CS Scholars Programming Hw3 - Written
Due Date: Friday 07/23 EOD

Name:

AndrewID:

For full credit on the assignment, complete either all Review + Core problems (#1-#9) or all Core + Spicy problems (#1-#3, #7-#12).

Bonus problems are related to the Advanced Track content, and are optional.

Core Problems - Written [30pts]

#1 - Loop Control Variables: While Loops - 9pts

Can attempt after While Loops lecture

Each of the following problem prompts could be implemented using a while loop. Identify the start value, continuing condition, and update action for the loop control variable you would use in that while loop. Assume that the loop control variable will be outputted at the beginning of the loop, and no conditional will be used. We've given an example of what this looks like in the first line

Ex) Output the numbers from 1 to 10, inclusive.

A) Output all even numbers between 2 and 20, exclusive on 20 (but not 2).

B) Output the numbers from 10 to 1, inclusive on both.

C) Output the numbers 3, 9, 15, 21.

Prompt	Start Value	Continuing Condition	Update Action
Ex	1	x <= 10	x = x + 1
A			
B			
C			

#2 - Code Tracing While Loops - 12pts

Can attempt after While Loops lecture

Given the following block of code, fill out a variable table that shows the values of the variables at the **end** of each iteration of the loop. You may not need to fill out values for every listed iteration.

```
x = 0
y = 10
z = 0
while x <= y:
    x = x + 3
    y = y + 1
    z = (x + y) - z
    print(x, y, z)
```

	x value	y value	z value
Pre-loop	0	10	0
Iter 1			
Iter 2			
Iter 3			
Iter 4			
Iter 5			
Iter 6			
Iter 7			
Iter 8			

#3 - Loop Control Variables: For Loops - 9pts

Can attempt after For Loops lecture

For each of the following range expressions, list all the values the loop variable will be set to over the course of the range. For example, `range(1, 5)` produces `1`, `2`, `3`, `4`.

Range Expression	**Numbers Produced**
`range(3)`	
`range(4, 8)`	
`range(1, 10, 3)`	

Programming Problems

Each of these problems should be solved in the starter file available on the course website. They should be submitted to the Gradescope assignment Hw3 - Programming to be autograded.

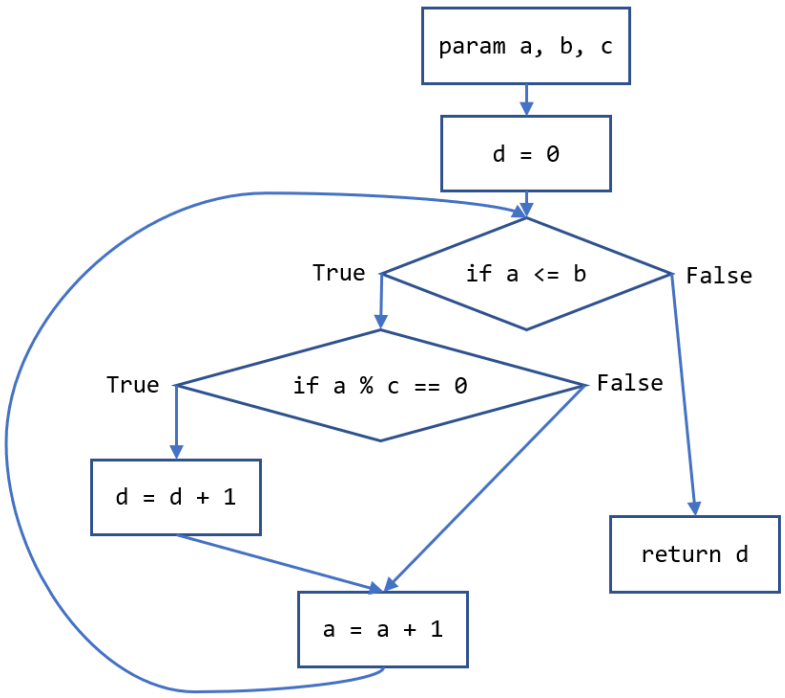
Make sure to 'Run file as Script' to check your work before submitting, then check the autograder feedback after you submit!

