Name: Samiksha Suryawanshi
Enroll No: 0801CS221122

Programming Practices

MINI PROJECT

- **TOPIC : 3D Vector Mathematical Utility**
  (A Comprehensive Tool for VectorOperations)
- **CODE:**

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

// Structure to represent a 3D vector
typedef struct
{
    double x;
    double y;
    double z;
} Vector3D;

// Function to add two 3D vectors
Vector3D addVectors(Vector3D v1, Vector3D v2)
{
    return (Vector3D){v1.x + v2.x, v1.y + v2.y, v1.z + v2.z};
}

// Function to subtract two 3D vectors
Vector3D subtractVectors(Vector3D v1, Vector3D v2)
{
    return (Vector3D){v1.x - v2.x, v1.y - v2.y, v1.z - v2.z};
}

// Function to calculate the dot product of two 3D vectors
double dotProduct(Vector3D v1, Vector3D v2)
{
    return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}

// Function to calculate the cross product of two 3D vectors
```

```
Vector3D crossProduct(Vector3D v1, Vector3D v2)
{
    return (Vector3D){
        v1.y * v2.z - v1.z * v2.y,
        v1.z * v2.x - v1.x * v2.z,
        v1.x * v2.y - v1.y * v2.x};
}

// Function to calculate the magnitude of a vector
double magnitude(Vector3D v)
{
    return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

// Function to scale a vector by a scalar
Vector3D scalarMultiply(Vector3D v, double scalar)
{
    return (Vector3D){v.x * scalar, v.y * scalar, v.z * scalar};
}

// Function to calculate the unit vector of a vector
Vector3D unitVector(Vector3D v)
{
    double mag = magnitude(v);
    if (mag == 0.0)
    {
        return (Vector3D){0.0, 0.0, 0.0}; // Avoid division by zero
    }
    return (Vector3D){v.x / mag, v.y / mag, v.z / mag};
}

// Function to calculate the scalar projection of vector v1 onto v2
double scalarProjection(Vector3D v1, Vector3D v2)
{
    double dot = dotProduct(v1, v2);
    double mag2 = dotProduct(v2, v2);
    if (mag2 == 0.0)
    {
        return 0.0; // Avoid division by zero
    }
```

```
    return dot / sqrt(mag2);
}

// Function to calculate the vector triple product
Vector3D vectorTripleProduct(Vector3D a, Vector3D b, Vector3D c)
{
    Vector3D result = crossProduct(a, crossProduct(b, c));
    return result;
}

// Function to calculate the angle (in degrees) between two vectors
double angleBetweenVectors(Vector3D v1, Vector3D v2)
{
    double dot = dotProduct(v1, v2);
    double mag1 = magnitude(v1);
    double mag2 = magnitude(v2);

    // Check for division by zero (to avoid NaN)
    if (mag1 == 0 || mag2 == 0)
    {
        return 0.0;
    }

    double cosTheta = dot / (mag1 * mag2);
    return acos(cosTheta) * 180.0 / M_PI; // Convert radians to degrees
}

// Function to display the equation of a plane given normal vector and a point on the
plane
void displayPlaneEquation(Vector3D normal, Vector3D point)
{
    printf("Plane Equation: %.2fx + %.2fy + %.2fz = %.2f\n", normal.x, normal.y, normal.z,
          normal.x * point.x + normal.y * point.y + normal.z * point.z);
}

// Function to display the equation of a line given a point on the line and its direction
vector
void displayLineEquation(Vector3D point, Vector3D direction)
{
    printf("Line Equation: x = %.2f + %.2ft, y = %.2f + %.2ft, z = %.2f + %.2ft\n",
```

```
        point.x, direction.x, point.y, direction.y, point.z, direction.z);
}

int main()
{
    int choice;
    Vector3D vector1, vector2, vector3;

    // Input for vector1
    printf("Enter components of the first 3D vector (x y z): ");
    scanf("%lf %lf %lf", &vector1.x, &vector1.y, &vector1.z);

    // Input for vector2
    printf("Enter components of the second 3D vector (x y z): ");
    scanf("%lf %lf %lf", &vector2.x, &vector2.y, &vector2.z);

