SCIENTIFIC CALCULATOR

16 March, 2024

Submitted By :-

-Nosin C. Adhikari(THA081BEI026)
-Oasis Paudel(THA081BEI027)
-Sujana Shrestha(THA081BEI045)
-Sumit Paudel(THA081BEI046)

Submitted To:-

Department of Computer and Electronics
Thapathali Campus

Introduction

A scientific calculator is an advanced computing tool capable of performing complex mathematical operations such as arithmetic, trigonometry, logarithms, exponents, and statistical functions.

Why Implement It in C?

- C is a powerful and efficient programming language.
- It provides fast execution.
- It allows implementation of mathematical functions .

Objective of the Project

- To develop a scientific calculator .
- To enhance problem-solving skills using C programming.
- To demonstrate functions, loops, conditionals, and user input handling.

Main , sum & sub function

```c
int main() {
    int operation;
    char ans;

start:
    printf("|-------------------------------------------------------------------------|\n");
    printf("|                          Calculator                                     |\n");
    printf("|-------------------------------------------------------------------------|\n");
    printf("|  Display:                                                               |\n");
    printf("|---------------|-----------------|---------------|-----------------|\n");
    printf("| [ 1] Add      | [ 2] Sub        | [ 3] Mult     | [ 4] Div        |\n");
    printf("|---------------|-----------------|---------------|-----------------|\n");
    printf("| [ 5] Trig     | [ 6] x^n        | [ 7] log      | [ 8] !          |\n");
    printf("|---------------|-----------------|---------------|-----------------|\n");
    printf("| [ 9] e^x      | [10] BaseConv   | [11] Const    | [12] roots fx   |\n");
    printf("|---------------|-----------------|---------------|-----------------|\n");
    printf("| [13] Perm     | [14] Comb       | [15] EqSolver | [16] S-D        |\n");
    printf("|---------------|-----------------|---------------|-----------------|\n");
    printf("| [17] MatrixOps| [18] Stats      | [19] FracConv | [20] Vector Ops |\n");
    printf("|---------------|-----------------|---------------|-----------------|\n");
    printf("Enter a number for the operation: ");
    scanf("%d", &operation);

    switch (operation) {
        case 1: sum(); break;
        case 2: sub(); break;
        case 3: multiplication(); break;
        case 4: division(); break;
        case 5: trig(); break;
        case 6: power(); break;
        case 7: logi(); break;
        case 8: factorial(); break;
        case 9: epower(); break;
        case 10: baseN(); break;
        case 11: valOfconst(); break;
        case 12: squareroot(); break;
        case 13: permutation(); break;
        case 14: combination(); break;
        case 15: eqnsolv(); break;
        case 16: sdfunction(); break;
        case 17: matrixoperation(); break;
        case 18: stat(); break;
        case 19: fraconversion(); break;
        case 20: vectorOperations(); break;
        default: printf("INVALID INPUT!\n"); break;
    }

    printf("Do you want to use the calculator again? (Y/N): ");
    scanf(" %c", &ans);

    if (ans == 'y' || ans == 'Y') {
        goto start;
    } else if (ans == 'n' || ans == 'N') {
        printf("Thank you for using the calculator\n");
    }

    return 0;
}
```

```c
void sum() {
    int n, i;
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    float *arr = (float *)malloc(n * sizeof(float)); // Dynamic allocation
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter the elements whose sum is to be calculated\n");
    for (i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
    }
    float sum = 0;
    for (i = 0; i < n; i++) {
        sum += arr[i];
    }
    printf("The sum of the elements is = %f\n", sum);
    free(arr); // Free allocated memory
}

void sub() {
    int n, i;
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    float *arr = (float *)malloc(n * sizeof(float)); // Dynamic allocation
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter the elements whose difference is to be calculated\n");
    for (i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
    }
    float result = arr[0];
    for (i = 1; i < n; i++) {
        result -= arr[i];
    }
    printf("The difference of the elements is = %f\n", result);
    free(arr); // Free allocated memory
}
```

Multiplication ,division & power function

```c
void multiplication() {
    int n, i;
    printf("Enter the number of elements \n");
    scanf("%d", &n);
    float *arr = (float *)malloc(n * sizeof(float)); // Dynamic allocation
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter the elements whose multiplication is to be calculated \n");
    for (i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
    }
    float multi = 1;
    for (i = 0; i < n; i++) {
        multi *= arr[i];
    }
    printf("Multiplication is = %f\n", multi);
    free(arr); // Free allocated memory
}

void division() {
    float num1, num2;
    printf("Enter two numbers\n");
    scanf("%f%f", &num1, &num2);
    if (num2 == 0) {
        printf("Error: Division by zero is not allowed.\n");
        return;
    }
    float division = num1 / num2;
    printf("The division is = %.2f\n", division);
}

void power() {
    float number, n;
    printf("Enter the number whose power is to be calculated \n");
    scanf("%f", &number);
    printf("Enter the power of the number to be calculated \n");
    scanf("%f", &n);
    float result = pow(number, n);
    printf("The power %.2f of a number %.2f is = %.2f\n", n, number, result);
```

