```c
#include <stdio.h>

int main() {
    float num = 0.00063;
    float result = num * num;
    float reciprocal = 1 / result;
    unsigned int *ptr = (unsigned int *)&reciprocal;

    for (int i = 31; i >= 0; i--) {
        printf("%d", (*ptr >> i) & 1);
    }
    printf("\n");

    return 0;
}
```

Explanation

#include <stdio.h>
This line includes a standard library in C that allows us to perform input and output operations.

int main() {
This line marks the beginning of the main function, which is the entry point of our program.

float num = 0.00063;
Here, we're declaring a variable called num and assigning it the value 0.00063. This represents the number we want to calculate the square of.

float result = num * num;
We're calculating the square of num by multiplying it by itself, and storing the result in a variable called result.

float reciprocal = 1 / result;
Next, we're finding the reciprocal (inverse) of the squared value. This effectively gives us 0.00063^−2

0.00063^−2 , the reciprocal of 0.00063^-2
1/0.00063^2 , which is what you requested.

unsigned int *ptr = (unsigned int *)&reciprocal;
Here, we're creating a pointer called ptr that points to the memory address of the reciprocal variable. We're casting it to an unsigned integer pointer so that we can interpret the bits of the floating-point number as individual bits of an integer.

for (int i = 31; i >= 0; i--) {
This is a loop that iterates through each bit of the 32-bit representation of the floating-point number.

printf("%d", (*ptr >> i) & 1);
Within the loop, we're printing each bit of the floating-point number. We shift the bits of the number to the right by i positions (*ptr >> i), then use a bitwise AND operation (& 1) to isolate the rightmost bit. We print this bit to the console.

printf("\n");
Finally, we print a newline character to move to the next line in the output.

return 0;
This line indicates that the program has executed successfully and returns a status code of 0 to the operating system, indicating successful completion of the program.

Noise

```
[OriginalAudio, Fs] = wavread("C:\Users\Adit\Desktop\file_example_WAV_1MG.wav");
playsnd(OriginalAudio, Fs)
Fmin = 20;
Fmax = 20e3;
analyze(OriginalAudio, Fmin, Fmax, Fs, length(OriginalAudio))
Fmax = 2e3;
analyze(OriginalAudio, Fmin, Fmax, Fs, length(OriginalAudio))
Fmax = 20e3;
OriginalAudio_NoiseOnly = OriginalAudio(1*Fs : 4*Fs);
playsnd(OriginalAudio_NoiseOnly, Fs)     // to confirm that the excerpt contains only background noise
analyze(OriginalAudio_NoiseOnly, Fmin, Fmax, Fs, length(OriginalAudio_NoiseOnly))
Fmax = 1e3;
analyze(OriginalAudio_NoiseOnly, Fmin, Fmax, Fs, length(OriginalAudio_NoiseOnly))
FIR_coefficients = wfir();
FilteredAudio = convol(FIR_coefficients, OriginalAudio);
analyze(FilteredAudio, Fmin, Fmax, Fs, length(FilteredAudio))
```

Loading the Audio File:
[OriginalAudio, Fs] = wavread("C:\Users\Adit\Desktop\file_example_WAV_1MG.wav");: This line loads a WAV audio file named "file_example_WAV_1MG.wav" from the specified path. It stores the audio data in the variable OriginalAudio and the sampling frequency in Fs.

Playing the Original Audio:
playsnd(OriginalAudio, Fs): This line plays the original audio stored in OriginalAudio at the sampling frequency Fs.

Analyzing the Original Audio:
Fmin = 20; Fmax = 20e3; analyze(OriginalAudio, Fmin, Fmax, Fs, length(OriginalAudio)): This line

analyzes the frequency content of the original audio using the analyze function. It specifies the minimum frequency Fmin, maximum frequency Fmax, sampling frequency Fs, and the length of the audio.

Changing Frequency Range and Analyzing Again:
Fmax = 2e3; analyze(OriginalAudio, Fmin, Fmax, Fs, length(OriginalAudio)): This line changes the maximum frequency to 2000 Hz and analyzes the original audio again.

Extracting a Segment for Noise Analysis:
OriginalAudio_NoiseOnly = OriginalAudio(1*Fs : 4*Fs);: This line extracts a 3-second segment of the original audio starting from 1 second and ending at 4 seconds. This segment is intended to contain only background noise.

Playing the Extracted Noise Segment:
playsnd(OriginalAudio_NoiseOnly, Fs): This line plays the extracted noise segment to confirm that it contains only background noise.

Analyzing the Extracted Noise Segment:
analyze(OriginalAudio_NoiseOnly, Fmin, Fmax, Fs, length(OriginalAudio_NoiseOnly)): This line analyzes the frequency content of the extracted noise segment.

Changing Frequency Range for Noise Analysis:
Fmax = 1e3; analyze(OriginalAudio_NoiseOnly, Fmin, Fmax, Fs, length(OriginalAudio_NoiseOnly)): This line changes the maximum frequency to 1000 Hz and analyzes the extracted noise segment again.

Filtering the Original Audio:
FIR_coefficients = wfir(); FilteredAudio = convol(FIR_coefficients, OriginalAudio);: This line designs a Finite Impulse Response (FIR) filter and applies it to the original audio using convolution to produce FilteredAudio.

Analyzing the Filtered Audio:
analyze(FilteredAudio, Fmin, Fmax, Fs, length(FilteredAudio)): This line analyzes the frequency content of the filtered audio.