Review Problems - Programming [40pts]

#4 - Flow Chart to Program - 10pts

Can attempt after While Loops lecture

Given the control flow chart below, write a function `mysteryFunction(a, b, c)` that implements the control flow chart correctly. We recommend that you use a while loop.



#5 - `printSquare(n)` - 10pts

Can attempt after For Loops lecture

Write a function `printSquare(n)` which prints an ascii art square out of asterisks based on the integer `n`. For example, `printSquare(5)` would print the following:

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

Note that the square is five lines long, with each line having five asterisks. As another example, `printSquare(8)` would look like:

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

You'll want to create a loop (we recommend a for loop) where each iteration prints a single line of the square. To draw multiple stars on a single line, consider using a separate loop, or a loop nested in the printing loop

Note: `n` is guaranteed to be positive.

#6 - `printPrimeFactors(x)` - 20pts

Can attempt after Algorithmic Thinking and Style lecture

Write the function `printPrimeFactors(x)` which takes a positive integer `x` and prints all of its prime factors in a nice format.

A prime factor is a number that is both prime and evenly divides the original number (with no remainder). So the prime factors of 70 are 2, 5, and 7, because 2 * 5 * 7 = 70. Note that 10 is not a prime factor because it is not prime, and 3 is not a prime factor because it is not a factor of 70.

Prime factors can be repeated when the same factor divides the original number multiple times; for example, the prime factors of 12 are 2, 2, and 3, because 2 and 3 are both prime and 2 * 2 * 3 = 12. The prime factors of 16 are 2, 2, 2, and 2, because 2 * 2 * 2 * 2 = 16. We'll display repeated factors on a single line as a power expression; for example, 16 would display 2 ** 4, because 2 is repeated four times.

Here's a high-level algorithm to solve this problem. To find factors manually, iterate through all possible factors. When you find a viable factor, repeatedly **divide the number** by that factor until it no longer evenly divides the number. Our algorithm looks something like this:

1. Repeat the following procedure over all possible factors (2 to x)
    a. If x is evenly divisible by the possible factor
        i. Set a number count to be 0
        ii. Repeat the following procedure until x is not divisible by the possible factor
            1. Set count to be count plus 1
            2. Set x to x divided by the factor
        iii. If the number count is exactly 1
            1. Print the factor by itself
        iv. If the number count is greater than 1
            1. Print "f ** c", where f is the factor and c is the count

As an example, if you call `printPrimeFactors(600)`, it should print

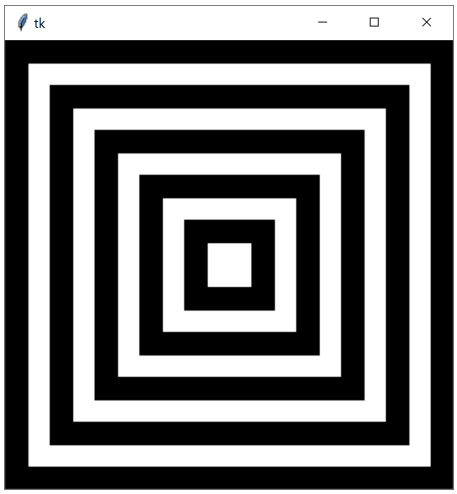
```
2 ** 3
3
5 ** 2
```

Core Problems - Programming [30pts]

#7 - `drawIllusion(canvas)` - 10pts

Can attempt after For Loops lecture

Write the function `drawIllusion(canvas)` which takes a Tkinter canvas and draws the illusion shown below. You must use a **loop** to do this; don't hardcode a large number of rectangles. We recommend that you use a for loop specifically.



Hint: it's easiest to make this illusion by drawing **overlapping squares**. Start with the largest black square, then draw the next-largest white square, etc. You'll need to draw 10 squares total. The canvas is 400px wide, so each square should be 20 pixels smaller on each side than the previous one (with the last square being exactly 40 pixels wide).