Power ,log ,factorial & epower function

```c
void squareroot() {
    float number;
    printf("Enter the number whose root is to be calculated\n");
    scanf("%f", &number);
    if (number < 0) {
        printf("Error: Cannot calculate square root of a negative number.\n");
        return;
    }
    float result = sqrt(number);
    printf("The square root of the number %.2f is = %.2f\n", number, result);
}


void logi() {
    double number, result;
    printf("Enter the number\n");
    scanf("%lf", &number);
    if (number <= 0) {
        printf("Value is not possible for non-natural numbers \n");
        return;
    }
    result = log(number);
    printf("The log BASE VALUE e of number %.2lf is = %.2lf\n", number, result);
    result = log10(number);
    printf("The log BASE VALUE 10 of number %.2lf is = %.2lf\n", number, result);
}

void factorial() {
    int fact = 1, n;
    printf("Enter the number whose factorial is to be calculated\n");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    printf("The factorial of the number %d is %d\n", n, fact);
}

void epower() {
    double n;
    printf("Enter the power of e\n");
    scanf("%lf", &n);
    double result = exp(n);
    printf("The power %.2lf of e is %.2lf\n", n, result);
}
```

Statistics function

```c
void stat() {
    int n;
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    float *arr = (float *)malloc(n * sizeof(float)); // Dynamic allocation
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter all the elements \n");
    for (int i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
    }

    float sum = 0, mean, sd, deviation = 0, median, mode;
    int maxcount = 0;

    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    mean = sum / n;
    printf("The mean of %d elements is = %.2f\n", n, mean);

    for (int i = 0; i < n; i++) {
        deviation += pow((mean - arr[i]), 2);
    }
    sd = sqrt(deviation / (n - 1));
    printf("The standard deviation is = %.2f\n", sd);

    // Sorting for median calculation
    float *sortedArr = (float *)malloc(n * sizeof(float));
    for (int i = 0; i < n; i++) {
        sortedArr[i] = arr[i];
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (sortedArr[i] > sortedArr[j]) {
                float temp = sortedArr[i];
                sortedArr[i] = sortedArr[j];
                sortedArr[j] = temp;
            }
        }
    }

    if (n % 2 == 0) {
        median = (sortedArr[n / 2 - 1] + sortedArr[n / 2]) / 2;
    } else {
        median = sortedArr[n / 2];
    }
    printf("The median of the given numbers is = %.2f\n", median);

    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (sortedArr[i] == sortedArr[j]) {
                count++;
            }
        }
        if (count > maxcount) {
            maxcount = count;
            mode = sortedArr[i];
        }
    }
    printf("The mode is = %.2f\n", mode);

    free(arr); // Free allocated memory
    free(sortedArr); // Free allocated memory
}
```

Trigonometric and base conversion function

```c
void trig() {
    int userInput;
    printf("[1] HYPERBOLIC [2] TRIGONOMETRIC [3] INVERSE TRIGONOMETRIC\n");
    scanf("%d", &userInput);
    switch (userInput) {
        case 1: hyperbolicTrig(); break;
        case 2: normalTrig(); break;
        case 3: inverseTrig(); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}

void hyperbolicTrig() {
    int userInput;
    float angle;
    printf("[1] sinh     [2] cosh     [3] tanh\n");
    scanf("%d", &userInput);
    printf("Enter the angle in radians: ");
    scanf("%f", &angle);
    switch (userInput) {
        case 1: printf("sinh(%.2f) = %.2f\n", angle, sinh(angle)); break;
        case 2: printf("cosh(%.2f) = %.2f\n", angle, cosh(angle)); break;
        case 3: printf("tanh(%.2f) = %.2f\n", angle, tanh(angle)); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}

void normalTrig() {
    int userInput;
    float angleDeg, angleRad;
    printf("[1] sin      [2] cos      [3] tan\n");
    scanf("%d", &userInput);
    printf("Enter the angle in degrees: \n");
    scanf("%f", &angleDeg);
    angleRad = angleDeg * M_PI / 180.0;
    switch (userInput) {
        case 1: printf("sin(%.2f) = %.2f\n", angleDeg, sin(angleRad)); break;
        case 2: printf("cos(%.2f) = %.2f\n", angleDeg, cos(angleRad)); break;
        case 3: printf("tan(%.2f) = %.2f\n", angleDeg, tan(angleRad)); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}
```

```c
void inverseTrig() {
    int userInput;
    float parameter;
    printf("[1] sin?¹      [2] cos?¹      [3] tan?¹\n");
    scanf("%d", &userInput);
    printf("Enter the parameter: \n");
    scanf("%f", &parameter);
    switch (userInput) {
        case 1:
            if (parameter > 1 || parameter < -1) {
                printf("INVALID INPUT!\n");
                break;
            }
            printf("sin?¹(%.2f) = %.2f\n", parameter, asin(parameter));
            break;
        case 2:
            if (parameter > 1 || parameter < -1) {
                printf("INVALID INPUT!\n");
                break;
            }
            printf("cos?¹(%.2f) = %.2f\n", parameter, acos(parameter));
            break;
        case 3:
            printf("tan?¹(%.2f) = %.2f\n", parameter, atan(parameter));
            break;
        default: printf("INVALID INPUT!\n"); break;
    }
}

void baseN() {
    int inputType;
    printf("Type of input: [1] BIN  [2] QUI  [3] OCT  [4] DEC  [5] HEX\n");
    scanf("%d", &inputType);
    switch (inputType) {
        case 1: binary(); break;
        case 2: quinary(); break;
        case 3: octal(); break;
        case 4: decimal(); break;
        case 5: {
            char hex[20];
            printf("HEX: ");
            scanf("%s", hex);
            hexadecimal(hex);
            break;
        }
        default: printf("INVALID INPUT!\n"); break;
    }
}
```

