

Maths Assignment report

by James Millns

FILE	6668_JAMES_MILLNS_MATHS_ASSIGNMENT_REPORT_112721_1132365 946.DOCX		
TIME SUBMITTED	25-APR-2018 10:17PM (UTC+0100)	WORD COUNT	5426
SUBMISSION ID	87480828	CHARACTER COUNT	30630

Maths for Software Development

Name

James Edward Millns

Student No

1708455

Email

1708455@student.uwtsd.ac.uk

Course

BSC (Hons) Software Engineering

Module Name

Maths for Software Development

Module Code

M4X01015

Lecturer

Gordon Dickers

Table of Contents

1. Menu & Quit.....	3
2. Length of a Vector	7
3. Normalize	10
4. Distance between two Points.....	12
5. Cross Product and Dot Product of two vectors	15
6. Multiply a (4x4) and (4x1) Matrix.....	17
7. Mean, Median, Mode and Standard Deviation.....	22
8. Union and Intersection.....	28

1. Menu & Quit

Brief

Create a testing harness for a maths library application which will allow the user to choose an option to perform a task and then return them to the menu, while giving them the option to quit the application.

Problems Identified

- Display menu to console.
- Allow user to enter choice.
- Prevent user from entering invalid choice.
- Process the user's choice.
- Return to the menu when the option is finished.
- Allow user to quit application.

Solutions Required

- Use functions which print the menu elements.
- Use a function which reads the users input.
- Check the users input and determine if a valid entry, if not then prompt user to try again.
- Use a while loop which contains the application options.
- Give while loop a condition which will change to false if user wishes to exit, ending the loop.

Implementation

```
void PrintHeader()
{
    printf("#####\r\n");
    printf("##### Maths #####\r\n");
    printf("##### Library #####\r\n");
    printf("#####\r\n\r\n");
}
```

```

void PrintMenu()
{
    PrintHeader();
    printf("1. Find the length of a vector\n");
    printf("2. Normalize a vector\n");
    printf("3. Distance between two points\n");
    printf("4. Cross Product of two vectors\n");
    printf("5. Multiply a 4x4 and 4x1 column matrix\n");
    printf("6. Mean, Median, Mode and Standard Deviation of a list of integers\n");
    printf("7. Union and Intersection of two sets\n");
    printf("8. Quit Application\n");
    printf("\nPlease select an option >> ");
}

```

```

int ReadInteger()
{
    int number;
    int checkInput;

    checkInput = scanf("%i", &number);
    IgnoreInput();

    while (checkInput != 1)
    {
        printf("\nInvalid Option, please try again >>");
        checkInput = scanf("%i", &number);
        IgnoreInput();
    }

    return number;
}

```

The above function uses the scan function in a dual purpose, it simultaneously reads the value into the number variable, and the checkInput takes the return value of the scan function. If the scan function read an integer, then it would return 1 as it scanned 1 value, if an invalid character was entered, then scan f would return 0 to say it did not scan anything. If the checkInput is not equal to 1, then it means the user entered an invalid character, this is used as a condition for a while loop which prompts the user to enter a value until a valid character is entered.

The function below is used to clear the buffer when the user enters an invalid character as the scan function will not remove it.

```

void IgnoreInput()
{
    char ch;
    while ((ch = getchar()) != '\n' && ch != EOF);
}

```

```

bool ProcessMenu(int option)
{
    switch (option)
    {
        case 1:
            Option1();
            break;
        case 2:
            Option2();
            break;
        case 3:
            Option3();
            break;
        case 4:
            Option4();
            break;
        case 5:
            Option5();
            break;
        case 6:
            Option6();
            break;
        case 7:
            Option7();
            break;
        case 8:
            if (IsQuitting())
            {
                return true;
            }
            break;
        default:
            printf("Invalid Option\r\n");
            break;
    }
    return false;
}

```

The above function uses a switch statement to choose the correct function to execute based on the users chosen option. It returns a bool value which is used in as a condition in the main.

```

bool IsQuitting()
{
    string answer;
    char text[100];
    printf("\r\nAre you sure you want to quit? ");
    scanf("%s", text);
    answer = text;

    if (answer == "YES" || answer == "Yes" || answer == "yes" || answer == "Y" || answer == "y")
    {
        return true;
    }
    return false;
}

```

The above function reads a text input from the user, which is then checked to return a bool. This asks the user if they want to quit the application when they choose the quit option.

```

int main()
{
    int option;

    while (true)
    {
        PrintMenu();
        option = ReadInteger();
        if (ProcessMenu(option))
        {
            break;
        }
        system("cls");
    }
    return 0;
}

```

The above image is the main function, the while loop has a true condition which means it will loop continuously until broken out of, inside the loop is where the application operates. The menu is printed with the PrintMenu function, and the user then enters their choice of option.

Their option is then fed into the ProcessMenu function which will execute the option then return a bool to an if statement. If the user chooses the quit option, then true will be returned which will break out of the loop, any other option will return false, continuing with the while loop.

Testing

```

#####
##### Maths #####
##### Library #####
#####

1. Find the length of a vector
2. Normalize a vector
3. Distance between two points
4. Cross Product of two vectors
5. Multiply a 4x4 and 4x1 column matrix
6. Mean, Median, Mode and Standard Deviation of a list of integers
7. Union and Intersection of two sets
8. Quit Application

Please select an option >> 8

Are you sure you want to quit? y
Press any key to continue . . .

```

The test above shows the quit option working correctly.

2. Length of a Vector

Brief

Create a function which calculates the length of a vector entered by a user.

Problems Identified

- Allow user to enter numbers for the vector.
- Store values for the vector.
- Implement vector length calculation formula: $(x^2 + y^2 + z^2)^{0.5}$

Solutions Required

- Use function to read in a double variable and check its validity.
- Create Vector class.
- Create Vector object to store values entered by user.
- Use pow() function provided in <conio.h> to calculate the square and square root.

Implementation

```
double ReadDouble()
{
    double number;
    int checkInput;

    checkInput = scanf("%lf", &number);
    IgnoreInput();

    while (checkInput != 1)
    {
        printf("\nInvalid Option, please try again >>");
        checkInput = scanf("%lf", &number);
        IgnoreInput();
    }

    return number;
}

void IgnoreInput()
{
    char ch;
    while ((ch = getchar()) != '\n' && ch != EOF);
}
```

The above function works the same way as the ReadInteger function, but it reads a double instead.

```

class Vector
{
private:    //Attributes
    double _x, _y, _z;
    Point _a = Point();
    Point _b = Point();

public:    //Constructor
    Vector();
    ~Vector();

    //Properties
    double GetX();
    double GetY();
    double GetZ();
    void SetX(double value);
    void SetY(double value);
    void SetZ(double value);
    void SetPointA();
    void SetPointB();

    //Methods
    void SetXYZ(int number);
    double SquareRoot(double number);
    double CalculateLength();
    double NormalizeAxis(double axis);
    void NormalizeVector();
    void ConvertPointsToVector();
};

```

Above is the vector class with all the attributes set to private to keep data encapsulated, and all the methods set to public.

```

void Vector::SetXYZ(int number)
{
    printf("\n\nEnter values of Vector %i\n", number);
    printf("\n\nEnter x component >>");
    _x = ReadDouble();
    printf("\n\nEnter y component >>");
    _y = ReadDouble();
    printf("\n\nEnter z component >>");
    _z = ReadDouble();
}

```

The SetXYZ function reads the users input into the vector objects attributes.

```

double Vector::SquareRoot(double number)
{
    return pow(number, 0.5);
}

double Vector::CalculateLength()
{
    return SquareRoot(pow(_x, 2) + pow(_y, 2) + pow(_z, 2));
}

```

The above functions use the pow function included in the <conio.h> library to perform the index mathematical functions to calculate the length with the $(x^2 + y^2 + z^2)^{0.5}$ formula.

```
void Option1()
{
    Vector vector = Vector();
    system("cls");
    PrintHeader();
    printf("1. Find the length of a vector\n\n");
    vector.SetXYZ(1);
    printf("\nThe Length of vector (%0.2f, %0.2f, %0.2f) is %0.2f.\n\n", vector.GetX(), vector.GetY(), vector.GetZ(), vector.CalculateLength());
    system("pause");
}
```

The function above creates a vector object which the user then inputs the values into, and the values are then used to calculate the length which is then displayed.

