Communication System Lab
Assignment 2

Name- Pushkal Mishra
Roll- EE20BTECH11042

Designing Digital Filters-

A digital filter is a system that performs mathematical operations on sampled, discrete-time signals to reduce or enhance certain frequency components of the input signal. In this assignment, we are given ideal low pass and band pass filters which have to be sampled at a specific frequency (design parameter). Also since these ideal filters extend to infinity in both time directions they are not implementable as they will require infinite energy, so we use some specific window functions which select out some time instances of the impulse response of the filter and independently scale them. In this assignment we use the hamming window.

Selecting sampling frequency-

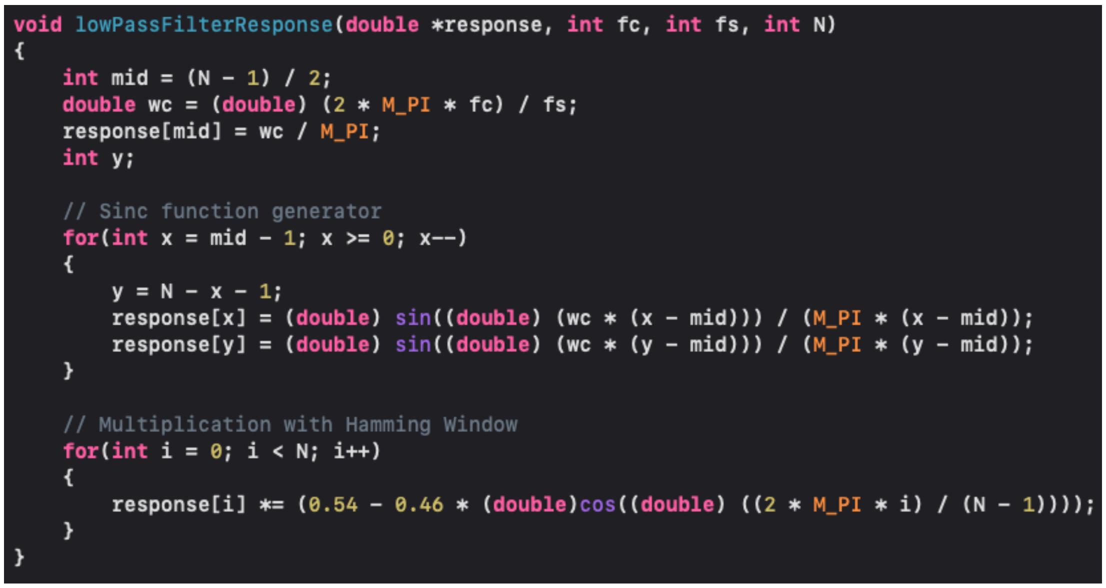
Given digital frequency ω_C and analog cutoff frequency f_C we can find out the required sampling frequency using the equation below-

$$\omega_C = \frac{2\pi f_C}{f_S}$$

1. **Half Band Low Pass Filter-**

   Here we have ω_C as $\pi/2$ and f_C as 400 Hz, substituting this into the above formula, we get f_S as 1600 Hz.

