|  |
| --- |
| S/W LIBRARIES Interface |
| Matrix, Quaternion & Data Reading |
| RAMKIRAN |

# Introduction

There are three libraries that are introduced in this document. They are

1. Library on Matrix operation (Mat)
2. Library on Quaternion operations (Quat)
3. Library on Data reading from text files (dRead)

In this document a comprehensive description of how these libraries are implemented and how to interface these libraries to a c++ code were described. Methodology for modification to the existing elements is also explained along with compilation scheme.

# General Structure, Interface and Compilation

All the libraries that are described are done to support programming in c/c++ only. Hence the general structure and interface are developed that best suits ‘C programming’ language, which is well established.

## General Structure

This describes the methodology to implement any library in c/c++. Let describe a library **‘myLib’** which comes with a set of predefined function **‘f1(), f2, …, fn()’** andpredefined global variable **‘v1, v2 …. vm’**. The idea of any library is that these functions and variable are in frequent use and hence it’s nice to standardize these functions with greater degree of discipline and use them in one’s code. To do that we have to make

1. **Header file**: Contains all the declarations of functions and global variables
2. **Source file**: Contains all the definitions and assertions of the above describes functions and variable respectively.

In this case a header file could be like **‘myLib.h’** and the file looks like

**myLib.h**

**----------------------------------------------------------------------------------------------------**

**#include < … > // include functions like <stdio.h, math.h> … etc**

**// varibles**

**Type v1;**

**Type v2;**

**.**

**.**

**.**

**Type vm;**

**// functions**

**return\_type f1( input arguments ); // function declarations**

**return\_type f2( input arguments ); // function declarations**

**.**

**.**

**.**

**return\_type fn( input arguments ); // function declarations**

**----------------------------------------------------------------------------------------------------**

and source file could be **‘myLib.cpp’**

**myLib.cpp**

**----------------------------------------------------------------------------------------------------**

**#include < … > // include functions like <stdio.h, math.h> … etc**

**// varibles assertions**

**v1 = 1.0;**

**v2 = true; …**

**vm = 180.0/3.141592;**

**// functions definitions**

**return\_type f1( input arguments )**

**{**

**…. Function f1 content …..**

**}**

**…f2,f3…**

**return\_type fn( input arguments )**

**{**

**…. Function fm content …..**

**}**

**----------------------------------------------------------------------------------------------------**

## 2.2 Interface

To use this library (myLib) in one’s code (let it be mainCode.cpp … your code), we have to include the header file along with its path.

**mainCode.cpp**

**----------------------------------------------------------------------------------------------------**

**#include “myLib.h “ // include mylib.h .. ‘./’ here represents current folder**

**.**

**.**

**.**

**main()**

**{**

**f1( input ); f1 of mylib function call here**

**}**

**----------------------------------------------------------------------------------------------------**

Make sure that the global variable and function calls that are described doesn’t clash with the existing functions. If any queries regarding this please go through c/c++ interfaces *or* ‘Dealing with multiple source files’.

## 2.3 Compilation

While compilation just include the source file (myLib.cpp) along with one’s main code for proper execution.

e.g.

***>> g++ mainCode.cpp myLib.cpp –o out***

*or*

***>> g++ -c myLib.cpp***

***>> g++ mainCode.cpp myLib.o –o out***

# Quaternion Library Usage

Quaternion library captures all the major quaternion operations. This library comes with two files as described above namely **‘QUAT.h’** **& ‘QUAT.cpp’**.

QUAT.h contains all the function declarations of quaternion functions while QUAT.cpp contains all the definitions. Here a ‘Quat’ c++ class is declared which acts as a data container which specifically stores normalised quaternions.

**Interface:**

Copy QUAT.h and QUAT.cpp in the current working directory. Include QUAT.h in the main program and build with QUAT.cpp while compiling. This mentioned operation will include all QUAT library functionalities in one’s code. If any queries are there please refer to previous section.

