25th Annual High School Programming Contest

April 12, 2022

Department of Mathematical and Digital Sciences
Bloomsburg University

1. Happiness

Alice, Bob, and Charlie are billionaires. Each has a net worth in billions that we will denote by *a*, *b*, and *c* respectively. These values are whole numbers.

Naturally, these three billionaires would be embarrassed to have a net worth that is not larger than the group average, so each calculates how many more billions he or she would need in order to be wealthier than average. That is the key to their happiness.

To illustrate the idea, suppose *a* = 3, *b* = 2, and *c* = 5. Their average wealth would be 3.33 billion. Alice, having only 3 billion, is ashamed. But if she had 1 billion more, then the average would be 3.67, and in that case her 4 billion would be above average. That's better!

Continuing this example, let's look at Bob. He has 2 billion, which is below average. You might be tempted to think that he needs another 2 billion to be above average, but in that case the amounts would be *a* = 3, *b* = 4, and *c* = 5, increasing the average to 4; although Bob's wealth would be equal to the average, it would not be greater. He would need 3 billion more to rise *above* the average and thereby achieve happiness.

Finally, there is Charlie. With his 5 billion, he is already above average, so he needs nothing more.

Write a program that prompts the user for the net worth of Alice, Bob, and Charlie (in billions), and outputs the additional amounts needed for them to be happy. In the example considered above, Alice needs 1 more, Bob needs 3 more, and Charlie needs nothing more.

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

```
Enter net worth for Alice, Bob, and Charlie: 3 2 5
Billions more needed for happiness: 1 3 0

Enter net worth for Alice, Bob, and Charlie: 8 6 8
Billions more needed for happiness: 0 3 0

Enter net worth for Alice, Bob, and Charlie: 6 3 3
Billions more needed for happiness: 0 2 2
```

2. Chuckwalla

Chuckwalla is a dice game for one player. The game consists of a single roll of three dice. The goal is to arrange the rolled numbers to form a true arithmetic statement of the form *X* OP *Y* = *Z*, where OP denotes either addition or multiplication. If this is possible, the player wins, and otherwise the player loses.

Write a program that prompts the user for three rolled numbers and outputs a win/lose message as shown below.

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

```
Enter three rolled numbers: 2 5 3
Chuckwalla! You WIN.

Enter three rolled numbers: 1 3 6
No Chuckwalla! You LOSE.

Enter three rolled numbers: 6 3 2
Chuckwalla! You WIN.

Enter three rolled numbers: 2 4 5
No Chuckwalla! You LOSE.
```

The first example (**2 5 3**) is a winning roll because 2 + 3 = 5. The third example (**6 3 2**) is a winning roll because $3 \times 2 = 6$.

3. Inheritance

The Knaster Inheritance Procedure is a system for fairly dividing an object of value among three or more people. The basic idea is that the person who values the object most gets it and then buys out the others with a certain amount of cash. Let *n* denote the number of people. The procedure consists of the following steps.

1. Each participant submits a sealed bid stating the cash value of the object. Value is subjective: the object may be worth more to one person than to another.
2. The bids are revealed and the highest bidder gets the item. (You may assume the highest bid is unique.) This person then deposits into a pot an amount equal to (*n*-1)/*n* times the highest bid. For example, if the highest bid among 4 participants is $1000 then the high bidder gets the object and deposits $750 (3/4 of $1000) into the pot.
3. Each other participant withdraws 1/*n* of the object's value according to his or her bid.
4. Any remaining money in the pot is divided equally among all participants.

You may assume that all bids are whole numbers. Fractional deposits and withdrawals are truncated. To illustrate the entire process, suppose 3 people have inherited a tambourine. Their bids are as follows:

- John: 2400
- Paul: 3000
- George: 2700

Paul is the high bidder so he gets the tambourine and deposits 2000 into the pot (2/3 of his bid). Next, John takes 800 (1/3 of his bid) and George takes 900 (1/3 of his bid). That leaves 300 in the pot, which is shared equally among all three participants (100 each). This leads to the following results.

- John gets 900 (800 initially plus 100 at the end).
- Paul gets the tambourine and pays 1900 (2000 initial deposit minus 100 received at the end).
- George gets 1000 (900 initially plus 100 at the end).