Testing

```
#####
##### Maths #####
##### Library #####
#####

1. Find the length of a vector

Enter values of Vector 1

Enter x component >>2

Enter y component >>3

Enter z component >>4

The Length of vector (2.00, 3.00, 4.00) is 5.39.
Press any key to continue . . .
```

The above image shows the function working correctly.

3. Normalize

Brief

Create an option which will normalise a vector entered by the user.

Problems Identified

- Allow user to enter values for vector.
- Calculate length of vector.
- Divide each axis by length to calculate normalised vector.
- Display normalised vector to screen.

Solutions Required

- Use functions from before to allow user to enter values.
- Use method from before to calculate length.
- Create method to divide each axis by length.
- Create function to display results.

Implementation

```
double Vector::NormalizeAxis(double axis)
{
    return (axis) / CalculateLength();
}
```

```
void Vector::NormalizeVector()
{
    printf("The Normalized vector is (%0.2f, %0.2f, %0.2f).\r\n", NormalizeAxis(_x), NormalizeAxis(_y), NormalizeAxis(_z));
}
```

```
void Option2()
{
    Vector vector = Vector();
    system("cls");
    PrintHeader();
    printf("2. Normalize a vector\r\n");
    vector.SetXYZ(1);
    vector.NormalizeVector();
    system("pause");
}
```

The images above show how the length is used to calculate the normalised vector and display it to the screen.

Testing

```
#####  
##### Maths #####  
##### Library #####  
#####  
  
2. Normalize a vector  
  
Enter values of Vector 1  
  
Enter x component >>2  
  
Enter y component >>3  
  
Enter z component >>4  
The Normalized vector is (0.37, 0.56, 0.74).  
Press any key to continue . . .
```

The test above shows the code working correctly.

4. Distance between two Points

Brief

Allow user to enter two points which then converts those two points into a vector which can then be used to calculate the length of the vector which is also the distance between the two points.

Problems Identified

- Read values from user for both points.
- Convert points into a vector.
- Use vector length calculator.
- Display results.

Solutions Required

- Use read double function from before.
- Create method to convert two points to a vector.
- Use methods from before to calculate length and display results.

Implementation

```
class Vector
{
private:    //Attributes
    double _x, _y, _z;
    Point _a = Point();
    Point _b = Point();

public:    //Constructor
    Vector();
    ~Vector();

    //Properties
    double GetX();
    double GetY();
    double GetZ();
    void SetX(double value);
    void SetY(double value);
    void SetZ(double value);
    void SetPointA();
    void SetPointB();

    //Methods
    void SetXYZ(int number);
    double SquareRoot(double number);
    double CalculateLength();
    double NormalizeAxis(double axis);
    void NormalizeVector();
    void ConvertPointsToVector();
};
```

```

class Point
{
private:
    double _x, _y, _z;
public:
    Point();
    ~Point();
    double GetX();
    double GetY();
    double GetZ();
    void SetXYZ(int number);
};

```

The Vector class contains two Point objects which are defined by the point class above.

```

void Point::SetXYZ(int number)
{
    printf("\r\nEnter Co-ordinates of Point %i\r\n", number);
    printf("\r\nEnter x Co-ordinate >>");
    _x = ReadDouble();
    printf("\r\nEnter y Co-ordinate >>");
    _y = ReadDouble();
    printf("\r\nEnter z Co-ordinate >>");
    _z = ReadDouble();
}

```

The method above asks the user to enter values for the point.

```

//Properties
double Point::GetX()
{
    return _x;
}
double Point::GetY()
{
    return _y;
}
double Point::GetZ()
{
    return _z;
}

```

```

void Vector::ConvertPointsToVector()
{
    _x = _a.GetX() - _b.GetX();
    _y = _a.GetY() - _b.GetY();
    _z = _a.GetZ() - _b.GetZ();
}

```

The methods above show how the values in the Point objects are accessed and used to create the values for the Vector object.

The answer is then calculated using the CalculateLength method in the Vector object.

Testing

```
#####  
##### Maths #####  
##### Library #####  
#####  
  
3. Distance between two points  
  
Enter Co-ordinates of Point 1  
  
Enter x Co-ordinate >>1  
  
Enter y Co-ordinate >>2  
  
Enter z Co-ordinate >>3  
  
Enter Co-ordinates of Point 2  
  
Enter x Co-ordinate >>4  
  
Enter y Co-ordinate >>5  
  
Enter z Co-ordinate >>6  
  
The distance between the two points is 5.20  
Press any key to continue . . .
```

The above test shows the code working correctly.

5. Cross Product and Dot Product of two vectors

Brief

Allow user to enter values for two vectors which are then used to calculate the cross product and displaying the results.

Problems Identified

- Read double from user
- Save values of two vectors
- Use values in both vectors to create values for a new vector.
- Use values in both vectors to calculate dot product
- Display values for new vector and dot product.

Solutions Required

- Use read double function from before.
- Create three vector objects
- Create function which returns a vector.
- Use $(i(a_2*b_3 - b_2*a_3) - j(a_1*b_3 - a_3*b_1) + k(a_1*b_2 - a_2*b_1))$ formula to calculate cross product.
- Use $(a_1 * b_1 + a_2 * b_2 + a_3 * b_3)$ to calculate dot product.
- Create function to display results.

Implementation

```
Vector CrossProduct(Vector* vector1, Vector* vector2)
{
    Vector crossProduct = Vector();
    crossProduct.SetX((vector1->GetY()*vector2->GetZ()) - (vector1->GetZ()*vector2->GetY()));
    crossProduct.SetY((vector1->GetX()*vector2->GetZ()) - (vector1->GetZ()*vector2->GetX()));
    crossProduct.SetZ((vector1->GetX()*vector2->GetY()) - (vector1->GetY()*vector2->GetX()));
    return crossProduct;
}
```

The function above takes in 2 vector objects to perform the $(i(a_2*b_3 - b_2*a_3) - j(a_1*b_3 - a_3*b_1) + k(a_1*b_2 - a_2*b_1))$ formula to calculate cross product which is then returned into a Vector object.

```
double DotProduct(Vector* vector1, Vector* vector2)
{
    return (vector1->GetX() * vector2->GetX()) + (vector1->GetY() * vector2->GetY()) + (vector1->GetZ() * vector2->GetZ());
}
```

The function above uses the $(a_1 * b_1 + a_2 * b_2 + a_3 * b_3)$ to calculate dot product.

```

void Option4()
{
    Vector vector1 = Vector();
    Vector vector2 = Vector();
    Vector crossProduct = Vector();
    system("cls");
    PrintHeader();
    printf("4. Cross Product and Dot Product of two vectors\n\n");
    vector1.SetXYZ(1);
    vector2.SetXYZ(2);
    crossProduct = CrossProduct(&vector1, &vector2);
    printf("The Cross product of the vectors");
    printf("(%0.2f, %0.2f, %0.2f) and", vector1.GetX(), vector1.GetY(), vector1.GetZ());
    printf("(%0.2f, %0.2f, %0.2f) is:", vector2.GetX(), vector2.GetY(), vector2.GetZ());
    printf("\n\n(%0.2f, %0.2f, %0.2f)", crossProduct.GetX(), crossProduct.GetY(), crossProduct.GetZ());
    printf("\n\nThe Dot product is %0.2f", DotProduct(&vector1, &vector2));
    system("pause");
}

```

The above function creates 3 vector objects which the user enters values for 2 of the vectors, the CrossProduct function then returns the calculated values into the vector object, the values for the vectors and the dot product are then printed.