    // Menu for vector operations
    printf("\nChoose a vector operation:\n");
    printf("1. Vector Addition\n");
    printf("2. Vector Subtraction\n");
    printf("3. Dot Product\n");
    printf("4. Cross Product\n");
    printf("5. Vector Projection\n");
    printf("6. Magnitude\n");
    printf("7. Scalar Projection\n");
    printf("8. Vector Triple Product\n");
    printf("9. Angle Between Vectors\n");
    printf("10. Unit Vector\n");
    printf("11. Plane Equation\n");
    printf("12. Line Equation\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    // User choice
    switch (choice)
    {
    case 1:
    {
        // Vector Addition
        Vector3D sum = addVectors(vector1, vector2);
```

```
        printf("Vector Addition: (%.2f, %.2f, %.2f) + (%.2f, %.2f, %.2f) = (%.2f, %.2f,
%.2f)\n",
                vector1.x, vector1.y, vector1.z, vector2.x, vector2.y, vector2.z, sum.x, sum.y,
sum.z);
        break;
    }
    case 2:
    {
        // Vector Subtraction
        Vector3D difference = subtractVectors(vector1, vector2);
        printf("Vector Subtraction: (%.2f, %.2f, %.2f) - (%.2f, %.2f, %.2f) = (%.2f, %.2f,
%.2f)\n",
                vector1.x, vector1.y, vector1.z, vector2.x, vector2.y, vector2.z, difference.x,
difference.y, difference.z);
        break;
    }
    case 3:
    {
        // Dot Product
        double dot = dotProduct(vector1, vector2);
        printf("Dot Product: %.2f\n", dot);
        break;
    }
    case 4:
    {
        // Cross Product
        Vector3D cross = crossProduct(vector1, vector2);
        printf("Cross Product: (%.2f, %.2f, %.2f) x (%.2f, %.2f, %.2f) = (%.2f, %.2f,
%.2f)\n",
                vector1.x, vector1.y, vector1.z, vector2.x, vector2.y, vector2.z, cross.x, cross.y,
cross.z);
        break;
    }
      case 5:
    {
        // Vector Projection
        double dot = dotProduct(vector1, vector2);
        double magSquared = dotProduct(vector2, vector2);
        Vector3D projection = scalarMultiply(vector2, dot / magSquared);
```

```
        printf("Vector Projection: (%.2f, %.2f, %.2f) projected onto (%.2f, %.2f, %.2f) =
(%.2f, %.2f, %.2f)\n",
                vector1.x, vector1.y, vector1.z, vector2.x, vector2.y, vector2.z, projection.x,
projection.y, projection.z);
        break;
    }

    case 6:
    {
        // Magnitude
        double mag1 = magnitude(vector1);
        double mag2 = magnitude(vector2);
        printf("Magnitude of Vector 1: %.2f\n", mag1);
        printf("Magnitude of Vector 2: %.2f\n", mag2);
        break;
    }
    case 7:
    {
        // Scalar Projection
        double scalarProj = scalarProjection(vector1, vector2);
        printf("Scalar Projection of Vector 1 onto Vector 2: %.2f\n", scalarProj);
        break;
    }
    case 8:
    {
        // Vector Triple Product
        Vector3D vector3;
        printf("Enter components of the third 3D vector (x y z): ");
        scanf("%lf %lf %lf", &vector3.x, &vector3.y, &vector3.z);
        Vector3D tripleProduct = vectorTripleProduct(vector1, vector2, vector3);
        printf("Vector Triple Product: ((%.2f, %.2f, %.2f) x (%.2f, %.2f, %.2f)) x (%.2f, %.2f,
%.2f) = (%.2f, %.2f, %.2f)\n",
                vector1.x, vector1.y, vector1.z, vector2.x, vector2.y, vector2.z, vector3.x,
vector3.y, vector3.z, tripleProduct.x, tripleProduct.y, tripleProduct.z);
        break;
    }
    case 9:
    {
        // Angle Between Vectors
        double angle = angleBetweenVectors(vector1, vector2);
```