Another Hint: start by considering what the loop control variable should be. Which values need to change as you move to the next square? How do those values relate to the loop control variable? Consider our approach to drawing a grid in lecture as well.

#8 - `isPowerful(x)` - 10pts

Can attempt after Algorithmic Thinking and Style lecture

Write a function `isPowerful(x)` that checks whether an integer is powerful. A positive integer x is powerful if, for every prime y that divides x, y^2 also divides x. Note that we've provided `isPrime(x)` in the starter file for your use.

Hint: consider our discussion from lecture!

#9 - Fix the Style - 10pts

Can attempt after Algorithmic Thinking and Style lecture

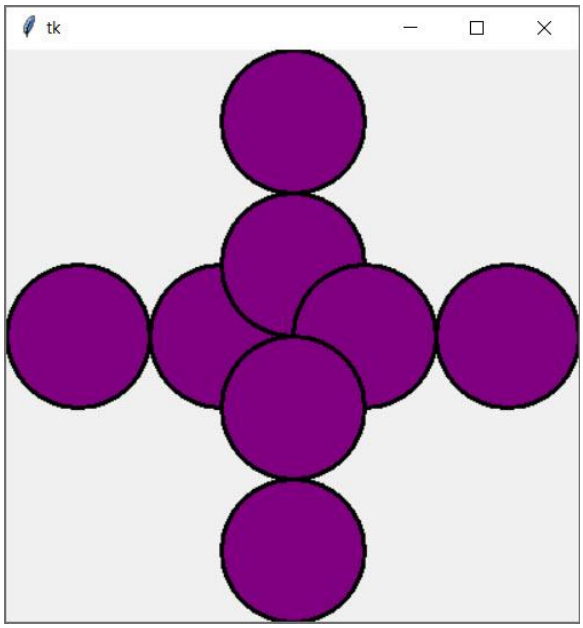
The programming starter file contains a program `drawPlus`, which draws an interesting plus sign using circles. The program is functionally correct, but we've written it with terrible style.

Fix this program so that it has better style according to the guidelines we discussed in class, **without** rewriting the main logic. Then, in a comment above the fixed program, write out a list of the changes you made to fix the style. You must make valid changes that cover at least two clarity style principles and and at least two robustness style principles.

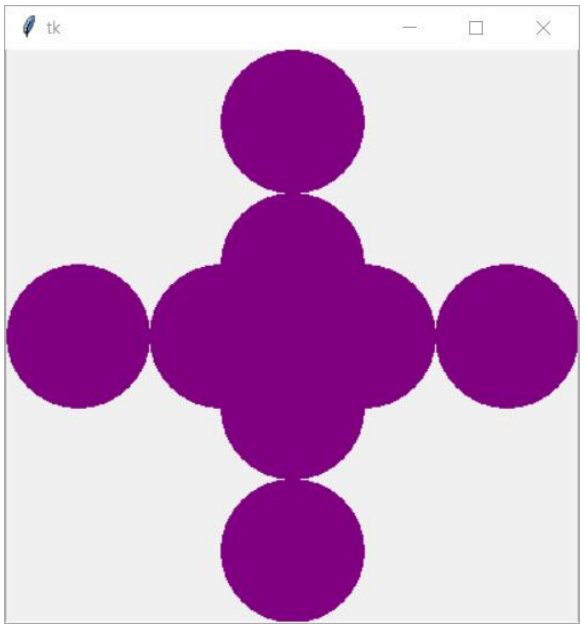
This problem is not autograded; we will hand-grade it instead.

Make sure your fixed program still produces the same outputs as before!

