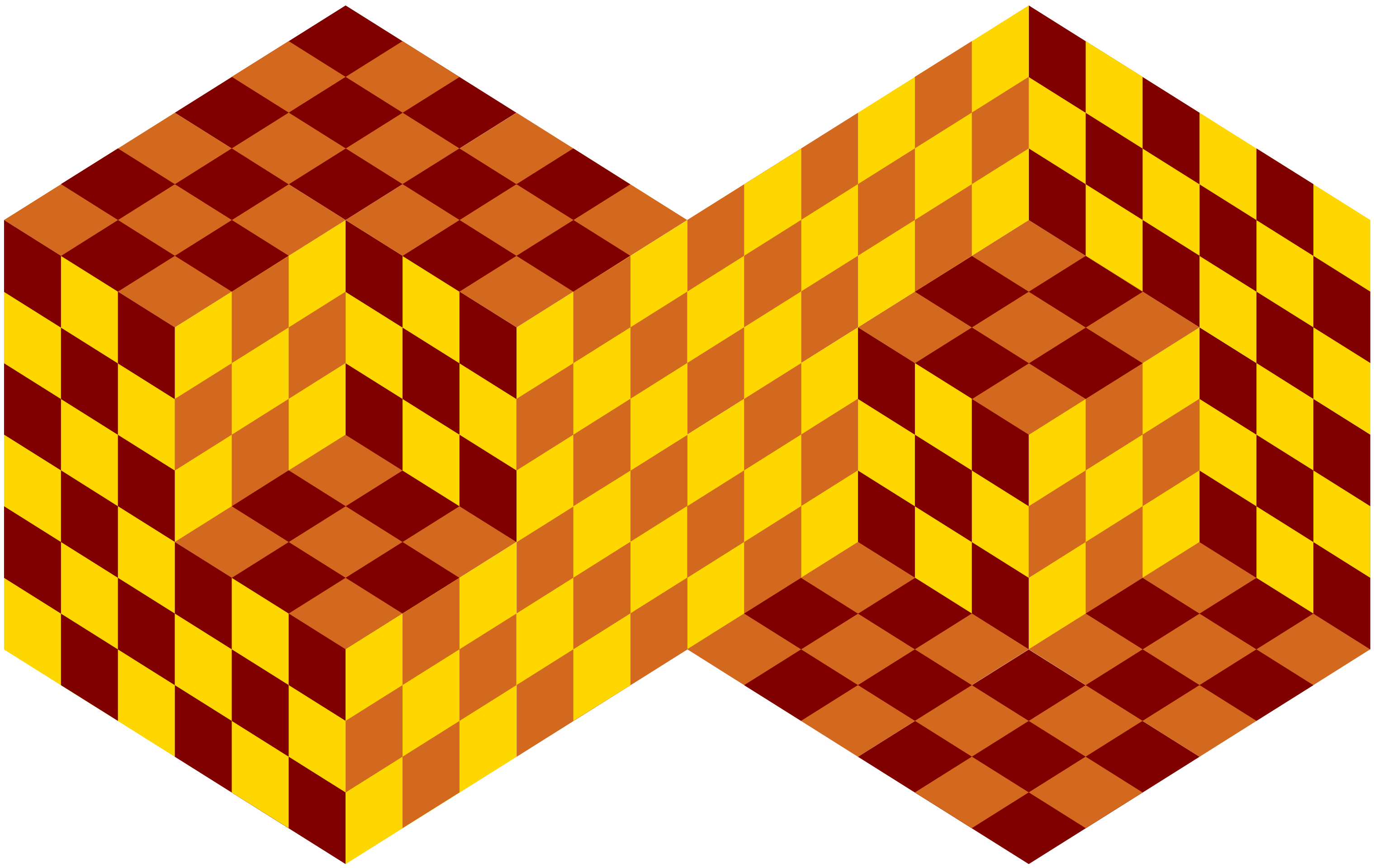
23rd Annual High School Programming Contest

April 10, 2018



Department of Mathematical and Digital Sciences
Bloomsburg University

1. Okapi

The game of *Okapi* is played by rolling three dice. A payout in dollars is determined by the rolled numbers according to the following rule:

- If the three numbers are the same, the player wins the sum of those three numbers.
- If only two of the numbers are the same, the player wins the sum of the two equal numbers.
- For three different numbers, the player wins nothing.