```
        printf("Angle Between Vectors 1 and 2: %.2f degrees\n", angle);
        break;
    }
    case 10:
    {
        // Unit Vector
        Vector3D unitVec1 = unitVector(vector1);
        Vector3D unitVec2 = unitVector(vector2);
        printf("Unit Vector of Vector 1: (%.2f, %.2f, %.2f)\n", unitVec1.x, unitVec1.y,
unitVec1.z);
        printf("Unit Vector of Vector 2: (%.2f, %.2f, %.2f)\n", unitVec2.x, unitVec2.y,
unitVec2.z);
        break;
    }
    case 11:
    {
        // Plane Equation
        Vector3D normal, point;
        printf("Enter the normal vector of the plane (x y z): ");
        scanf("%lf %lf %lf", &normal.x, &normal.y, &normal.z);
        printf("Enter a point on the plane (x y z): ");
        scanf("%lf %lf %lf", &point.x, &point.y, &point.z);
        displayPlaneEquation(normal, point);
        break;
    }
    case 12:
    {
        // Line Equation
        Vector3D linePoint, lineDirection;
        printf("Enter a point on the line (x y z): ");
        scanf("%lf %lf %lf", &linePoint.x, &linePoint.y, &linePoint.z);
        printf("Enter the direction vector of the line (x y z): ");
        scanf("%lf %lf %lf", &lineDirection.x, &lineDirection.y, &lineDirection.z);

        // Display the line equation
        displayLineEquation(linePoint, lineDirection);
        break;
    }
    case 13:
    {
```

```
        // Unique Vector
        // Implement the logic for the unique vector operation
        printf("This is the Unique Vector operation.\n");
        break;
    }
    default:
    {
        printf("Invalid choice. Please choose a valid option from the menu.\n");
    }
    }

    return 0;
}
```

Output:

```
Enter components of the first 3D vector (x y z): 1 2 3
Enter components of the second 3D vector (x y z): 1 2 4

Choose a vector operation:
1. Vector Addition
2. Vector Subtraction
3. Dot Product
4. Cross Product
5. Vector Projection
6. Magnitude
7. Scalar Projection
8. Vector Triple Product
9. Angle Between Vectors
10. Unit Vector
11. Plane Equation
12. Line Equation
Enter your choice: 10
Unit Vector of Vector 1: (0.27, 0.53, 0.80)
Unit Vector of Vector 2: (0.22, 0.44, 0.87)
```

Profiling

```
cndc-7@cndc7-OptiPlex-3050-AIO:~$ gcc one.c -lm
cndc-7@cndc7-OptiPlex-3050-AIO:~$ .\a.out
.a.out: command not found
cndc-7@cndc7-OptiPlex-3050-AIO:~$ ./a.out
Enter components of the first 3D vector (x y z): 1 2 3
Enter components of the second 3D vector (x y z): 1 2 4

Choose a vector operation:
1. Vector Addition
2. Vector Subtraction
3. Dot Product
4. Cross Product
5. Vector Projection
6. Magnitude
7. Scalar Projection
8. Vector Triple Product
9. Angle Between Vectors
10. Unit Vector
11. Plane Equation
12. Line Equation
Enter your choice: 6
Magnitude of Vector 1: 3.74
Magnitude of Vector 2: 4.58
cndc-7@cndc7-OptiPlex-3050-AIO:~$ gcc -pg -o 1 one.c -lm
cndc-7@cndc7-OptiPlex-3050-AIO:~$ gprof 1 gmon.out>file.txt
cndc-7@cndc7-OptiPlex-3050-AIO:~$ gedit file.txt
cndc-7@cndc7-OptiPlex-3050-AIO:~$
```

Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	1	0.00	0.00	__do_global_dtors_aux
0.00	0.00	0.00	1	0.00	0.00	addVectors
0.00	0.00	0.00	1	0.00	0.00	subtractVectors

% the percentage of the total running time of the

time   program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self   the number of seconds accounted for by this
seconds   function alone. This is the major sort for this
           listing.

calls   the number of times this function was invoked, if
           this function is profiled, else blank.

 self   the average number of milliseconds spent in this
ms/call   function per call, if this function is profiled,
           else blank.

 total   the average number of milliseconds spent in this
ms/call   function and its descendents per call, if this
           function is profiled, else blank.

name   the name of the function. This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.

                        Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) no time propagated

index % time   self children   called   name
                0.00   0.00   1/1   crossProduct [5]
[1]   0.0   0.00   0.00   1   addVectors [1]

                0.00   0.00   1/1   crossProduct [5]

```
[2]      0.0    0.00    0.00       1         subtractVectors [2]
-----------------------------------------------
               0.00    0.00       1/1           crossProduct [5]
[20]     0.0    0.00    0.00       1         __do_global_dtors_aux [20]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:

index
: A unique number given to each element of the table.
Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function is in the table.

