1 Advanced — Cruising to K-State

Problem Statement

After a nice Thanksgiving break, you and your friend are ready to head back to K-State. However, you have got into an argument as to who will get back first. Your friend believes that going faster, even if it means speeding, ensures that you'll get to your destination first, but you believe that his fuel efficiency will be so bad that he'll have to make multiple stops at gas stations. Being the programmer that you are, you have decided to solve this problem once and for all with code.

Write a program that accepts the distance(in miles) to Manhattan, the speed(in miles per hour) for driver 1, and the speed(mph) for driver 2. The program then displays which driver arrived first, what time they did it in, and how much sooner they arrived. All values are allowed a maximum error of .01 (ie: for an answer of 1.78, 1.79 or 1.77 will suffice).

Important to know

- Assume that each driver has a 15.0 gallon gas tank and they are always going at least 55.0 mph.

- Every time someone refills, they lose 0.5 hours as they stop for gas, refill, get up to speed, etc.

- As speed increases, the amount of power required to overcome drag, friction, and rolling resistance increases exponentially. This results in a lower fuel efficiency as more fuel is used to cover less distance regardless of how fast the car is going. Thus, knowing that on average the best fuel efficiency of 24.7 miles per gallon (mpg) is achieved around 55.0 mph, use the equation $\text{mpg} = 24.7 \frac{55}{(speed - 55)^{1.35} + 55}$ to approximate the mpg for each driver.

Example 1	Example 2
Input: Enter distance(mi) to Manhattan, KS: 170.1 **Input:** Enter driver 1's speed(mph): 75.0 **Input:** Enter driver 2's speed(mph): 79.0 **Output:** Driver 1 arrived first in 2.27 hours. **Output:** Driver 1 arrived 0.39 hours sooner!	**Input:** Enter distance(mi) to Manhattan, KS: 133.7 **Input:** Enter driver 1's speed(mph): 70.82 **Input:** Enter driver 2's speed(mph): 74.98 **Output:** Driver 2 arrived first in 1.78 hours. **Output:** Driver 2 arrived 0.10 hours sooner!

1 Advanced — Cruising to K-State

Solution

```
static void Main(string[] args)
{
    Console.Write("Enter distance(mi) to Manhattan, KS: ");
    double distance = Double.Parse(Console.ReadLine());

    Console.Write("Enter driver 1's speed(mph): ");
    double speed1 = Double.Parse(Console.ReadLine());
    double mpg1 = 24.7 * (55 / (Math.Pow(speed1 - 55, 1.35) + 55));

    Console.Write("Enter driver 2's speed(mph): ");
    double speed2 = Double.Parse(Console.ReadLine());
    double mpg2 = 24.7 * (55 / (Math.Pow(speed2 - 55, 1.35) + 55));

    double gasTank = 15;

    int refillTimes1 = (int)((distance / mpg1) / gasTank);
    int refillTimes2 = (int)((distance / mpg2) / gasTank);

    double finalTime1 = refillTimes1 * .5 + (distance / speed1);
    double finalTime2 = refillTimes2 * .5 + (distance / speed2);

    if (finalTime1 < finalTime2)
    {
        Console.WriteLine("\nDriver 1 arrived first in " + string.Format("{0:N2}", finalTime1) + " hours.");
        Console.WriteLine("Driver 1 arrived " + string.Format("{0:N2}", finalTime2 - finalTime1) + " hours sooner!
    }
    else
    {
        Console.WriteLine("\nDriver 2 arrived first in " + string.Format("{0:N2}", finalTime2) + " hours.");
        Console.WriteLine("Driver 2 arrived " + string.Format("{0:N2}", finalTime1 - finalTime2) + " hours sooner!
    }

    Console.ReadKey();
}
```

2 - Advanced - Camping Trip

Problem Statement

You have been planning a camping trip with your family this Fall, but Kansas weather can be tricky! Looking at the weather forecast for the next week, find the warmest *d* consecutive days (*i.e. highest average temperature*) for your trip.