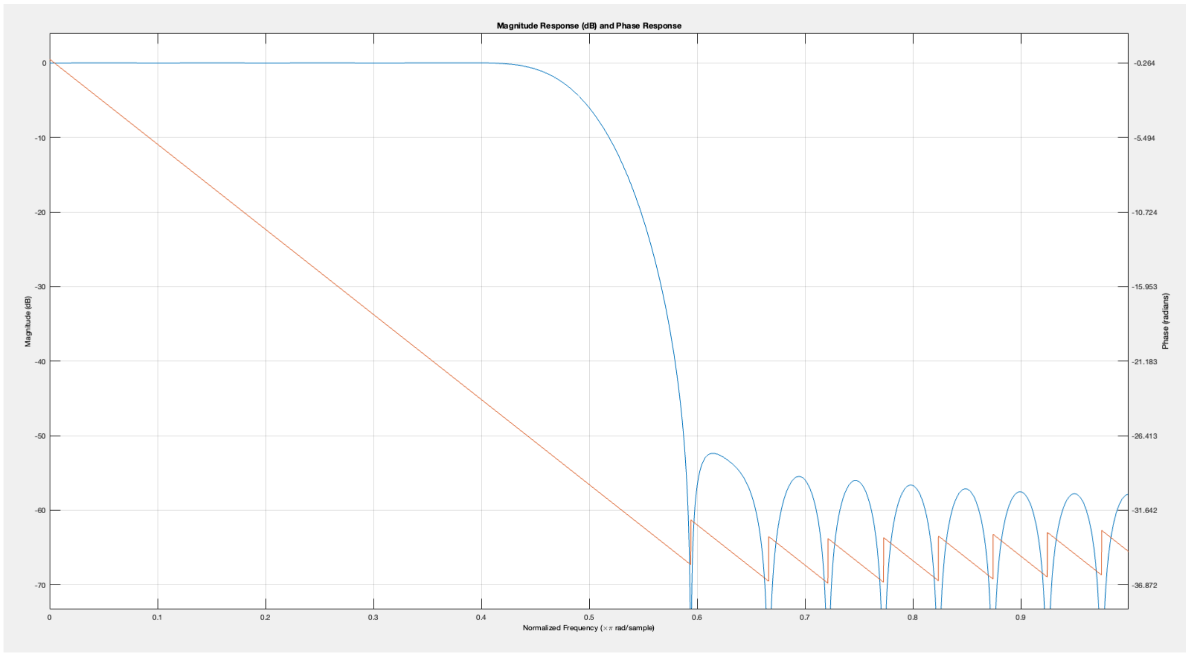
Implementation in C



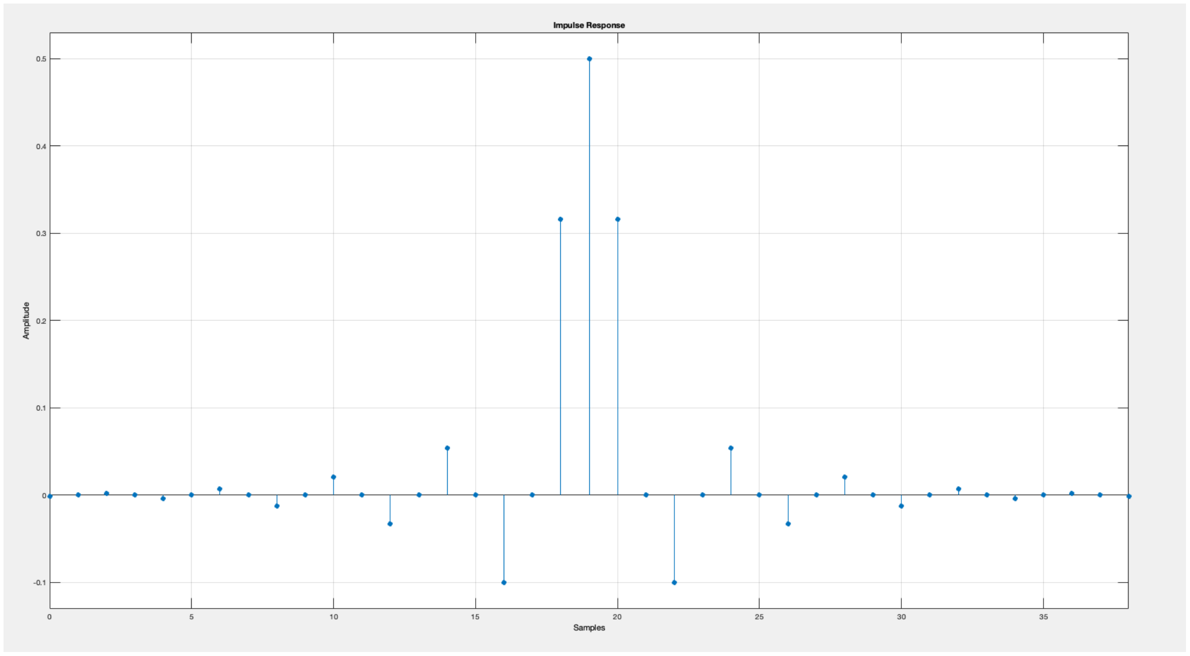
```c
void lowPassFilterResponse(double *response, int fc, int fs, int N)
{
    int mid = (N - 1) / 2;
    double wc = (double) (2 * M_PI * fc) / fs;
    response[mid] = wc / M_PI;
    int y;

    // Sinc function generator
    for(int x = mid - 1; x >= 0; x--)
    {
        y = N - x - 1;
        response[x] = (double) sin((double) (wc * (x - mid))) / (M_PI * (x - mid));
        response[y] = (double) sin((double) (wc * (y - mid))) / (M_PI * (y - mid));
    }

    // Multiplication with Hamming Window
    for(int i = 0; i < N; i++)
    {
        response[i] *= (0.54 - 0.46 * (double)cos((double) ((2 * M_PI * i) / (N - 1))));
    }
}
```

In the above implementation, first we compute the sinc values at the required times instead of computing it for infinite time, then we multiply the corresponding time domain elements with the hamming window coefficients corresponding to that time. The pointer which stores the data is given to the function as argument and allocated memory in the "main.c" file.

