Software Unit Tests
for
Odyssey Space Research

Software Engineer Evaluation

Version 0.4

Prepared by: **Frank Putnam, Jr.**

Date of submission: **Monday, June 16, 2019**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
0.1	Frank Putnam, Jr.	Initial draft	2019/06/09
0.2	Frank Putnam, Jr.	Revision after peer review	2019/06/10
0.3	Frank Putnam, Jr.	Revised functions calling format	2019/06/13
0.4	Frank Putnam, Jr.	Corrections	2019/06/16

1 Introduction

1.1 Document Purpose

The purpose of this document is to outline software unit test specifications for a C++ project to evaluate my software engineering skills.

1.2 Testing Scope

The unit tests described in this document will determine the Software Quality Attributes of correctness and robustness.

Correctness – Software produces the expected output for each input as defined in the Functional Requirements.

Robustness – Software's ability to cope with erroneous input and errors while executing.

1) Check that:
    a. Matrix size defined by input variables is not greater than available system memory.
    b. Variables cannot be directly accessed and manipulated by user.
    c. Functions return an error code value of 0 if the task is accomplished.
    d. Functions return an error code value greater than 0, specifying the error, if the task is not accomplished.
2) The Matrix_header contains:
    a. The Number_of_rows and Number_of_columns such that matrix operations can check dimensions of the input matrices.
    b. The Matrix_element_type such that matrix operations can check inappropriate data types of the input matrices.
3) The enumeration of the Matrix_element_types prevents the use of unsupported types.

1.3 Intended Audience

The unit tests will be evaluated by at least two senior software developers from Odyssey Space Research.

1.4 Definitions, Acronyms and Abbreviations

SRS – Software Requirements Specifications

IEEE – Institute of Electrical and Electronics Engineers

IDE - integrated development environment.

BYTE_TYPE – Variable consisting of 8 bits.
Value range from 0 to 255.

FLOAT_TYPE – Floating point variable consisting of 32 bits.
Value range 3.4E +/- 38 (7 digits).

SINT_TYPE - Signed integer variable consisting of 32 bits.
Value range from -2,147,483,648 to +2,147,483,647.
(note: I believe the form is 2's complement but was unable to confirm.)

UINT_TYPE – Unsigned integer variable consisting of 32 bits.
Value range from 0 to 4,294,967,295.

Class – A "… type specifier that governs the lifetime, linkage, and memory location of objects." (source https://docs.microsoft.com/en-us/cpp/cpp/c-cpp-language-and-standard-libraries?view=vs-2019 [accessed 2019-06-08])

Overloaded – Multiple functions with the same name that are differentiated by their input variables declarations.

Unit-based access – The first element is addressed by (1, 1) instead of (0, 0).

1.5 Document Conventions

In general this document follows the IEEE formatting requirements. Arial font size 11 or 12 is used throughout the document for text. *Italics* are used for comments and ***bold italics*** are used for code operators. Document text is single spaced and maintains 1" margins.

1.6 References and Acknowledgments

1) edX: **Introduction to C++** (Provided by Microsoft)). Presented by Gerry O'Brien, Kate Gregory, and James McNellis.

2) Coursera On-line Education: **Matrix Algebra for Engineers** (The Hong Kong University of Science and Technology). Presented by Jeffrey R. Chasnov.

3) Wikipedia: *https://en.wikipedia.org/wiki/Matrix_multiplication*

4) **A Firmware Development Standard** Version 1.4 *by* Jack G. Ganssle

5) **"C/C++ Language and Standard Libraries"**
Microsoft
Accessed June 8, 2019
https://docs.microsoft.com/en-us/cpp/cpp/c-cpp-language-and-standard-libraries?view=vs-2019

2 Unit Test Descriptions

2.1 UT01→[FR1] The matrix structure.

The matrix shall be defined with a structure composed of an enumeration _Matrix_element_type, uint _Number_of_rows, uint _Number_of_columns, and pointer to Matrix_array.

2.2 UT02→[FR2] The matrix class constructor-default.

The matrix class shall have a default constructor of 1 row and 1column of bytes.

Input: OSR_matrix Default_matrix ();

Expected Result:	Default_matrix.Element_type	= BYTE_TYPE
	Default_matrix.Number_of_rows	= 1
	Default_matrix.Number_of_columns	= 1
	Default_matrix._Number_of_elements	= 1
	Default_matrix._Matrix_size_in_bytes	= 1
	Default_matrix.Matrix_array	=> {0}

2.3 UT03→[FR3] The matrix class constructor-custom.

The matrix class shall have a custom constructor of n rows and m columns of _Matrix_element_type.