Input

Your program should take the following input:

- The length of the trip in days. You may assume that this number is at least 1 and at most 7.
- Seven temperatures (as whole numbers).

Output

Your program should output which days you decide to go camping on, as well as the average temperature. Note that if there is a tie, you should always go camping on the later days. Your computed averages
should be within 0.01 of the averages shown

Examples

Input	Output
Trip Length: **2** Temperatures: **80 77 42 50 75 60 56**	Camping on days: **1, 2** Average temperature: **78.50**
Trip Length: **3** Temperatures: **60 77 42 50 75 60 56**	Camping on days: **5, 6, 7** Average temperature: **63.67**

```
days = int(input('Please enter the length of your trip: '))
temps = [int(temp) for temp in input('Please enter the all 7 temperatures, separated by a space:').split(' ')]
max_temp = temps[0]
max_i = 0
i = 0

while i <= 7 - days:
    total = sum(temps[i:i+days])
    if total >= max_temp:
        max_temp = total
        max_i = i
    i += 1
print('Camping on day(s): {}'.format(', '.join([str(day) for day in range(max_i + 1, max_i + 1 + days)])))
print('Average temperature: {:.2f}'.format(max_temp / days))

```

3 Advanced — Martian Calendar
Problem Statement

This problem involves a Martian calendar, defined as follows. The calendar begins at year 0. A year that is divisible by 2 but not divisible by 10 has 668 days. All other years have 669 days. Each year is divided into 24 months. In a 668-day year, months 6, 12, 18, and 24 have 27 days, and all other months have 28 days. A 669-day year is the same, except month 24 has 28 days.

Write a program that takes as input three integers giving the month, day, and year of a starting date and three integers giving the month, day, and year of an ending date, and produces as output the number of days in the given range, including both the starting and ending dates. You may assume that all dates are valid according to the above definition, and that the ending date is no earlier than the starting date (the dates may, however, be the same).

Example 1:

```
Enter starting month: 23
Enter starting day: 28
Enter starting year: 0
Enter ending month: 1
Enter ending day: 1
Enter ending year: 1

Days in range: 29
```

Example 2:

```
Enter starting month: 3
Enter starting day: 6
Enter starting year: 57
Enter ending month: 3
Enter ending day: 6
Enter ending year: 57

Days in range: 1
```

Example 3:

```
Enter starting month: 24
Enter starting day: 27
Enter starting year: 18
Enter ending month: 2
Enter ending day: 10
Enter ending year: 22

Days in range: 2046
```

```
1  // 3 Advanced - Martian Calendar
2
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace MartianCalendar
10 {
11     class Program
12     {
13         static void Main(string[] args)
14         {
15             Console.Write("Enter starting month: ");
16             int m1 = Convert.ToInt32(Console.ReadLine());
17             Console.Write("Enter starting day: ");
18             int d1 = Convert.ToInt32(Console.ReadLine());
19             Console.Write("Enter starting year: ");
20             int y1 = Convert.ToInt32(Console.ReadLine());
21             Console.Write("Enter ending month: ");
22             int m2 = Convert.ToInt32(Console.ReadLine());
23             Console.Write("Enter ending day: ");
24             int d2 = Convert.ToInt32(Console.ReadLine());
25             Console.Write("Enter ending year: ");
26             int y2 = Convert.ToInt32(Console.ReadLine());
27             Console.WriteLine();
28             Console.Write("Days in range: ");
29             Console.WriteLine(DayNumber(m2, d2, y2) - DayNumber(m1, d1, y1) + 1);
30             Console.ReadLine();
31         }
32
33         // Returns the day number within the entire perpetual calendar,
34         // where 1/1/0 is day 1.
35         static int DayNumber(int month, int day, int year)
36         {
37             int daysInPrevYears = (year - 1) * 668 + year / 2 + (year - 1) / 10;
38             return daysInPrevYears + (month - 1) * 28 - (month - 1) / 6 + day;
39         }
40     }
41 }
42
```

4 Advanced - Tiling

Problem Statement

A room which is shaped like an isosceles right triangle needs to be tiled with equal-sized, square tiles. In such a room, two equal-length walls come together in a 90-degree corner. The tiling must start in the 90-degree corner. The pieces that are cut off to make the tiles fit cannot be used anywhere else. There are two sizes of tiles available at different prices. Given the length (in whole inches) of one of the two equal-length sides of the room and for each size of tile, the size (in whole inches) and cost per tile (dollars), determine which tile size gives the lowest total cost. Display which tiles to use and the total cost for those tiles.