Write a program that prompts the user for three dice rolls and outputs the payout.

The following test cases illustrate the required I/O format. User input is shown in bold.

```
Enter dice rolls: 3 3 3
The payout is $9.

Enter dice rolls: 5 3 5
The payout is $10.

Enter dice rolls: 5 4 6
The payout is $0.
```

2. Vowel Shifter

Write a program that prompts the user for a sentence and modifies it by shifting each vowel like this:

- `a → e     A → E`
- `e → i     E → I`
- `i → o     I → O`
- `o → u     O → U`
- `u → a     U → A`

In other words, each "a" in the original sentence becomes an "e", each "e" in the original sentence becomes an "i", and so on, and similarly for capital letters.

The following test cases illustrate the required I/O format.

```
Enter a sentence.
Your powers are weak, old man!
Yuar puwirs eri wiek, uld men!

Enter a sentence.
I find your lack of faith disturbing.
O fond yuar leck uf feoth dostarbong.

Enter a sentence.
I can take you as far as Anchorhead.
O cen teki yua es fer es Enchurhied.
```

As shown by the third test case, capitalized vowels may appear anywhere within the input sentence.

3. Carry Count

When calculating the sum of two numbers by hand, we first add the digits in the 1's position, and if the result overflows (i.e., is greater than 9) then we carry the leftmost 1 in the result to the 10's position. This process is repeated at the 10's position, then at the 100's position, and so on. For example, if we are calculating 1523 + 817 by hand, we write:

```
①   ①
1 5 2 3
  8 1 7
-------
2 3 4 0
```

In this case, there are two carries (shown in circles).

For another example, consider how 98705 and 1335 would be added by hand:

```
① ① ①   ①
  9 8 7 0 5
    1 3 3 5
-----------
1 0 0 0 4 0
```

In this case, there are four carries.

Write a program that prompts the user for two positive integers and outputs the number of carries that would occur when performing addition by hand.

The following test cases illustrate the required I/O format.

```
Enter two positive integers to be added: 12345678 87654321
There will be 0 carries.

Enter two positive integers to be added: 18 1023
There will be 1 carry.

Enter two positive integers to be added: 1523 817
There will be 2 carries.

Enter two positive integers to be added: 2847356 58365001
There will be 4 carries.
```

Be sure to handle "carry" vs. "carries" in a grammatically correct way.

4. Stacking Boxes

Given a quantity of square boxes, your job is to arrange them into triangular stacks with none left over. You might need more than one stack to do this, but it is always possible with at most four stacks. For example, if you have 34 boxes, then you can stack them like this:

```
        *
       ***
      *****      *
     *******    ***
    *********  *****
```

A single box by itself counts as a stack of height one, so 35 boxes could be stacked like this:

```
        *
       ***
      *****      *
     *******    ***
    *********  *****  *
```

There are different ways in which a given number of boxes may be arranged into triangular stacks. We want the arrangement with the highest possible first stack, and for the remaining possibilities we choose the one with the highest possible second stack, and so on.

Write a program that prompts the user for the number of boxes and outputs the size of each stack in descending order. Each stack height in the output must be greater than zero.

The following test cases illustrate the required I/O format.

```
How many boxes? 35
Height of 1st stack: 5
Height of 2nd stack: 3
Height of 3rd stack: 1

How many boxes? 75
Height of 1st stack: 8
Height of 2nd stack: 3
Height of 3rd stack: 1
Height of 4th stack: 1

How many boxes? 100
Height of 1st stack: 10

How many boxes? 125
Height of 1st stack: 11
Height of 2nd stack: 2
```

5. Wealth Redistribution

Given a list of integers from 1 to *n*, we subtract 1 from the maximum value and add 1 to the minimum. This process is repeated until the list is *stabilized*, which means that the maximum and minimum values in the list differ by at most one.

To illustrate, consider the case *n* = 8. Initially the list looks like this:

```
1 2 3 4 5 6 7 8
```

After applying the redistribution process, we have:

```
2 2 3 4 5 6 7 7
```

Now there are two maximum values and two minimum values. We are not interested in the order of values in the list, so we can subtract 1 from either maximum and add 1 to either minimum. For example:

```
2 3 3 4 5 6 6 7
```

We can continue in this way until the list is stabilized:

```
3 3 3 4 5 6 6 6
3 3 4 4 5 5 6 6
3 4 4 4 5 5 5 6
4 4 4 4 5 5 5 5
```

You can see that when starting with a list of size 8, it takes 6 rounds of redistribution to stabilize.

