
37         if order > max_num:
38             max_num = order
39
40 if len(kiosk3) > 0:
41     # Store initial order number
42     last = kiosk3[0]
43
44     for order in kiosk3:
45         # If this order is less than the previous one
46         if order < max_num:
47             # Increase out of order count
48             out_of_order += 1
49         # Check if this order is larger
50         if order > max_num:
51             max_num = order
52
53 print("Orders out of order: {}".format(out_of_order))
54
55
```

Problem Statement - A2

Box Collisions

Two dimensional (2D) video games often resolve multiple collisions every frame. If every collision was precise to the pixel, the game could quickly lose performance. One performant way to handle all these collisions is to use an Axis-Aligned Bounding Box (AABB). Every collidable object in a game utilizing AABBs is given a fixed (i.e. non-rotating) rectangle, and collision checks simply test if two rectangles intersect. This can be seen in the figure below.



Example of an AABB Intersection



Example of no AABB Intersection

Write a program that implements AABBs as described above. This program should accept a total number of rectangles, a constant width for all rectangles, a constant height for all rectangles, and unique x/y coordinates for the bottom left corner of each rectangle. The program should then print the total number of collisions detected. Each collision should only be counted once (i.e. A colliding with B is the same as B colliding with A). Also note that a collision should only occur if the rectangles are overlapping and not just touching. You can assume that the values for number of rectangles, width, height, x, and y are non-zero positive integers. You can also assume that the max number of rectangles will be at most 4. This means you can also assume that the max number of possible collisions is 6.

Example 1	Example 2
<p>Input: Enter number of rectangles: 2 Input: Enter width for rectangles: 2 Input: Enter height for rectangles: 2 Input: Enter x for rectangle 1: 4 Input: Enter y for rectangle 1: 2 Input: Enter x for rectangle 2: 6 Input: Enter y for rectangle 2: 2 Output: Number of collisions: 0</p>	<p>Input: Enter number of rectangles: 3 Input: Enter width for rectangles: 4 Input: Enter height for rectangles: 4 Input: Enter x for rectangle 1: 1 Input: Enter y for rectangle 1: 1 Input: Enter x for rectangle 2: 5 Input: Enter y for rectangle 2: 2 Input: Enter x for rectangle 3: 3 Input: Enter y for rectangle 3: 3 Output: Number of collisions: 2</p>

Solution - A2

Box Collisions

```
using System;

public class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Enter # rectangles: ");
        int n = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Enter Width: ");
        int w = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Enter Height: ");
        int h = Convert.ToInt32(Console.ReadLine());

        int[] x = new int[n];
        int[] y = new int[n];

        for(int i = 0; i < n; i++)
        {
            Console.WriteLine("Enter Rectangle " + (i+1) + "'s X: ");
            x[i] = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter Rectangle " + (i + 1) + "'s Y: ");
            y[i] = Convert.ToInt32(Console.ReadLine());
        }

        int nCollisions = 0;

        for(int i = 0; i < n; i++)
        {
            for(int j = i + 1; j < n; j++)
            {
                if( x[i] + w > x[j] &&
                    x[i] < x[j] + w &&
                    y[i] + h > y[j] &&
                    y[i] < y[j] + h )
                {
                    nCollisions += 1;
                }
            }
        }

        Console.WriteLine("Number of collisions: " + nCollisions);
    }
}
```

3 Advanced — Magic Squares Problem Statement

A 3×3 *magic square* is nine integers arranged in a 3×3 grid such that each row, column, and diagonal sums to the same value. Furthermore, each of the nine values must be distinct, are we restrict these values to being greater than 0 and less than 100. For example, each of the following is a magic square:

2	9	4	20	16	15
7	5	3	12	17	22
6	1	8	19	18	14

Write a program that takes as input three integers for the top row, from left to right, and an integer for the leftmost element of the middle row, and displays a magic square containing these values, if possible. If no magic square exists with the given values in the given locations, a message to this effect should be displayed. You may assume that the four input values are distinct and in the proper range.

Example 1:

```
Enter value at row 0, column 0: 20
Enter value at row 0, column 1: 16
Enter value at row 0, column 2: 15
Enter value at row 1, column 0: 12
```

```
20 16 15
12 17 22
19 18 14
```

Example 2:

```
Enter value at row 0, column 0: 1
Enter value at row 0, column 1: 16
Enter value at row 0, column 2: 3
Enter value at row 1, column 0: 9
```

No magic square exists.

Example 3:

```
Enter value at row 0, column 0: 94
Enter value at row 0, column 1: 69
Enter value at row 0, column 2: 89
Enter value at row 1, column 0: 79
```

No magic square exists.


