Software Requirements Specification

for

Odyssey Space Research

Software Engineer Evaluation

Version 0.2

Prepared by **Frank Putnam, Jr.**

Date of submission: **Friday, June 9, 2019**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
0.1	Frank Putnam, Jr.	Initial draft	2019/06/08
0.2	Frank Putnam, Jr.	Revision after peer review	2019/06/09

1 Introduction

1.1 Document Purpose

The purpose of this document is to outline software specifications of a C++ project to evaluate my software engineering abilities. The customer requirements were deliberately vague in order to review my personal decisions related to the implementation.

1.2 Product Scope

The product is a C++ class implementing a 2 dimensional matrix of form n x m, where n and m are set by the user when the object is constructed. The class will support operations of addition, scalar multiplication, matrix multiplication, and transpose (my choice). The overloaded C++ operators of ***get*** and ***set*** will manipulate data of a specific element in the matrix using unit-based access.
Unit tests will be created by me and included in the zipped git repository submission.

1.3 Intended Audience

The product 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 Overall Description

2.1 Product Overview

In order for Odyssey Space Research to evaluate my software engineering skills I was presented with a set of requirements describing a project for developing a C++ class of two dimensional matrixes.

The class will provide such functionalities as user defined size; operations of addition, scalar and matrix multiplication, and one of my own choice; and two overloaded operators to manipulate specified elements of the matrix. Unit tests are included as part of the development.

Not only did OSR want to see functioning code, they also wanted to preview my methods for making decisions about the implementation.

2.2 Product Functionality

1) C++ class implementing a 2 dimensional matrix of form is **n** x **m.**

2) **n** and **m** are:
    a. unsigned integers greater than or equal to 1.
    b. set by the user when the object is constructed.

3) Matrix element type is defined by the user at time of construction.

4) Operations of
    a. addition
    b. scalar multiplication
    c. matrix multiplication
    d. transpose

5) Overloaded operators
    a. ***get*** to obtain data from a specific element of the matrix.
    b. ***set*** to load data into a specific element of the matrix.
    c. Matrix element access is unit-based.

6) Constructors
    a. default initialization to **n** = 1_row and **m** = 1_column.
    b. custom initialization to **n** = Number_of_rows and **m** = Number_of_columns.

2.3 Design and Implementation Constraints

1) Use of standard C++ libraries.

2) IDE will be Microsoft Visual Studio 2019 Professional (trial version) with Visual C++ Redistributables.

2.4 Assumptions and Dependencies

The inherently subjective evaluation will factor the following items into the analysis:

a) Does the code compile?
b) Does it do what it should?
c) Does it pass all unit tests?
d) Are the unit tests sufficient?
e) How readable and maintainable is the code?
f) Is the code robust enough to handle any edge cases?
g) Were good habits exhibited that would be expected of a professional at your level of expertise?
h) Were C++ best practices followed (or failed to be followed)?
i) What extra features to these bare requirements were added?
j) Is the code efficient?
k) What design and implementation decisions were made, and why?
l) Were relevant comments and other documentation added?

2.5 External Interface Requirements

2.5.1 User Interfaces

The matrix header file **'OSR_matrix.h'** is included in the development environment.

2.5.2 Hardware Interfaces

Development and testing was performed on a Hewlett Packard Pavilion series laptop containing

a) x64 processor - AMD A8-6410 APU (4 CPUs) with AMD Radeon R5 graphics at 2.00 GHz.
b) DRAM - 16.0 GB.
c) Hard drive - Samsung 860 EVO SSD 1.00 TB.
d) Microsoft Windows 64-bit operating system.
    1) Windows 10 Home edition
    2) Version 1809
    3) OS build 17763.503

The C++ matrix class should run on any hardware platform as long as it is compiled for that platform.

2.5.3 Software Interfaces

For the default constructor:

```
OSR_matrix New_default_matrix{}; // matrix of bytes with 1 row and 1 column.
```

For the custom constructor:

```
OSR_matrix New_custom_matrix{n, m, element_type};
// matrix of element_type with n rows and m columns.
```

For the overloaded operator ***get*** a unit-based row number and unit-based column number are input and the value contained in the specified element of the matrix is returned.

For the overloaded operator ***set*** a unit-based row number, unit-based column number and the value to be placed in the specified element of the matrix are input.

For the operations of addition, and matrix multiplication two matrices are input and one matrix is returned.

For the operation of scalar multiplication one matrix and a scalar value are input and one matrix is returned.

For the operation of transpose one matrix is input and one matrix is returned.

2.6 Functional Requirements

2.6.1 **[FR1]** The matrix class shall have a default constructor of 1 row and 1column of bytes.

2.6.2 **[FR2]** The matrix class shall have a custom constructor of n rows and m columns of Matrix_element_type.

2.6.3 **[FR3]** 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.

2.6.4 **[FR4]** The matrix elements shall be accessed via unit-based addressing. First element is accessed by (1,1). Last element accessed by (n,m).

2.6.5 **[FR5]** The overloaded operator **get_element_from** shall obtain data of Matrix_element_type from a specific element in the matrix.

Element_contents = **get_element_from**(Row, Column, The_matrix);

2.6.6 **[FR6]** 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);

2.6.7 **[FR7]** 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.

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

2.6.8 **[FR8]** 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**.

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \textbf{scalar} = k \quad \mathbf{B} = \begin{bmatrix} ka & kb \\ kc & kd \end{bmatrix}$$

2.6.9 [FR9] 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.

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

2.6.10 [FR10] 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.**

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & *** & a_{1m} \\ a_{21} & a_{22} & *** & a_{2m} \\ * & * & *** & * \\ * & * & *** & * \\ a_{n1} & a_{n2} & *** & a_{nm} \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & *** & a_{n1} \\ a_{12} & a_{22} & *** & a_{n2} \\ * & * & *** & * \\ * & * & *** & * \\ a_{1m} & a_{2m} & *** & a_{nm} \end{bmatrix}$$

2.6.11 [FR11] The matrix shall be defined as a structure composed of an enumeration Matrix_element_type, uint Number_of_rows, uint Number_of_columns, and Pointer_to_array.

2.6.12 [FR12] 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.

2.6.13 [FR13] The function **get_size_of** shall return the number of bytes that make up the matrix array pointed to by the matrix header.

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

2.6.14 [FR14] The matrix elements shall be initialized at instantiation by providing a list of values.

OSR_matrix matrix_2_by_3(Matrix_header, TWO_ROWS, THREE_COLUMNS, SINT_TYPE) = ({1,-3,5},{-2,4,-6});

2.6.15 [FR15] 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.

2.6.16 [FR16] The matrix elements of type FLOAT_TYPE shall be initialized at instantiation to 0.0 if no list of values is provided.

3 Other Non-functional Requirements

3.1 Software Quality Attributes

Correctness – Software produces the expected output for each input as defined in the Functional Requirements and verified by unit test.

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

1) Check that matrix size defined by input variables is not greater than available system memory.

2) Class variables are private to prevent direct access by user and manipulated via functions.

3) Functions return value of 0 if able to accomplish task and if not, an error code greater than 0 specifying the error.

4) The Matrix_header contains the Number_of_rows and Number_of_columns such that matrix operations can check dimensions of the input matrices.

5) The Matrix_header contains the Matrix_element_type such that matrix operations can check inappropriate data types of the input matrices.
(i.e., calling the .set function of an integer matrix with a floating point value).

6) The enumeration of the Matrix_element_types will prevent the use of unsupported types.