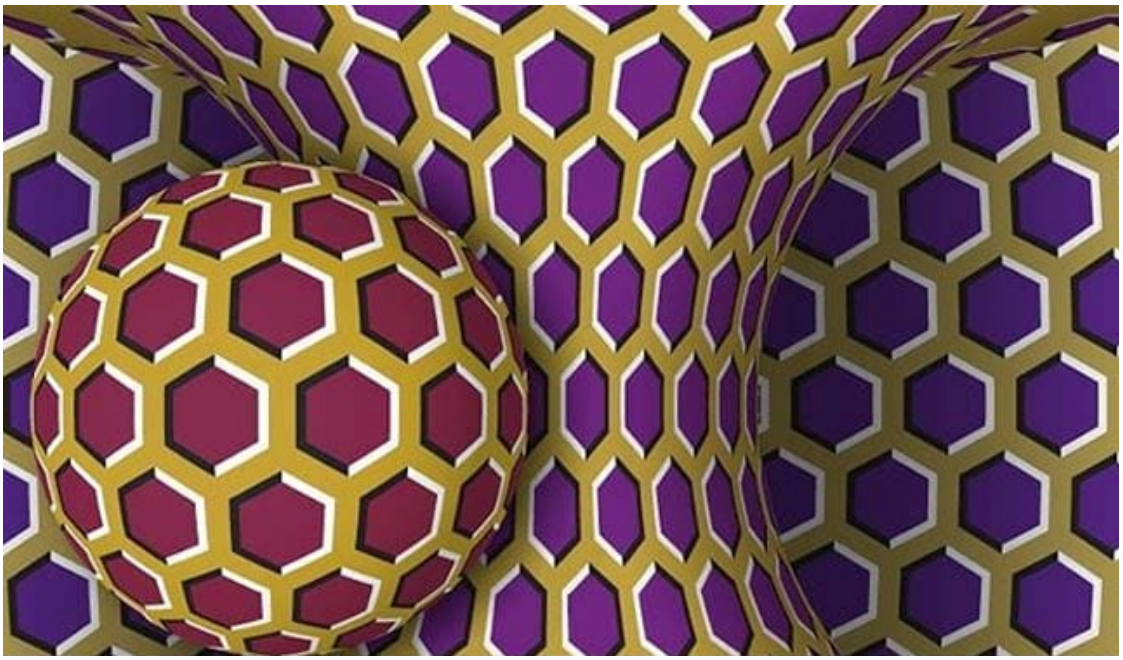
24th Annual High School Programming Contest

April 9, 2019



Department of Mathematical and Digital Sciences
Bloomsburg University

1. Good Luck

This problem deals with numbers comprised of three decimal digits. We allow leading zeros, so numbers like 007 and 023 count. Among all 3-digit numbers, those that start and end with the same digit are *lucky*; all others are unlucky. Here is a way to transform any 3-digit number *K* into a lucky number.

- Let *x* be the number formed by writing the digits of *K* in ascending order.
- Let *y* be the number formed by writing the digits of *K* in descending order.
- Let *z* be the number consisting of the median digit of *K* written 3 times.
- Calculate *x* + *y* - *z*.

For example, if *K* = 895 then we calculate as follows.

- x = 589
- y = 985
- z = 888

The resulting lucky number is 686 = 589 + 985 - 888.

Write a program that prompts the user to enter a 3-digit number and then outputs the corresponding lucky number. The following execution snapshots show the required I/O format.

```
Enter a 3-digit number: 123
Good luck: 222

Enter a 3-digit number: 501
Good luck: 414

Enter a 3-digit number: 23
Good luck: 121

Enter a 3-digit number: 845
Good luck: 757

Enter a 3-digit number: 7
Good luck: 707
```

HALF CREDIT: Your program may read the input as three separate digits.

```
Enter a 3-digit number: 1  2  3
Good luck: 222

Enter a 3-digit number: 0  2  3
Good luck: 121
```

2. Euclidean Cake

Happy birthday! Your best friend has baked a cake for you. It's rectangular. You have resolved to eat just one slice each day and it must be a square slice (tastes better that way). Each day, you will eat the largest square slice that can be obtained with a single cut of the knife.

To illustrate, consider an 8 x 12 cake. The following artwork shows the cake in its original state and each day after a slice has been removed and eaten.