Testing

```

#####
##### Maths #####
##### Library #####
#####

4. Cross Product and Dot Product of two vectors

Enter values of Vector 1

Enter x component >>1

Enter y component >>2

Enter z component >>3

Enter values of Vector 2

Enter x component >>4

Enter y component >>5

Enter z component >>6
The Cross product of the vectors(1.00, 2.00, 3.00) and(4.00, 5.00, 6.00) is:
(-3.00, -6.00, -3.00)
The Dot product is 32.00
Press any key to continue . . .

```

The above test shows the code working correctly.

6. Multiply a (4x4) and (4x1) Matrix

Brief

Create a function that allows the user to enter values for a 4x4 matrix and a 4x1 column matrix, and then display the resulting 4x1 column matrix when both are multiplied together.

Problems Identified

- Read values from user.
- Store values in a matrix.
- Make sure no unexpected values are in the array to start with.
- Print out matrix as user is entering values
- Multiply matrices together.
- Display results.

Solutions Required

- Use read number function from before
- Create an array with 16 addresses for the 4x4 matrix
- Create 2 arrays with 4 addresses, one for the initial 4x1, the other for the resulting matrix.
- Set all the initial values in the arrays to 0.
- Use switch function to print out updating versions of the matrices.
- Create function to multiply both matrices and store values in result matrix.
- Create function to display result matrix.

Implementation

```
int squareMatrix[16], colMatrix[4], resultMatrix[4];
```

```
void InitialiseArray(int array[], int size)
{
    for (int col = 0; col < size; col++)
    {
        array[col] = 0;
    }
}
```

The above images show the 3 arrays being created and showing the function which will set the initial values to 0 with the use of a for loop.

```

void PrintColumnMatrix(int index, int matrix[])
{
    switch (index)
    {
        case 0:
            printf("A\r\nB\r\nC\r\nD\r\n");
            break;
        case 1:
            printf("%i\r\nB\r\nC\r\nD\r\n", matrix[0]);
            break;
        case 2:
            printf("%i\r\n%i\r\nC\r\nD\r\n", matrix[0], matrix[1]);
            break;
        case 3:
            printf("%i\r\n%i\r\n%i\r\nD\r\n", matrix[0], matrix[1], matrix[2]);
            break;
        case 4:
            printf("%i\r\n%i\r\n%i\r\n%i\r\n", matrix[0], matrix[1], matrix[2], matrix[3]);
            break;
    }
}

```

The above function uses a switch statement which will be used in a for loop, depending on the loop iteration will depend on what is displayed. This was done to show an updating display as the user entered values into the 4x1 column matrix. There is another function like this for the 4x4 matrix which has 16 cases but it was too large to capture in one image.

```

void MultiplyMatrices(int arrayx[], int arrayy[], int result[])
{
    for (int index = 0; index < 4; index++)
    {
        result[index] = (arrayx[0+(4*index)]*arrayy[0]) + (arrayx[1+(4 * index)] * arrayy[1]) + (arrayx[2+(4 * index)] * arrayy[2]) + (arrayx[3+(4 * index)] * arrayy[3]);
    }
}

```

The above function multiplies the rows of the 4x4 matrix with the column matrix to calculate the values for the resulting matrix.

```

void Options5()
{
    int squareMatrix[16], colMatrix[4], resultMatrix[4];

    InitialiseArray(squareMatrix, 16);
    InitialiseArray(colMatrix, 4);

    for (int index = 0; index < 16; index++)
    {
        system("cls");
        PrintHeader();
        printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
        PrintSquareMatrix(index, squareMatrix);
        printf("Enter Values for Square Matrix >> ");
        squareMatrix[index] = ReadInteger();
    }

    for (int index = 0; index < 4; index++)
    {
        system("cls");
        PrintHeader();
        printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
        PrintColumnMatrix(index, colMatrix);
        printf("Enter Values for Column Matrix >> ");
        colMatrix[index] = ReadInteger();
    }

    MultiplyMatrices(squareMatrix, colMatrix, resultMatrix);
    system("cls");
    PrintHeader();
    printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
    PrintSquareMatrix(16, squareMatrix);
    printf("\r\n");
    PrintColumnMatrix(4, colMatrix);
    printf("\r\nThe resulting matrix is:\r\n");
    PrintColumnMatrix(4, resultMatrix);

    system("pause");
}

```

The above function contains all the code to execute the option. The arrays are first declared and initialised, then the user enters the values for the 4x4 matrix and the 4x1 column matrix. The values are then passed to the multiply function and then displayed on screen.

Testing

```

#####
##### Maths #####
##### Library #####
#####

5. Multiply a 4x4 and 4x1 column matrix
A B C D
E F G H
I J K L
M N O P
Enter Values for Square Matrix >>

```

This shows the initial screen when the user selects the option.

```
#####  
##### Maths #####  
##### Library #####  
#####  
5. Multiply a 4x4 and 4x1 column matrix  
1 2 3 4  
5 6 7 8  
9 0 9 8  
7 6 5 P  
Enter Values for Square Matrix >> 4
```

This is what the screen looks like when the user is about to enter the last value for the 4x4 matrix.

```
#####  
##### Maths #####  
##### Library #####  
#####  
5. Multiply a 4x4 and 4x1 column matrix  
A  
B  
C  
D  
Enter Values for Column Matrix >>
```

The user is then presented with the 4x1 matrix entry.

```
#####  
##### Maths #####  
##### Library #####  
#####  
  
5. Multiply a 4x4 and 4x1 column matrix  
1 2 3 4  
5 6 7 8  
9 0 9 8  
7 6 5 4  
  
5  
6  
7  
8  
  
The resulting matrix is:  
70  
174  
172  
138  
Press any key to continue . . .
```

The above image shows the code working correctly after all the calculations have been performed

7. Mean, Median, Mode and Standard Deviation

Brief

Create a set of functions that allow the user to enter a list of numbers and then display the mean/average, median, mode and standard deviation of that list of numbers.

Problems Identified

- Allow user to choose the size of the list
- Allow the user to enter the values into the list.
- Calculate the mean/average
- Calculate the median
- Calculate the mode
- Calculate the standard deviation.
- Display the list sorted in ascending order.
- Display the results calculated.

