Lab 2 Debugger Report

Debugging Report Requirements

• Must set up relevant breakpoints and watched variables around the suspected buggy code section.

• Take screenshots of the exact moment where the debugger captures the incorrect program behaviour by using debugger stepping.

• Show how your proposed bug-fix eliminates the incorrect program behaviour with debugger output evidence.

Original Code:

```
Matrix Matrix::copy() { //GOTTA DEBUG THIS

    // Member Function - Create a Copy of This Matrix (NOT a copy constructor)
    // The function is intended:
        // 1. Create an instance of a matrix of the same dimensions as itself
        // 2. Copy all the elements of itself to the new copied instance
        // 3. Return the instance of the Matrix
    // However, the implementation is faulty with two semantic bugs.
    // fix the code using VSCode IDE Debugger r or Debugging Message Printout using cout, and produce a simple debugging report


    Matrix copy = Matrix();

    for(int i = 0; i < rowsNum; i++)
        for(int j = 0; j < colsNum; j++)
            copy.setElement(matrixData[j][i], j, i);

    return copy;

}
```

Bug 1

Observation

```
Matrix copy = Matrix();

Matrix m1 = Matrix();
Matrix m2 = Matrix(4,4);

String str1 = m1.toString(); //str1 = “0 0 0 \n0 0 0 \n0 0 0 \n”
String str2 = m2.toString(); //str2 = “0 0 0 0 \n0 0 0 0 \n0 0 0 0 \n0 0 0 0 \n”
```

Analysis

From the image below, it shows that the function, copy(), uses the default constructor, which creates a matrix with the size of [3][3]. This causes a problem due to the function not considering the size of the intended matrix to be copied.

```
Matrix::Matrix() { // Default Constructor

    /* // This is a sample constructor with the default matrix size set to 3x3
    // Two key concepts here:
    //  1. in C++, use the keyword new for heap memory allocation calls
    //  2. When a C++ class has heap data members, allocate heap memory for them
in the constructor
    //     THEREFORE, you may need to add a destructor to deallocate the memory.
    //     (You need to add it yourself!!)
    */

    rowsNum = 3;
    colsNum = 3;
    matrixData = new int*[rowsNum];

    for(int i = 0; i < rowsNum; i++) {
        matrixData[i] = new int[colsNum];
    }

    for(int i = 0; i < rowsNum; i++)
        for(int j = 0; j < colsNum; j++)
            matrixData[i][j] = 0;

}
```

Fix

Use the additional constructor that allows for a custom matrix size via Matrix (int row, int col). This allows for initializing a matrix with the same dimensions.

```
Matrix::Matrix( int row, int col ) {// Additional Constructor

    if (row <= 0 ){
        row = 3;
    }
    if (col <= 0 ){
        col = 3;
    }

    rowsNum = row;
    colsNum = col;
    matrixData = new int*[row];


    for(int i = 0; i < row; i++) {
        matrixData[i] = new int[col];
    }

    for(int i = 0; i < row; i++)
        for(int j = 0; j < col; j++)
            matrixData[i][j] = 0;

}
```

In the copy() function, the line will now be modified the line shown below.

```
Matrix copy = Matrix(rowsNum, colsNum);
```

The overall function now looks like this.

```
Matrix Matrix::copy() {

    Matrix copy = Matrix(rowsNum, colsNum);

    for(int i = 0; i < rowsNum; i++)
        for(int j = 0; j < colsNum; j++)
            copy.setElement(matrixData[j][i], j, i);

    return copy;
```

```
}
```

Bug 2

Observation

The first for loop will access the first dimension of the matrix, while the second one (the nested for loop) will access the next dimension.

```
int data[4][5] = {{1,2,3,4,5},{6,7,8,9,0},{0,0,1,2,3},{0,0,0,4,5}};
Matrix m2 = Matrix(4,5);
String str2 = m2.toString(); //str2 = "1 2 3 4 5\n6 7 8 9 0 \n0 0 1 2 3 \n0 0 0 4
5\n"
```

```
for(int i = 0; i < rowsNum; i++)
        for(int j = 0; j < colsNum; j++)
            copy.setElement(matrixData[j][i], j, i);
```

```
String str3 = "1 6 0 0 \n2 7 0 0 \n3 8 1 0 \n4 9 2 4 \n5 0 3 5\n";
```

Analysis

The innermost statement uses the function setElement to access the value at [j][i], while [i] and [j] are the first and second dimensions respectively. So this will lead to either accessing the wrong value or an out of bounds access.

Fix

Swap [j][i] to [i][j] so that the indices are not swapped, as shown below.

```
copy.setElement(matrixData[i][j], i, j);
```

Overall Code:

```
Matrix Matrix::copy() { //DEBUGGED

    // Member Function - Create a Copy of This Matrix (NOT a copy constructor)
    // The function is intended:
        // 1. Create an instance of a matrix of the same dimensions as itself
        // 2. Copy all the elements of itself to the new copied instance
        // 3. Return the instance of the Matrix
    // However, the implementation is faulty with two semantic bugs.
    // fix the code using VSCode IDE Debugger r or Debugging Message Printout
using cout, and produce a simple debugging report


    Matrix copy = Matrix(rowsNum, colsNum);

    for(int i = 0; i < rowsNum; i++)
        for(int j = 0; j < colsNum; j++)
            copy.setElement(matrixData[i][j], i, j);

    return copy;

}
```

Resulting in passed test case.

```
=== testCopy() ===
Passed
```