% time
: This is the percentage of the `total' time that was spent
in this function and its children. Note that due to
different viewpoints, functions excluded by options, etc,
these numbers will NOT add up to 100%.

self
: This is the total amount of time spent in this function.

children
: This is the total amount of time propagated into this
function by its children.

called
: This is the number of times the function was called.
If the function called itself recursively, the number
only includes non-recursive calls, and is followed by
a `+' and the number of recursive calls.

name
: The name of the current function. The index number is
printed after it. If the function is a member of a
cycle, the cycle number is printed between the
function's name and the index number.

For the function's parents, the fields have the following meanings:

selfThis is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

selfThis is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an
entry for the cycle-as-a-whole. This entry shows who called the
cycle (as parents) and the members of the cycle (as children.)
The `+' recursive calls entry shows the number of function calls that
were internal to the cycle, and the calls entry for each member shows,
for that member, how many times it was called from other members of
the cycle.

Index by function name

Debugging

```
Reading symbols from a.out...
(gdb) break main
Breakpoint 1 at 0x17cf: file main.c, line 114.
(gdb) run
Starting program: /home/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at main.c:114
114     {
(gdb) next
119         printf("Enter components of the first 3D vector (x y z): ");
(gdb) next 127
Enter components of the first 3D vector (x y z): 1 2 3
Enter components of the second 3D vector (x y z): 1 2 4

Choose a vector operation:
1. Vector Addition
2. Vector Subtraction
3. Dot Product
4. Cross Product
5. Vector Projection
6. Magnitude
7. Scalar Projection
8. Vector Triple Product
9. Angle Between Vectors
10. Unit Vector
11. Plane Equation
```

```
12. Line Equation
Enter your choice: 10
Unit Vector of Vector 1: (0.27, 0.53, 0.80)
Unit Vector of Vector 2: (0.22, 0.44, 0.87)
[Inferior 1 (process 3144) exited normally]
(gdb) continue
```

Latex

Code:

```
\documentclass[12pt]{article}
\usepackage{geometry}
\usepackage{titlesec}
\usepackage{enumitem}

\geometry{margin=1in}
\titleformat{\section}{\normalfont\Large\bfseries}{\thesection}{1em}{}
\titleformat{\subsection}{\normalfont\large\bfseries}{\thesubsection}{1em}{}
\titleformat{\subsubsection}{\normalfont\normalsize\bfseries}{\thesubsubsection}{
1em}{}

\begin{document}

\title{\begin{center}
    \textbf{3D Vector Mathematical Utility}:
        Comprehensive Tool for Vector Operations
\end{center}}



\maketitle

\section{Introduction}
The \textit{3D Vector Mathematical Utility},  is a powerful program designed to
perform a wide range of operations on 3D vectors. This mathematical tool covers
essential concepts related to 3D vectors and provides users with an interactive
interface to select and execute various operations. In this presentation, we will
explore the main topics and aims of this project, highlighting its key features and
potential applications.

\section{Objective}
The main objective of the \textit{3D Vector Mathematical Utility} is to provide
users with a versatile and user-friendly tool for performing various operations on
3D vectors. By offering a comprehensive set of functionalities, this program aims
to simplify vector-related calculations and provide clear, formatted output for each
operation selected.
```

\section{Main Topics and Aims}

\subsection{Vector Operations}
\textbf{Aim:} Enable users to perform essential operations involving 3D vectors. \\
\textbf{Key Features:} Addition, subtraction, dot product, cross product, vector projection, magnitude, scalar projection, vector triple product, angle between vectors, unit vector, plane equation, line equation.

\subsection{User Interaction}
\textbf{Aim:} Provide an intuitive and interactive menu for users to select desired operations easily. \\
\textbf{Key Features:} Interactive interface, menu-driven system, user-friendly experience.

\subsection{Error Handling}
\textbf{Aim:} Ensure safe execution of operations by implementing error handling mechanisms. \\
\textbf{Key Features:} Division by zero prevention, error detection, graceful error handling.