(a) The cake as given: 8 x 12.
(b) The cake after an 8 x 8 slice has been eaten.
(c) The cake after a 4x4 slice has been eaten.
(d) The cake after another 4x4 slice (all that remains) has been eaten.

Write a program that outputs the size of the cake each day until it is gone. The user may enter the length and width in either order, but your program must display all dimensions with the shortest side first.

```
Enter dimensions of cake: 30 42
Cake on day 1: 30 x 42
Cake on day 2: 12 x 30
Cake on day 3: 12 x 18
Cake on day 4: 6 x 12
Cake on day 5: 6 x 6
Cake on day 6: gone

Enter dimensions of cake: 21 13
Cake on day 1: 13 x 21
Cake on day 2: 8 x 13
Cake on day 3: 5 x 8
Cake on day 4: 3 x 5
Cake on day 5: 2 x 3
Cake on day 6: 1 x 2
Cake on day 7: 1 x 1
Cake on day 8: gone

Enter dimensions of cake: 12 36
Cake on day 1: 12 x 36
Cake on day 2: 12 x 24
Cake on day 3: 12 x 12
Cake on day 4: gone
```

HALF CREDIT: You may assume the dimensions of the cake are such that it will be sliced at most 3 times.

3. Exam Strategy

Course grades are determined by four exams, each consisting of 100 questions. Each question is worth a single point, so possible exam scores are between 0 and 100 inclusive. The average of the four exam scores is converted to a letter grade according to the following scale.

- A = [90, 100]
- B = [80, 90)
- C = [70, 80)
- D = [60, 70)
- F = [0, 60)

The notation [*x*, *y*) means at least *x* and less than *y*.

So far, you have taken 3 exams. You have prepared well for the 4th exam and you are confident that you can answer every question correctly. But you do not want to waste your time answering more questions than necessary. Write a program that prompts the user for the first 3 exam scores and then outputs the smallest number of questions that can be answered on the 4th exam to earn the highest possible grade.

The following execution snapshot illustrates the required I/O format:

```
First three exam scores: 63 48 91
Answer 78 questions on the fourth exam to earn a grade of C.
```

Explanation: If you earn 100 points on the last exam, your average would be 75.5, so a C is the highest possible grade you could earn. But you would also earn a C with only 78 points on the last exam. Any fewer points, however, would result in a grade lower than C.

More examples:

```
First three exam scores: 85 93 80
Answer 62 questions on the last exam to earn a grade of B.

First three exam scores: 91 88 95
Answer 86 questions on the last exam to earn a grade of A.

First three exam scores: 50 55 61
Answer 74 questions on the last exam to earn a grade of D.

First three exam scores: 40 53 38
Skip the fourth exam.
```

Note that in the last example, the program advises the user to skip the fourth exam. This is because there is no possible way to pass the course. Might as well stay home.

HALF CREDIT: Write a program that outputs the highest and lowest possible course grade.

```
First three exam scores: 85 93 80
Highest possible grade: B
Lowest possible grade: D
```

4. Numeronym

A numeronym (nu-MER-o-nym) is an abbreviation formed by replacing part of a word with a number. In particular, the first and last letter of the word are kept, and the second letter is kept if it is not a vowel (aeoiu). The intervening letters are deleted and in their place the number of deleted letters is written.

The word *robot*, for example, is abbreviated by *r3t*. The first and last letter (*r* and *t*) are kept and the intervening three letters are replaced by the number 3.

The word *droid* is abbreviated by *dr2d*. The first and last letter (*d* and *d*) are kept, and the second letter (*r*) is not a vowel so it is also kept. The two letters between *dr* and *d* are replaced by the number 2.

The number substitution takes place only if the resulting abbreviation would be shorter than the original word. So, for example, *star* is abbreviated by *star*, not *st1r*.

Write a program that prompts the user for a line of text and then outputs the numeronymic abbreviation of each word. A *word* is defined to be a maximal sequence of alphabetic letters. Other characters such as punctuation symbols are treated just like spaces, namely as separators between words.

```
Enter text: Beware my great-grandfather's ghost (behind you)!
B4e my gr2t-gr8r's gh2t (b4d you)!
```

The following table shows each word of this text and its abbreviation.