```
42     {
43         Console.WriteLine("No magic square exists.");
44     }
45     Console.ReadLine();
46 }
47
48 private static void Read(int row, int col)
49 {
50     Console.Write("Enter value at row " + row + ", column " + col + "\n");
51     _square[row, col] = Convert.ToInt32(Console.ReadLine());
52     _used[_square[row, col]] = true;
53 }
54
55 private static bool Fill(int row, int col, int aRow, int aCol, int bRow, int bCol)
56 {
57     _square[row, col] = _sum - _square[aRow, aCol] - _square[bRow, bCol];
58     if (_square[row, col] < 1 || _square[row, col] > 99 || _used[_square[row, col]])
59     {
60         return false;
61     }
62     _used[_square[row, col]] = true;
63     return true;
64 }
65 }
66 }
67
```


4 Advanced — Card Sequence Problem Statement

A deck of cards has been arranged into a particular sequence. The deck contains 32 cards, each of which is uniquely identified by a *rank* and a *suit*. The eight ranks are 7, 8, 9, 10, Jack, Queen, King, and Ace. The four suits are Hearts, Diamonds, Clubs, and Spades. Hearts and Diamonds are colored red, whereas Clubs and Spades are colored black.

The deck is arranged so that, no matter how it is cut by moving cards from the top to the bottom, the colors of the first five cards uniquely determine the first card. The colors of the first two cards determine the suit as follows:

- Black-Black: Spades
- Black-Red: Clubs
- Red-Black: Diamonds
- Red-Red: Hearts

The colors of the third, fourth, and fifth cards determine the rank as follows:

- The rank is initialized to 7.
- If the third card is black, add 4.
- If the fourth card is black, add 2.
- If the fifth card is black, add 1.

If the rank, as computed above, is 11, 12, 13, or 14, it is interpreted as Jack, Queen, King, or Ace, respectively.

Furthermore, the color of the sixth card in the sequence is determined by the rank and suit of the first card. If the first card is the 8 of Hearts, the 8 of Clubs, or any 9, the sixth card is the same color as the first card; otherwise, the sixth card is the opposite color from the first card. Having the color of the sixth card, we can now identify the second card, from which we can determine the color of the seventh card, etc.

Write a program that takes as input a sequence of five letters, each of which is either “B” (denoting black) or “R” (denoting red), and produces as output the ranks and suits of these five cards in the order they appear. You may choose whether to have the program accept the input as five separate letters or as a single string. You may assume that each character is either “B” or “R”. Your output must be the full name of each card, as shown in the examples below.

Example 1:

Enter colors: BBRRB

8 of Spades
9 of Clubs
Queen of Hearts
10 of Diamonds
Ace of Clubs

Example 2:

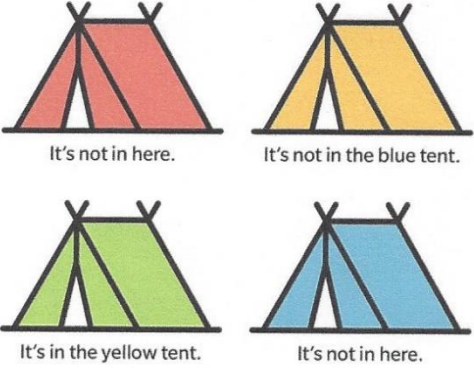
Enter colors: RBRBR

9 of Diamonds
Jack of Clubs
7 of Diamonds
8 of Clubs
10 of Hearts