Solutions Required

- Read integers from user using functions from before.
- Create a dynamic array which can hold the number of values the user enters.
- Use loop to read multiple values from user.
- Create function to add all values together and then divide the product by the number of values to find the average.
- Create function which looks at the size of the list to determine if odd or even. If odd then return the middle number, if even then return the average of the two middle numbers to find the median.
- Create function to check through the list of numbers to find the most popular.
- Create function to find Standard deviation with use of formula.
- Display the results returned.

Implementation

```
void Option6()
{
    int maxNumbers, mode;
    int* listOfIntegers;
    double mean, median, stdDev;

    system("cls");
    PrintHeader();
    printf("6. Mean, Median, Mode and Standard Deviation of a list of integers\r\nEnter size of list: ");
    maxNumbers = ReadInteger();
    listOfIntegers = new int[maxNumbers];
    SetArray(listOfIntegers, maxNumbers);
    BubbleSort(listOfIntegers, maxNumbers);
    PrintList(listOfIntegers, maxNumbers, "list");
    mean = CalculateMean(listOfIntegers, maxNumbers);
    median = CalculateMedian(listOfIntegers, maxNumbers);
    mode = CalculateMode(listOfIntegers, maxNumbers);
    stdDev = CalculateStdDev(listOfIntegers, maxNumbers, mean);

    delete[] listOfIntegers;
    system("pause");
}
```

The above function creates the variables required for the function, as the user is able to enter the size of the list, a dynamic array needs to be used, hence the need for an int pointer.

```
void SetArray(int list[], int listSize)
{
    printf("\r\nEnter the list\r\n");
    for (int i = 0; i < listSize; i++)
    {
        printf("%i >> ", i + 1);
        list[i] = ReadInteger();
    }
}
```

The above function prompts the user to enter the values into the array.

```

void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

void BubbleSort(int list[], int size)
{
    bool swapped;
    for (int i = 0; i < size - 1; i++)
    {
        swapped = false;
        for (int j = 0; j < size - i - 1; j++)
        {
            if (list[j] > list[j + 1])
            {
                swap(&list[j], &list[j + 1]);
                swapped = true;
            }
        }

        if (swapped == false)
        {
            break;
        }
    }
}

```

Once the user enters the values, the array is then passed into the above function which then sorts the values in ascending order.

```

void PrintList(int list[], int listSize, string object)
{
    printf("\n\nThe sorted %s is: ", object.c_str());
    for (int i = 0; i < listSize; i++)
    {
        printf("%i ", list[i]);
    }
}

```

The list of values is then printed to the screen after being sorted.

```

double CalculateMean(int list[], int listSize)
{
    double total = 0;

    for (int i = 0; i < listSize; i++)
    {
        total = total + list[i];
    }

    printf("\n\nThe Mean is %.2f\n", total / listSize);
    return total / listSize;
}

```

To calculate the mean, the list of values are added together using a for loop, then divided by the size of the list.

```
double CalculateMedian(int list[], int listSize)
{
    double median;
    if (listSize % 2 == 0)
    {
        median = (list[(listSize / 2) - 1] + list[(listSize / 2)]) / 2;
    }
    else
    {
        median = list[((listSize + 1) / 2) - 1];
    }
    printf("\n\nThe Median is %.2f\n\n", median);
    return median;
}
```

To calculate the median, the list size is checked to see if it is odd or even. If it is odd then the median will be a single value, however If the list size is even, then there will be no single middle number, so the 2 middle numbers need to be averaged to find the median.

```
int CalculateMode(int list[], int listSize)
{
    int number = list[0];
    int mode = number;
    int count = 1;
    int modeCount = 1;

    for (int i = 1; i < listSize; i++)
    {
        if (list[i] == number)
        {
            ++count;
        }
        else
        {
            if (count > modeCount)
            {
                modeCount = count;
                mode = number;
            }
            count = 1;
            number = list[i];
        }
    }
    printf("\n\nThe Mode is %i\n\n", mode);
    return mode;
}
```

To find the mode, the first number in the list is recorded and set as the mode by default. The current count and mode count are then set to one, this is so we can compare them to check for a new mode.

The for loop checks if the next number is equal to the current number, if it is then the current count is incremented, if it is not then it checks if the current count is greater than the mode count, if it is then the mode count is set to the current count because a

new mode has been found, If not then the mode count stays the same, and the current count is reset and the number is set to the next value in the list to check if that is the new mode.

The main problem with this function is it will not display multiple modes, for example in the list 1,4,4,5,7,7, the modes are 4 and 7, but only the first mode found will be displayed as 4 with this function.

```
double CalculateStdDev(int list[], int listSize, double mean)
{
    //      --- N      /      - \ 2      -
    //      | \      | X - X |      X = Mean
    // SD = | /      \ i      /      N = Size
    //      | --- i = 1
    //      \ |      N - 1
    //      \ |

    double stdDev;
    double numerator = 0;
    int denominator = listSize - 1;

    for (int i = 0; i < listSize; i++)
    {
        numerator = numerator + pow(list[i] - mean, 2);
    }
    stdDev = pow(numerator / denominator, 0.5);
    printf("\n\nThe Standard Deviation is %0.2f\n\n", stdDev);
    return stdDev;
}
```

The above function calculated the standard deviation by finding the product of the mean being subtracted from the list elements squared, which is then divided by the size of the list minus 1 all square-rooted. This is separated by calculating the numerator and the denominator separately and passing them into a pow function.

```
#####  
##### Maths #####  
##### Library #####  
#####  
  
6. Mean, Median, Mode and Standard Deviation of a list of integers  
Enter size of list: 5  
  
Enter the list  
1 >> 2  
2 >> 4  
3 >> 6  
4 >> 8  
5 >> 9  
  
The sorted list is: 2 4 6 8 9  
  
The Mean is 5.80  
  
The Median is 6.00  
  
The Mode is 2  
  
The Standard Deviation is 2.86  
Press any key to continue . . .
```

```
#####  
##### Maths #####  
##### Library #####  
#####  
  
6. Mean, Median, Mode and Standard Deviation of a list of integers  
Enter size of list: 5  
  
Enter the list  
1 >> 2  
2 >> 6  
3 >> 3  
4 >> 8  
5 >> 5  
  
The sorted list is: 2 3 5 6 8  
  
The Mean is 4.80  
  
The Median is 5.00  
  
The Mode is 8  
  
The Standard Deviation is 2.39  
Press any key to continue . . .
```

The above tests show the code working correctly with the bubble sort being tested in the second image.

8. Union and Intersection

Brief

Create function which allows user to enter values into two sets, the union and the intersection of both sets is then calculated and displayed.

Problems Identified

- Allow user to choose size of both sets.
- Allow user to enter values into both sets.
- Compare sets to find the union
- Compare sets to find the intersection
- Sort sets
- Display sets

Solutions Required

- Use functions from before to make dynamic arrays.
- Use functions from before to read values from user into the arrays
- Create function to find size of the union set
- Create dynamic array for the union with the found size
- Create function to put all necessary values from both sets into the union array
- Create function to find size of intersection set
- Create dynamic array for the intersection with the found size
- Create function to put all necessary values from both sets into the union
- Use if statement to check if the sets intersect, if they don't then print out message warning there is no intersection.
- Use function from before to sort the list in ascending order.
- Create function to display results.