`drawPlus(canvas, True)`



`drawPlus(canvas, False)`



Spicy Problems - Programming [40pts]

#10 - `longestDigitRun(n)` - 10pts

Can attempt after While Loops lecture

Write the function `longestDigitRun(n)` that takes an integer `n` and returns the digit that has the longest consecutive run in that number. So `longestDigitRun(117773732)` returns 7 (because there is a run of 3 consecutive 7's), and `longestDigitRun(676886)` returns 8 (because there are 2 consecutive 8's). You are guaranteed that there will never be a tie for longest run.

#11 - `printTriangle(n)` - 10pts

Can attempt after For Loops lecture

Write a function `printTriangle(n)` which prints an ascii art triangle out of asterisks based on the integer `n`. For example, `printTriangle(5)` would print the following:

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

Note that the triangle is five lines long, with the top and bottom line each having only one asterisk, the second and second-from bottom lines each having two asterisks, etc. So `printTriangle(9)` would look like:

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

You'll want to create a loop (we recommend a for loop) where each iteration prints a single line of the triangle. To draw multiple stars on a single line, consider using a nested loop. Note that `n` is guaranteed to be positive and odd.

Hint: how can the program switch from increasing to decreasing? Consider using a conditional to check when you hit the midpoint, or two separate loops (one going up, one going down).

#12 - `drawThreadPattern(canvas, spokes, skips)` - 20pts

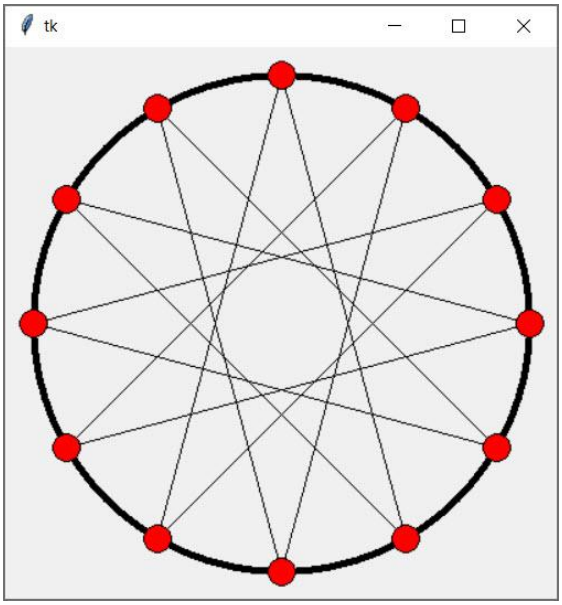
Can attempt after Algorithmic Thinking and Style lecture

In this problem, you'll create a thread pattern with graphics. A thread pattern (also known as a string art pattern) is made by creating a circle with several spokes placed around the perimeter, then running a thread around different spokes to create a desired pattern. Read more here: https://www.guidepatterns.com/35-string-art-patterns.php

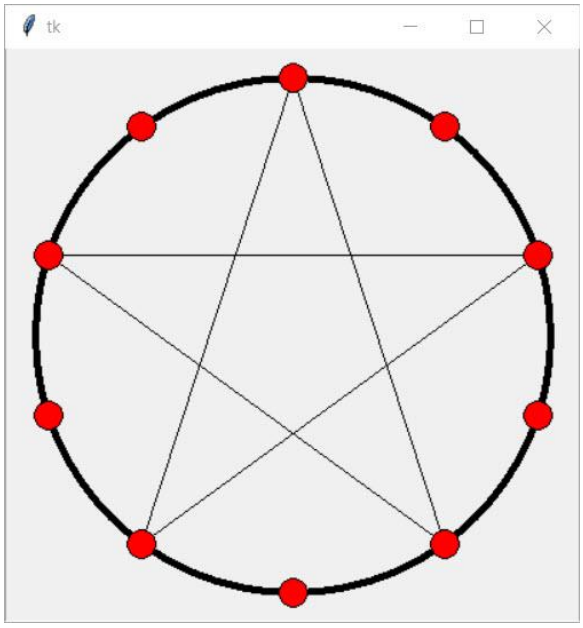
We'll create thread patterns algorithmically by specifying a number of spokes to place around the circle and how many spokes should be skipped each time the thread is moved to the next spoke. The thread will continue going between spokes until it reaches the spoke it originally started on. We'll number the spokes starting with 0 at the top, then increasing going counter-clockwise. So a circle with eight spokes would have spoke 2 where 3 would be on a clock.

