Problem 7.1 We are tasked with finding all 3 solutions for the function $f(x) = x^3 - 5x^2 - 12x + 19 = 0$ via newtons method, for this problem I figured the best way forward was to use a modified and redacted version of my C code with improvements made thanks to suggestions by Ed Bueler.

```c
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdbool.h>

//------------------------INTERFACE------------------------
#define MIN -10.0
#define MAX 10.0
#define STEP 0.05
#define EPSILON 1e-6
#define STORAGE 100

double f(double x)
{
    return pow(x, 3) - 5 * pow(x, 2) - 12 * x + 19;
}

//---------------------------------------------------------

// Derivative using central finite differences
double deriv(double x, double h)
{
    return (f(x+h) - f(x-h)) / (2.0*h);
}

bool signCheck(double prev, double curr)
{
    // Check if the sign has changed, considering EPSILON for near-zero values
    if ((prev > EPSILON && curr < -EPSILON) || (prev < -EPSILON && curr > EPSILON))
     {
        return true;
    }
    return false;
}

// Newton's Method for finding zeros
double duke_newton(double x0, double h, double tolerance, int max_iterations)
{
    double x = x0;
    double x_new;

    for (int i = 0; i < max_iterations; i++) {
        x_new = x - f(x) / deriv(x, h);

        if (fabs(x_new - x) < tolerance) {
            return x_new;
        }
        x = x_new;
    }
    printf("Did not converge within %d iterations for given range.\n", max_iterations);
    return x_new;
}

```

```c
int main()
{
    double h = EPSILON; // For notational convenience
    double x;
    int zeroCount = 0;
    int i = 0;
    double zeros[STORAGE];
    zeros[i] = MIN; // Start with MIN to capture potential zero at the boundary
    ++i;

    // Find zeros by checking where f(x) changes sign
    double prev_val = f(MIN);
    for (x = MIN + STEP; x <= MAX; x += STEP) {
        double curr_val = f(x);
        if (signCheck(prev_val, curr_val)) {
            zeros[i] = x - STEP; // Adjust back to where the sign change was detected
            i++;
            zeroCount++;
        }
        prev_val = curr_val;
    }
    zeros[i] = MAX; // End with MAX to capture potential zero at the boundary

    // Refine the zero estimates using Newton's method
    for(i = 1; i < zeroCount + 1; i++) { // Start from 1 to skip the MIN boundary check
        double guess = zeros[i];
        double out = duke_newton(guess, h, EPSILON, 6);
        printf("Zero at x: %.5f, f(x): %.5f\n", out, f(out));
    }

    printf("Number of zeros found: %d\n", zeroCount);
    return 0;
}
```

Our output ends up being:

```
Zero at x: -2.56524, f(x): -0.00000
Zero at x: 1.15555, f(x): 0.00000
Zero at x: 6.40970, f(x): -0.00000
Number of zeros found: 3
```

Note that the reason f(x) values end up as positive or negative zero, is because these are approximations and are not actually true zero, rather they are values which are sufficiently close to zero and are simply cut off within the first 5 digits. Source code is available in https://github.com/Funkematics/Optimizations/tree/m as "NewtProb.c", you may compile using by running "make".