Implementation

```
int setASize, setBSize, unionSize, intersectionSize;
int* setA;
int* setB;
int* unionOfSets;
int* intersectionOfSets;
```

The above image shows the variables required for the function to work.

```

system("cls");
PrintHeader();

printf("7. Union and Intersection of two sets\nEnter size of Set A: ");
setASize = ReadInteger();
setA = new int[setASize];
SetArray(setA, setASize);

printf("\nEnter size of Set B: ");
setBSize = ReadInteger();
setB = new int[setBSize];
SetArray(setB, setBSize);

BubbleSort(setA, setASize);
BubbleSort(setB, setBSize);
PrintList(setA, setASize, "set A");
PrintList(setB, setBSize, "set B");

```

Much like the previous function, the user enters how large they want the array to be, then enters the values for the list. The list is then sorted and displayed.

```

unionSize = FindUnionSize(setA, setB, setASize, setBSize);
unionOfSets = new int[unionSize];
SetUnion(unionOfSets, setA, setB, unionSize, setASize, setBSize);
BubbleSort(unionOfSets, unionSize);
PrintList(unionOfSets, unionSize, "Union");

```

To find the Union, a function was required to first find the size of the union. Once found the dynamic array could be created, and the values could be passed in.

```

int FindUnionSize(int setA[], int setB[], int setASize, int setBSize)
{
    int unionSize;
    int count = 0;
    bool matchFound;

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == false)
        {
            count++;
        }
    }
    unionSize = setBSize + count;
    return unionSize;
}

```

The above function checks if each element of set A is in set B, if an element of set A is not in set B, then the count will be incremented. This is because for each element

of A that is not in B, the union will be that much larger than B. for example, say B has 5 elements, and A has 3 elements that are not in B, the union will be $5 + 3 = 8$.

```
void SetUnion(int setUnion[], int setA[], int setB[], int unionSize, int setASize, int setBSize)
{
    int count = 0;
    bool matchFound;

    for (int i = 0; i < setBSize; i++)
    {
        setUnion[i] = setB[i];
    }

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == false)
        {
            setUnion[setBSize + count] = setA[i];
            count++;
        }
    }
}
```

Once the union size has been found and the union array has been created, the values can be passed in. First all the values of B are passed into the union as the foundation, then much like the previous function, all the elements of A are compared with B, when a non-match is found, the number is placed into the next address of the union.

```
intersectionSize = FindIntersectionSize(setA, setB, setASize, setBSize);
if (intersectionSize > 0)
{
    intersectionOfSets = new int[intersectionSize];
    SetIntersection(intersectionOfSets, setA, setB, intersectionSize, setASize, setBSize);
    BubbleSort(intersectionOfSets, intersectionSize);
    PrintList(intersectionOfSets, intersectionSize, "Intersection");

    delete[] intersectionOfSets;
}
else
{
    printf("\r\nThere is no Intersection");
}
```

To find the intersection, much like the union, the size needs to be determined first to create the dynamic array. If the size returned is 0, then the array cannot be created, so a check is implemented before the creation with an if statement shown above.

```

int FindIntersectionSize(int setA[], int setB[], int setASize, int setBSize)
{
    int intersectionSize = 0;
    bool matchFound;

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == true)
        {
            intersectionSize++;
        }
    }
    return intersectionSize;
}

```

To find the intersection size, the same logic is used from finding the union except with a reverse of logic. The intersection is all the numbers that are shared between each set, so when a match is found, the size is increased.

```

void SetIntersection(int setIntersection[], int setA[], int setB[], int intersectionSize, int setASize, int setBSize)
{
    int count = 0;
    bool matchFound;

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == true)
        {
            setIntersection[count] = setA[i];
            count++;
        }
    }
}

```

Once the size has been found and determined to be greater than 0, the function above is executed which checks through the sets for shared numbers, and when one is found, it is placed into the intersection set.

```

printf("\r\n");
delete[] setA;
delete[] setB;
delete[] unionOfSets;

system("pause");

```

Before the option functions complete, the dynamic arrays are deleted to prevent memory leak.

Testing

```

#####
##### Maths #####
##### Library #####
#####

7. Union and Intersection of two sets
Enter size of Set A: 5

Enter the list
1 >> 1
2 >> 2
3 >> 4
4 >> 5
5 >> 3

Enter size of Set B: 5

Enter the list
1 >> 3
2 >> 6
3 >> 4
4 >> 7
5 >> 5

The sorted set A is: 1 2 3 4 5
The sorted set B is: 3 4 5 6 7
The sorted Union is: 1 2 3 4 5 6 7
The sorted Intersection is: 3 4 5
Press any key to continue . . .

```

The above test shows the code working when there is an intersection.

```
#####  
##### Maths #####  
##### Library #####  
#####  
  
7. Union and Intersection of two sets  
Enter size of Set A: 5  
  
Enter the list  
1 >> 1  
2 >> 2  
3 >> 3  
4 >> 4  
5 >> 5  
  
Enter size of Set B: 5  
  
Enter the list  
1 >> 6  
2 >> 7  
3 >> 8  
4 >> 9  
5 >> 10  
  
The sorted set A is: 1 2 3 4 5  
The sorted set B is: 6 7 8 9 10  
The sorted Union is: 1 2 3 4 5 6 7 8 9 10  
There is no Intersection 10  
Press any key to continue . . .
```

The above test shows the code works when there is no intersection.

Appendix

Source.cpp

```
#include "stdafx.h"
#include "Functions.h"

int main()
{
    int option;

    while (true)
    {
        PrintMenu();
        option = ReadInteger();
        if (ProcessMenu(option))
        {
            break;
        }
        system("cls");
    }
    return 0;
}
```

Stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// Maths Library.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information
```

```
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable:4996)
```

```
#include "stdafx.h"
#include <conio.h>
#include <string>
#include <stdarg.h>
#include <Windows.h>
#include <list>
```