Base conversion function

```c
void binary() {
    int bin, binTowanted;
    printf("[1] BIN to QUI  [2] BIN to OCT  [3] BIN to DEC  [4] BIN to HEX\n");
    scanf("%d", &binTowanted);
    printf("BINARY: ");
    scanf("%d", &bin);
    int totalDigit = log10(bin) + 1;
    switch (binTowanted) {
        case 1: anyTodec(bin, 5, 2, totalDigit); break;
        case 2: anyTodec(bin, 8, 2, totalDigit); break;
        case 3: anyTodec(bin, 10, 2, totalDigit); break;
        case 4: anyTodec(bin, 16, 2, totalDigit); break;
        default: printf("Conversion Input Error!\n"); break;
    }
}
void quinary() {
    int qui, quiTowanted;
    printf("[1] QUI to BIN  [2] QUI to OCT  [3] QUI to DEC  [4] QUI to HEX\n");
    scanf("%d", &quiTowanted);
    printf("QUINARY: ");
    scanf("%d", &qui);

    int totalDigit = log(qui) / log(5) + 1; // Calculate total digits in base 5
    switch (quiTowanted) {
        case 1:
            anyTodec(qui, 2, 5, totalDigit); // Convert to binary
            break;
        case 2:
            anyTodec(qui, 8, 5, totalDigit); // Convert to octal
            break;
        case 3:
            anyTodec(qui, 10, 5, totalDigit); // Convert to decimal
            break;
        case 4:
            anyTodec(qui, 16, 5, totalDigit); // Convert to hexadecimal
            break;
        default:
            printf("Conversion Input Error!\n");
            break;
    }
}
```

```c
void octal() {
    int oct, octTowanted;
    printf("[1] OCT to BIN  [2] OCT to QUI  [3] OCT to DEC  [4] OCT to HEX\n");
    scanf("%d", &octTowanted);
    printf("OCTAL: ");
    scanf("%d", &oct);

    int totalDigit = log(oct) / log(8) + 1; // Calculate total digits in base 8
    switch (octTowanted) {
        case 1:
            anyTodec(oct, 2, 8, totalDigit); // Convert to binary
            break;
        case 2:
            anyTodec(oct, 5, 8, totalDigit); // Convert to quinary
            break;
        case 3:
            anyTodec(oct, 10, 8, totalDigit); // Convert to decimal
            break;
        case 4:
            anyTodec(oct, 16, 8, totalDigit); // Convert to hexadecimal
            break;
        default:
            printf("Conversion Input Error!\n");
            break;
    }
}

void decimal() {
    int dec, decTowanted;
    printf("[1] DEC to BIN  [2] DEC to QUI  [3] DEC to OCT  [4] DEC to HEX\n");
    scanf("%d", &decTowanted);
    printf("DECIMAL: ");
    scanf("%d", &dec);

    int totalDigit = log10(dec) + 1; // Calculate total digits in base 10
    switch (decTowanted) {
        case 1:
            anyTodec(dec, 2, 10, totalDigit); // Convert to binary
            break;
        case 2:
            anyTodec(dec, 5, 10, totalDigit); // Convert to quinary
            break;
        case 3:
            anyTodec(dec, 8, 10, totalDigit); // Convert to octal
            break;
```

Base conversion function

```c
void hexadecimal(char hex[]) {
    int hexTowanted, totalDigit = strlen(hex), realValue = 0, decimal = 0, power;

    printf("[1] HEX to BIN  [2] HEX to QUI  [3] HEX to OCT  [4] HEX to DEC\n");
    scanf("%d", &hexTowanted);

    for (int i = 0; i < totalDigit; i++) {
        switch (hex[i]) {
            case '0': realValue = 0; break;
            case '1': realValue = 1; break;
            case '2': realValue = 2; break;
            case '3': realValue = 3; break;
            case '4': realValue = 4; break;
            case '5': realValue = 5; break;
            case '6': realValue = 6; break;
            case '7': realValue = 7; break;
            case '8': realValue = 8; break;
            case '9': realValue = 9; break;
            case 'A': realValue = 10; break;
            case 'B': realValue = 11; break;
            case 'C': realValue = 12; break;
            case 'D': realValue = 13; break;
            case 'E': realValue = 14; break;
            case 'F': realValue = 15; break;
            default: printf("Invalid Hexadecimal Input!\n"); return;
        }

        // Power means 1x16^power + 2x16^power + ...
        power = totalDigit - i - 1;
        decimal += realValue * pow(16, power);
    }

    switch (hexTowanted) {
        case 1: decTobin(decimal); break;
        case 2: decToqui(decimal); break;
        case 3: decTooct(decimal); break;
        case 4: decTodec(decimal); break;
        default: printf("Conversion Error!\n"); break;
    }
}
```

```c
void anyTodec(int userNum, int destinationType, int userNumtype, int totalDigit) {
    int dec = 0, rem;

    if (userNumtype == 2) { // Binary to Decimal
        for (int i = 0; i < totalDigit; i++) {
            rem = userNum % 10;
            if (rem > 1) {
                printf("\nINVALID INPUT!\n");
                return;
            }
            dec += rem * pow(2, i);
            userNum /= 10;
        }
    } else if (userNumtype == 5) { // Quinary to Decimal
        for (int i = 0; i < totalDigit; i++) {
            rem = userNum % 10;
            if (rem > 4) {
                printf("\nINVALID INPUT!\n");
                return;
            }
            dec += rem * pow(5, i);
            userNum /= 10;
        }
    } else if (userNumtype == 8) { // Octal to Decimal
        for (int i = 0; i < totalDigit; i++) {
            rem = userNum % 10;
            if (rem > 7) {
                printf("\nINVALID INPUT!\n");
                return;
            }
            dec += rem * pow(8, i);
            userNum /= 10;
        }
    } else if (userNumtype == 10) { // Decimal to Decimal
        dec = userNum;
    } else if (userNumtype == 16) { // Hexadecimal to Decimal
        // This case is handled in the hexadecimal function
        return;
    }

    // Sending the obtained decimal to be converted to the required base type
    switch (destinationType) {
        case 2: decTobin(dec); break;
        case 5: decToqui(dec); break;
        case 8: decTooct(dec); break;
        case 10: decTodec(dec); break;
        case 16: decTohex(dec); break;
        default: printf("Input error!\n"); break;
    }
}
```

