Question 2

A fellow developer has asked you to review the following code. Please provide your feedback:

```
1.  char *GetErrorString( int x )
2.  {
3.  char errorString[20];
4.  switch ( x )
5.  {
6.  case 0:
7.  errorString = "Success -- No error.";
8.  break;
9.  case 2:
10. errorString = "Overflow!";
11. break;
12. }
13. errorString[19] = 0;
14. return errorString;
15. }
16. void main( void )
17. {
18. int err = DoSomething();
19. if ( err )
20. {
21. printf( "%s\n", GetErrorString( err ) );
22. }
23. }
```

Feedback:
Hey,

It looks like you're trying to write some error handling for the function DoSomething() in C. C isn't my native programming language but I still might be able to help you.

The first bit of advice would be to try to follow style guides for C. Does Apple have a linter available for C? Please update the code to follow standard conventions for indentations and braces. It will help make your code easier to read, which in turn makes it easier to find bugs.

Related, please provide documentation comments in your code so that others (and your future-self) know what the code does.

Similarly, writing unit tests to provide adequate code coverage is very helpful. This can help prevent future changes from introducing bugs and makes it explicit how you expect it to run.

It looks like this code doesn't actually compile. Have you looked at the error codes you get from compiling at it? Maybe we should look at it together. Here are some changes I can see off the top of my head (not including style changes):

Line 1 - Refactor x to a more descriptive variable name such as "errorCode" to make it more clear what the code is doing.

Line 3 - GetErrorString returns errorString, a local array only available in the scope of GetErrorString. When it returns, the memory will be freed and trying to print it will cause a segmentation fault for accessing memory it no longer has access to. To fix this use "static char errorString[20];"

Line 7, 10 - This kind of assignment doesn't work in C. Try using strcpy such as "strcpy(errorString, "Success -- No error.");" and "strcpy(errorString, "Overflow!");"

Line 12 - It is good practice to handle the default case in the switch statement even if you intend to just pass using a semicolon or empty brackets just to make it clear what is meant to happen. Though, in this case, I believe it should return a generic errorString.

Line 13 - This line is unnecessary now that we have properly declared errorString. Furthurmore, as it stands, it sets the "." in the case 0 string to null.

Line 16 - main should return the error code so this should be "int main(void)"

Line 19 - This means that GetErrorString(0) never gets called. Is this the only place GetErrorString is used? If so then maybe case 0 is unnecessary. Similarly, since there is no implementation of case 1, nothing gets printed for err == 1, the same as if err == 0.

Line 23 - Since we changed main to return an int, we should "return err" here.

It looks like there's a lot of work to be done on this. Perhaps we could find a resource for us to brush up on our C together.

Good luck,
~Marvin

Question 3

You are asked to filter a noisy but slowly-shifting sensor signal with a low-pass, finite impulse response (FIR) filter. Derive the mean delay and expected SNR boost (noise standard deviation of output compared to input) for two filter variations:
a) N filter taps uniformly weighted: y[n] = (x[n] +...+x[n-N+1])/N
b) M filter taps harmonically weighted: z[n] = (M*x[n]+(M-1)*x[n-1] ... +1*x[n-M+1]) / (M*(M+1)/2)

Assumptions:

1) Unstructured white noise with mean, μ = 0 and standard deviation = σ
2) Slowly shifting means the timescale of the signal >> the sampling timescale of the sensor so we can assume the signal is the DC offset of the sensor

We decompose x[n] = x_s[n] + x_n[n] for signal and noise components of x.
By assumption 1, x_n[n], x_n[n-1]... are independent.
By assumption 2, x_s = x_s[n] = x_s[n-1] = ...

For mean delay:

a)

Mean delay = (0τ + 1τ + ... + (N+1)τ) / N
Since 1 + .. + N + (N+1) = (N + 1) * (N + 2) / 2
Mean delay = τ * (N + 1) (N + 2) / 2N
Where τ is the sampling period
In the limit as N goes to infinity
Mean delay = τN/2

b)