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       THREE_COLUMNS,
                                        OSR_matrix::SINT_TYPE);
```

Expected Result:	Matrix._Matrix_elements_type	= SINT_TYPE
	Matrix._Number_of_rows	= 2
	Matrix._Number_of_columns	= 3
	Matrix._Number_of_elements	= 6
	Matrix. _Matrix_size_in_bytes	=24
	Matrix.Matrix_array	=> {0 0 0}
		{0 0 0}

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       TWO_COLUMNS,
                                        OSR_matrix::UINT_TYPE);

Expected Result:    Matrix._Matrix_elements_type     = UINT_TYPE
                    Matrix._Number_of_rows           = 2
                    Matrix._Number_of_columns        = 2
                    Matrix._Number_of_elements       = 4
                    Matrix. _Matrix_size_in_bytes    =16
                    Matrix.Matrix_array               => {0 0}
                                                         {0 0}
```

```
Input:  OSR_matrix* Matrix = new OSR_matrix(THREE_ROWS,
                                            OUR_COLUMNS,
                                            OSR_matrix::FLOAT_TYPE);

Expected Result:    Matrix->_Matrix_elements_type   = FLOAT_TYPE
                    Matrix->_Number_of_rows         = 3
                    Matrix->_Number_of_columns      = 4
                    Matrix->_Number_of_elements     = 12
                    Matrix->_Matrix_size_in_bytes   = 48
                    Matrix->Matrix_array             => {0.0  0.0  0.0  0.0}
                                                        {0.0  0.0  0.0  0.0}
                                                        {0.0  0.0  0.0  0.0}
```

```
Input:  OSR_matrix* Matrix = new OSR_matrix(THREE_ROWS,
                                            FOUR_COLUMNS,
                                            OSR_matrix::BYTE_TYPE);

Expected Result:    Matrix->_Matrix_elements_type   = BYTE_TYPE
                    Matrix->_Number_of_rows         = 3
                    Matrix->_Number_of_columns      = 4
                    Matrix->_Number_of_elements     = 12
                    Matrix->_Matrix_size_in_bytes   = 12
                    Matrix->Matrix_array             => {0 0 0 0}
                                                        {0 0 0 0}
                                                        {0 0 0 0}
```

2.4 UT04→[FR4] Four types of matrix elements.

The Matrix_element_type shall be any one of the following 4 types declared by the user at instantiation. BYTE_TYPE, FLOAT_TYPE, INT_TYPE, and UINT_TYPE.

The function **get_type_of_element** shall return the _Matrix_elements_type from the matrix header.

2.5 UT05→[FR5] Unit-based addressing.

The matrix elements shall be accessed via unit-based addressing. First element is accessed by (1,1). Last element accessed by (n,m).

Use enumerated type for indexes.

{To-do}

2.6 UT06→[FR6] The matrix elements initialization.

The matrix elements shall be initialized at instantiation by providing a list of values.

```
Input: OSR_matrix Matrix_2_by_3_of_sint(TWO_ROWS,
                                        THREE_COLUMNS,
                                        SINT_TYPE) = ({1,-3,5},{-2,4,-6});
```

```
Expected Result:    A = { 1, -3,  5}
                        {-2,  4, -6}
```

2.7 UT07→[FR7] Non float elements initialization.

The matrix elements of type BYTE_TYPE, INT_TYPE, and UINT_TYPE shall be initialized at instantiation to 0 if no list of values is provided.

{ Part of UT03 }

2.8 UT08→[FR8] Float elements initialization.

The matrix elements of type FLOAT_TYPE shall be initialized at instantiation to 0.0 if no list of values is provided.

{ Part of UT03 }

2.9 UT09→[FR9] The overloaded operator get_element.

The overloaded operator get_element shall obtain data of Matrix_element_type from a specific element in the matrix.

```
Input:  Matrix_2_by_3_of_sint => { 1 -3  5}
                                 {-2  4 -6}
     int Element_contents = 0;

     Matrix_2_by_3_of_sint. get_element ( ROW_1,
                                          COLUMN_2,
                                          &Element_contents);

Expected Result: Element_contents = -3


Input:  Matrix_2_by_3_of_uint  => {1 3 5}
                                  {2 4 6}
unsigned Element_contents = 0;

Matrix_2_by_3_of_uint. get_element ( ROW_2,
                                     COLUMN_2,
                                     &Element_contents);

Expected Result: Element_contents = 4;


Input:  Matrix_2_by_3_of_float  => {1.1  -1.2  1.3}
                                   {2.1  -2.2  2.3}
float Element_contents = 0.0;

Matrix_2_by_3_of_uint. get_element ( ROW_2,
                                     COLUMN_2,
                                     &Element_contents);

Expected Result: Element_contents = -2.2;
```

```
Input:  Matrix_2_by_3_of_uint  => {1 3 5}
                                  {2 4 6}
char Element_contents = 0;

Matrix_2_by_3_of_uint. get_element ( ROW_2,
                                     COLUMN_1,
                                     &Element_contents);

Expected Result: Element_contents = 2;
```