Base conversion function

```c
void decTobin(int dec) {
    int rem, bin[30], count = 0;
    while (dec != 0) {
        rem = dec % 2;
        dec = dec / 2;
        bin[count] = rem;
        count++;
    }
    printf("BINARY: ");
    for (int i = count - 1; i >= 0; i--) {
        printf("%d", bin[i]);
    }
    printf("\n");
}

void decToqui(int dec) {
    int rem, qui[10], count = 0;
    while (dec != 0) {
        rem = dec % 5;
        dec = dec / 5;
        qui[count] = rem;
        count++;
    }
    printf("QUINARY: ");
    for (int i = count - 1; i >= 0; i--) {
        printf("%d", qui[i]);
    }
    printf("\n");
}

void decTooct(int dec) {
    int rem, oct[10], count = 0;
    while (dec != 0) {
        rem = dec % 8;
        dec = dec / 8;
        oct[count] = rem;
        count++;
    }
    printf("OCTAL: ");
    for (int i = count - 1; i >= 0; i--) {
        printf("%d", oct[i]);
    }
    printf("\n");
}
```

```c
void decTodec(int dec) {
    printf("DECIMAL: %d\n", dec);
}

void decTohex(int dec) {
    char hex[20];
    int count = 0;
    while (dec != 0) {
        int rem = dec % 16;
        if (rem < 10) {
            hex[count] = rem + '0';
        } else {
            hex[count] = rem - 10 + 'A';
        }
        count++;
        dec /= 16;
    }
    printf("HEX: ");
    for (int i = count - 1; i >= 0; i--) {
        printf("%c", hex[i]);
    }
    printf("\n");
}
```

Combination & Equation solver function

```c
void combination() {
    int n, r, factN = 1, factNR = 1, factR = 1, C;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    printf("Enter the value of r: ");
    scanf("%d", &r);
    if (n < r) {
        printf("Error: The value of n must be greater than r\n");
        return;
    }
    for (int i = 1; i <= n; i++) {
        factN *= i;
    }
    for (int i = 1; i <= r; i++) {
        factR *= i;
    }
    for (int i = 1; i <= (n - r); i++) {
        factNR *= i;
    }
    C = factN / (factNR * factR);
    printf("Combination C(%d, %d) = %d\n", n, r, C);
}
```

```c
void eqnsolv() {
    int choice;
    printf("\nEquation Solver:\n");
    printf(" 1. Linear equation\n");
    printf(" 2. Quadratic equation\n");
    printf(" 3. Cubic equation\n");
    printf("Choose the equation solver:\n");
    scanf("%d", &choice);
    switch (choice) {
        case 1: linearеqn(); break;
        case 2: quadraticeqn(); break;
        case 3: cubiceqn(); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}
```

```c
void linearеqn() {
    float a, b, x;
    printf("Enter coefficients a and b for the equation ax + b = 0:\n");
    scanf("%f%f", &a, &b);
    if (a != 0) {
        x = -b / a;
        printf("Solution: x = %.2f\n", x);
    } else {
        printf("No solution (a cannot be zero in a linear equation).\n");
    }
}

void quadraticeqn() {
    float a, b, c, discriminant, x1, x2;
    printf("Enter coefficients a, b, and c for the quadratic equation ax^2 + bx + c = 0:\n");
    scanf("%f %f %f", &a, &b, &c);
    discriminant = b * b - 4 * a * c;
    if (discriminant > 0) {
        x1 = (-b + sqrt(discriminant)) / (2 * a);
        x2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("Two real solutions: x1 = %.2f, x2 = %.2f\n", x1, x2);
    } else if (discriminant == 0) {
        x1 = -b / (2 * a);
        printf("One real solution: x = %.2f\n", x1);
    } else {
        printf("No real solutions (discriminant is negative).\n");
    }
}
```