```
using namespace std;
```

```
// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

Functions.cpp

```
#include "stdafx.h"
#include "Functions.h"
#include "Vector.h"
#include <list>

void PrintHeader()
{
    printf("#####\r\n");
    printf("##### Maths #####\r\n");
    printf("##### Library #####\r\n");
    printf("#####\r\n\r\n");
}

void PrintMenu()
{
    PrintHeader();
    printf("1. Find the length of a vector\r\n");
    printf("2. Normalize a vector\r\n");
    printf("3. Distance between two points\r\n");
    printf("4. Cross Product and Dot Product of two vectors\r\n");
    printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
    printf("6. Mean, Median, Mode and Standard Deviation of a list of\n");
    printf("integers\r\n");
    printf("7. Union and Intersection of two sets\r\n");
    printf("8. Quit Application\r\n");
    printf("\r\nPlease select an option >> ");
}

int ReadInteger()
{
    int number;
    int checkInput;

    checkInput = scanf("%i", &number);
    IgnoreInput();

    while (checkInput != 1)
    {
        printf("\r\nInvalid Option, please try again >>");
        checkInput = scanf("%i", &number);
        IgnoreInput();
    }

    return number;
}

double ReadDouble()
{
    double number;
    int checkInput;

    checkInput = scanf("%lf", &number);
    IgnoreInput();

    while (checkInput != 1)
    {
        printf("\r\nInvalid Option, please try again >>");
        checkInput = scanf("%lf", &number);
        IgnoreInput();
    }

    return number;
}
```

```
}

void IgnoreInput()
{
    char ch;
    while ((ch = getchar()) != '\n' && ch != EOF);
}

bool IsQuitting()
{
    string answer;
    char text[100];
    printf("\r\nAre you sure you want to quit? ");
    scanf("%s", text);
    answer = text;

    if (answer == "YES" || answer == "Yes" || answer == "yes" || answer == "Y" ||
answer == "y")
    {
        return true;
    }
    return false;
}

bool ProcessMenu(int option)
{
    switch (option)
    {
    case 1:
        Option1();
        break;
    case 2:
        Option2();
        break;
    case 3:
        Option3();
        break;
    case 4:
        Option4();
        break;
    case 5:
        Option5();
        break;
    case 6:
        Option6();
        break;
    case 7:
        Option7();
        break;
    case 8:
        if (IsQuitting())
        {
            return true;
        }
        break;
    default:
        printf("Invalid Option\r\n");
        break;
    }
    return false;
}
```

```

double DotProduct(Vector* vector1, Vector* vector2)
{
    return (vector1->GetX() * vector2->GetX()) + (vector1->GetY() * vector2->GetY()) + (vector1->GetZ() * vector2->GetZ());
}

Vector CrossProduct(Vector* vector1, Vector* vector2)
{
    Vector crossProduct = Vector();
    crossProduct.SetX((vector1->GetY()*vector2->GetZ()) - (vector1->GetZ()*vector2->GetY()));
    crossProduct.SetY((vector1->GetX()*vector2->GetZ()) - (vector1->GetZ()*vector2->GetX()));
    crossProduct.SetZ((vector1->GetX()*vector2->GetY()) - (vector1->GetY()*vector2->GetX()));
    return crossProduct;
}

void InitialiseArray(int array[], int size)
{
    for (int col = 0; col < size; col++)
    {
        array[col] = 0;
    }
}

void PrintColumnMatrix(int index, int matrix[])
{
    switch (index)
    {
        case 0:
            printf("A\r\nB\r\nC\r\nD\r\n");
            break;
        case 1:
            printf("%i\r\nB\r\nC\r\nD\r\n", matrix[0]);
            break;
        case 2:
            printf("%i\r\n%i\r\nC\r\nD\r\n", matrix[0], matrix[1]);
            break;
        case 3:
            printf("%i\r\n%i\r\n%i\r\nD\r\n", matrix[0], matrix[1], matrix[2]);
            break;
        case 4:
            printf("%i\r\n%i\r\n%i\r\n%i\r\n", matrix[0], matrix[1], matrix[2],
matrix[3]);
            break;
    }
}

void PrintSquareMatrix(int index, int matrix[])
{
    switch (index)
    {
        case 0:
            printf("A B C D\r\nE F G H\r\nI J K L\r\nM N O P\r\n");
            break;
        case 1:
            printf("%i B C D\r\nE F G H\r\nI J K L\r\nM N O P\r\n",
matrix[0]);
            break;
        case 2:
            printf("%i %i C D\r\nE F G H\r\nI J K L\r\nM N O P\r\n",
matrix[0], matrix[1]);
            break;
    }
}

```

```

    case 3:
        printf("%i %i %i D\r\nE F G H\r\nI J K L\r\nM N O P\r\n",
matrix[0], matrix[1], matrix[2]);
        break;
    case 4:
        printf("%i %i %i %i\r\nE F G H\r\nI J K L\r\nM N O P\r\n",
matrix[0], matrix[1], matrix[2], matrix[3]);
        break;
    case 5:
        printf("%i %i %i %i\r\n%i F G H\r\nI J K L\r\nM N O P\r\n",
matrix[0], matrix[1], matrix[2], matrix[3], matrix[4]);
        break;
    case 6:
        printf("%i %i %i %i\r\n%i %i G H\r\nI J K L\r\nM N O P\r\n",
matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5]);
        break;
    case 7:
        printf("%i %i %i %i\r\n%i %i %i H\r\nI J K L\r\nM N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6]);
        break;
    case 8:
        printf("%i %i %i %i\r\n%i %i %i %i\r\nI J K L\r\nM N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7]);
        break;
    case 9:
        printf("%i %i %i %i\r\n%i %i %i %i\r\n%i J K L\r\nM N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8]);
        break;
    case 10:
        printf("%i %i %i %i\r\n%i %i %i %i\r\n%i %i K L\r\nM N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9]);
        break;
    case 11:
        printf("%i %i %i %i\r\n%i %i %i %i\r\n%i %i %i L\r\nM N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9], matrix[10]);
        break;
    case 12:
        printf("%i %i %i %i\r\n%i %i %i %i\r\n%i %i %i %i\r\nM N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9], matrix[10], matrix[11]);
        break;
    case 13:
        printf("%i %i %i %i\r\n%i %i %i %i\r\n%i %i %i %i\r\n%i N O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9], matrix[10], matrix[11], matrix[12]);
        break;
    case 14:
        printf("%i %i %i %i\r\n%i %i %i %i\r\n%i %i %i %i\r\n%i %i O
P\r\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9], matrix[10], matrix[11], matrix[12], matrix[13]);
        break;

```

```

        case 15:
            printf("%i %i %i %i\n%i %i %i %i\n%i %i %i %i\n%i %i %i %i\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9], matrix[10], matrix[11], matrix[12], matrix[13],
matrix[14]);

            break;

        case 16:
            printf("%i %i %i %i\n%i %i %i %i\n%i %i %i %i\n%i %i %i %i\n", matrix[0], matrix[1], matrix[2], matrix[3], matrix[4], matrix[5], matrix[6],
matrix[7], matrix[8], matrix[9], matrix[10], matrix[11], matrix[12], matrix[13],
matrix[14], matrix[15]);

            break;
    }
}

void MultiplyMatrices(int array4x4[], int array4x1[], int result[])
{
    for (int index = 0; index < 4; index++)
    {
        result[index] = (array4x4[0+(4*index)]*array4x1[0]) + (array4x4[1+(4 *
index)] * array4x1[1]) + (array4x4[2+(4 * index)] * array4x1[2]) + (array4x4[3+(4 *
index)] * array4x1[3]);
    }
}

void SetArray(int list[], int listSize)
{
    printf("\nEnter the list\n");
    for (int i = 0; i < listSize; i++)
    {
        printf("%i >> ", i + 1);
        list[i] = ReadInteger();
    }
}

void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

void BubbleSort(int list[], int size)
{
    bool swapped;
    for (int i = 0; i < size - 1; i++)
    {
        swapped = false;
        for (int j = 0; j < size - i - 1; j++)
        {
            if (list[j] > list[j + 1])
            {
                swap(&list[j], &list[j + 1]);
                swapped = true;
            }
        }

        if (swapped == false)
        {
            break;
        }
    }
}