Write the function `drawThreadPattern`, which takes three parameters: the canvas, the number of spokes in the circle, and the number of spokes to skip each time. This function should draw the thread pattern with the given parameters. You'll need to draw the circle (filling the screen with a small margin on each side), the spokes (as much smaller circles placed evenly around the circle), and the threads (as lines between spokes). A few examples are shown here:

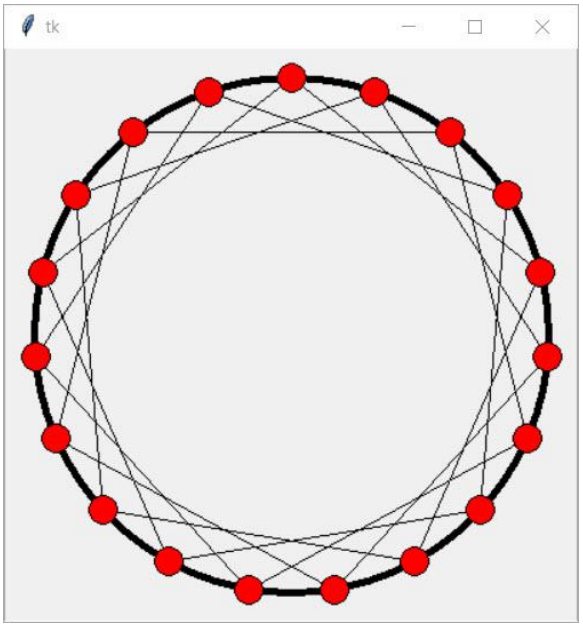
`drawThreadPattern(canvas, 12, 5)`



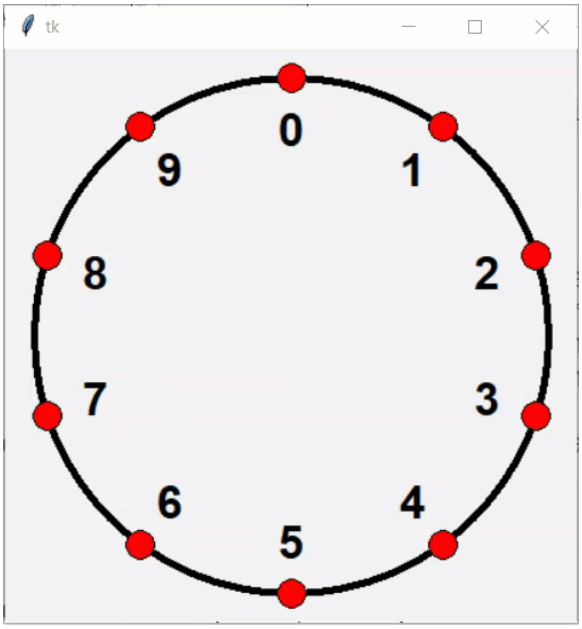
`drawThreadPattern(canvas, 10, 4)`



`drawThreadPattern(canvas, 19, 15)`



To provide further clarification, you can find a gif at this link - http://www.krivers.net/CSS-m21/hw/thread_pattern.gif - of a pattern with ten spokes that shows what each spoke's number would be. The gif shows how the thread pattern would be created in real time, starting from the top spoke and skipping 3 spokes each time. You do not need to create this image or animation- this is just here to help you understand the problem! Here's a still version of the gif:



Note: for this problem, anything we do not specify is a design decision. For example, you can decide the size of the margin, as long as it is reasonably close to the examples shown above.

Hint: how can you calculate where the spokes on the circle should be placed? Use trigonometry! If you know the center coordinate of a circle and its radius, you can calculate the coordinate of any point on the perimeter just by using its radius and SOH-CAH-TOA (https://www.mathsisfun.com/algebra/sohcahtoa.html).