Write a program that prompts the user for the bids and outputs the results. The participants will be denoted by A, B, C, and so on. You may assume there are 3-5 participants (but as mentioned below, for half credit you may solve the problem assuming exactly 3 participants).

The following execution snapshots (continued on the next page) illustrate the required I/O format.

```
Enter bids: 2400 3000 2700
A gets 900.
B gets the object and pays 1900.
C gets 1000.
```

```
Enter bids: 60 80 52 40
A gets 20.
B gets the object and pays 55.
C gets 18.
D gets 15.
```

The explanation for the second example is as follows.

1. B is the high bidder at 80. Since there are 4 participants, B deposits 60 into the pot (3/4 of 80); however, B will receive 5 back at the end (see below), so in total B pays 55.
2. The other participants withdraw 1/4 of their bids:
   - A gets 15.
   - C gets 13.
   - D gets 10.
3. That leaves 22 dollars in the pot, which is divided equally among all; that is, each receives 5 more. (22 divided four ways would actually be 5.50, but remember, fractional amounts are truncated in this problem for simplicity.)

Half credit: Solve this problem assuming exactly 3 participants.

4. Decoder

General Argon sends a message to his field commander concerning tomorrow's battle. He encrypts the message just in case it should fall into enemy hands. The idea is to replace each letter of the alphabet with its index: A=1, B=2, C=3, and so on. If the message is, say, ATTACK AT DAWN, then ATTACK is encrypted as 120201311 since A=1, T=20, T=20, A=1, C=3, and K=11.

The commander must decode the message by translating the digit sequence back into letters. Unfortunately, there may be several ways to do this depending on how the encoding is broken into numbers between 1 and 26. For example, there are four ways in which the encoding of ATTACK (120201311) can be decoded:

1. ATTACAA (1, 20, 20, 1, 3, 1, 1)
2. ATTACK (1, 20, 20, 1, 3, 11)
3. ATTMAA (1, 20, 20, 13, 1, 1)
4. ATTMK (1, 20, 20, 13, 11)

Note that when decoding 120201311 you cannot start with L=12, because 0 comes after 12 and there is no letter whose encoding starts with 0.

Write a program that prompts the user for a word in uppercase letters, encodes it, and outputs all possible ways in which the encoding could be decoded. Display the results in dictionary order. The following execution snapshot illustrates the required I/O format.

```
Enter a word: ATTACK
ATTACAA
ATTACK
ATTMAA
ATTMK

Enter a word: FIGHT
FIGHT

Enter a word: CHARGE
CHAAHGE
CHARGE
CHKHGE
```

Half credit: Write a program prompts the user for a word in uppercase letters, encodes it, and outputs a single decoding, namely the first one to occur in dictionary order. In other words, decode from left to right and whenever there are multiple ways to obtain the next letter, choose the one that starts with the alphabetically smaller letter; for example, decode 13 as AC, not M.

```
Enter a word: ATTACK
ATTACAA

Enter a word: CHARGE
CHAAHGE
```

5. Switchback

Write a program that prompts the user for a line of text and a width, and displays the text on separate lines of the specified width (number of characters per line). The text is written from left to right on the first line, right to left on the second line, and so on, winding back and forth from one line to the next as shown in the following execution snapshots.

```
Enter text: papaya peach pear persimmon pineapple
Enter width: 8
papaya p
aep hcae
r persim
enip nom
apple

Enter text: panda pelican penguin pig porcupine
Enter width: 10
panda peli
iugnep nac
n pig porc
     enipu
```

Note that a line of output may begin or end with a space, as in the following example.