Equation solver ,S-D , matrix operation function

```
void cubiceqn() {
    float a, b, c, d;
    printf("Enter the coefficients a, b, c, and d of the cubic equation ax^3 + bx^2 + cx + d = 0:\n");
    scanf("%f %f %f %f", &a, &b, &c, &d);
    if (a == 0) {
        printf("The equation is not cubic.\n");
        return;
    }
    float delta0 = b * b - 3 * a * c;
    float delta1 = 2 * b * b * b - 9 * a * b * c + 27 * a * a * d;
    float discriminant = delta1 * delta1 - 4 * delta0 * delta0 * delta0;
    if (discriminant >= 0) {
        float C = cbrt((delta1 + sqrt(discriminant)) / 2);
        float realPart = -1 / (3 * a) * (b + C + delta0 / C);
        printf("The equation has one real root: %f\n", realPart);
    } else {
        float r = sqrt(delta0);
        float phi = acos(delta1 / (2 * r * r * r));
        float root1 = -2 * r * cos(phi / 3) / (3 * a) - b / (3 * a);
        float root2 = -2 * r * cos((phi + 2 * M_PI) / 3) / (3 * a) - b / (3 * a);
        float root3 = -2 * r * cos((phi + 4 * M_PI) / 3) / (3 * a) - b / (3 * a);
        printf("The equation has three real roots:\n");
        printf("Root 1: %f\n", root1);
        printf("Root 2: %f\n", root2);
        printf("Root 3: %f\n", root3);
    }
}
```

```
void sdfunction() {
    float a, b;
    printf("Enter the numerator \n");
    scanf("%f", &a);
    printf("Enter the denominator\n");
    scanf("%f", &b);
    if (b == 0) {
        printf("Error: Denominator cannot be zero.\n");
        return;
    }
    float res = a / b;
    printf("The value of %.2f/%.2f is = %.2f\n", a, b, res);
}
```

```
void matrixoperation() {
    int choice;
    printf("\nMatrix Operations:\n");
    printf(" 1. Matrix Addition\n");
    printf(" 2. Matrix Subtraction\n");
    printf(" 3. Matrix Multiplication\n");
    printf(" 4. Matrix Transpose\n");
    printf("Choose the matrix operation:\n");
    scanf("%d", &choice);
    switch (choice) {
        case 1: matrixaddition(); break;
        case 2: matrixsubtraction(); break;
        case 3: matrixmultiply(); break;
        case 4: matrixtranspos(); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}
```

Matrix operation function

```c
void matrixaddition() {
    int i, j, R, C;
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &R, &C);

    int **A = (int **)malloc(R * sizeof(int *));
    int **B = (int **)malloc(R * sizeof(int *));
    int **add = (int **)malloc(R * sizeof(int *));
    for (i = 0; i < R; i++) {
        A[i] = (int *)malloc(C * sizeof(int));
        B[i] = (int *)malloc(C * sizeof(int));
        add[i] = (int *)malloc(C * sizeof(int));
    }

    printf("Enter Matrix A:\n");
    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            scanf("%d", &A[i][j]);

    printf("Enter Matrix B:\n");
    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            scanf("%d", &B[i][j]);

    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            add[i][j] = A[i][j] + B[i][j];

    printf("Sum Matrix:\n");
    for (i = 0; i < R; i++) {
        for (j = 0; j < C; j++)
            printf("%d ", add[i][j]);
        printf("\n");
    }

    for (i = 0; i < R; i++) {
        free(A[i]);
        free(B[i]);
        free(add[i]);
    }
    free(A);
    free(B);
    free(add);
}
```

```c
void matrixsubtraction() {
    int i, j, R, C;
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &R, &C);

    int **A = (int **)malloc(R * sizeof(int *));
    int **B = (int **)malloc(R * sizeof(int *));
    int **sub = (int **)malloc(R * sizeof(int *));
    for (i = 0; i < R; i++) {
        A[i] = (int *)malloc(C * sizeof(int));
        B[i] = (int *)malloc(C * sizeof(int));
        sub[i] = (int *)malloc(C * sizeof(int));
    }

    printf("Enter Matrix A:\n");S
    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            scanf("%d", &A[i][j]);

    printf("Enter Matrix B:\n");
    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            scanf("%d", &B[i][j]);

    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            sub[i][j] = A[i][j] - B[i][j];

    printf("Subtraction Matrix:\n");
    for (i = 0; i < R; i++) {
        for (j = 0; j < C; j++)
            printf("%d ", sub[i][j]);
        printf("\n");
    }

    for (i = 0; i < R; i++) {
        free(A[i]);
        free(B[i]);
        free(sub[i]);
    }
    free(A);
    free(B);
    free(sub);
}
```