Bode Plot for Half Band LPF



Impulse Response of Half Band LPF

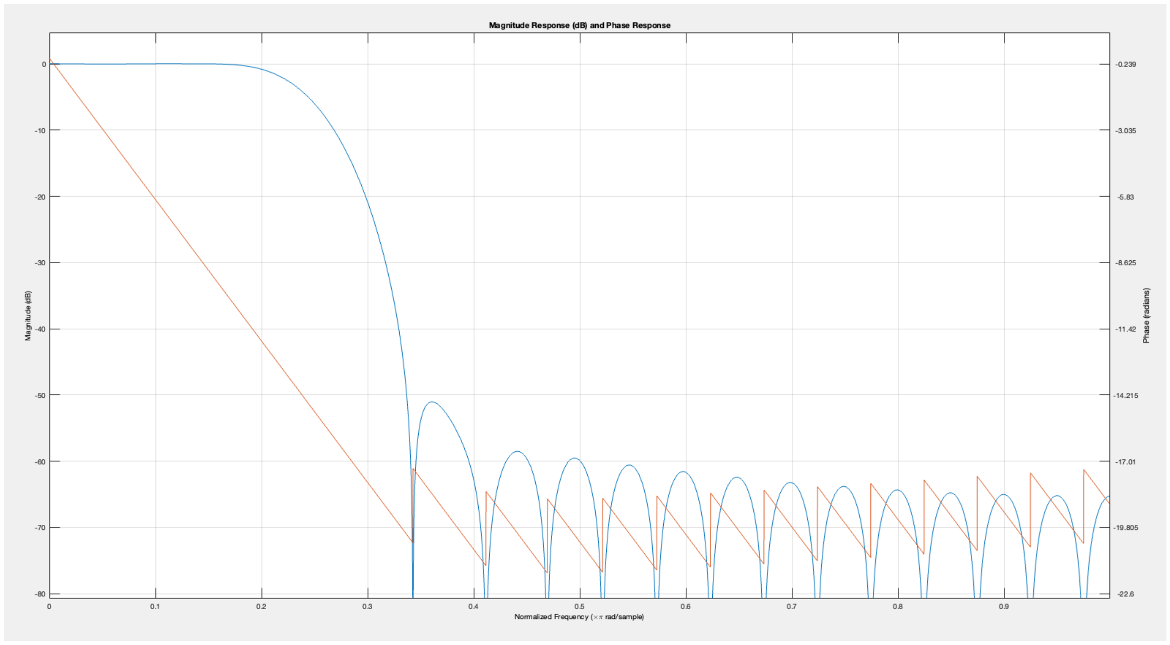


2. **Low Pass Filter-**

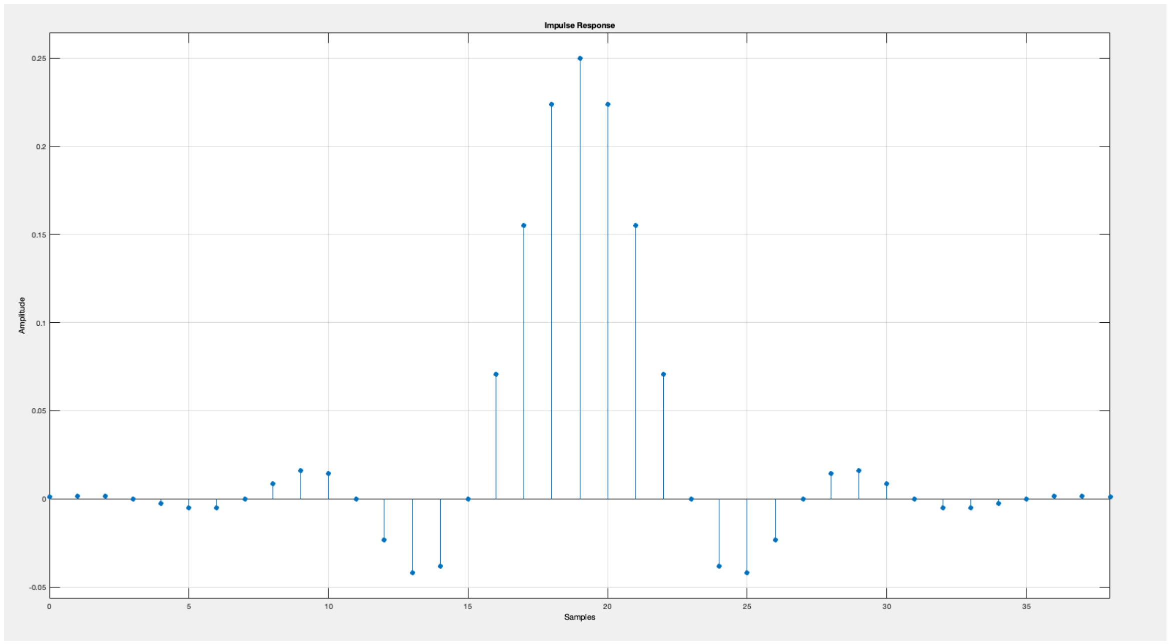
Here we have ω_C as $\pi/4$ and f_C as 400 Hz, substituting this into the above formula, we get f_S as 3200 Hz.