```

```
        }
    }
}

void PrintList(int list[], int listSize, string object)
{
    printf("\r\nThe sorted %s is: ", object.c_str());
    for (int i = 0; i < listSize; i++)
    {
        printf("%i ", list[i]);
    }
}

double CalculateMean(int list[], int listSize)
{
    double total = 0;

    for (int i = 0; i < listSize; i++)
    {
        total = total + list[i];
    }
    printf("\r\n\r\nThe Mean is %0.2f\r\n", total / listSize);
    return total / listSize;
}

double CalculateMedian(int list[], int listSize)
{
    double median;
    if (listSize % 2 == 0)
    {
        median = (list[(listSize / 2) - 1] + list[(listSize / 2)]) / 2;
    }
    else
    {
        median = list[((listSize + 1) / 2) - 1];
    }
    printf("\r\nThe Median is %0.2f\r\n", median);
    return median;
}

int CalculateMode(int list[], int listSize)
{
    int number = list[0];
    int mode = number;
    int count = 1;
    int modeCount = 1;

    for (int i = 1; i < listSize; i++)
    {
        if (list[i] == number)
        {
            ++count;
        }
        else
        {
            if (count > modeCount)
            {
                modeCount = count;
                mode = number;
            }
            count = 1;
            number = list[i];
        }
    }
}
```

```

    }
}
printf("\r\nThe Mode is %i\r\n", mode);
return mode;
}

double CalculateStdDev(int list[], int listSize, double mean)
{
    //
    //      --- N      /      \ 2
    //      \      | X - X |      X = Mean
    // SD =  /      \ i      /      N = Size
    //      --- i = 1
    //      \      |
    //      \      | N - 1
    //
    double stdDev;
    double numerator = 0;
    int denominator = listSize - 1;

    for (int i = 0; i < listSize; i++)
    {
        numerator = numerator + pow(list[i] - mean, 2);
    }
    stdDev = pow(numerator / denominator, 0.5);
    printf("\r\nThe Standard Deviation is %0.2f\r\n", stdDev);
    return stdDev;
}

int FindUnionSize(int setA[], int setB[], int setASize, int setBSize)
{
    int unionSize;
    int count = 0;
    bool matchFound;

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == false)
        {
            count++;
        }
    }
    unionSize = setBSize + count;
    return unionSize;
}

void SetUnion(int setUnion[], int setA[], int setB[], int unionSize, int setASize, int setBSize)
{
    int count = 0;
    bool matchFound;

    for (int i = 0; i < setBSize; i++)
    {
        setUnion[i] = setB[i];
    }

```

```
    }

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == false)
        {
            setUnion[setBSize + count] = setA[i];
            count++;
        }
    }
}

int FindIntersectionSize(int setA[], int setB[], int setASize, int setBSize)
{
    int intersectionSize = 0;
    bool matchFound;

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == true)
        {
            intersectionSize++;
        }
    }
    return intersectionSize;
}

void SetIntersection(int setIntersection[], int setA[], int setB[], int
intersectionSize, int setASize, int setBSize)
{
    int count = 0;
    bool matchFound;

    for (int i = 0; i < setASize; i++)
    {
        matchFound = false;
        for (int j = 0; j < setBSize; j++)
        {
            if (setA[i] == setB[j])
            {
                matchFound = true;
            }
        }
        if (matchFound == true)
        {
            setIntersection[count] = setA[i];
            count++;
        }
    }
}
```

```
        }
    }
}

void Option1()
{
    Vector vector = Vector();
    system("cls");
    PrintHeader();
    printf("1. Find the length of a vector\n");
    vector.SetXYZ(1);
    printf("\n\nThe Length of vector (%0.2f, %0.2f, %0.2f) is %0.2f.\n",
vector.GetX(), vector.GetY(), vector.GetZ(), vector.CalculateLength());
    system("pause");
}

void Option2()
{
    Vector vector = Vector();
    system("cls");
    PrintHeader();
    printf("2. Normalize a vector\n");
    vector.SetXYZ(1);
    vector.NormalizeVector();
    system("pause");
}

void Option3()
{
    Vector vector = Vector();
    system("cls");
    PrintHeader();
    printf("3. Distance between two points\n");
    vector.SetPointA();
    vector.SetPointB();
    vector.ConvertPointsToVector();
    printf("\n\nThe distance between the two points is %0.2f\n",
vector.CalculateLength());
    system("pause");
}

void Option4()
{
    Vector vector1 = Vector();
    Vector vector2 = Vector();
    Vector crossProduct = Vector();
    system("cls");
    PrintHeader();
    printf("4. Cross Product and Dot Product of two vectors\n");
    vector1.SetXYZ(1);
    vector2.SetXYZ(2);
    crossProduct = CrossProduct(&vector1, &vector2);
    printf("The Cross product of the vectors");
    printf("(%0.2f, %0.2f, %0.2f) and", vector1.GetX(), vector1.GetY(),
vector1.GetZ());
    printf("(%0.2f, %0.2f, %0.2f) is:", vector2.GetX(), vector2.GetY(),
vector2.GetZ());
    printf("\n\n(%0.2f, %0.2f, %0.2f)", crossProduct.GetX(), crossProduct.GetY(),
crossProduct.GetZ());
    printf("\n\nThe Dot product is %0.2f\n", DotProduct(&vector1, &vector2));
    system("pause");
}
```

```

void Option5()
{
    int squareMatrix[16], colMatrix[4], resultMatrix[4];

    InitialiseArray(squareMatrix, 16);
    InitialiseArray(colMatrix, 4);

    for (int index = 0; index < 16; index++)
    {
        system("cls");
        PrintHeader();
        printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
        PrintSquareMatrix(index, squareMatrix);
        printf("Enter Values for Square Matrix >> ");
        squareMatrix[index] = ReadInteger();
    }

    for (int index = 0; index < 4; index++)
    {
        system("cls");
        PrintHeader();
        printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
        PrintColumnMatrix(index, colMatrix);
        printf("Enter Values for Column Matrix >> ");
        colMatrix[index] = ReadInteger();
    }
    MultiplyMatrices(squareMatrix, colMatrix, resultMatrix);
    system("cls");
    PrintHeader();
    printf("5. Multiply a 4x4 and 4x1 column matrix\r\n");
    PrintSquareMatrix(16, squareMatrix);
    printf("\r\n");
    PrintColumnMatrix(4, colMatrix);
    printf("\r\nThe resulting matrix is:\r\n");
    PrintColumnMatrix(4, resultMatrix);

    system("pause");
}

void Option6()
{
    int maxNumbers, mode;
    int* listOfIntegers;
    double mean, median, stdDev;

    system("cls");
    PrintHeader();
    printf("6. Mean, Median, Mode and Standard Deviation of a list of integers\r\nEnter size of list: ");
    maxNumbers = ReadInteger();
    listOfIntegers = new int[maxNumbers];
    SetArray(listOfIntegers, maxNumbers);
    BubbleSort(listOfIntegers, maxNumbers);
    PrintList(listOfIntegers, maxNumbers, "list");
    mean = CalculateMean(listOfIntegers, maxNumbers);
    median = CalculateMedian(listOfIntegers, maxNumbers);
    mode = CalculateMode(listOfIntegers, maxNumbers);
    stdDev = CalculateStdDev(listOfIntegers, maxNumbers, mean);

    delete[] listOfIntegers;
}

```

```
        system("pause");
    }

void Option7()
{
    int setASize, setBSize, unionSize, intersectionSize;
    int* setA;
    int* setB;
    int* unionOfSets;
    int* intersectionOfSets;

    system("cls");
    PrintHeader();

    printf("7. Union and Intersection of two sets\r\nEnter size of Set A: ");
    setASize = ReadInteger();
    setA = new int[setASize];
    SetArray(setA, setASize);

    printf("\r\nEnter size of Set B: ");
    setBSize = ReadInteger();
    setB = new int[setBSize];
    SetArray(setB, setBSize);

    BubbleSort(setA, setASize);
    BubbleSort(setB, setBSize);
    PrintList(setA, setASize, "set A");
    PrintList(setB, setBSize, "set B");

    unionSize = FindUnionSize(setA, setB, setASize, setBSize);
    unionOfSets = new int[unionSize];
    SetUnion(unionOfSets, setA, setB, unionSize, setASize, setBSize);
    BubbleSort(unionOfSets, unionSize);
    PrintList(unionOfSets, unionSize, "Union");

    intersectionSize = FindIntersectionSize(setA, setB, setASize, setBSize);
    if (intersectionSize > 0)
    {
        intersectionOfSets = new int[intersectionSize];
        SetIntersection(intersectionOfSets, setA, setB, intersectionSize,
setASize, setBSize);
        BubbleSort(intersectionOfSets, intersectionSize);
        PrintList(intersectionOfSets, intersectionSize, "Intersection");

        delete[] intersectionOfSets;
    }
    else
    {
        printf("\r\nThere is no Intersection");
    }
    printf("\r\n");
    delete[] setA;
    delete[] setB;
    delete[] unionOfSets;

    system("pause");
}
```