Matrix operation function

```c
void matrixmultiply() {
    int i, j, k, R1, C1, R2, C2;
    printf("Enter rows and columns of first matrix: ");
    scanf("%d %d", &R1, &C1);
    printf("Enter rows and columns of second matrix: ");
    scanf("%d %d", &R2, &C2);

    if (C1 != R2) {
        printf("Matrix multiplication is not possible (C1 must be equal to R2).\n");
        return;
    }

    int **m = (int **)malloc(R1 * sizeof(int *));
    int **m1 = (int **)malloc(R2 * sizeof(int *));
    int **mul = (int **)malloc(R1 * sizeof(int *));
    for (i = 0; i < R1; i++) {
        m[i] = (int *)malloc(C1 * sizeof(int));
        mul[i] = (int *)malloc(C2 * sizeof(int));
    }
    for (i = 0; i < R2; i++) {
        m1[i] = (int *)malloc(C2 * sizeof(int));
    }

    printf("Enter elements of first matrix:\n");
    for (i = 0; i < R1; i++) {
        for (j = 0; j < C1; j++)
            scanf("%d", &m[i][j]);
    }
    printf("Enter elements of second matrix:\n");
    for (i = 0; i < R2; i++) {
        for (j = 0; j < C2; j++)
            scanf("%d", &m1[i][j]);
    }

    for (i = 0; i < R1; i++)
        for (j = 0; j < C2; j++) {
            mul[i][j] = 0;
            for (k = 0; k < C1; k++) {
                mul[i][j] += m[i][k] * m1[k][j];
            }
        }

    printf("Matrix multiplication:\n");
    for (i = 0; i < R1; i++) {
        for (j = 0; j < C2; j++)
            printf("%d ", mul[i][j]);
        printf("\n");
    }

    for (i = 0; i < R1; i++) {
        free(m[i]);
        free(mul[i]);
    }
    for (i = 0; i < R2; i++) {
        free(m1[i]);
    }
    free(m);
    free(m1);
    free(mul);
}
```

```c
void matrixtranspose() {
    int R, C, i, j;
    printf("Enter the number of rows and columns: ");
    scanf("%d%d", &R, &C);
    int **m = (int **)malloc(R * sizeof(int *));
    int **transpose = (int **)malloc(C * sizeof(int *));
    for (i = 0; i < R; i++) {
        m[i] = (int *)malloc(C * sizeof(int));
    }
    for (i = 0; i < C; i++) {
        transpose[i] = (int *)malloc(R * sizeof(int));
    }
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < R; i++) {
        for (j = 0; j < C; j++) {
            scanf("%d", &m[i][j]);
        }
    }
    for (i = 0; i < R; i++) {
        for (j = 0; j < C; j++) {
            transpose[j][i] = m[i][j];
        }
    }
    printf("Original Matrix:\n");
    for (i = 0; i < R; i++) {
        for (j = 0; j < C; j++) {
            printf("%d ", m[i][j]);
        }
        printf("\n");
    }
    printf("\nTranspose Matrix:\n");
    for (i = 0; i < C; i++) {
        for (j = 0; j < R; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < R; i++) {
        free(m[i]);
    }
    for (i = 0; i < C; i++) {
        free(transpose[i]);
    }
    free(m);
    free(transpose);
}
```

Value of constant , squareroot & permutation function

```c
void valOfconst() {
    int userInput;
    printf("[1]  (c)     [2]  (g)     [3]  (G)     [4]  (g0)\n");
    printf("[5]  (g0)    [6]  (e)     [7]  (m?)    [8]  (m?)\n");
    printf("[9]  (m?)    [10] (u)     [11] (N?)    [12] (k)\n");
    printf("[13] (x)     [14] (R)     [15] (F)     [16] (?0)\n");
    printf("[17] (u)     [18] (?)     [19] (K?)    [20] (R?)\n");
    printf("[21] (meV)   [22] (eV)    [23] (K)     [24] (M?)\n");
    printf("[25] (atm)   [26] (Å)     [27] (c1)    [28] (c2)\n");
    printf("[29] (?0)    [30] (C)     [31] (h)     [32] (h)\n");
    printf("[33] (?)     [34] (?)     [35] (b)     [36] (m?)\n");
    printf("[37] (e/m?)  [38] (??)    [39] (R?)    [40] (G0)\n");

    printf("Enter the value: ");
    scanf("%d", &userInput);

    switch (userInput) {
        case 1: printf("Speed of light in vacuum = 3.00 * 10^8 m/s\n"); break;
        case 2: printf("Acceleration due to gravity = 9.81 m/s^2\n"); break;
        case 3: printf("Gravitational constant = 6.67 * 10^-11 N·m^2/kg^2\n"); break;
        case 4: printf("Permittivity of free space = 8.85 * 10^-12 F/m\n"); break;
        case 5: printf("Permeability of free space = 4? * 10^-7 N/A^2\n"); break;
        case 6: printf("Elementary charge = 1.602 * 10^-19 C\n"); break;
        case 7: printf("Electron mass = 9.11 * 10^-31 kg\n"); break;
        case 8: printf("Proton mass = 1.67 * 10^-27 kg\n"); break;
        case 9: printf("Neutron mass = 1.675 * 10^-27 kg\n"); break;
        case 10: printf("Atomic mass unit = 1.66 * 10^-27 kg\n"); break;
        case 11: printf("Avogadro's number = 6.022 * 10^23 mol^-1\n"); break;
        case 12: printf("Boltzmann constant = 1.38 * 10^-23 J/K\n"); break;
        case 13: printf("Stefan-Boltzmann constant = 5.67 * 10^-8 W/m^2·K^4\n"); break;
        case 14: printf("Universal gas constant = 8.314 J/mol·K\n"); break;
        case 15: printf("Faraday constant = 9.65 * 10^4 C/mol\n"); break;
        case 16: printf("Magnetic flux quantum = 2.07 * 10^-15 Wb\n"); break;
        case 17: printf("Fine-structure constant = 1/137\n"); break;
        case 18: printf("Euler-Mascheroni constant = 0.577\n"); break;
        case 19: printf("Hartree energy = 4.36 * 10^-18 J\n"); break;
        case 20: printf("Rydberg constant = 1.097 * 10^7 m^-1\n"); break;
        case 21: printf("Millielectron volt = 1.602 * 10^-22 J\n"); break;
        case 22: printf("Electron volt = 1.602 * 10^-19 J\n"); break;
        case 23: printf("Kelvin = unit of temperature\n"); break;
        case 24: printf("Unified atomic mass unit = 1.66 * 10^-27 kg\n"); break;
        case 25: printf("Standard atmosphere = 101325 Pa\n"); break;
        case 26: printf("Angstrom unit = 1 * 10^-10 m\n"); break;
        case 27: printf("First radiation constant = 3.74 * 10^8 W·m^2·K4\n"); break;
        case 28: printf("Second radiation constant = 1.44 * 10^-2 m·K\n"); break;
        case 29: printf("Impedance of free space = 377 O\n"); break;
        case 30: printf("Ohm = unit of electrical resistance\n"); break;
        case 31: printf("Planck's constant = 6.626 * 10^-34 J·s\n"); break;
        case 32: printf("Reduced Planck's constant = 1.055 * 10^-34 J·s\n"); break;
        case 33: printf("Frequency = unit of frequency (Hz)\n"); break;
        case 34: printf("Wavelength = unit of wavelength (m)\n"); break;
        case 35: printf("Magnetic field strength = unit of magnetic field strength (T)\n"); break;
        case 36: printf("Atomic mass unit = 1.66 * 10^-27 kg\n"); break;
        case 37: printf("Electron charge-to-mass ratio = 1.76 * 10^11 C/kg\n"); break;
        case 38: printf("Planck's constant in terms of photon = 6.626 * 10^-34 J·s\n"); break;
        case 39: printf("Von Klitzing constant = 2.58 * 10^-5 O\n"); break;
        case 40: printf("Quantum conductance = 7.75 * 10^-5 S\n"); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}
```

```c
void squareroot() {
    float number;
    printf("Enter the number whose root is to be calculated\n");
    scanf("%f", &number);
    if (number < 0) {
        printf("Error: Cannot calculate square root of a negative number.\n");
        return;
    }
    float result = sqrt(number);
    printf("The square root of the number %.2f is = %.2f\n", number, result);
}
void permutation() {
    int n, r, factN = 1, factNR = 1, P;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    printf("Enter the value of r: ");
    scanf("%d", &r);
    if (n < r) {
        printf("Error: The value of n must be greater than r\n");
        return;
    }
    for (int i = 1; i <= n; i++) {
        factN *= i;
    }
    for (int i = 1; i <= (n - r); i++) {
        factNR *= i;
    }
    P = factN / factNR;
    printf("Permutation P(%d, %d) = %d\n", n, r, P);
}
```