***QUAT library interface:***

**mainCode.cpp**

**----------------------------------------------------------------------------------------------------**

**#include “QUAT.h “ // include QUAT.h**

**.**

**.**

**main()**

**{**

**Quat functions are available here**

**}**

**----------------------------------------------------------------------------------------------------**

***QUAT library Compilation:***

***>> g++ mainCode.cpp QUAT.cpp –o out***

*or*

***>> g++ -c QUAT.cpp***

***>> g++ mainCode.cpp QUAT.o –o out***

**----------------------------------------------------------------------------------------------------**

**Functionalities of QUAT Library:**

There are many functionalities of quaternion library and all the declarations of functionalities are captured in the public sector of ***Quat*** Class in ***QUAT.h*** and their corresponding definitions in ***QUAT.cpp***.

***QUAT.h***

**#ifndef \_QUAT\_H\_**

**#define \_QUAT\_H\_**

**Include files and standard libraries**

**Note “MAT.h” another external library is also included here**

**#include <stdio.h>**

**#include <iostream>**

**#include <stdlib.h>**

**#include <vector>**

**#include <unistd.h>**

**#include <math.h>**

**#include <limits.h>**

**#include "MAT.h"**

**class Quat**

**{**

**Initialization, Constructer and destructor declarations**

**public :**

**Quat();**

**Quat(double,double,double);**

**Quat(double,double,double,double);**

**//Quat(const Mat);**

**//Quat(const Quat&);**

**~Quat();**

**Setter and getter**

**double q(int);**

**void setq(int,double);**

**void setq(double,double,double);**

**void setq(double,double,double,double);**

**//void setq(const Mat);**

**void SetRotSeq(int,int,int);**

**double eul(int);**

**double dcm(int,int);**

**Mat dcm();**

**void sete(double,double,double);**

**void sete(double inp[3]);**

**//void sete(const Mat);**

**void setd(double inp[3][3]);**

**void setd(const Mat);**

**Quat& operator= (const Quat &Q);**

**Operators**

**Quat operator+ (const Quat &Q);**

**Quat operator- (const Quat &Q); // q1-q2 = ~q1 \* q2**

**Quat operator\* (const Quat &Q);**

**Quat operator~ (void );**

**Quat operator/ ( Quat &Q); // q1/q2 = q1 \* ~q2**

**Display Options**

**void PrintQuat();**

**void PrintEul();**

**void PrintDCM();**

**void PrintRotSeq();**

**private:**

**double Q[4];**

**double EUL[3];**

**double IR1,IR2,IR3;**

**double DCM[3][3];**

**void norm();**

**double sign(double);**

**void ComputeDcm();**

**void ComputeEul();**

**double maximum(double ary[],int size);**

**};**

**#endif**

**Initialization, Constructer and destructor declarations**

**‘Quat’** can be used as any other data type like int, double, float, char etc. Following explains the initialisation of quaternions. Note normalization is part of initialization.

Initialisation examples

Quat q1;

Initializes quaternion q1 with default value (0,0,0,1)

Quat q2(1,2,3);

Initializes quaternion q2

q2(1) = 1/sqrt(1\*1+2\*2+3\*3)

q2(2) = 2/sqrt(1\*1+2\*2+3\*3)

q2(3) = 3/sqrt(1\*1+2\*2+3\*3)

q2(4) = 0;

Here values q4 = sqrt(1 - 1\*1+2\*2+3\*3) is impossible

Quat q3(0.5,0.5,0.5,0.5);

Initializes quaternion q3 by

q3(1) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)

q3(2) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)

q3(3) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)

q3(4) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)

Quat q4(1,2,3,4);

Initializes quaternion q3 by

q4(1) = 1/sqrt(1^2 + 2^2 + 3^2 + 4^2)

q4(2) = 2/sqrt(1^2 + 2^2 + 3^2 + 4^2)

q4(3) = 3/sqrt(1^2 + 2^2 + 3^2 + 4^2)

q4(4) = 4/sqrt(1^2 + 2^2 + 3^2 + 4^2)

Quat q5 = q6;

Initializes q5 with the values of q6.

Note : here q1,q2,q3,q4,q5,q6 are the Quat variables declared randomly. Any name of one’s liking can be given as variable names.