Example 1

Inputs:
Length of one side: 480
Tile1 length: 12
Tile 1 cost: 0.44
Tile 2 length: 18
Tile 2 cost: 0.67
Output:
The second tiles are cheapest at 253.26

Example 2

Inputs:
Length of one side: 480
Tile1 length: 11
Tile 1 cost: 0.53
Tile 2 length: 18
Tile 2 cost: 1.47
Output:
The first tiles are cheapest at 524.7

4 Advanced – Tiling

Solution

```
int main()
{
	int length, tile1size, tile2size;
	int Ntile1, Ntile2, x1, x2, y1, y2;
	float tile1price, tile2price, costtile1, costtile2;
	char q;
	q = 'Q';
	while (q == 'Q') {
		std::cout << "\nEnter length of side(inches): ";
		std::cin >> length;
		std::cout << "\nEnter tile 1 size(inches): ";
		std::cin >> tile1size;
		std::cout << "\nEnter tile 1 price(dollars): ";
		std::cin >> tile1price;
		std::cout << "\nEnter tile 2 size(inches): ";
		std::cin >> tile2size;
		std::cout << "\nEnter tile 2 price(dollars): ";
		std::cin >> tile2price;

		//tile1
		x1 = 1;
		y1 = length / tile1size;
		if (length == tile1size*y1) {
			x1 = 0;//if length is multiple of tilesize then no extra tile needed
		}
		Ntile1 = (y1 + x1)*(y1 + x1 + 1) / 2; //sum of 1 + 2 + .. y1+x1
		std::cout << "\nNum tile 1: " << Ntile1;
		costtile1 = Ntile1 * tile1price;

		//tile2
		x2 = 1;
		y2 = length/ tile2size ;
		if (length == tile2size*y2) {
			x2 = 0; //if length is multiple of tilesize then no extra tile
		}
		Ntile2 = (y2 + x2)*(y2 + x2 + 1) / 2; //sum of 1 + 2 + .. y2+x2
		std::cout << "\nNum tile 2: " << Ntile2;
		costtile2 = Ntile2 * tile2price;

		//compare
		if (costtile1 < costtile2) { std::cout << "\nTile 1 costs less: "
				<< costtile1; }
		else {
			std::cout << "\nTile 2 costs the same or less: " << costtile2;
		}
			std::cin >> q;

	}
		return 0;
	}
```

5 - Advanced - Partner Up

Problem Statement

You are hosting a holiday party. When you invited guests to your party, you requested each guest to bring one partner so all of the party games would work out. Each of your invitations had a unique identifier (an integer) that your guests (and their partner) had to present to join the party. Before the festivities started, you went to double check the guest list to make sure everyone brought their partner. If multiple guests were not able to bring a partner, they should be paired together in the order that they arrived. Finally, if anyone, find out which guest does not end up with a partner.

Input

Your program should take the following input:

- The number of people (guests and partners) at the party.
- The identifier of each guest (and partner if applicable) from their invitation.

You may assume that second input is in the order that the guests arrived in. You may also assume that at least one guest arrived, but not all. You may **not** assume that guests and their partner arrive together.

Output

Your program should output the identification numbers of the unaccompanied guests that you were able to pair, then any unaccompanied guest that was left out. Your output should clearly indicate which guests who have been paired up.