Implementation is the same as above, only a different instance of the function was called in the "main.c" file.

Bode Plot for LPF



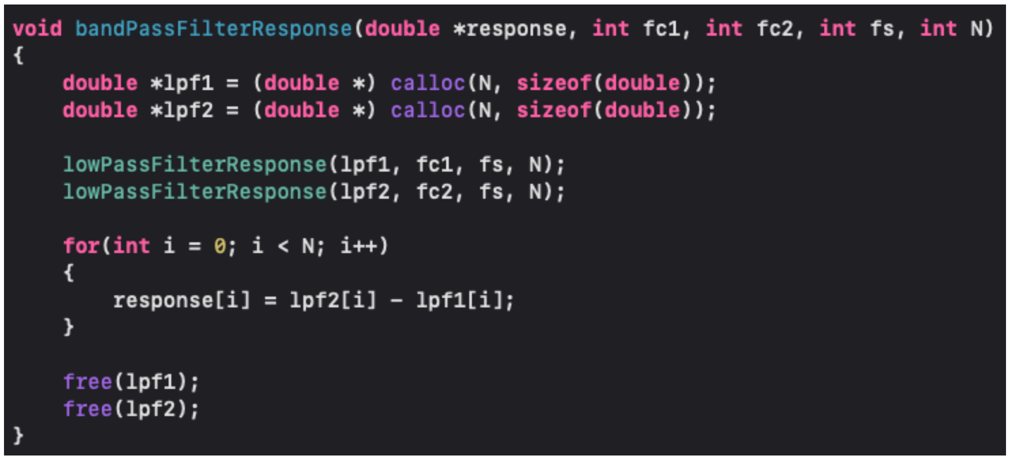
Impulse Response of LPF



3. Band Pass Filter-

A bandpass filter only allows a certain range of frequencies to pass through the system, which means that it will have a maximum and minimum frequency pass component. We can essentially use two different low pass filters, one with higher and other with lower cutoff frequency and subtract their corresponding outputs which will cancel out the lower frequency component in the filter with a higher cutoff frequency. This leaves us with two rectangular frequency components on either side of the y-axis.