Fraction to decimal conversion function

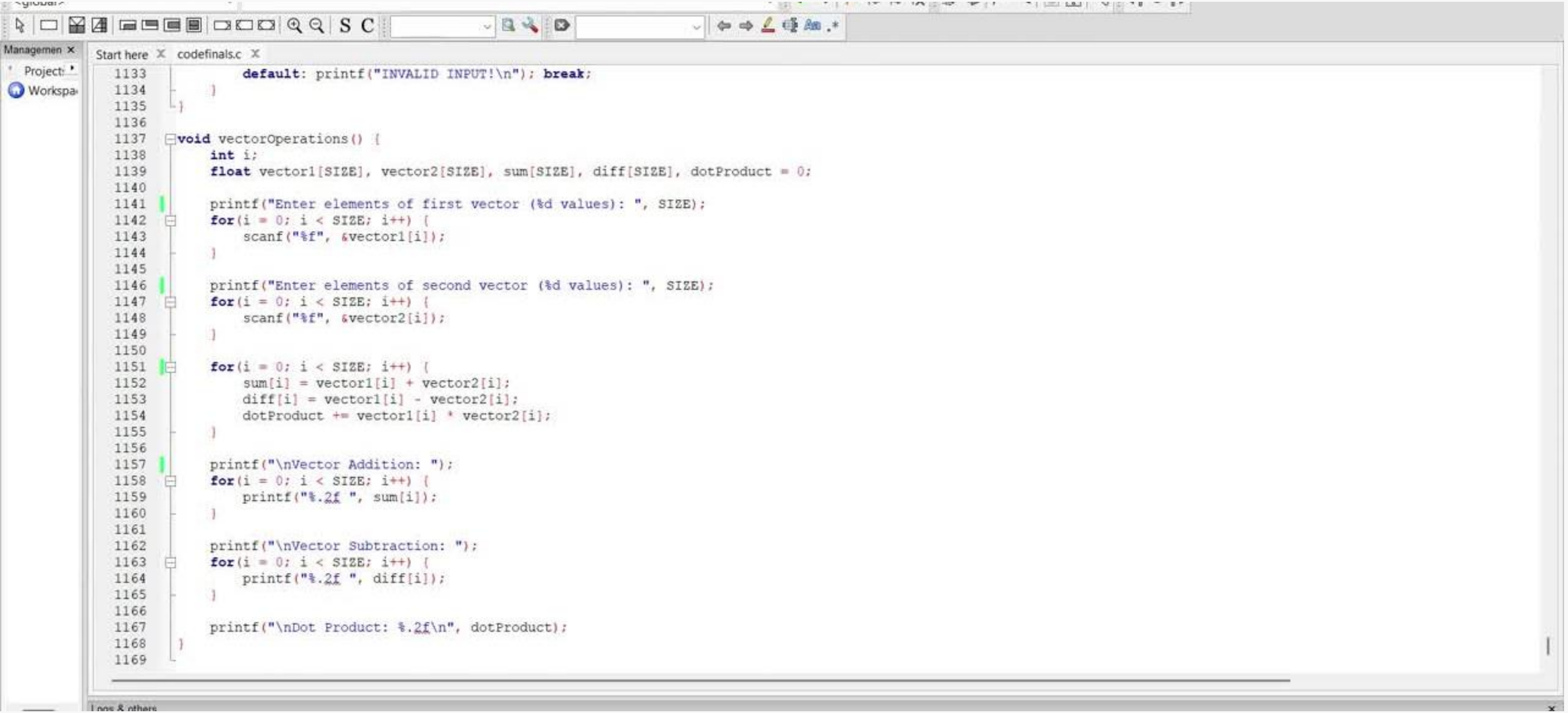
```c
void fraconversion() {
    int choice;
    printf("\nFraction Conversion\n");
    printf(" 1. Improper fraction to mixed fraction\n");
    printf(" 2. Mixed fraction to improper fraction\n");
    printf(" 3. Decimal to fraction\n");
    printf("Choose the fraction conversion:\n");
    scanf("%d", &choice);
    switch (choice) {
        case 1: mixedfraction(); break;
        case 2: improperfraction(); break;
        case 3: decimaltofrac(); break;
        default: printf("INVALID INPUT!\n"); break;
    }
}

void mixedfraction() {
    int numerator, denominator;
    printf("Enter the numerator: ");
    scanf("%d", &numerator);
    printf("Enter the denominator: ");
    scanf("%d", &denominator);
    if (denominator == 0) {
        printf("Error: Denominator cannot be zero.\n");
        return;
    }
    int wholePart = numerator / denominator;
    int newNumerator = numerator % denominator;
    if (newNumerator == 0) {
        printf("Mixed Fraction: %d\n", wholePart);
    } else if (wholePart == 0) {
        printf("Mixed Fraction: %d/%d\n", newNumerator, denominator);
    } else {
        printf("Mixed Fraction: %d %d/%d\n", wholePart, newNumerator, denominator);
    }
}
```

```c
void improperfraction() {
    int wholePart, numerator, denominator;
    printf("Enter the whole part: ");
    scanf("%d", &wholePart);
    printf("Enter the numerator: ");
    scanf("%d", &numerator);
    printf("Enter the denominator: ");
    scanf("%d", &denominator);
    if (denominator == 0) {
        printf("Error: Denominator cannot be zero.\n");
        return;
    }
    int improperNumerator = (wholePart * denominator) + numerator;
    printf("Improper Fraction: %d/%d\n", improperNumerator, denominator);
}

void decimaltofrac() {
    float decimal;
    printf("Enter a decimal number: ");
    scanf("%f", &decimal);
    int numerator = (int)(decimal * 1000);
    int denominator = 1000;
    while (numerator % 2 == 0 && denominator % 2 == 0) {
        numerator /= 2;
        denominator /= 2;
    }
    printf("Decimal to fraction: %d/%d\n", numerator, denominator);
}
```