```
1 // 4 - Advanced Card Sequence
2 // See also the solution to the beginning problem from this round.
3 using System;
4 using System.Collections.Generic;
5 using System.Linq;
6 using System.Text;
7 using System.Threading.Tasks;
8
9 namespace Advanced.CardSequence
10 {
11     internal class Program
12     {
13         static void Main(string[] args)
14         {
15             Console.Write("Enter colors: ");
16             string cols = Console.ReadLine();
17             Console.WriteLine();
18             string[] ranks = { "7", "8", "9", "10", "Jack", "Queen", "King", "Ace" };
19             for (int i = 0; i < 5; i++)
20             {
21                 string suit;
22                 if (cols[i] == 'B')
23                 {
24                     if (cols[i + 1] == 'B') suit = "Spades";
25                     else suit = "Clubs";
26                 }
27                 else
28                 {
29                     if (cols[i + 1] == 'B') suit = "Diamonds";
30                     else suit = "Hearts";
31                 }
32                 int rank = 0;
33                 if (cols[i + 2] == 'B') rank += 4;
34                 if (cols[i + 3] == 'B') rank += 2;
35                 if (cols[i + 4] == 'B') rank += 1;
36                 Console.WriteLine(ranks[rank] + " of " + suit);
37                 if ((rank == 1 && (suit == "Hearts" || suit == "Clubs")) || rank == 2)
38                     cols += cols[i];
39                 else
40                     cols += (char)('B' + 'R' - cols[i]);
41             }
42             Console.ReadLine();
43         }
44     }
45 }
46
```

A5 Problem Statement

Happy Campers
EASY Your annual family camping trip usually involves some sort of practical joke and, this year, the prank is on you. Your family has hidden your sleeping bag in one of four tents. Outside each they put a sign, but only one sign is truthful. Can you determine which sign that is, and which tent your sleeping bag is in?



The image shows four tents arranged in a 2x2 grid. Each tent has a sign below it. The top-left tent is red and has a sign that says "It's not in here." The top-right tent is yellow and has a sign that says "It's not in the blue tent." The bottom-left tent is green and has a sign that says "It's in the yellow tent." The bottom-right tent is blue and has a sign that says "It's not in here."

There are many ways to solve logic problems like the one on the left. One approach is to consider the four cases: which signs are true if the sleeping bag is (1) in the red tent, (2) in the yellow tent, (3) in the blue tent, finally, (4) in the green tent. For

example, if the sleeping bag is in the red tent, then the sign for the red tent is false, the sign for the yellow tent is true, the sign for the blue tent is true, and the sign for the green tent is false. Since in this case, there are two signs that are true and thus, the sleeping bag cannot be in the red tent.

Write a program that accepts two characters that determine the sign for each tent: 'T' mean the statement is positive. 'N' means it is negative. 'H' means here. The letters R, Y, B, G mean red, yellow, blue and green respectively. The situation shown in the image would be specified by

'N H N B N H T Y'.

Print out which tent contains the sleeping bag and which tent's sign is true. The output for this situation would be: Sleeping bag is in blue tent; the red tent's sign is true. If there is no situation in which only one sign is true, print out: no solution.

Formatting of output not important as long as answer is clear where bag is and which sign is true.

Example 1: input: N H T G T R T Y

output: no solution (both case 1 "in red tent" and case 3 "in blue tent" make only one sign true)

Example 2: input: N B T G N R T B

output: in red tent; red sign true

```
1
2 // LogicA.cpp : This file contains the 'main' function. Program execution ↗
   begins and ends there.
3 //
4
5 #include <iostream>
6
7 int main()
8 {
9     char TR, TY, TB, TG, CR, CY, CB, CG;
10    char PR, PY, PB, PG;
11    int IsTR, IsTY, IsTB, IsTG;
12    char Quit;
13    Quit = 'Q';
14    while (Quit == 'Q')
15    {
16        std::cout << "\nEnter T or N for red";
17        std::cin >> TR;
18        std::cout << "\nEnter color for red";
19        std::cin >> CR;
20
21        std::cout << "\nEnter T or N for yellow";
22        std::cin >> TY;
23        std::cout << "\nEnter color for yellow";
24        std::cin >> CY;
25
26        std::cout << "\nEnter T or N for blue";
27        std::cin >> TB;
28        std::cout << "\nEnter color for blue";
29        std::cin >> CB;
30
31        std::cout << "\nEnter T or N for green";
32        std::cin >> TG;
33        std::cout << "\nEnter color for green";
34        std::cin >> CG;
35        IsTR = 0; IsTY = 0; IsTB = 0; IsTG = 0;
36        PR = 'X'; PY = 'V'; PB = 'W'; PG = 'Z';
37
38        std::cout << "\n Case: " << TR << CR << TY << CY << TB << CB << TG << ↗
           CG << "\n1";
39
40        // case 1: bag is in red tent
41
42        if (TR == 'T' && (CR == 'H' || CR == 'R')) { IsTR++; PR = 'R'; }
43        if (TR == 'N' && (CR == 'B' || CR == 'Y' || CR == 'G')) { IsTR++; PR = ↗
           'R'; }
44        if (TY == 'T' && CY == 'R') { IsTR++; PR = 'Y'; }
45        if (TB == 'T' && CB == 'R') { IsTR++; PR = 'B'; }
46        if (TG == 'T' && CG == 'R') { IsTR++; PR = 'G'; }
```