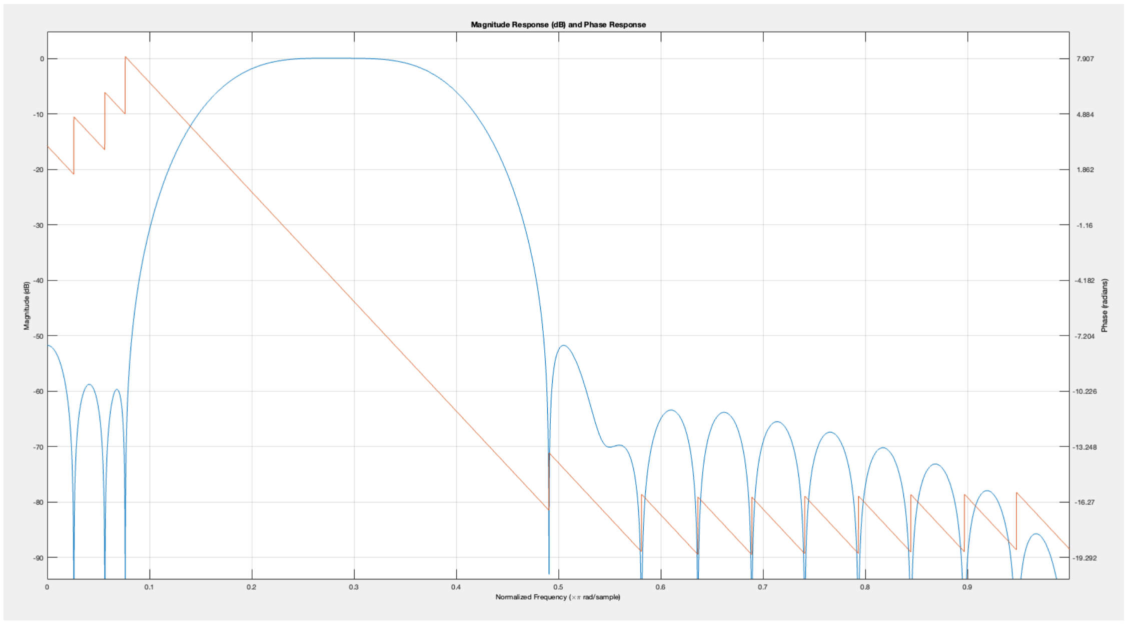
Implementation in C



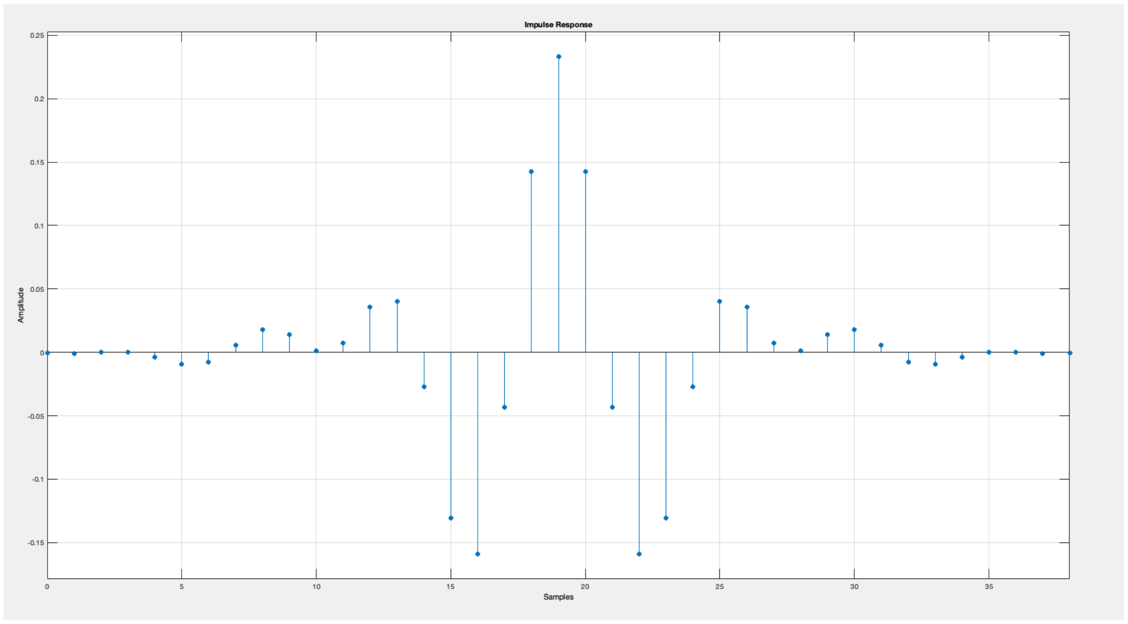
```c
void bandPassFilterResponse(double *response, int fc1, int fc2, int fs, int N)
{
    double *lpf1 = (double *) calloc(N, sizeof(double));
    double *lpf2 = (double *) calloc(N, sizeof(double));

    lowPassFilterResponse(lpf1, fc1, fs, N);
    lowPassFilterResponse(lpf2, fc2, fs, N);

    for(int i = 0; i < N; i++)
    {
        response[i] = lpf2[i] - lpf1[i];
    }

    free(lpf1);
    free(lpf2);
}
```

In the above implementation, we just call the two LPF functions and subtract their impulse responses which in essence gives out a band pass filter. Note that the hamming window is not required here since that was already taken care of in the LPF function implementation.