Write a program that prompts the user for the size of the list and outputs the number of steps required to reach stabilization. You may assume that the size will be at most 1000.

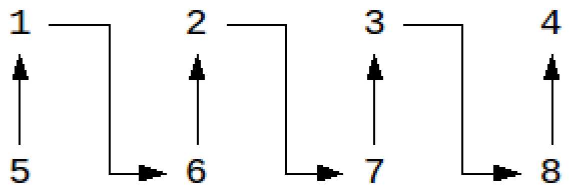
The following test cases illustrate the required I/O format.

```
Size of list: 8
6 steps to stabilization.

Size of list: 15
28 steps to stabilization.

Size of list: 1000
124750 to stabilization.
```

6. Faro Shuffle

There are many ways to shuffle a deck of cards. This problem investigates one particular method known as the *Faro shuffle*, which is used in card tricks and in the theory of parallel processing. The basic idea is to split a deck with an even number of cards into halves and then interleave the cards by taking from the top half and the bottom half alternatingly. Depending on which half you start with, this procedure is known as an *in-shuffle* or an *out-shuffle*. An *in-shuffle* starts with the bottom half.

To illustrate, suppose there are 8 cards. We denote this by (1, 2, 3, 4, 5, 6, 7, 8). After splitting the deck, the top half is (1, 2, 3, 4) and the bottom half is (5, 6, 7, 8). An in-shuffle merges the two halves like this:



In words, an in-shuffle transforms (1, 2, 3, 4, 5, 6, 7, 8) into (5, 1, 6, 2, 7, 3, 8, 4).

Any deck that is repeatedly in-shuffled will eventually return to its original ordering. With 8 cards, it takes 6 in-shuffles to do this:

```
(1, 2, 3, 4, 5, 6, 7, 8)
(5, 1, 6, 2, 7, 3, 8, 4)
(7, 5, 3, 1, 8, 6, 4, 2)
(8, 7, 6, 5, 4, 3, 2, 1)
(4, 8, 3, 7, 2, 6, 1, 5)
(2, 4, 6, 8, 1, 3, 5, 7)
(1, 2, 3, 4, 5, 6, 7, 8)
```

Write a program that prompts the user for the number of cards and outputs the number of in-shuffles needed to return the deck to its initial ordering.

The following test cases illustrate the required I/O format.

```
Number of cards in the deck: 8
The deck will return to its initial ordering after 6 in-shuffles.

Number of cards in the deck: 26
The deck will return to its initial ordering after 18 in-shuffles.

Number of cards in the deck: 52
The deck will return to its initial ordering after 52 in-shuffles.
```

You may assume that the user enters a positive even number.

7. Back and Forth

A robot is moving along a straight line. Initially it is at point *A*. It walks to point *B* at a constant velocity of 1 unit per second. Immediately after reaching *B* it turns around and walks to *A* at the same speed. After reaching point *A* it immediately turns around once again and heads to *B*. This process continues indefinitely and gives the robot a sense of fulfillment.

For example, if *A* = 5 and *B* = 13, then after *T* = 27 steps the robot will be at location 10. The following picture shows the robot's location and direction of movement after each unit of time from 0 to 27 seconds.

A 5	6	7	8	9	10	11	12	B 13
→ 0	→ 1	→ 2	→ 3	→ 4	→ 5	→ 6	→ 7	→ 8
← 16	← 15	← 14	← 13	← 12	← 11	← 10	← 9	← 8
→ 16	→ 17	→ 18	→ 19	→ 20	→ 21	→ 22	→ 23	→ 24
					← 27	← 26	← 25	← 24

Write a program that takes as input the points *A* and *B* (positive integers) and a time *T* (in seconds), and outputs the location of the robot at time *T*.

Note that *A* and *B* will be different, but *A* may be either smaller or larger than *B*. Note also that *T* may be quite large (up to 2^{63}-1), as shown in the fourth test case below.