Mean delay = (M * 0τ + (M-1)* 1τ + (M-2) * 2τ +...+ 2 * (M+1-2)τ + 1 * (M+1)τ) / (M*(M+1)/2)

$$= 2/(M * (M + 1)) \sum_{i=0}^{M+1} (M - i) * i\tau = 2\tau/(M * (M + 1)) \sum_{i=0}^{M+1} Mi - i^2 = 2\tau/(M * (M + 1))(M \sum_{i=0}^{M+1} i - \sum_{i=0}^{M+1} i^2)$$

$$= 2\tau/(M * (M + 1))(M * (M + 1)(M + 2)/2 - (M + 1)(M + 1 + 1)(2(M + 1) + 1) / 6)$$

Mean delay $= \tau/(M * (M + 1))(M * (M + 1)(M + 2) - (M + 1)(M + 2)(2M + 3) / 3)$
Where τ is the sampling period
In the limit as M goes to infinity
Mean delay $= \tau/(M^2)(M^3 - 2/3M^3) = \tau M/3$

For SNR:

a)

y[n] = (x[n] +...+x[n-N+1])/N
y[n] = (x_s[n] + x_n[n] +...+x_s[n-N+1] + x_n[n-N+1])/N
y[n] = N/N*x_s + (x_n[n] +... + x_n[n-N+1])/N
y[n] = x_s + (x_n[n] +... + x_n[n-N+1])/N

E(y[n]) = E(x_s) + (μ + ... + μ) / N
E(y[n]) = E(x_s) + (0 + ... + 0) / N
E(y[n]) = x_s

variance(y[n]) = variance(x_s) + variance((x_n[n] +... + x_n[n-N+1]) / N)
variance(y[n]) = 0 + (σ^2 + ... + σ^2) / N^2
variance(y[n]) = N * σ^2 / N^2
variance(y[n]) = σ^2 / N

Therefore the standard deviation of y[n] is σ / sqrt(N) and the SNR is sqrt(N)

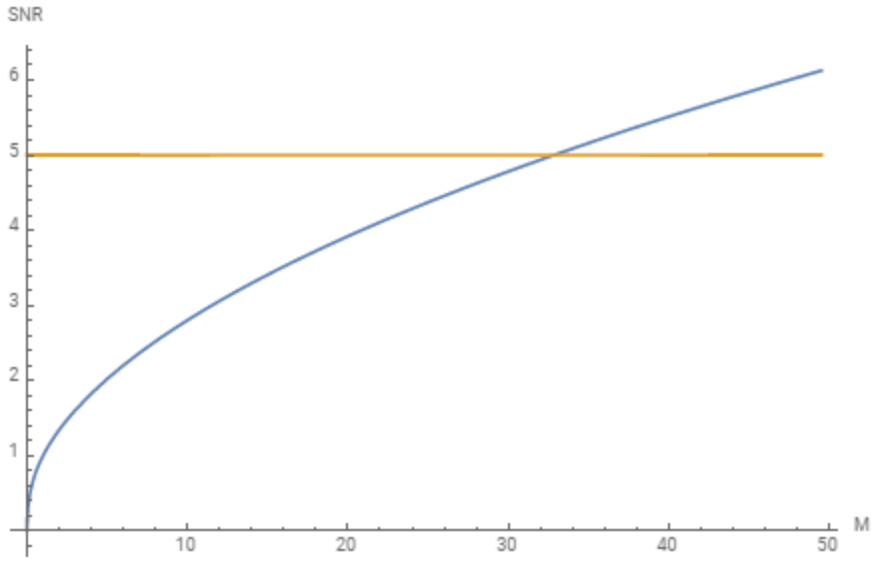
b)

z[n] = (M*x[n]+(M-1)*x[n-1] ... +1*x[n-M+1]) / (M*(M+1)/2)
z[n] = (M*x_s+(M-1)*x_s ... +1*x_s) / (M*(M+1)/2) + (M*x_n[n]+(M-1)*x_n[n-1] ... +1*x_n[n-M+1]) / (M*(M+1)/2)
z[n] = (M*x_s+(M-1)*x_s ... +1*x_s) / (M*(M+1)/2) + (M*x_n[n]+(M-1)*x_n[n-1] ... +1*x_n[n-M+1]) / (M*(M+1)/2)
Since M+(M-1) ... +1 = M*(M+1)/2
z[n] = x_s + (M*x_n[n]+(M-1)*x_n[n-1] ... +1*x_n[n-M+1]) / (M*(M+1)/2)