2.10 UT10→[FR10] The overloaded operator set_element.

The overloaded operator set_element shall load data of Matrix_element_type into a specific element of the matrix.

```
Input: Matrix_2_by_3_of_sint =>  {0 0 0}
                                 {0 0 0}

Matrix_2_by_3_of_sint .set_element (ROW_2, COLUMN_1, -5);


Expected Result: Matrix_2_by_3_of_sint => { 0 0 0}
                                          {-5 0 0}


Input: Matrix_3_by_3_of_uint=     {0 0 0}
                                  {0 0 0}
                                  {0 0 0}

Matrix_2_by_3_of_uint .set_element (ROW_3, COLUMN_2, 5);


Expected Result: Matrix_3_by_3_of_uint=       {0 0 0}
                                              {0 0 0}
                                              {0 5 0}


Input: Matrix_2_by_2_of_float => {0.0  0.0}
                                 {0.0  0.0}

Matrix_2_by_2_of_float .set_element (ROW_2, COLUMN_2, -15.3);

Expected Result: Matrix_2_by_2_of_float =>    {0.0    0.0}
                                              {0.0  -15.3}
```

```
Input: Matrix_1_by_5_of_bytes =>          {0 0 0 0 0}

Matrix_1_by_5_of_bytes.set_element (ROW_1, COLUMN_4, 0xFE);

Expected Result: Matrix_1_by_5_of_bytes =>   {0 0 0 0xFE 0}
```

2.11 UT11→[FR11] The class operation of addition.

The class shall support the operation of addition. Given two matrices A and B create a third matrix C with the elements of A and B added together.

```
Input:
        Sint_matrix_A =>    { 1 -2}
                            {-3  4}

        Sint_matrix_B =>    {-5  6}
                            { 7 -8}

        Sint_matrix_C =>    {0 0}
                            {0 0}

        [Overload operator =]

        Sint_matrix_C = Sint_matrix_A;

Expected Result: Sint_matrix_C =>       { 1 -2}
                                        {-3  4}

        [Overload operator+ =]

        Sint_matrix_C += Sint_matrix_B;

Expected Result: Sint_matrix_C =>       {-4   4}
                                        { 4  -4}
```

```
Input:
        Uint_matrix_A =>    {1 2}
                            {3 4}

        Uint_matrix_B =>    {5 6}
                            {7 8}

        Uint_matrix_C =>    {0 0}
                            {0 0}


        [Overload operator =]

        Uint_matrix_C = Uint_matrix_A;


Expected Result: Uint_matrix_C =>       {1 2}
                                        {3 4}

        [Overload operator+ =]

        Uint_matrix_C += Uint_matrix_B;

Expected Result: Uint_matrix_C =>       {  6   8}
                                        {10 12}



Input:
        Byte_matrix_A =>    {1 2}
                            {3 4}

        Byte_matrix_B =>    {5 6}
                            {7 8}

        Byte_matrix_C =>    {0 0}
                            {0 0}

        [Overload operator =]

        Byte_matrix_C = Byte_matrix_A;

Expected Result: Byte_matrix_C =>       {1 2}
                                        {3 4}

        [Overload operator+ =]

        Byte_matrix_C += Byte_matrix_B;

Expected Result: Byte_matrix_C =>       {  6   8}
                                        {10 12}
```

2.12 UT12→[FR12] The class operation of scalar multiplication.

The class shall support the operation of scalar multiplication. Given the matrix A and a scalar value K create a second matrix B with the elements of A multiplied by the scalar K.

Input: int Scalar_K = 5;

```
    Matrix_sint =>      {1 -2}
                        {-3 4}

    Scaled_matrix =>    {0 0}
                        {0 0}


    [Overload operator =]

    Scaled_matrix *= Matrix_sint;
```

Expected Result:

```
    Scaled_matrix =>    {1 -2}
                        {-3,4}


    [Overload operator *=]

    Scaled_matrix *= Scalar_K;
```

```
Expected Result => {   5   -10}
                   {-15    20}
```

Input: unsigned Scalar_K = 3;

```
    Matrix_Uint =>      {5 6}
                        {7 8}

    Scaled_matrix =>    {0 0}
                        {0 0}


    [Overload operator =]

    Scaled_matrix = Matrix_uint;
```

Expected Result:

```
    Scaled_matrix =>    {5 6}
                        {7 8}
```

[Overload operator *=]

Scaled_matrix *= Scalar_K;

```
Expected Result => {15  18}
                   {21  24}
```

Input: float Scalar_K = 2.5;

```
Matrix_float =>      {-5.1   6.2}
                     { 7.3  -8.4}

Scaled_matrix =>     {0.0  0.0}
                     {0.0  0.0}
```