Vector operation function

```c
void vectorOperations() {
    int i;
    float vector1[SIZE], vector2[SIZE], sum[SIZE], diff[SIZE], dotProduct = 0;

    // Input for first vector
    printf("Enter elements of first vector (%d values): ", SIZE);
    for(i = 0; i < SIZE; i++) {
        scanf("%f", &vector1[i]);
    }

    // Input for second vector
    printf("Enter elements of second vector (%d values): ", SIZE);
    for(i = 0; i < SIZE; i++) {
        scanf("%f", &vector2[i]);
    }

    // Perform operations
    for(i = 0; i < SIZE; i++) {
        sum[i] = vector1[i] + vector2[i];
        diff[i] = vector1[i] - vector2[i];
        dotProduct += vector1[i] * vector2[i];
    }

    // Display results
    printf("\nVector Addition: ");
    for(i = 0; i < SIZE; i++) {
        printf("%.2f ", sum[i]);
    }

    printf("\nVector Subtraction: ");
    for(i = 0; i < SIZE; i++) {
        printf("%.2f ", diff[i]);
    }

    printf("\nDot Product: %.2f\n", dotProduct);
}
```

Demo video


Start here
codefinals.c
default: printf("INVALID INPUT!\n"); break;
}
}
void vectorOperations() {
int i;
float vector1[SIZE], vector2[SIZE], sum[SIZE], diff[SIZE], dotProduct = 0;
printf("Enter elements of first vector (%d values): ", SIZE);
for(i = 0; i < SIZE; i++) {
scanf("%f", &vector1[i]);
}
printf("Enter elements of second vector (%d values): ", SIZE);
for(i = 0; i < SIZE; i++) {
scanf("%f", &vector2[i]);
}
for(i = 0; i < SIZE; i++) {
sum[i] = vector1[i] + vector2[i];
diff[i] = vector1[i] - vector2[i];
dotProduct += vector1[i] * vector2[i];
}
printf("\nVector Addition: ");
for(i = 0; i < SIZE; i++) {
printf("%.2f ", sum[i]);
}
printf("\nVector Subtraction: ");
for(i = 0; i < SIZE; i++) {
printf("%.2f ", diff[i]);
}
printf("\nDot Product: %.2f\n", dotProduct);
}

Conclusion

In conclusion, developing a scientific calculator in C demonstrates the power and flexibility of the C programming language in handling complex mathematical operations. Through the implementation of functions for arithmetic, trigonometry, logarithms, exponentiation, and more, we can create an efficient and user-friendly tool for performing various calculations.
This project not only enhances our understanding of C programming concepts, such as functions, loops, conditional statements, and memory management, but also improves problem-solving skills by applying mathematical logic to real-world applications.