As above
E(z[n]) = x_s

variance(z[n]) = variance(M*x_n[n]+(M-1)*x_n[n-1] ... +1*x_n[n-M+1]) / (M*(M+1)/2) ^ 2
variance(z[n]) = (M^2 * σ^2 + (M-1)^2 * σ^2 + ... + 1^2 * σ^2) / (M*(M+1)/2) ^ 2
Since M^2 + (M-1)^2 .. + 1^2 = M(M+1)(2M+1) / 6
variance(z[n]) = σ^2 * M(M+1)(2M+1) / (6 * (M*(M+1)/2) ^ 2)
variance(z[n]) = σ^2 * (2M+1) / (3/2 * M * (M+1))

Therefore the standard deviation of y[n] is
σ * sqrt((2M+1) / (3/2 * M * (M+1)))
And the SNR is
sqrt((3/2 * M * (M+1)) / (2M+1))



What depths (N and M) for each setup are needed to boost the SNR by a factor of 5?

a)

sqrt(N) = 5
N = 25

b)

sqrt((24 * M * (M+1)) / (2M+1)) = 5
M =32.8 therefore you need M=33 to achieve a SNR of 5

Which setup a) or b) has the lowest mean delay at the required depth?

Assuming asymptotic behavior:

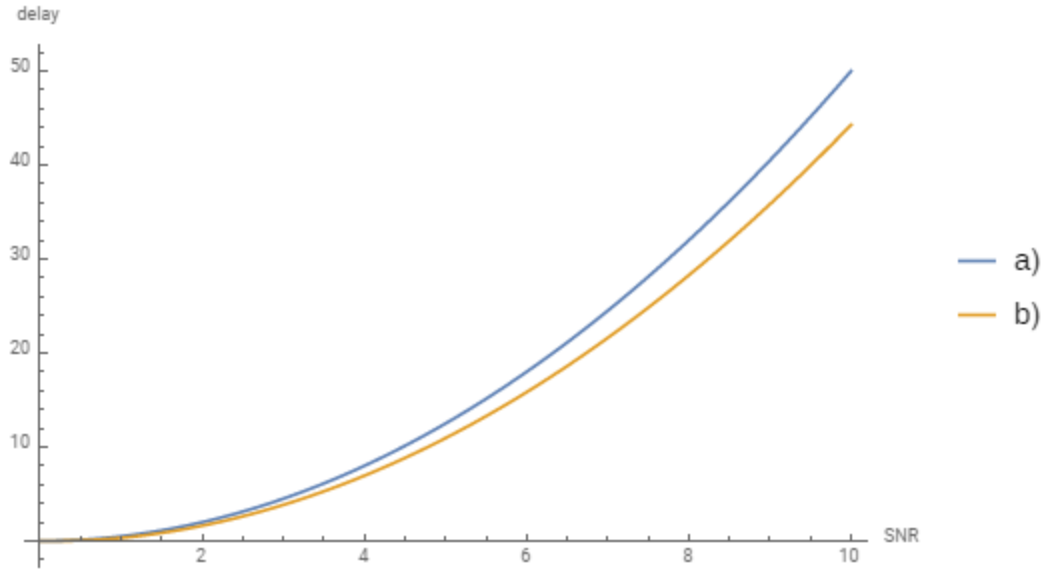
τ25/2 > τ33/3
12.5 > 11
Therefore b) has the lowest mean delay at the required depth to achieve a SNR of 5.

If instead we use the explicit form of the delay we have
τ * (N + 1) (N + 2) / 2N >
$\tau/\ (M\ *\ (M\ +\ 1))(M\ *\ \ (M\ +\ 1)(M\ +\ 2)\ -\ (M\ +\ 1)(M\ +\ 2)(2M\ +\ 3)\ /\ 3)$
(25 + 1) (25 + 2) / (2*25) >
$1/\ (33\ *\ (33\ +\ 1))(33\ *\ \ (33\ +\ 1)(33\ +\ 2)\ -\ (33\ +\ 1)(33\ +\ 2)(2\ *\ 33\ +\ 3)\ /\ 3)$
14.04 > 10.61
Since b) still has a lower mean delay and in fact the difference is larger than the asymptotic behavior, we can use the asymptotic forms for the rest of this analysis.

Which setup would you recommend gives the best tradeoff between SNR and delay?

It may help to plot your results.

So, solving for M and N as a function of SNR and then calculating the delay as a function of SNR using asymptotic behavior we get:



Therefore for the same SNR b) always provides a better delay.