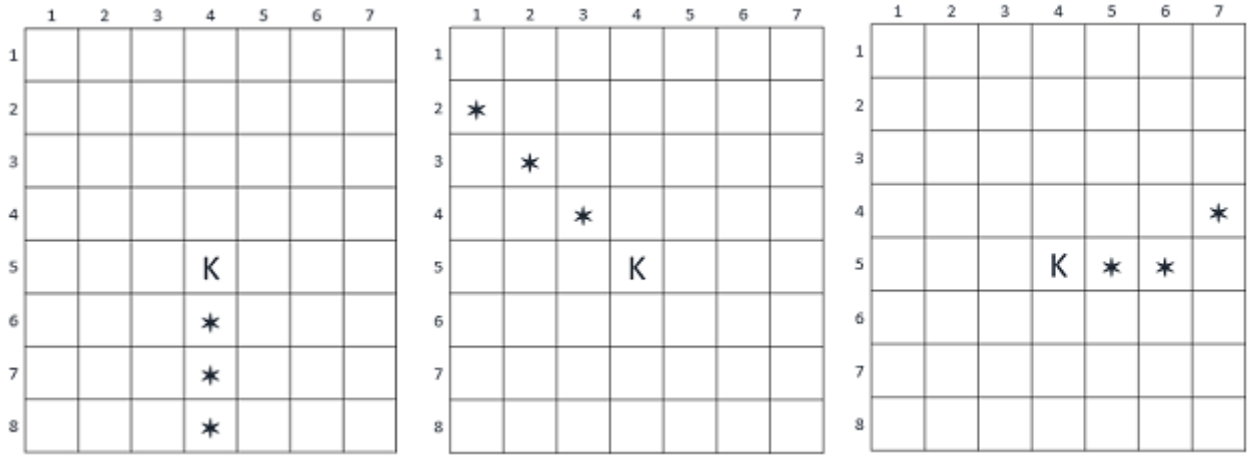
```
Enter text: Peter Piper picked a peck of pickled peppers.
Enter width: 7
Peter P
ip repi
cked a
fo kcep
 pickle
eppep d
rs.
```

Half credit: Solve the problem assuming width = 3.

```
Enter text: birch maple oak
bir
 hc
map
 el
oak
```

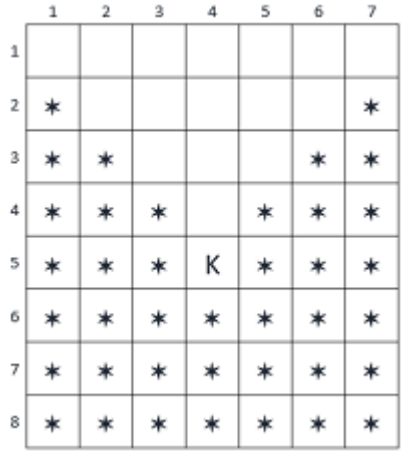
6. Minimal Boundary Paths

In chess, a king can move one square in any direction (horizontally, vertically, or diagonally). Suppose a king is placed on a certain square of an M x N board. We define a *boundary path* to be the sequence of squares occupied by the king as it moves to the boundary of the board. To illustrate the idea, suppose the king is at square (5, 4) on an 8 x 7 board. The following images show three different boundary paths.



Note that each of the boundary paths shown above consist of 3 moves and that it is impossible to reach the boundary in fewer moves. For this reason, we refer to these boundary paths as *minimal*. The starting square is considered part of a boundary path.

Write a program that prompts the user for the dimensions of the board and the starting position of the king, and outputs the fraction of squares that belong to a minimal boundary path. The following diagram shows the union of all minimal paths for the example considered above.



There are 56 squares on the board and, as you can see, 40 of them belong to a minimal boundary path, so the fraction of such squares is 40/56.

The fraction will not be reduced to lowest terms: the denominator will always equal the number of squares on the board.

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

```
Enter dimensions (rows and columns): 8 7
Enter starting square: 5 4
Fraction of squares on a minimal boundary path: 40/56
```

7. Smallest Divisible

Write a program that prompts the user for a positive integer n ≤ 40 and outputs the smallest positive integer divisible by every integer from 1 to n.

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

```
Enter a positive integer: 4
Smallest divisible by 1-4: 12

Enter a positive integer: 6
Smallest divisible by 1-6: 60

Enter a positive integer: 15
Smallest divisible by 1-15: 360360

Enter a positive integer: 40
Smallest divisible by 1-40: 5342931457063200
```

8. Look and Say

The *look-and-say sequence* is an infinite sequence of positive integers that begins like this:

1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ...

Each term after the first is determined by the one before it according to the look-and-say rule. To explain this rule, consider the term 1211. To read it aloud, one might say, "ONE 1, ONE 2, and TWO 1s." This verbal expression is encoded as 11 12 21, which gives us the next term of the sequence, namely 111221.