**Setter and getter**

Setter is a function (method) in Quat to set a variable. Similarly getter is used to get the variable in the given quaternion. The following will explain all the methods related to setter and getters.

1. **(double)q(int): Getter**

**Gets the double value of particular quaternion**

**Quat qValue(0.5,0.5,0.5,0.5);**

**double q1 = qValue.q(1);**

**here q1 will be having the value of first quaternion**

**qValue.q(2), qValue.q(3), qValue.q(4) are used to get the second, third and fourth value of the quaternion ‘qValue’**

1. **void setq(int,double) : Setter**

**Never use this function**

1. **void setq(double,double,double) : Setter**

**Sets the value of the quaternion similar to initialization of three values (q1,q2,q3) quaternion initialization**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.setq(0.5,0.5,0.5);**

**re-writes qValue to**

**qValues(4) = sqrt(1- (0.5^2 + 0.5^2 + 0.5^2)) = 0.5**

**qValues(1) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

**qValues(2) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

**qValues(3) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

1. **void setq(double,double,double,double) : Setter**

**Sets the value of quaternion similar to initialization with 4 quaternions.**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.setq(0.5,0.5,0.5,0.5);**

**re-writes qValue to**

**qValues(1) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

**qValues(2) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

**qValues(3) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

**qValues(4) = 0.5/sqrt(0.5^2 + 0.5^2 + 0.5^2 + 0.5^2)**

1. **void SetRotSeq(int,int,int) : Setter**

**Sets the rotation sequence used to compute Euler angles. Default rotation sequence is set to 3-2-1**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.setRotSeq(3,2,1);**

1. **double eul(int): Getter**

**Gets the Euler angles as per the rotation sequence. Output is in radians.**

**Quat qValue; // initializes to 0,0,0,1**

**double psi = qValues.eul(1);**

**returns first Euler angle and copies to psi**

1. **double dcm(int,int) : Getter**

**Gets the element of DCM for a quaternion.**

**Quat qValue; // initializes to 0,0,0,1**

**double dcm31 = qValues.dcm(3,1);**

**returns (3,1) element of qValue DCM and copies to dcm31**

1. **Mat dcm(); Getter**

**Gets dcm matrix in Mat format (to be discussed)**

**Quat qValue; // initializes to 0,0,0,1**

**Mat dcm = qValues.dcm();**

**Returns dcm matrix of qValue**

1. **void sete(double,double,double): Setter**

**Sets the Euler angles and stores as quaternion. Inputs are taken in radians**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.sete(10\*3.141592/180,20\*3.141592/180,30\*3.141592/180);**

**Sets qValue to Euler angles (10,20,30) deg as per rotation sequence**

1. **void sete(double inp[3]): Setter**

**Same as above, Only that input is a double array**

1. **void setd(double inp[3][3]): Setter**

**Quat qValue; // initializes to 0,0,0,1**

**double dcmInp = {{1,0,0},{0,1,0},{0,0,1}}**

**qValues.setd(dcmInp);**

**Sets the DCM matrix given in a double array ‘dcmInp’ and re-computes the qValue.**

1. **void setd(const Mat): Setter**

**Same as above only for input is using Mat.**

**Operators**

Operators are used like any other operations of +,-,\*,/ . All the implementation details of the operators are given below.