Beware	my	great	grandfather	s	ghost	behind	you
B4e	my	gr2t	gr8r	s	gh2t	b4d	you

Here are some more execution snapshots.

```
Enter text: I think it's today, yeah.
I th2k it's t3y, y2h.

Enter text: tangerine trees and marmalade skies
t7e tr2s and m7e sk2s

Enter text: Alright, alright... (fade out)
Al4t, al4t... (f2e out)
```

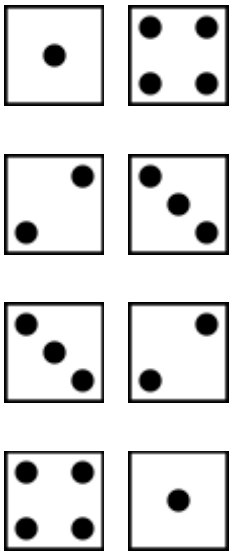
HALF CREDIT: Assume the input consists of letters and spaces only (no punctuation or other symbols).

```
Enter text: tangerine trees and marmalade skies
t7e tr2s and m7e sk2s
```

5. Dice Sums

Write a program that prompts the user for two positive integers, *N* and *S*, and then outputs the number of ways that *N* rolls of a 6-sided die can sum to *S*. You may assume $N \leq 20$.

For example, if $N = 2$ and $S = 5$ then the program outputs 4. The 2-roll sequences that sum to 5 are shown below.



The following execution snapshots illustrate the required I/O format.

```
Enter number of rolls and target sum: 5 10
Number of 5-roll sequences summing to 10: 126

Enter number of rolls and target sum: 8 24
Number of 8-roll sequences summing to 24: 98813

Enter number of rolls and target sum: 13 40
Number of 13-roll sequences summing to 40: 570374701

Enter number of rolls and target sum: 15 50
Number of 15-roll sequences summing to 50: 26199964377

Enter number of rolls and target sum: 20 60
Number of 20-roll sequences summing to 60: 81987009993775
```

HALF CREDIT: There are different ways to solve this problem, but some of them might be too slow except for small values of *N*. For half credit, you may assume $N \leq 10$.

6. Text Square

Write a program that prompts the user to enter a line of text and then writes it clockwise around the perimeter of a square. The number of characters per side of the square is the smallest possible for which all of the characters in the text will fit.

The following execution snapshot illustrates the required I/O format.

```
Enter text: AND IN HER EYES YOU SEE NOTHING

AND IN HE
        R
G
N       E
I       Y
H       E
T       S
O
N EES UOY
```

Here the text wraps around a 9 x 9 square (each row and column has 9 characters, including spaces).

See the next page for more examples.

HALF CREDIT: Solve the problem as described above except that # symbols appear all the way around the square.

```
Enter text: AND IN HER EYES YOU SEE NOTHING

#########
#       #
#       #
#       #
#       #
#       #
#       #
#       #
#########
```

The dimensions of the square are just as they would be for the full-credit version of the problem.

Enter text: **WORDS ARE FLOWING OUT LIKE ENDLESS RAIN INTO A PAPER CUP**

```
WORDS ARE FLOWI
P              N
U              G
C
               O
R              U
E              T
P
A              L
P              I
               K
A              E

O              E
TNI NIAR SSELDN
```

Enter text: **KEEPING AN EYE ON THE WORLD GOING BY MY WINDOW**

```
KEEPING AN EY
            E

W           O
O           N
D
N           T
I           H
W           E

Y           W
M           O
 YB GNIOG DLR
```

7. XYZ

An *XYZ sequence* is a sequence of letters containing only *X*'s, *Y*'s, and *Z*'s. We will call such a sequence *special* if contains neither *XY* nor *YZ*.

For example, *XZYXZY* and *ZYXXZ* are special. On the other hand, *XZXYXZ* is not special because it contains *XY*, and *ZYYYZZ* is not special because it contains *YZ*. Note that *YXYYZX* is not special since it contains both *XY* and *YZ*.

There are 16 special *XYZ* sequences of length 3:

```
XXX XXZ XZX XZY XZZ YXX YXZ YYX
YYY ZXX ZXZ ZYX ZYY ZZX ZZY ZZZ
```

Write a program that prompts the user for the sequence length and then outputs the number of special *XYZ* sequences. You may assume that the length will be at most 40.