[Overload operator =]

Scaled_matrix = Matrix_float;

Expected Result:

```
Scaled_matrix =>     {-5.1   6.2}
                     { 7.3  -8.4}
```

[Overload operator *=]

Scaled_matrix *= Scalar_K;

```
Expected Result => {-12.75   15.50}
                   { 18.25  -21.00}
```

Input: char Scalar_K = 0x0F;

```
Matrix_char =>       {5 6}
                     {7 8}

Scaled_matrix =>     {0 0}
                     {0 0}
```

[Overload operator =]

Scaled_matrix = Matrix_char;

Expected Result:

```
Scaled_matrix =>     {5 6}
                     {7 8}
```

[Overload operator *=]

Scaled_matrix *= Scalar_K;

```
Expected Result => {0x4B  0x5A}
                   {0x69  0x78}
```

{To-do}

2.13 UT13→[FR13] The class operation of matrix multiplication.

The class shall support the operation of matrix multiplication. Given two matrices A and B create a third matrix C with the elements of A row multiplied by the elements of B column and then added together.

```
Input:  Matrix_A =>  {1 2}
                     {3 4}

        Matrix_B =>  {5 6}
                     {7 8}

        Matrix_C =>  {0 0}
                     {0 0}
```

[Overload operator *]

Matrix_C = Matrix_A * Matrix_B;

```
Expected Result:   A = {a b}     B =  {e f }     C =  {ae+bg  af+bh}
                       {c d}          {g h}           {ce+dg  cf+dh}

                       {1 2}          {5 6}           {19  22}
                       {3 4}          {7 8}           {43  50}
```

2.14 UT14→ [FR14] The class operation of transpose.

The class shall support the operation of transpose. Given a matrix A create a second matrix A^T whose elements are the transpose of A.

```
Input:  A_matrix  => {1   2   3   4}
                     {5   6   7   8}
                     {9 10 11 12}
```

A^T_matrix => nullptr

Input: [Overload Function Transpose]

A^T_matrix = A_matrix.Transpose();

```
AT_matrix => {1 5   9}
             {2 6 10}
             {3 7 11}
             {4 8 12}
```

2.15 UT15→[FR15] The function get_number_of_elements.

The function **get_number_of_elements** shall return the _Number_of_elements from the matrix header.

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       THREE_COLUMNS,
                                        OSR_matrix::SINT_TYPE);

        Number_of_matrix_elements = Matrix.get_number_of_elements();
```

Expected Result: Number_of_matrix_elements = 6

2.16 UT16→ [FR16] The function get_size_in_bytes

The function **get_size_in_bytes** shall return the number of bytes that make up the matrix array pointed to by the matrix header.

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       THREE_COLUMNS,
                                        OSR_matrix::SINT_TYPE);

        Matrix_size = Matrix.get_size_in_bytes ();

Expected Result: Number_of_matrix_elements = 24 bytes*
                              2   rows by 3 columns = 6 elements
                              6   elements x 4 bytes per element = 24 bytes
```

*NOTE: A matrix of unsigned or float will return the same number of bytes.

```
Input:  OSR_matrix Byte_matrix = OSR_matrix (  TWO_ROWS,
                                               THREE_COLUMNS,
                                               OSR_matrix::BYTE_TYPE);

        Matrix_size = Byte_matrix.get_size_in_bytes ();

Expected Result:     Matrix_ size = 4 bytes
                          2 rows by 2 columns = 4 elements
                              3   elements x 1 byte per element = 4 bytes
```

2.17 UT17→ [FR17] The function get_number_of_rows.

The function **get_number_of_rows** shall return the _Number_of_rows for the matrix array pointed to by the matrix header.

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       THREE_COLUMNS,
                                        OSR_matrix::SINT_TYPE);

        Number_of_matrix_rows = Matrix.get_number_of_rows();

Expected Result: Number_of_matrix_rows = 2
```

2.18 UT18→ [FR18] The function get_number_of_columns.

The function **get_number_of_columns** shall return the _Number_of_columns for the matrix array pointed to by the matrix header.

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       THREE_COLUMNS,
                                        OSR_matrix::SINT_TYPE);

        Number_of_matrix_columns = Matrix.get_number_of_columns();
```

Expected Result: Number_of_matrix_columns = 3

2.19 UT19→ [FR19] The function get_array.

The function **get_array** shall return the pointer to the Matrix_array.

```
Input:  OSR_matrix Matrix = OSR_matrix(TWO_ROWS,
                                       THREE_COLUMNS,
                                        OSR_matrix::SINT_TYPE);

        *Matrix_array_pointer = Matrix.get_array();
```

Expected Result: Matrix_array_pointer => Matrix.Matrix_array