Software Unit Tests
for
Odyssey Space Research

Software Engineer Evaluation

Version 0.2

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

Date of submission: **Friday, June 10, 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

Introduction

Document Purpose

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

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.

Intended Audience

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

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).

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.

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

Overall Description

Unit Tests

UT01 **[FR1]** 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_array.

Input: OSR_matrix Matrix_of_sint(TWO_ROWS,
THREE_COLUMNS,
SINT_TYPE);

Expected Result:

Matrix_of_sint.Element_type	= SINT_TYPE
Matrix_of_sint.Number_of_rows	= 2
Matrix_of_sint.Number_of_columns	= 3
Matrix_of_sint.Pointer_to_array	=> {0, 0, 0} {0, 0, 0}

UT02 **[FR2]** 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.Pointer_to_array	=> {0}

UT03 **[FR3]** The matrix class shall have a custom constructor of n rows and m columns of Matrix_element_type.

Input: OSR_matrix Matrix_2_by_3_of_sint(TWO_ROWS,
THREE_COLUMNS,
SINT_TYPE);

Expected Result: **A** = {0, 0, 0}
{0, 0, 0}

Input: OSR_matrix Matrix_3_by_3_of_uint(THREE_ROWS,
THREE_COLUMNS,
UINT_TYPE);

Expected Result: **A** = {0, 0, 0}
{0, 0, 0}
{0, 0, 0}

Input: OSR_matrix Matrix_1_by_5_of_bytes(ONE_ROW,
FIVE_COLUMNS,
BYTE_TYPE);

Expected Result: **A** = {0, 0, 0, 0, 0}

Input: OSR_matrix Matrix_2_by_2_of_float(TWO_ROWS,
TWO_COLUMNS,
FLOAT_TYPE);

Expected Result: **A** = {0.0, 0.0}
{0.0, 0.0}

UT04 **[FR4]** 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.

Input:

Expected Result:

{ Part of UT03 }

UT05 **[FR5]** 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.

Input:

Expected Result:

UT06 **[FR6]** 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}
```

UT07 **[FR7]** 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.

Input:

Expected Result:

{ Part of UT03 }

UT08 **[FR8]** The matrix elements of type FLOAT_TYPE shall be initialized at instantiation to 0.0 if no list of values is provided.

Input:

Expected Result:

{ Part of UT03 }

UT09 **[FR9]** The overloaded operator get_element_from shall obtain data of Matrix_element_type from a specific element in the matrix.

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

```
sint Element_contents = get_element_from( ROW_1,
                                          COLUMN_3,
                                          Matrix_2_by_3_of_sint);
```

Expected Result: Element_contents = 5

```
Input:  OSR_matrix Matrix_2_by_3_of_uint(TWO_ROWS,
                                        THREE_COLUMNS,
                                        UINT_TYPE) = ({1,3,5},{2,4,6});
```

```
sint Element_contents = get_element_from( ROW_1,
                                          COLUMN_3,
                                          Matrix_2_by_3_of_uint);
```

Expected Result: Error

UT10 **[FR10]** The overloaded operator set_element_to shall load data of Matrix_element_type into a specific element of the matrix.

```
set_element_to(Value_of_Matrix_element_type, Row, Column, The_matrix);
```

```
Input: OSR_matrix Matrix_2_by_3_of_sint(  TWO_ROWS,
                                          THREE_COLUMNS,
                                          SINT_TYPE);

set_element_to(-5, ROW_2, COLUMN_1, Matrix_2_by_3_of_sint);
```

```
Expected Result:     A = { 0, 0, 0}
                         {-5, 0, 0}
```

```
Input: OSR_matrix Matrix_3_by_3_of_uint(  THREE_ROWS,
                                          THREE_COLUMNS,
                                          UINT_TYPE);

set_element_to(5, ROW_3, COLUMN_2, Matrix_3_by_3_of_uint);
```

```
Expected Result:     A = {0, 0, 0}
                         {0, 0, 0}
                         {0, 5, 0}
```

```
Input: OSR_matrix Matrix_1_by_5_of_bytes(  ONE_ROW,
                                           FIVE_COLUMNS,
                                           BYTE_TYPE);

set_element_to(254, ROW_1, COLUMN_4, Matrix_1_by_5_of_bytes);
```

```
Expected Result:     A = {0, 0, 0, 254, 0}
```

```
Input: OSR_matrix Matrix_2_by_2_of_float( TWO_ROWS,
                                          TWO_COLUMNS,
                                          FLOAT_TYPE);

set_element_to(15.3, ROW_2, COLUMN_2, Matrix_2_by_2_of_float);

