General Idea

Create a small numerical library that also comes with a test harness that will be used to test the math functions. We are to implement sin, cos, arcsin, arccos, arctan, and log. We will compare our implemented functions to the ones in the `<math.h>` library and output them into a table to see the differences.

Pseudocode

These functions will be placed in a file called `mathlib.c`. Each implementation will be a separate function. Include a function to find the absolute value. Define Epsilon as 10^{-10}. Include a square root function.

For sin:
Create a function that will take in a value, *x*.
Have *y* start at *x*'s value.
Create a for-loop that will end once the absolute value of the outcome's value from the inside of the loop is greater than or equal to our Epsilon. We will initialize it to begin at 1 and update it by adding 1.

Multiply *x* by negative 1, then multiply that with *x* times *x*.

Multiply the for-loop's updated value by 2, then use this to divide the previous value. Then divide this output by 2 times the updated value plus 1.

The output will be added to *y* as it goes through the loop.

It will return the *y*.
Our test harness will go through the range of $(0,2\pi)$ with steps of 0.05π as it runs through the data.

For cos:
Create a function that will take in a value, *x*.
Have *previous*, a variable, start at 1.
Have *y* start at 1.
Create a for-loop that will end once the absolute value of the outcome's value from the inside of the loop is greater than or equal to our Epsilon. We will initialize it to begin at 1 and update it by adding 1.

Multiply *previous* by negative 1, then multiply that with *x* times *x*.

Multiply the for-loop's updated value by 2, then use this to divide the *previous* value. Then divide this output by 2 times the updated value minus 1.

The output will be added to *y* as it goes through the loop.

It will return the *y*.
Our test harness will go through the range of $(0,2\pi)$ with steps of 0.05π as it runs through the data.

For arcsin:
Create a function that will take in a value, *x*.
Have *previous*, a variable, start at 1.
Have *y* start at *x* + 1 so it can pass through the condition.
Create a do-while loop that will end once the conditional of the absolute value of our *y* minus *previous* is less than or equal to Epsilon.

The *y* value will contain the updated *previous* value as it goes through the loop.

Find the output of sin(*previous*) - *x*.

Find the output of the previous logic and use that to divide by cos(*previous*).

It will return the *previous* value.
Our test harness will go through the range [−1,1) with steps of 0.05 as it runs through the data.

For arccos:
Create a function that will take in a value, x.
Using arcsin from our previous implementation, we subtract arcsin(x) from π divided by 2 to get arccos(x).
Our test harness will go through the range [−1,1) with steps of 0.05 as it runs through the data.

For arctan:
Create a function that will take in a value, x.
Find the square root of x^2 +1.
Then use that to find the arccos of 1 divided by the output of the square root of x^2 +1.
Make the range from 1 to 10 and with steps of 0.05.

For log:
Set *previous* to begin at 1.
In a do-while loop, subtract x from exponential function to the power of *previous*.
Divide this output by the exponential function to the power of *previous*.
Add this output with *previous*.
Make the variable, *previous*, the output in the end.
The loop will terminate once the exponential function to the power of *previous* minus the x value is greater than or equal to Epsilon.
Return *previous*.

For `mathlib-test.c`:
This file will contain the `main()` program and will be used to test our math library.
Compiled program must be called `mathlib-test`.
Using `getopt()`, we are to program command-line options, such as being able to run all tests when pressing `-a` for example.
The other types of tests are `-s` to run sin tests, `-c` to run cos tests, `-S` to run arcsin tests, `-C` to run arccos tests, `-T` to run arctan tests, and `-l` to run log tests.
Set variables for each function needed, except `-a`, and set them to all false. This requires us to `include <stdbool.h>`.
Using the assignment's example of `getopt()`, replace commands and place our own logic.
When an argument is passed, the case will first check if the variable corresponding to that argument is `false`. If it is, it will go through. The variable will be set to `true` in the case's logic.
By setting the variable to `true` in the logic, this is so no same case goes twice if we were to call a combination of `-a` and any other argument.
Case a will hold all of the cases and be the same as their logic.
Inside all of the case's logic is first printing out the columns and dashes.
There is a for-loop where we create the initialization, conditionals, and step according to the above correlating functions.

The logic inside the for-loops is using our function, then using the math library's function, and finding the difference between those two.
Then we will use the `printf` statement given to us in the assignment to print out the values.