```
47     if (TY == 'N' && (CY == 'H' || CY == 'B' || CY == 'G' || CY == 'Y')) ↗
        { IsTR++; PR = 'Y'; }
48     if (TB == 'N' && (CB == 'H' || CB == 'B' || CB == 'G' || CB == 'Y')) ↗
        { IsTR++; PR = 'B'; }
49     if (TG == 'N' && (CG == 'H' || CG == 'B' || CG == 'G' || CG == 'Y')) ↗
        { IsTR++; PR = 'G'; }
50     std::cout << "\n 2";
51
52     // case 2: bag is in yellow tent
53
54     if (TY == 'T' && (CY == 'H' || CY == 'Y')) { IsTY++; PY = 'Y'; }
55     if (TY == 'N' && (CY == 'B' || CY == 'R' || CY == 'G')) { IsTY++; PY = ↗
        'Y'; }
56     if (TR == 'T' && CR == 'Y') { IsTY++; PY = 'R'; }
57     if (TB == 'T' && CB == 'Y') { IsTY++; PY = 'B'; }
58     if (TG == 'T' && CG == 'Y') { IsTY++; PY = 'G'; }
59     if (TR == 'N' && (CR == 'H' || CR == 'B' || CR == 'G' || CR == 'R')) ↗
        { IsTY++; PY = 'R'; }
60     if (TB == 'N' && (CB == 'H' || CB == 'B' || CB == 'G' || CB == 'R')) ↗
        { IsTY++; PY = 'B'; }
61     if (TG == 'N' && (CG == 'H' || CG == 'B' || CG == 'G' || CG == 'R')) ↗
        { IsTY++; PY = 'G'; }
62
63     // case 3: bag is in blue tent
64
65     if (TB == 'T' && (CB == 'H' || CB == 'B')) { IsTB++; PB = 'B'; }
66     if (TB == 'N' && (CB == 'R' || CB == 'Y' || CB == 'G')) { IsTB++; PB = ↗
        'B'; }
67     if (TY == 'T' && CY == 'B') { IsTB++; PB = 'Y'; }
68     if (TR == 'T' && CR == 'B') { IsTB++; PB = 'R'; }
69     if (TG == 'T' && CG == 'B') { IsTB++; PB = 'G'; }
70     if (TY == 'N' && (CY == 'H' || CY == 'Y' || CY == 'G' || CY == 'R')) ↗
        { IsTB++; PB = 'Y'; }
71     if (TR == 'N' && (CR == 'H' || CR == 'Y' || CR == 'G' || CR == 'R')) ↗
        { IsTB++; PB = 'R'; }
72     if (TG == 'N' && (CG == 'H' || CG == 'Y' || CG == 'G' || CG == 'R')) ↗
        { IsTB++; PB = 'G'; }
73
74     // case 4: bag is in green tent
75
76     if (TG == 'T' && (CG == 'H' || CG == 'G')) { IsTG++; PG = 'G'; }
77     if (TG == 'N' && (CG == 'R' || CG == 'Y' || CG == 'B')) { IsTG++; PG = ↗
        'G'; }
78     if (TY == 'T' && CY == 'G') { IsTG++; PG = 'Y'; }
79     if (TB == 'T' && CB == 'G') { IsTG++; PG = 'B'; }
80     if (TR == 'T' && CR == 'G') { IsTG++; PG = 'R'; }
81     if (TY == 'N' && (CY == 'H' || CY == 'B' || CY == 'Y' || CY == 'R')) ↗
        { IsTG++; PG = 'Y'; }
82     if (TB == 'N' && (CB == 'H' || CB == 'B' || CB == 'Y' || CB == 'R')) ↗
```

