traceMatrix.c

```c
// Write a function that computes the trace of a square matrix.
#include <stdio.h>

int main(){
    int n, i, j;
    printf("Enter the number of rows/columns: ");
    scanf("%d", &n);

    int matrix[n][n];
    printf("Enter the elements of the matrix:\n");
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }

    int trace = 0;
    for(i = 0; i < n; i++){
        trace += matrix[i][i];
    }

    printf("The trace of the matrix is: %d\n", trace);
    return 0;
}

/**
 * Example Input/Output:
 * Enter the number of rows/columns: 3
 * Enter the elements of the matrix:
 * Element [0][0]: 1
 * Element [0][1]: 2
 * Element [0][2]: 3
 * Element [1][0]: 4
 * Element [1][1]: 5
 * Element [1][2]: 6
 * Element [2][0]: 7
 * Element [2][1]: 8
 * Element [2][2]: 9
 * The trace of the matrix is: 15
 */

```

sinx.c

```c
/**
 * write a C program that computes the sine of an angle in radians using the Taylor series * * expansion.
 *
 * sin(x) = x - x^3/3! + x^5/5! - x^7/7! + ...
 *
 */

#include <stdio.h>
#include <math.h>

int main() {
    float xdeg, sine, term;
    int nStep, sign;

    // input number of terms
    printf("Enter number of terms for Taylor series approximation: ");
    scanf("%d", &nStep);
    // Input angle in degrees
    printf("Enter angle in degrees: ");
    scanf("%f", &xdeg);

    // convert degrees to radians
    float x = (xdeg * 3.14159) / 180.0;

    for(int i = 0; i < nStep; i++) {
        // Calculate each term of the Taylor series
        sign = (i % 2 == 0) ? 1 : -1; // alternate signs
        int exponent = 2 * i + 1;

        // Calculate factorial
        int factorial = 1;
        for(int j = 1; j <= exponent; j++) {
            factorial = factorial * j;
        }

        term = sign * (pow(x, exponent) / factorial);
        sine += term;

    }

    printf("Sine of %.2f degrees is approximately: %f\n", xdeg, sine);

    return 0;
}
```

quadratic.c

```c
// Write a C program to solve a quadratic equation of the form ax^2 + bx + c = 0 with real coefficients.

#include <stdio.h>
#include <math.h>

int main() {
    float a, b, c;
    float discriminant, root1, root2;

    // Input coefficients a, b and c
    printf("Enter coefficients a, b and c: ");
    scanf("%f %f %f", &a, &b, &c);

    // Calculate the discriminant
    discriminant = b * b - 4 * a * c;

    // Check the nature of the roots based on the discriminant
    if (discriminant > 0) {
        // Two distinct real roots
        root1 = (-b + sqrt(discriminant)) / (2 * a);
        root2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("Roots are real and different.\n");
        printf("Root 1: %.2f\n", root1);
        printf("Root 2: %.2f\n", root2);
    } else if (discriminant == 0) {
        // One real root
        root1 = -b / (2 * a);
        printf("Roots are real and the same.\n");
        printf("Root: %.2f\n", root1);
    } else {
        // No real roots
        printf("No real roots exist.\n");
        // calculate complex roots (optional)
        float realPart = -b / (2 * a);
        float imagPart = sqrt(-discriminant) / (2 * a);
        printf("Root 1: %.2f + %.2fi\n", realPart, imagPart);
        printf("Root 2: %.2f - %.2fi\n", realPart, imagPart);

    }

    return 0;
}


/*******************************
 *
 * Sample Input/Output:
 *
 * * Case 1:
 * Enter coefficients a, b and c: 1 -5 6
 * Roots are real and different.
```

```
 * Root 1: 3.00
 * Root 2: 2.00
 * ******************************
 * * Case 2:
 * Enter coefficients a, b and c: 1 -4 4
 * Roots are real and the same.
 * Root: 2.00
 * ******************************
 * * Case 3:
 * Enter coefficients a, b and c: 1 2 5
 * No real roots exist.
 * Root 1: -1.00 + 2.00i
 * Root 2: -1.00 - 2.00i
 * ******************************
 * // End of the program
 *******************************/
```

gcd2.c

```c
// write a c program to find gcd of two numbers using euclidean algorithm
#include <stdio.h>

int gcd(int a, int b) {
    // Euclidean algorithm to find GCD
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main() {
    int a, b;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    // check for non-negative integers
    if (a < 0 || b < 0) {
        printf("Please enter non-negative integers only.\n");
    } else {
        int result = gcd(a, b);
        printf("The GCD is: %d\n", result);
    }
    return 0;
}
```

gcd.c

```c
// write a c program to find gcd of two numbers using euclidean algorithm

#include <stdio.h>

int main() {
    int a, b;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    // check for non-negative integers
    if (a < 0 || b < 0) {
        printf("Please enter non-negative integers only.\n");
    }else{

    // Euclidean algorithm to find GCD
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }

    printf("The GCD is: %d\n", a);
 }
    return 0;
}
```