Examples

Input	Output
Attendance: **2** Guest ID(s): **2 5**	Guest **#2 and #5** got paired.
Attendance: **5** Guest ID(s): **3 2 1 4 2**	Guest **#3 and #1** got paired. Guest **#4** did not bring a partner.
Attendance: **8** Guest ID(s): **3 2 2 7 3 1 4 5**	Guest **#7 and #1** got paired. Guest **#4 and #5** got paired.

```python
guests = int(input("Enter number of guests: ")) # not needed for this particular model solution
guest_list = [int(guest) for guest in input("Enter guest list: ").split(' ')]
seen = []
odds = []
for i, guest in enumerate(guest_list):
    if guest not in seen and guest not in guest_list[i + 1:]:
        odds.append(guest)
    else:
        seen.append(guest)

odd_count = len(odds)
if odd_count > 0:
    for pair in zip(odds[0::2], odds[1::2]):
        if len(pair) > 1:
            print('Guest #{} and #{} got paired.'.format(pair[0], pair[1]))
    if odd_count % 2 == 1:
        print('Guest #{} did not bring a partner.'.format(odds[-1]))

```

6 Advanced — World Series
Problem Statement

The World Series is a series of 7 baseball games between the same two teams, which we will call A and B. A team must win 4 games to win the World Series. Games cannot end in a tie. For this problem, we will assume that the probability that team A wins a specific game is the same for each game played, and that all 7 games are always played, even if a team reaches 4 wins sooner.

Your task is to write a program that takes as input the probability p that team A wins a given game, and produces as output the probability that team A wins the World Series. You may assume that p is a floating-point value, $0 \leq p \leq 1$.

The probability you need to compute can be expressed in terms of a function $P(k, n, p)$ that gives the probability that team A wins at least k games out of a series of n games, assuming that the probability it wins any given game is p. $P(0, n, p) = 1$ for all n and p, because a team will always win at least 0 games. If $k > n$, then $P(k, n, p) = 0$, as it is impossible to win more games than are played. Otherwise, there are two ways for team A to win at least k games: either team A wins the first game (probability p), then wins at least $k-1$ of the next $n-1$ games (probability $P(k-1, n-1, p)$), or team A loses the first game (probability $1-p$), then wins k of the next $n-1$ games (probability $P(k, n-1, p)$). Thus, if $0 < k \leq n$, we can compute $P(k, n, p)$ using $P(k-1, n-1, p)$ and $P(k, n-1, p)$:

$$P(k, n, p) = pP(k-1, n-1, p) + (p-1)P(k, n-1, p).$$

For example, using these facts, the following values for $P(k, n, 0.1)$ can be computed:

	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$
$k=0$	1	1	1	1	1	1	1	1
$k=1$	0	0.1	0.19	0.271	0.3439	0.40951	0.468559	0.5217031
$k=2$	0	0	0.01	0.028	0.0523	0.08146	0.114265	0.1496944
$k=3$	0	0	0	0.001	0.0037	0.00856	0.01585	0.0256915
$k=4$	0	0	0	0	0.0001	0.00046	0.00127	0.002728

For this example, the probability of winning the World Series would be $P(4, 7, 0.1) = 0.002728$.

Example 1:

```
Enter the probability of winning a game: 1
The probability of winning the series is: 1
```

Example 2:

```
Enter the probability of winning a game: 0.55
The probability of winning the series is: 0.608287796875
```

Note: Your results should agree with the above in at least the first 3 digits, not counting any initial 0 digits; e.g., 0.00272... for the example shown in the table above.

```
1  // 6 Advanced - World Series
2
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace WorldSeriesAdvanced
10 {
11     class Program
12     {
13         static void Main(string[] args)
14         {
15             Console.Write("Enter the probability of winning a game: ");
16             double p = Convert.ToDouble(Console.ReadLine());
17             Console.Write("The probability of winning the series is: ");
18             Console.WriteLine(Probability(4, 7, p));
19             Console.ReadLine();
20         }
21
22         static double Probability(int wins, int games, double winProb)
23         {
24             if (wins == 0)
25             {
26                 return 1;
27             }
28             else if (wins > games)
29             {
30                 return 0;
31             }
32             else
33             {
34                 return winProb * Probability(wins - 1, games - 1, winProb) + (1 -
                       winProb) * Probability(wins, games - 1, winProb);
35             }
36         }
37     }
38 }
39
```