The following execution snapshots illustrate the required I/O format.

```
Sequence length: 2
Number of special XYZ sequences: 7

Sequence length: 3
Number of special XYZ sequences: 16

Sequence length: 5
Number of special XYZ sequences: 86

Sequence length: 25
Number of special XYZ sequences: 1828587033
```

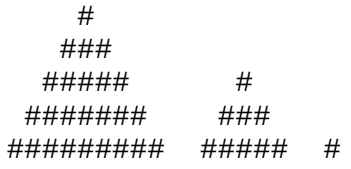
HALF CREDIT: Solve the problem for lengths at most 15. With this restriction, a simpler approach is possible.

8. Pyramids

Given a collection of identical 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:

Each row of a stack has an odd number of boxes.

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



Usually there are many possible ways to arrange a given number of boxes into triangular stacks. We want the combination with the highest possible first stack, and if there are several such combinations then we want the highest possible second stack, and so on.

Write a program that draws the stacks for a user-specified number of boxes. The execution snapshots below illustrate the required I/O format.

```
How many boxes? 35

    #
   ###
  #####       #
 #######     ###
#########  #####  #


How many boxes? 79

      #
     ###
    #####           #
   #######         ###
  #########       #####
 ###########     #######     #
#############  #########  ###  #
```

9. Latin Square Failure

A Latin square of order *N* is an *N* x *N* grid with the first *N* letters of the alphabet in each row and column. For example, here is a Latin square of order 3:

```
A B C
C A B
B C A
```

And here is a Latin square of order 4:

```
C B A D
B A D C
A D C B
D C B A
```

We would like to have a technique for constructing a Latin square of a given order, starting with a given letter in the top-left position. One idea would be to traverse the grid row by row, and left to right within each row, putting at each position the least letter that has not already appeared in that row or column.

To illustrate the idea, suppose that you want to construct a Latin square of order 4 that starts with the letter B. The first row would be constructed like this:

```
B . . .     B A . .     B A C .     B A C D
```

Continuing to construct the second row:

```
B A C D     B A C D     B A C D     B A C D
A           A B         A B D       A B D C
```

Continuing to construct the third row:

```
B A C D     B A C D     B A C D     B A C D
A B D C     A B D C     A B D C     A B D C
C           C D         C D A       C D A B
```

And finally the fourth row:

```
B A C D     B A C D     B A C D     B A C D
A B D C     A B D C     A B D C     A B D C
C D A B     C D A B     C D A B     C D A B
D           D C         D C B       D C B A
```

You can see that the completed grid is in fact a Latin square.

Unfortunately, this technique does not always work. If you try to construct a Latin square of order 4 that starts with C, you will run out of options at the end of the second row:

```
C A B D
A B C ?
```

Write a program that prompts the user for the order of the square and the starting letter, and then outputs the grid as it would appear just before the construction fails. If the construction succeeds, the output will be a completed Latin square.

The following execution snapshots illustrate the required I/O format.

```
Enter order and starting letter: 4 C
C A B D
A B C
```

```
Enter order and starting letter: 5 C
C A B D E
A B C E D
B C A
```

```
Enter order and starting letter: 6 B
B A C D E F
A B D C F E
C D A B
```

```
Enter order and starting letter: 7 C
C A B D E F G
A B C E D G F
B C A F G D E
D E F A B C
```

```
Enter order and starting letter: 8 A
A B C D E F G H
B A D C F E H G
C D A B G H E F
D C B A H G F E
E F G H A B C D
F E H G B A D C
G H E F C D A B
H G F E D C B A
```

Note that consecutive letters in a row are separated with a space for clarity.

10. Splitting Up

Write a program that prompts the user for a string of non-zero digits and then splits the string into a sorted sequence of two or more integers. There may be several ways to do this. For example, 81314527 can be split in four ways:

```
8, 13, 14, 527
8, 131, 4527
81, 314, 527
813, 14527
```

When there are several solutions, we want the one that minimizes the last number. If there are several of *these*, ties are broken by taking the sequence with the largest first value; if there are still ties, they are broken by taking the sequence with the largest second value, and so on.

You may assume that the input string has at most 9 digits.

```
Enter a string of non-zero digits: 81314527
81, 314, 527

Enter a string of non-zero digits: 23149768
23, 149, 768

Enter a string of non-zero digits: 483769325
4, 8, 37, 69, 325

Enter a string of non-zero digits: 987654321
98, 765, 4321

Enter a string of non-zero digits: 999999999
9, 9, 9, 9, 9, 9, 9, 9, 9
```

The commas separating terms of the sequence are required.