Vector.cpp

```
#include "stdafx.h"
#include "Vector.h"
#include "Functions.h"
#include "Point.h"

//Constructor
Vector::Vector()
{
    _x = 0;
    _y = 0;
    _z = 0;
}

//Destructor
Vector::~Vector()
{
}

//Properties
double Vector::GetX()
{
    return _x;
}
double Vector::GetY()
{
    return _y;
}
double Vector::GetZ()
{
    return _z;
}
void Vector::SetX(double value)
{
    _x = value;
}
void Vector::SetY(double value)
{
    _y = value;
}
void Vector::SetZ(double value)
{
    _z = value;
}
void Vector::SetPointA()
{
    _a.SetXYZ(1);
}
void Vector::SetPointB()
{
    _b.SetXYZ(2);
}

//Methods
void Vector::SetXYZ(int number)
{
    printf("\r\nEnter values of Vector %i\r\n", number);
    printf("\r\nEnter x component >>");
    _x = ReadDouble();
    printf("\r\nEnter y component >>");
    _y = ReadDouble();
    printf("\r\nEnter z component >>");
    _z = ReadDouble();
}
```

```
}

double Vector::SquareRoot(double number)
{
    return pow(number, 0.5);
}

double Vector::CalculateLength()
{
    return SquareRoot(pow(_x, 2) + pow(_y, 2) + pow(_z, 2));
}

double Vector::NormalizeAxis(double axis)
{
    return (axis) / CalculateLength();
}

void Vector::NormalizeVector()
{
    printf("The Normalized vector is (%0.2f, %0.2f, %0.2f).\r\n",
NormalizeAxis(_x), NormalizeAxis(_y), NormalizeAxis(_z));
}

void Vector::ConvertPointsToVector()
{
    _x = _a.GetX() - _b.GetX();
    _y = _a.GetY() - _b.GetY();
    _z = _a.GetZ() - _b.GetZ();
}
```

Point.cpp

```
#include "stdafx.h"
#include "Point.h"
#include "Functions.h"

Point::Point()
{
    _x = 0;
    _y = 0;
    _z = 0;
}

Point::~Point()
{
}

//Properties
double Point::GetX()
{
    return _x;
}
double Point::GetY()
{
    return _y;
}
double Point::GetZ()
{
    return _z;
}
//Methods
void Point::SetXYZ(int number)
{
    printf("\r\nEnter Co-ordinates of Point %i\r\n", number);
    printf("\r\nEnter x Co-ordinate >>");
    _x = ReadDouble();
    printf("\r\nEnter y Co-ordinate >>");
    _y = ReadDouble();
    printf("\r\nEnter z Co-ordinate >>");
    _z = ReadDouble();
}
```

Functions.h

```
#pragma once
#include "Vector.h"
#include "Point.h"

extern void PrintHeader();
extern void PrintMenu();
extern int ReadInteger();
extern double ReadDouble();
extern void IgnoreInput();
extern bool IsQuitting();
extern bool ProcessMenu(int option);
extern double DotProduct(Vector* vector1, Vector* vector2);
extern Vector CrossProduct(Vector* vector1, Vector* vector2);
extern void InitialiseArray(int array[], int size);
extern void SetArray(int array[], int size);
extern void PrintColumnMatrix(int index, int matrix[]);
extern void PrintSquareMatrix(int index, int matrix[]);
extern void MultiplyMatrices(int array4x4[], int array4x1[], int result[]);
extern void SetArray(int list[], int listSize);
extern void swap(int *x, int *y);
extern void BubbleSort(int list[], int size);
extern void PrintList(int list[], int listSize, string object);
extern double CalculateMean(int list[], int listSize);
extern double CalculateMedian(int list[], int listSize);
extern int CalculateMode(int list[], int listSize);
extern double CalculateStdDev(int list[], int listSize, double mean);
extern int FindUnionSize(int setA[], int setB[], int setASize, int setBSize);
extern void SetUnion(int setUnion[], int setA[], int setB[], int unionSize, int
setASize, int setBSize);
extern int FindIntersectionSize(int setA[], int setB[], int setASize, int setBSize);
extern void SetIntersection(int setIntersection[], int setA[], int setB[], int
intersectionSize, int setASize, int setBSize);
extern void Option1();
extern void Option2();
extern void Option3();
extern void Option4();
extern void Option5();
extern void Option6();
extern void Option7();
```

Vector.h

```
#pragma once
#include "Point.h"

class Vector
{
private:    //Attributes
    double _x, _y, _z;
    Point _a = Point();
    Point _b = Point();

public:    //Constructor
    Vector();
    ~Vector();

    //Properties
    double GetX();
    double GetY();
    double GetZ();
    void SetX(double value);
    void SetY(double value);
    void SetZ(double value);
    void SetPointA();
    void SetPointB();

    //Methods
    void SetXYZ(int number);
    double SquareRoot(double number);
    double CalculateLength();
    double NormalizeAxis(double axis);
    void NormalizeVector();
    void ConvertPointsToVector();
};
```

Point.h

```
#pragma once
```

```
class Point
{
private:
    double _x, _y, _z;
public:
    Point();
    ~Point();
    double GetX();
    double GetY();
    double GetZ();
    void SetXYZ(int number);
};
```

Maths Assignment report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

86/100

PAGE 1

PAGE 2

PAGE 3



Comment 1

And code the maths library!

PAGE 4

PAGE 5



Comment 2

thorough. Did you explore other ways of parsing?

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15



Comment 3

no need for dot product

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26



Comment 4

ok but, for example, bimodal distributions are possible.



Comment 5

dividing by $(n-1)$ is the sample std dev. The population std dev divides by n alone.

PAGE 27



Comment 6

No mode.



Comment 7

test case could include multiple elements with the same value.



Comment 8

no mode

PAGE 28

PAGE 29

PAGE 30

PAGE 31



Comment 9

good



Comment 10

Good tests identified.

JAMES MILLNS

85.50

RESEARCH-INFORMED LITERATURE



Excellent investigation. Do read up on nuances of modes. PLEASE INCLUDE REFERENCES!!

3

KNOWLEDGE AND UNDERSTANDING



A sound understanding of maths and programming principles demonstrated.

13

ANALYSIS



Excellent analysis of problem and synthesis of solution. Some good test case generated but see comments in your report.

13

PRACTICAL APPLICATION AND DEP.



Excellent. Commenting good in places. Over engineered in places perhaps but overall a sound understanding of the underlying maths principles and programming practices.

53

SKILLS FOR PROFESSIONAL PRAC.



A well structured report, go demonstration of SE practices, test cases documented but no references! Include references.

3.5