The x coordinate is the adjacent leg of the triangle, and the radius is the hypotenuse, so we'll use the CAH equation. Make sure to offset by the center point!

$$x = center\,x + cosine(angle) \times r$$

The y coordinate is the opposite leg, so we'll want to use the SOH equation. Note that y coordinates are **reversed** in tkinter, so we need to subtract instead of adding (to move up instead of down).

$$y = center\,y - sine(angle) \times r$$

Another Hint: you'll want to use the math functions `math.cos` and `math.sin`. Note that these functions take **radians**, not degrees! If you'd rather work with degrees, use `math.radians` to convert them.

Bonus Problems [10pts]

Advanced Programming 1 - Recursion - 2pts

Assume you want to write a function `recursiveSum` that takes a positive integer, `n`, and **recursively** computes the sum from one to `n`.

For example, the result when calling the function on n=5 is 5+4+3+2+1 = 15.

What condition do you need to check for your **base case**?

What do you return in the **base case**?

What is the recursive call on a smaller problem in the **recursive case**?

How do you use the recursive call's result to solve the whole problem for `n` in the **recursive case**?

Advanced Programming 2 - Recursion - 3pts

In the programming starter file, write the function `powerSum(n, k)` that takes two non-negative integers `n` and `k` and returns the so-called power sum: $1^k + 2^k + ... + n^k$. You must use **recursion** to solve this problem: for loops, while loops, and the function `sum` are not allowed.

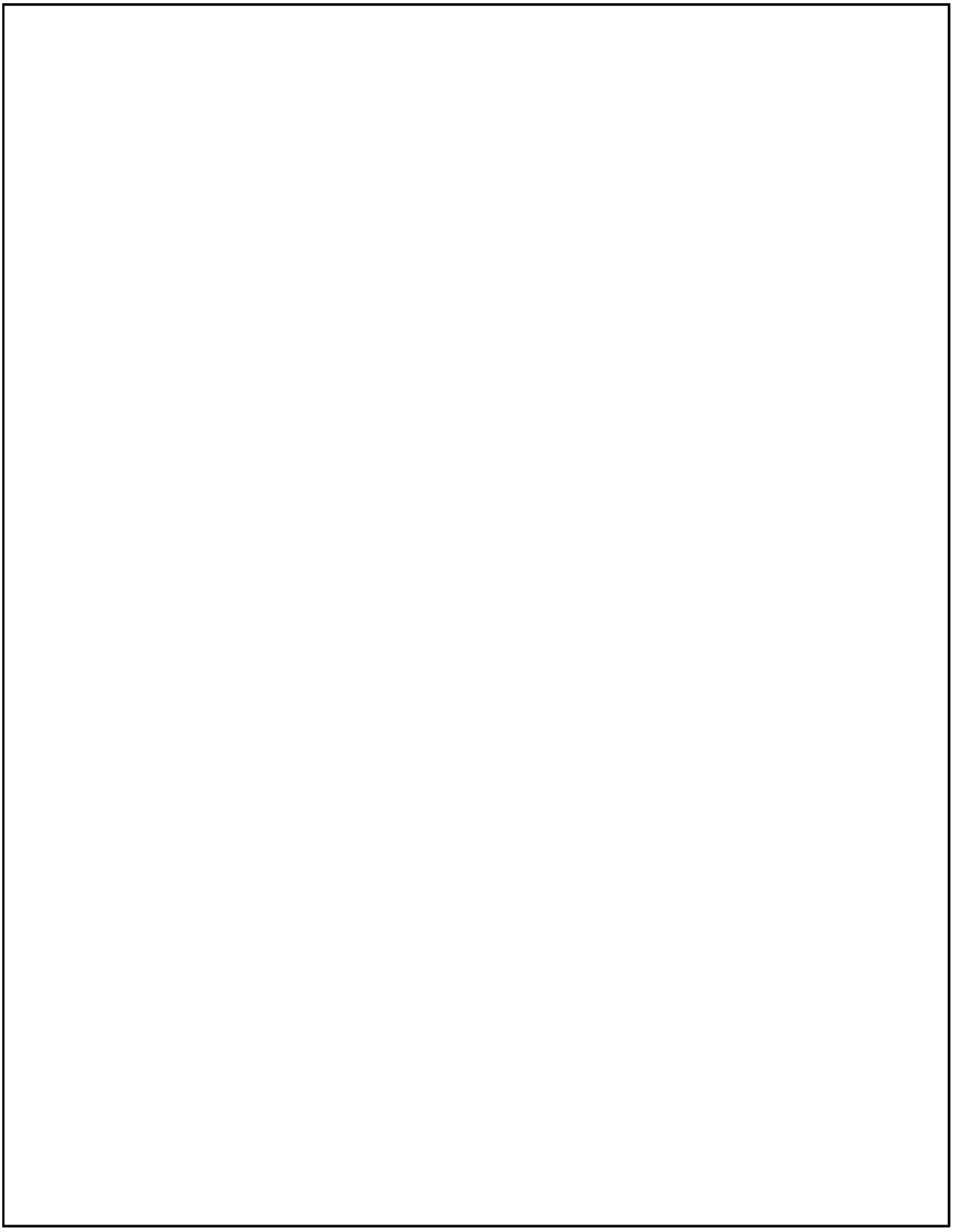
Note that the test function for `powerSum` is commented out; you'll need to uncomment it to test your function.