The following test cases illustrate the required I/O format.

```
Enter A, B, and T: 5 13 27
Location at time 27: 10

Enter A, B, and T: 1 500 12345
Location at time 12345: 370

Enter A, B, and T: 787 23 12345
Location at time 12345: 666

Enter A, B, and T: 3 97 12345678987654321
Location at time 12345678987654321: 84
```

HALF CREDIT: Solve this problem assuming *T* < 1,000,000.

8. Greatest Chain of Being

Given a list of positive integers, you can chain them together in some order to form a single number. You would like to form the greatest possible number in this way. For example, suppose the list contains 20, 3005, and 2. The six possible ways in which these numbers can be chained together are:

- `20, 3005, 2 → 2030052`
- `20, 2, 3005 → 2023005`
- `3005, 20, 2 → 3005202`
- **`3005, 2, 20 → 3005220`**
- `2, 20, 3005 → 2203005`
- `2, 3005, 20 → 2300520`

As you can see, 3005220 is the greatest number that can be formed by chaining 20, 3005, and 2.

Write a program that prompts the user for the numbers in a list and outputs the greatest chain.

The following test cases illustrate the required I/O format.

```
Size of list: 3
Numbers in list: 20 3 24
Greatest chain: 32420

Size of list: 4
Numbers in list: 4056 909 99 500
Greatest chain: 999095004056

Size of list: 7
Numbers in list: 41 26 50 1000 2345 3 2018
Greatest chain: 50413262345201810000
```

You may assume at most 10 numbers in the list, with each number being less than a million.

HALF CREDIT: Solve this problem assuming at most 6 numbers, each less than 100.

9. Greatest Prime Factor

For an integer $k \geq 2$, let's define $g(k)$ as the greatest prime factor of k. For example:

- $g(10) = 5$
- $g(17) = 17$
- $g(24) = 3$

Now define $s(n)$ as the sum of the greatest prime factors of the numbers from 2 to n.

For example, the greatest prime factors of $k = 2, 3, ..., 10$ are:

k	2	3	4	5	6	7	8	9	10
$g(k)$	2	3	2	5	3	7	2	3	5

Therefore $s(10) = 2 + 3 + 2 + 5 + 3 + 7 + 2 + 3 + 5 = 32$.

Write a program that prompts the user for an integer $n \leq 1000000$ and outputs $s(n)$.

The following test cases illustrate the required I/O format.

```
Input: 10
s(10) = 32.

Input: 5000
s(5000) = 2745481.

Input: 999999
s(999999) = 64937323257.
```

HALF CREDIT: solve this problem assuming $n \leq 1000$.

10. Longest Reversible Remainder (LRR)

We will say that a sequence of characters is *reversible* if it reads the same forwards as backwards, like ABCCBA and ROTATOR.

Given a sequence of characters, a *remainder* of the sequence is obtained by deleting one or more of its characters. For example, SADTOY is a remainder of STAR DESTROYER (spaces are ignored):

```
STAR DESTROYER
S-A- D--T-OY--
```

Write a program that prompts the user for a line of text and outputs the longest reversible remainder (LRR) and its length. If the LRR is not unique, output the one that occurs first when reading from left to right. For example, ABCCBDDCE has two reversible remainders of length 4: BCCB and CDDC. Reading from left to write, BCCB occurs first so that is the one we are looking for.

You may assume that the input consists only of uppercase letters (A-Z) and possibly spaces. The spaces are ignored for purposes of calculating the LRR. So "STAR DESTROYER", for example, is treated just like "STARDESTROYER".

The following test cases illustrate the required I/O format (including spaces between words).

```
Enter text: STAR DESTROYER
Longest reversible remainder: RESER 5

Enter text: RECKLESS IS HE
Longest reversible remainder: ESSSE 5

Enter text: AN ELEGANT WEAPON
Longest reversible remainder: ANELENA 7

Enter text: THE CIRCLE IS NOW COMPLETE
Longest reversible remainder: TECIRICET 9

Enter text: SET YOUR COURSE FOR THE HOTH SYSTEM
Longest reversible remainder: ETYOROUOROYTE 13

Enter text: SHUT DOWN ALL THE GARBAGE MASHERS ON THE DETENTION LEVEL
Longest reversible remainder: HTOAEGARAGEAOTH 15
```

You may assume that the input will have at most 50 letters.

HALF CREDIT: Solve this problem for inputs having at most 20 letters. With this restriction on the size of the input, the problem can be solved by generating all possible remainders and checking each one to see if it is reversible.