```
    { IsTG++; PG = 'B'; }
83     if (TR == 'N' && (CR == 'H' || CR == 'B' || CR == 'Y' || CR == 'R')) ↗
        { IsTG++; PG = 'R'; }
84
85     std::cout << "\nR " << IsTR << " PR " << PR;
86     std::cout << "\nY " << IsTY << " PY " << PY;
87     std::cout << "\nB " << IsTB << " PB " << PB;
88     std::cout << "\nG " << IsTG << " PG " << PG;
89
90     if (IsTR == 1) std::cout << "\nbag is in Red, sign is at tent " << PR;
91     if (IsTY == 1) std::cout << "\nbag is in Yellow, sign is at tent " << ↗
        PY;
92     if (IsTB == 1) std::cout << "\nbag is in Blue, sign is at tent " << ↗
        PB;
93     if (IsTG == 1) std::cout << "\nbag is in Green, sign is at tent " << ↗
        PG;
94
95     std::cout << "\nEnter quit";
96     std::cin >> Quit;
97 }
98 }
99
100 // Run program: Ctrl + F5 or Debug > Start Without Debugging menu
101 // Debug program: F5 or Debug > Start Debugging menu
102
103 // Tips for Getting Started:
104 // 1. Use the Solution Explorer window to add/manage files
105 // 2. Use the Team Explorer window to connect to source control
106 // 3. Use the Output window to see build output and other messages
107 // 4. Use the Error List window to view errors
108 // 5. Go to Project > Add New Item to create new code files, or Project > Add ↗
    Existing Item to add existing code files to the project
109 // 6. In the future, to open this project again, go to File > Open > Project ↗
    and select the .sln file
110
```

6 - Advanced - Super Secret

Problem Statement

Substitution ciphers are simple encryption algorithms where plaintext is replaced with ciphertext based off some fixed system. For example, we could shift or rotate the alphabet by some number of positions as the basis for encrypting a message. If we rotate the alphabet n positions, a letter is substituted with a letter that comes n places after it. For example, if we start with a normal English alphabet and rotate 13 positions, "A" in plaintext, would become "N" in ciphertext. To convert the ciphertext back into plaintext, you can just do the ROT13 cipher in reverse.

Write a program that will encrypt plaintext and decrypt ciphertext using the substitution cipher described. Your program should take in the number of positions to rotate as input. If a number other than "0" is given, your alphabet should wrap around for encrypting/decrypting a letter that would cause you to go past the end of the alphabet. For example, if you are rotating 5 positions, the letter "X" would be encrypted to be "C". The wrap would also apply to decryption, but in reverse. In rotating 5 positions, "C" would decrypt to "X".

The program should then take as input an "E" for encode or "D" for decode followed by the message to encode or decode. The message should be encrypted/decrypted, and the result should be printed out. Non-alpha characters should not be manipulated. Uppercase characters should remain uppercase and lowercase characters should remain lowercase.

Input

- A number n , where $0 \leq n < 26$, that represents the number of rotations.
- A character ('E' or 'D') that indicates encrypt or decrypt. This letter will be capitalized.
- A non-empty string that is the message to encode or decode

Output

Your program should output the message from the user that is encrypted or decrypted based off of the inputs. The output should maintain the case of the original message as well as any non-alpha characters.

Example

Input	Output
Enter the starting letter: 13 Do you want to (E)ncode or (D)ecode? E Enter a plaintext message: Hello, World!	Uryyb, Jbeyq!
Enter the starting letter: 18 Do you want to (E)ncode or (D)ecode? E Enter a plaintext message: Hello, World!	Zwddg, Ogjdv!
Enter the starting letter: 25 Do you want to (E)ncode or (D)ecode? D Enter a ciphertext message: Fn Bzsr!	Go Cats!

```
1 """
2 Advance problem 6: Super Secret
3 """
4 def process_text(text, alphabet, rotations):
5     result = ""
6     for c in text:
7         was_upper = False
8         if c.isupper():
9             was_upper = True
10            c = c.lower()
11
12            if c in alphabet:
13                rotate_index = (alphabet.index(c) +
14 rotations) % len(alphabet)
15                letter = alphabet[rotate_index]
16                if was_upper:
17                    letter = letter.upper()
18
19                result += letter
20            else: # non-alpha
21                result += c
22
23            return result
24
25 def main():
26     # alphabet for the regular plain text
27     plain_alphabet = "abcdefghijklmnopqrstuvwxyz"
28
29     # number of positions to rotate
30     rotations = int(input("Enter the number of
31 rotations: "))
32
33     choice = input("Do you want to (E)ncode or (D)
34 ecode? ")
35     if choice == 'E':
36         # encrypt a plain text message
37         plain_text = input("Enter a plaintext message
38 : ")
39         secret_text = process_text(plain_text,
40 plain_alphabet, rotations)
41
42 """
```



```
37     print(secret_text)
38     elif choice == 'D':
39         # decrypt an encrypted message
40         secret_text = input("Enter a ciphertext
message: ")
41         plain_text = process_text(secret_text,
plain_alphabet, -1 * rotations)
42
43         print(plain_text)
44
45
46 main()
47
```