Fun fact: no digit other than 1, 2, and 3 will ever appear in the sequence.

Write a program that prompts the user for a positive integer $n \leq 40$ and outputs the last 10 digits of the n-th term of the look-and-say sequence (or the entire term if has fewer than 10 digits).

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

```
Enter a positive integer: 5
Term 5 of the look-and-say sequence: 111221

Enter a positive integer: 8
Term 8 of the look-and-say sequence: 1113213211

Enter a positive integer: 20
Term 20 of the look-and-say sequence: 1312113211

Enter a positive integer: 30
Term 30 of the look-and-say sequence: 2113112211
```

9. Fibonacci Runs

The infinite sequence of integers 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, … is known as the Fibonacci sequence. The first two terms of the sequence are 0 and 1, and thereafter each term is the sum of the two immediately before it.

A *Fibonacci run of order n* is an arrangement of the integers from 1 to *n* with the property that the sum of any two consecutive terms is a Fibonacci number. For example, consider the sequence (4, 1, 2, 3, 5). The sums of consecutive terms of this sequence are:

4 + 1 = 5
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8

These sums (5, 3, 5, 8) are all Fibonacci numbers, so (4, 1, 2, 3, 5) is a Fibonacci run of order 5.

There does not exist a Fibonacci run of order *n* for every *n*, but if one does exist then there is only one (not counting the reverse sequence). For example, the Fibonacci run of order 5 is (4, 1, 2, 3, 5). We take this one instead of the reverse sequence (5, 3, 2, 1, 4) since it starts with the smaller number.

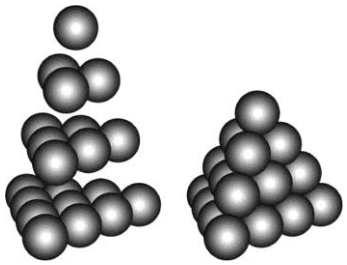
Write a program that prompts the user for a positive integer $n \leq 40$ and outputs the Fibonacci Run of order *n*.

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

```
Enter a positive integer: 5
Fibonacci run of order 5: 4 1 2 3 5

Enter a positive integer: 6
Fibonacci run of order 6: DOES NOT EXIST

Enter a positive integer: 10
Fibonacci run of order 10: DOES NOT EXIST

Enter a positive integer: 12
Fibonacci run of order 12: 4 9 12 1 7 6 2 11 10 3 5 8

Enter a positive integer: 33
Fibonacci run of order 33: 17 4 30 25 9 12 22 33 1 20 14 7 27 28 6 15
19 2 32 23 11 10 24 31 3 18 16 5 29 26 8 13 21
```

10. Tetrahedral Sums

A *tetrahedron*, or *triangular pyramid*, is obtained by stacking triangular arrangements of spheres.



The top layer consisting of a single sphere is considered to be a tetrahedron of height one, the top two layers form a tetrahedron of height two, the top three layers a tetrahedron of height three, and all four layers a tetrahedron of height four. This can be extended indefinitely to form a tetrahedron of any height.

Let $T(n)$ denote the n-th tetrahedral number, which is defined to be the number of spheres in the tetrahedron of height n. Looking at the picture, you can see that the first four tetrahedral numbers are:

- $T(1) = 1$.
- $T(2) = 4$.
- $T(3) = 10$.
- $T(4) = 20$.

The sequence of tetrahedral numbers continues with 35, 56, 84, 120, and so on.

There is a conjecture in number theory, called Pollock's conjecture, which states that every positive integer is the sum of at most five tetrahedral numbers. There are 241 known positive integers that cannot be expressed as the sum of only four tetrahedral numbers. The first five such integers are 17, 27, 33, 52, and 73. Let's denote these numbers $S(1)$, $S(2)$, …, $S(241)$.

Write a program that prompts the user for a positive integer n in the range [1, 241] and outputs $S(n)$.

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

```
Enter a positive integer: 1
S(1) = 17

Enter a positive integer: 2
S(2) = 27

Enter a positive integer: 10
S(10) = 137

Enter a positive integer: 100
S(100) = 3142

Enter a positive integer: 241
S(241) = 343867
```