Bode plot of Band Pass Filter

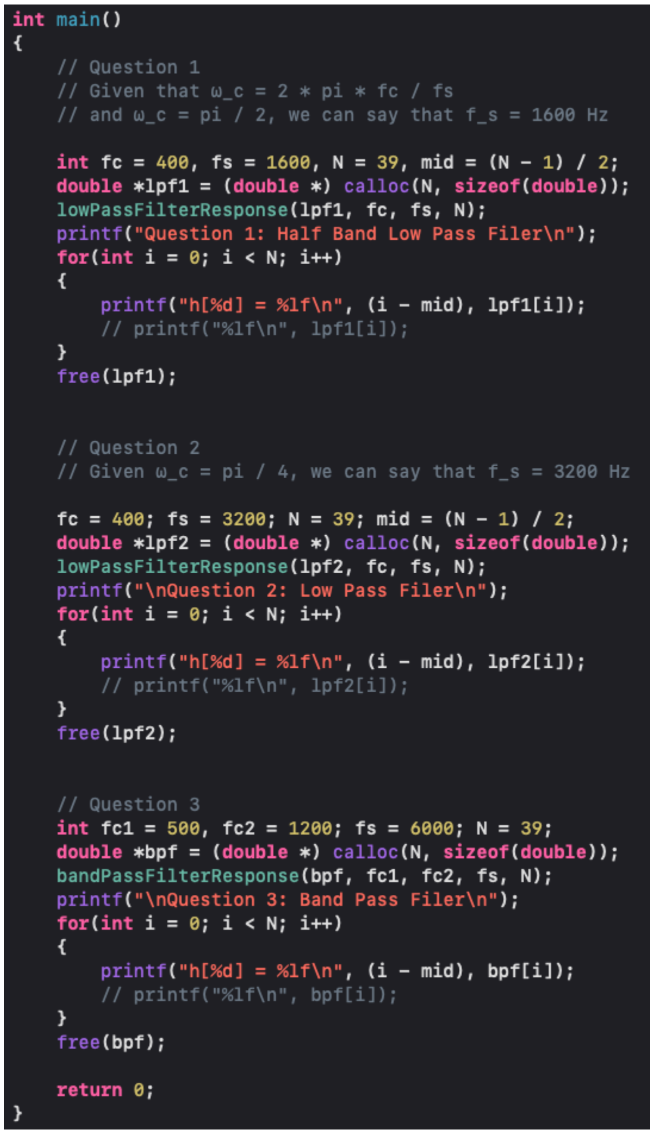


Impulse Response of Band Pass Filter



As seen from the bode plot, the cutoff frequencies are around 500 Hz and 1200 Hz (as required by the question).

Implementation in main.c $\quad \Rightarrow$



```c
int main()
{
    // Question 1
    // Given that ω_c = 2 * pi * fc / fs
    // and ω_c = pi / 2, we can say that f_s = 1600 Hz

    int fc = 400, fs = 1600, N = 39, mid = (N - 1) / 2;
    double *lpf1 = (double *) calloc(N, sizeof(double));
    lowPassFilterResponse(lpf1, fc, fs, N);
    printf("Question 1: Half Band Low Pass Filer\n");
    for(int i = 0; i < N; i++)
    {
        printf("h[%d] = %lf\n", (i - mid), lpf1[i]);
        // printf("%lf\n", lpf1[i]);
    }
    free(lpf1);


    // Question 2
    // Given ω_c = pi / 4, we can say that f_s = 3200 Hz

    fc = 400; fs = 3200; N = 39; mid = (N - 1) / 2;
    double *lpf2 = (double *) calloc(N, sizeof(double));
    lowPassFilterResponse(lpf2, fc, fs, N);
    printf("\nQuestion 2: Low Pass Filer\n");
    for(int i = 0; i < N; i++)
    {
        printf("h[%d] = %lf\n", (i - mid), lpf2[i]);
        // printf("%lf\n", lpf2[i]);
    }
    free(lpf2);


    // Question 3
    int fc1 = 500, fc2 = 1200; fs = 6000; N = 39;
    double *bpf = (double *) calloc(N, sizeof(double));
    bandPassFilterResponse(bpf, fc1, fc2, fs, N);
    printf("\nQuestion 3: Band Pass Filer\n");
    for(int i = 0; i < N; i++)
    {
        printf("h[%d] = %lf\n", (i - mid), bpf[i]);
        // printf("%lf\n", bpf[i]);
    }
    free(bpf);

    return 0;
}
```