\section{Potential Applications}
\textbf{Aim:} Highlight the practical applications of the \textit{3D Vector Mathematical Utility}. \\
\textbf{Key Areas:} Physics, computer graphics, engineering, geometry, and other fields utilizing 3D vector calculations.

\section{Conclusion}
The \textit{3D Vector Mathematical Utility} is a comprehensive and versatile tool for performing various operations on 3D vectors. With its user-friendly interface, extensive range of functionalities, and error handling mechanisms, this program simplifies vector-related calculations and provides clear, formatted output for each operation. Whether you are working in physics, computer graphics, engineering, or any field that involves 3D vector calculations, this utility will prove to be an invaluable asset. The \textit{3D Vector Mathematical Utility} is beneficial for your vector-related mathematical needs.

\hspace{1.5}\\
\textbf{\Large Output}\\
\\

\textbf{Enter components of the first 3D vector (x y z): 1 2 3 \\
Enter components of the second 3D vector (x y z): 1 2 4 \\}

\textbf{Choose a vector operation: \\
1. Vector Addition \\
2. Vector Subtraction \\
3. Dot Product \\
4. Cross Product \\
5. Vector Projection \\
6. Magnitude \\
7. Scalar Projection \\
8. Vector Triple Product \\
9. Angle Between Vectors \\
10. Unit Vector \\
11. Plane Equation \\
12. Line Equation \\}

Enter your choice: 10 \\
\textbf{Unit Vector of Vector 1: (0.27, 0.53, 0.80)} \\
\textbf{Unit Vector of Vector 2: (0.22, 0.44, 0.87)} \\

\end{document}

3D Vector Mathematical Utility: Comprehensive Tool for Vector Operations

1. Introduction

The *3D Vector Mathematical Utility*, is a powerful program designed to perform a wide range of operations on 3D vectors. This mathematical tool covers essential concepts related to 3D vectors and provides users with an interactive interface to select and execute various operations. In this presentation, we will explore the main topics and aims of this project, highlighting its key features and potential applications.

2. Objective

The main objective of the *3D Vector Mathematical Utility* is to provide users with a versatile and user-friendly tool for performing various operations on 3D vectors. By offering a comprehensive set of functionalities, this program aims to simplify vector related calculations and provide clear, formatted output for each operation selected.

3. Main Topics and Aims

3.1 Vector Operations

Aim: Enable users to perform essential operations involving 3D vectors. Key Features: Addition, subtraction, dot product, cross product, vector projection, magnitude, scalar projection, vector triple product, angle between vectors, unit vector, plane equation, line equation.

3.2 User Interaction

Aim: Provide an intuitive and interactive menu for users to select desired operations easily.
Key Features: Interactive interface, menu-driven system, user-friendly experience.

3.3 Error Handling

Aim: Ensure safe execution of operations by implementing error handling mechanisms.
Key Features: Division by zero prevention, error detection, graceful error handling.

4. Potential Applications

Aim: Highlight the practical applications of the *3D Vector Mathematical Utility*. Key Areas: Physics, computer graphics, engineering, geometry, and other fields utilizing 3D vector calculations.

5. Conclusion

The *3D Vector Mathematical Utility* is a comprehensive and versatile tool for performing various operations on 3D vectors. With its user-friendly interface, extensive range of functionalities, and error handling mechanisms, this program simplifies vector-related calculations and provides clear, formatted output for each operation. Whether you are working in physics, computer graphics, engineering, or any field that involves 3D vector calculations, this utility will prove to be an invaluable asset. The *3D Vector Mathematical Utility* is beneficial for your vector-related mathematical needs.

Output:

Enter components of the first 3D vector (x y z): 1 2 3
Enter components of the second 3D vector (x y z): 1 2 4

Choose a vector operation:
1. Vector Addition
2. Vector Subtraction
3. Dot Product
4. Cross Product
5. Vector Projection
6. Magnitude
7. Scalar Projection
8. Vector Triple Product
9. Angle Between Vectors
10. Unit Vector
11. Plane Equation
12. Line Equation

Enter your choice: 10
Unit Vector of Vector 1: (0.27, 0.53, 0.80)
Unit Vector of Vector 2: (0.22, 0.44, 0.87)