Expected Result:   A = {0.0,   0.0}
                       {0.0, 15.3}
```

UT11 **[FR11]** 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: OSR_matrix Byte_matrix_A( TWO_ROWS,
                                 TWO_COLUMNS,
                                 BYTE_TYPE) = ({1,2},{3,4});

OSR_matrix Byte_matrix_B( TWO_ROW,
                          TWO_COLUMNS,
                          BYTE_TYPE) = ({5,6},{7,8});

OSR_matrix Byte_matrix_C( TWO_ROW,
                          TWO_COLUMNS,
                          BYTE_TYPE);

[Overload operator +]

Byte_matrix_C = Byte_matrix_A + Byte_matrix_C;

Expected Result:   A = {1 2}    B = {5 6}    C = {  6   8}
                       {3 4}        {7 8}        {10 12}
```

UT12 **[FR12]** 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: uint Scalar_K = 5;

OSR_matrix Uint_matrix_A( TWO_ROWS,
                          TWO_COLUMNS,
                          UINT_TYPE) = ({1,2},{3,4});
```

```
OSR_matrix Scalar_matrix_B(  TWO_ROW,
                             TWO_COLUMNS,
                             UINT_TYPE);
```

[Overload operator *]

Scalar_matrix_B = Uint_matrix_A * Scalar_K;

```
Expected Result:    A = {1 2}      K = 5          C = {  5  10}
                        {3 4}                         {15  20}
```

UT13 **[FR13]** 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:  OSR_matrix Sint_matrix_A(  TWO_ROWS,
                                   TWO_COLUMNS,
                                    SINT_TYPE) = ({1,2},{3,4});

OSR_matrix Sint_matrix_B(  TWO_ROWS,
                           TWO_COLUMNS,
                           SINT_TYPE) = ({5,6},{7,8});

OSR_matrix Result_matrix_C(  TWO_ROWS,
                             TWO_COLUMNS,
                             SINT_TYPE);
```

[Overload operator *]

Result_matrix_C = Sint_matrix_A * Sint_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}
```

UT14 [FR14] The class shall support the operation of transpose. Given a matrix **A** create a second matrix $\mathbf{A}^T$ whose elements are the transpose of **A.**

```
Input:  OSR_matrix A_matrix (FOUR_ROWS,
                             TWO_COLUMNS,

                           ({1, 2, 3, 4},{5, 6, 7,8},{9, 10, 11, 12},{13, 14, 15,16});

OSR_matrix Transposed _matrix (FOUR_ROWS,
                               FOUR_COLUMNS,
                               SINT_TYPE);

Transposed_matrix = Transpose(A_matrix);
```

Expected Result: $\mathbf{A} = \begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$ $\mathbf{A}^T = \begin{matrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{matrix}$

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{matrix} \qquad \begin{matrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{matrix}$$

UT15 [FR15] The function **get_dimensions_of** shall return Number_of_rows and Number_of_columns from the matrix header.

get_dimensions_of(The_matrix, Rows, Columns); // Rows & Columns modified.

```
Input:  uint A_rows;
        uint A_columns;

        uint B_rows;
        uint B_columns;

        uint C_rows;
        uint C_columns;


OSR_matrix Byte_matrix_A(  TWO_ROWS,
                           TWO_COLUMNS,
                           BYTE_TYPE);
```

get_dimensions_of(Byte_matrix_A, A_Rows, A_Columns);

```
OSR_matrix Uint_matrix_B(  TWO_ROWS,
```

```
                                  THREE_COLUMNS,
                                  UINT_TYPE);
```

get_dimensions_of(Uint_matrix_B, B_Rows, B_Columns);

```
OSR_matrix Float_matrix_C(  THREE_ROWS,
                            TWO_COLUMNS,
                            FLOAT_TYPE);
```

get_dimensions_of(Float_matrix_C, C_Rows, C_Columns);

```
Expected Result:    A_rows      = 2
                    A_columns   = 2

                    B_rows      = 2
                    B_columns   = 3

                    C_rows      = 3
                    C_columns   = 2
```

UT16 **[FR16]** The function **get_size_of** shall return the number of bytes that make up the matrix array pointed to by the matrix header.

uint Matrix_size **= get_size_of**(The_matrix); // Returns size in bytes.

```
Input:  OSR_matrix Byte_matrix_A(  TWO_ROWS,
                                   TWO_COLUMNS,
                                   BYTE_TYPE);
```

uint Matrix_A_size **= Get_size_of**(Byte_matrix_A);

```
OSR_matrix Uint_matrix_B(  TWO_ROWs,
                           THREE_COLUMNS,
                           UINT_TYPE);
```

uint Matrix_B_size **= Get_size_of**(Uint_matrix_B);

```
OSR_matrix Float_matrix_C(  THREE_ROWS,
                            TWO_COLUMNS,
                            FLOAT_TYPE);
```

uint Matrix_C_size **= Get_size_of**(Float_matrix_C);

Expected Result: Matrix_A_size = 4 bytes
2 rows by 2 columns = 4 elements
4 elements x 1 byte per element = 4 bytes

Matrix_B_size = 24 bytes
2 rows by 3 columns = 6 elements
6 elements x 4 bytes per element = 24 bytes

Matrix_C_size = 24 bytes
3 rows by 2 columns = 6 elements
6 elements x 4 bytes per element = 24 bytes