CSI 201
practice using while loops!

1. Goals for today: Practice, Explore, Practice!

2. Example #1: Repeating a task many times. Output a user input number of stars!

```
int i = 0;
int n = 1;
cin >> n;
while(i < n) { //loop "n" times
  cout << "*";
  i++;
}
cout << endl; // end my line of stars
```

3. Example #2: Restricting user input (plus using && and ||). User must give 1-10!

```
int user_input = 0;
cin >> user_input;
while(user_input < 1 || user_input > 10) {
  cerr << "User input was not 1-10, reading again:" << endl;
  cin >> user_input;
}
cout << "Input is now 1-10" << endl;
```

4. Show how to keep reading user input until the input lies between 1 and 20.

5. Show how to keep reading user input until the input does not lie between 1 and 10.

6. Show how to make sure the user inputs a positive number.

7. Show how to output "Ipsum lorem. ", 5000 times.

8. Read in an integer from the user. Write a loop that continues as long as that number (which started off as input) is more than 1. Inside the loop, change that number to be half as much as it started. Does this loop ever run infinitely? Add a counter to count how many times the loop has run! What is this number telling us?

9. Create integers for an evil dragon's hitpoints and the user's hitpoints. Make a while loop that continues as long as both the dragon's hitpoints and the user's hitpoints are more than 0. Inside the loop, update (change) the dragon's hitpoints and the user's hitpoints to indicate that they have taken damage by each other's attacks. Outside of the loop, output who won the battle. This is rather static and stale right now, but you might imagine grabbing some randomness from the Day 5 sheet to make this a little more interesting with randomness.

10. How could we make sure the user inputs an even number? If you have not learned about modulus (%), this might be a good time. % gives the remainder after a division. Thus 7%2 is 1 because 7/2 is 3R1. It can describe divisibility of a number. For example, when something a%b is 0 then that a is divisible by b evenly. For example 12%4 is 0 because 12/4 is 3R0. Anyway, even without modulus there are also some interesting integer arithmetic tricks we could use here. Given some integer, what is the result of that int / 2 * 2? Now try to make sure the inputs are odd? How about numbers that are a multiple of 7? How about numbers that are NOT a multiple of 4?

11. Make an integer variable called counter and initialize it to 0. Write code to read in two integer variables that represent the b and c in a quadratic equation. Write a loop that goes from -1,000,000,000 to 1,000,000,000 that will iterate over values of the variable a. If the quantity $b^2 - 4ac$ is bigger than or equal to 0 for that given iteration of the loop, then increment the counter variable by 1. Recall there is no ^ operator in C++. When the loop is complete, output the value of the counter variable. This describes how many times the quantity was bigger than 0 for those two million different input values. There are a few interesting things to notice about this. Might it be possible to find this quantity by using math instead of brute force? Other interesting thoughts: Is there a reason I restricted the search from -1b to 1b and didnt push for -2b to 2b?

12. The Madhava formula for estimating pi is:

$$\sqrt{12}\sum_{k=0}^{\infty}\frac{(-3)^{-k}}{2k+1}$$

To implement this, you'll need to `#include<cmath>` at the top of your code as we've seen before. You'll also use `sqrt(12)` and `pow(-3,-k)` to compute those pieces of the formula. The large e-like symbol (Σ) is called a sigma and it denotes the sum of many terms. To start your code, create a double sum variable and initialize it to 0. Then, create a counter variable (like i) called k that goes from 0 to 10 in the loop. At each step in the loop, add the result of $\frac{(-3)^{-k}}{2k+1}$ to the sum. Outside of the loop, multiply the entire thing by $\sqrt{12}$. This will give an estimation of pi. The higher you allow your counter variable to go, the better your pi approximation. How many iterations do we need until the output looks like it has a good approximation of pi? You can also put this in your loop to see more digits of pi too: `cout << setprecision(15) << sqrt(12)*sum << endl;`