Advanced Computer Science 1 - Concurrency - 2pts

Exponentiation (raising a base to a power, like 2^4) can be computed concurrently by first multiplying pairs of bases (e.g., 2*2) together, and then multiplying those products together (4*4), and continuing until there is only one answer (16).

Draw a concurrency tree that computes 2^7. Note that you should only use **multiplication** as operations, not exponentiation. You can make a physical drawing and take a picture, or use an online image editing tool (like Google Drawings). Then upload your tree into the next page. You should be able to click on the square on the next page, then add your image directly onto the page. If that doesn't work, contact Prof. Kelly for assistance.

How many **total** steps does this tree take?	
How many **time** steps does this tree take?	



Advanced Computer Science 2 - Concurrency - 3pts

A factory with four workers produces custom t-shirts. To make a t-shirt, the workers follow these steps:

- [S] Set up supplies (for measuring) (5 minutes)
- [M] Measure the fabric (5 minutes)
- [S] Set up supplies (for cutting) (5 minutes)
- [C] Cut out the pattern (5 minutes)
- [S] Set up supplies (for sewing) (5 minutes)
- [W] Sew it all together (5 minutes)
- [F] Fold the shirt (5 minutes).

Note that setup occurs **once** before starting either measuring, cutting, or sewing. When you set up new supplies for a task, you put away the supplies for the previous task.

Continue to next page

Originally each worker made one shirt at a time, with all four workers working in parallel. Each of the cells in the following table represents five minutes, with the whole table representing an hour of work. Fill in the cells with the letters representing the steps to demonstrate the original system the factory used.

Worker	00:00	00:05	00:10	00:15	00:20	00:25	00:30	00:35	00:40	00:45	00:50	00:55
A												
B												
C												
D												

How many complete, folded shirts could be made by four workers in one hour with the original system?	

Recent budget changes have led to new restrictions on materials. Now the workers have to share a single sewing machine; in other words, only one worker can sew at any given point in time. They decide to use a new approach to make things more efficient.

Create a new schedule that uses **pipelining** to increase the efficiency of the shirt-making process. (**Hint:** think about the most efficient way to split up the tasks.)

Worker	00:00	00:05	00:10	00:15	00:20	00:25	00:30	00:35	00:40	00:45	00:50	00:55
A												
B												
C												
D												

How many complete, folded shirts could be made by four workers in one hour using the new pipeline?	