perplexity

give code for matrix addition, multiplication, transpose and inverse in c with detailed step breakdown and syntax and code walkthrough I need to be able to reconstruct this code tomorrow. without fail.

Here's a comprehensive C code for matrix operations (addition, multiplication, transpose, and inverse) with detailed explanations:

```
#include <stdio.h>
#include <stdlib.h>

// Function prototypes
void matrixAddition();
void matrixMultiplication();
void matrixTranspose();
void matrixInverse();
void printMatrix(float mat[^3][^3]);
float determinant3x3(float mat[^3][^3]);

int main() {
    int choice;

    do {
        printf("\nMatrix Operations Menu:\n");
        printf("1. Matrix Addition\n");
        printf("2. Matrix Multiplication\n");
        printf("3. Matrix Transpose\n");
        printf("4. Matrix Inverse (3x3 only)\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: matrixAddition(); break;
            case 2: matrixMultiplication(); break;
            case 3: matrixTranspose(); break;
            case 4: matrixInverse(); break;
            case 5: printf("Exiting...\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while(choice != 5);

    return 0;
}

// Matrix Addition
```

```
void matrixAddition() {
    int rows, cols;
    printf("\nEnter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    int mat1[rows][cols], mat2[rows][cols], sum[rows][cols];

    // Input matrices
    printf("Enter elements of first matrix:\n");
    for(int i=0; i<rows; i++)
        for(int j=0; j<cols; j++)
            scanf("%d", &mat1[i][j]);

    printf("Enter elements of second matrix:\n");
    for(int i=0; i<rows; i++)
        for(int j=0; j<cols; j++)
            scanf("%d", &mat2[i][j]);

    // Addition
    for(int i=0; i<rows; i++)
        for(int j=0; j<cols; j++)
            sum[i][j] = mat1[i][j] + mat2[i][j];

    // Display result
    printf("\nSum of matrices:\n");
    for(int i=0; i<rows; i++) {
        for(int j=0; j<cols; j++)
            printf("%d\t", sum[i][j]);
        printf("\n");
    }
}

// Matrix Multiplication
void matrixMultiplication() {
    int r1, c1, r2, c2;
    printf("\nEnter rows and columns for first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and columns for second matrix: ");
    scanf("%d %d", &r2, &c2);

    if(c1 != r2) {
        printf("Multiplication not possible!\n");
        return;
    }

    int mat1[r1][c1], mat2[r2][c2], product[r1][c2];

    // Input matrices
    printf("Enter elements of first matrix:\n");
    for(int i=0; i<r1; i++)
        for(int j=0; j<c1; j++)
            scanf("%d", &mat1[i][j]);

    printf("Enter elements of second matrix:\n");
    for(int i=0; i<r2; i++)
        for(int j=0; j<c2; j++)
```

```
            scanf("%d", &mat2[i][j]);

    // Multiplication
    for(int i=0; i<r1; i++) {
        for(int j=0; j<c2; j++) {
            product[i][j] = 0;
            for(int k=0; k<c1; k++)
                product[i][j] += mat1[i][k] * mat2[k][j];
        }
    }

    // Display result
    printf("\nProduct matrix:\n");
    for(int i=0; i<r1; i++) {
        for(int j=0; j<c2; j++)
            printf("%d\t", product[i][j]);
        printf("\n");
    }
}

// Matrix Transpose
void matrixTranspose() {
    int rows, cols;
    printf("\nEnter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    int mat[rows][cols], transpose[cols][rows];

    // Input matrix
    printf("Enter matrix elements:\n");
    for(int i=0; i<rows; i++)
        for(int j=0; j<cols; j++)
            scanf("%d", &mat[i][j]);

    // Transpose
    for(int i=0; i<rows; i++)
        for(int j=0; j<cols; j++)
            transpose[j][i] = mat[i][j];

    // Display result
    printf("\nTranspose matrix:\n");
    for(int i=0; i<cols; i++) {
        for(int j=0; j<rows; j++)
            printf("%d\t", transpose[i][j]);
        printf("\n");
    }
}

// Matrix Inverse (3x3 only)
float determinant3x3(float mat[^3][^3]) {
    return mat[^0][^0] * (mat[^1][^1]*mat[^2][^2] - mat[^2][^1]*mat[^1][^2])
         - mat[^0][^1] * (mat[^1][^0]*mat[^2][^2] - mat[^1][^2]*mat[^2][^0])
         + mat[^0][^2] * (mat[^1][^0]*mat[^2][^1] - mat[^1][^1]*mat[^2][^0]);
}

void matrixInverse() {
```

```
    float mat[^3][^3], inv[^3][^3];
    float det;

    printf("\nEnter 3x3 matrix elements:\n");
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            scanf("%f", &mat[i][j]);

    det = determinant3x3(mat);

    if(det == 0) {
        printf("Matrix is singular, inverse not possible!\n");
        return;
    }

    // Calculate inverse using adjugate matrix
    for(int i=0; i<3; i++) {
        for(int j=0; j<3; j++) {
            inv[i][j] = ((mat[(j+1)%3][(i+1)%3] * mat[(j+2)%3][(i+2)%3])
                        - (mat[(j+1)%3][(i+2)%3] * mat[(j+2)%3][(i+1)%3])) / det;
        }
    }

    printf("\nInverse matrix:\n");
    printMatrix(inv);
}

void printMatrix(float mat[^3][^3]) {
    for(int i=0; i<3; i++) {
        for(int j=0; j<3; j++)
            printf("%.2f\t", mat[i][j]);
        printf("\n");
    }
}
```

Code Walkthrough:

1. **Menu System:**
   - The program starts with a menu-driven interface
   - Users can choose between operations or exit
2. **Matrix Addition:**
   - Requires two matrices of same dimensions
   - Adds corresponding elements
   - Time Complexity: $O(n^2)$
3. **Matrix Multiplication:**
   - First matrix columns must match second matrix rows
   - Uses triple nested loops for dot product calculation
   - Time Complexity: $O(n^3)$
4. **Matrix Transpose:**

- Swaps rows and columns
- Creates new matrix with flipped indices
- Time Complexity: $O(n^2)$

5. **Matrix Inverse (3×3):**
   - Calculates determinant using rule of Sarrus
   - Uses compact index manipulation for adjugate matrix
   - Applies formula: inverse = (1/determinant) * adjugate
   - Handles singular matrices (determinant = 0)

Key Features:

- Input validation for matrix dimensions
- Clear error messages for invalid operations
- Formatted output for better readability
- Modular design with separate functions for each operation
- Handles dynamic matrix sizes for addition/multiplication/transpose
- Specialized 3×3 implementation for inverse

To Reconstruct Tomorrow:

1. Remember the main menu structure
2. For addition/multiplication: use nested loops
3. Transpose is simple index swapping
4. Inverse uses determinant and index manipulation pattern
5. Use helper functions for determinant and printing

Example Usage:

1. Compile with `gcc matrix_ops.c -o matrix_ops`
2. Run with `./matrix_ops`
3. Follow menu prompts to select operation
4. Input matrix elements when prompted
5. View results

This code provides a complete implementation of basic matrix operations in C, following good programming practices with modular design and error checking.

⁂