1. **Quat& operator= (const Quat )**

**operator=, copies one Quat variable to other Quat variable.**

**Quat q1;**

**Quat q2 = q1;**

**q1 = q2;**

**The above equality is valid because of member function operator=.**

1. **Quat operator+ (const Quat )**

**operator+ effectively adds two quaternions equivalent to quaternion multiplication.**

**Quat q1,q2,q3;**

**q3 = q1+q2; => q3 = qmultiplication(q1,q2)**

1. **Quat operator- (const Quat )**

**operator- does**

**Quat q1,q2,q3;**

**q3 = q1-q2; => q3 = qmultiplication(conjugate(q1),q2)**

**Used while error computation, input to a attitude controller**

1. **Quat operator\* (const Quat )**

**operator\* does exactly what Operator+ does**

1. **Quat operator~ (void )**

**Operator~ computes the conjugate of given quaternion**

**Quat q1,q2;**

**q2 = ~q1;**

**Stores conjugate of q1 in q2**

1. **Quat operator/ (Quat );**

**Operator/ does**

**Quat q1,q2,q3;**

**q3 = q1/q2; => q3 = qmultiplication(q1,conjugate(q2))**

**Display Options**

1. **void PrintQuat();**

**Prints all the quaternions**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.PrintQuat();**

1. **void PrintEul();**

**Prints all the Euler Angles**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.PrintEul();**

1. **void PrintDCM();**

**Prints all the elements of DCM**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.PrintDCM();**

1. **void PrintRotSeq();**

**Prints the Rotation Sequence used in that Quaternion element**

**Quat qValue; // initializes to 0,0,0,1**

**qValues.PrintRotSeq();**

This completes the Quaternion library. ‘**Private functions are not used in function calls’**. Don’t use functions declared in private zone.

Next we will discuss about the Mat (matrix) library.

**Compilation**

**g++ youfile.cpp QUAT.cpp -lm**

**Functionalities of MAT Library:**

Matrix Library is used to complete all the matrix operations with ease. Matrix operations like addition, multiplication, inverse, transpose etc can be easily done using matrix library. Evaluation of sub matrices and extending the existing matrices is also easily done using a matric library. Following gives the interface file for matrix library.

***MAT.h***

**#ifndef \_MAT\_H\_**

**#define \_MAT\_H\_**

**Include files and standard libraries**

**Note WARNING set 1 will display all the errors that a user is doing in the simulation**

**#define WARNING 0**

**#include <stdio.h>**

**#include <iostream>**

**#include <stdlib.h>**

**#include <vector>**

**#include <unistd.h>**

**#include <math.h>**

**#include <limits.h>**

**#include <string.h>**

**class Mat**

**{**

**Initialization, Constructer and destructor**

**public:**

**Mat();**

**Mat(int ,int ); //row column**

**Mat(const Mat& ); //deep copy;**

**~Mat();**

**Display elements**

**int rsize() const;**

**int csize() const;**

**void size();**

**void show();**

**Some Major matrix operators**

**Mat& operator= (const Mat& );**

**double& operator()(unsigned int ,unsigned int );**

**double& operator()(unsigned int );**

**void reset(int row,int col);**

**Setters**

**void set(int,double);**

**void set(double);**

**void set(int,int,double);**

**void set(const Mat& );**

**void set(int,int,const Mat& );**

**double get(int,int);**

**Getters**

**double get(int);**

**Mat row(int);**

**Mat col(int);**

**Mat submat(int,int,int,int);**

**Mat zero();**

**Mat Const(double );**

**Mat eye();**

**Mat eyec();**

**Mat eyer();**

**Mat diag();**

**void close(); -- Closes the matrix**

**Mat inv(); -- Computes inverse of matrix**

**double \*\*val;**

**private:**

**int r,c;**

**void allocate(int,int);**

**void release();**

**};**

**Mat operator+(Mat,double);**

**Some Major matrix operators**

**Mat operator+(double,const Mat);**

**Mat operator+(const Mat,const Mat);**

**Mat operator-(const Mat,double);**

**Mat operator-(double,const Mat);**

**Mat operator-(const Mat,const Mat);**

**Mat operator-(const Mat);**

**Mat operator\*(const Mat,double);**

**Mat operator\*(double,const Mat);**

**Mat operator\*(const Mat,const Mat);**

**Mat operator/(const Mat,double);**

**Mat operator~(const Mat);**

**Mat eye(int);**

**Some basic functionality**

**Mat fnorm(const Mat);**

**void MATsize(const Mat );**

**#endif**

**Initialization, Constructer and destructor**

‘Mat’ can be used as any other data type like int, double, float, char etc. Following explains the initialisation of matrices.

Initialisation examples

Mat m1;

Initializes matrix ‘m1’ with no value. Martix m1 will have no data, no rows and no columns.

Mat m1(3,4);

Initialises matrix m1 with size 3 rows and 4 columns. All the elements of the matrix are initialised with zeros.

Mat a(2,2);

a(1,1) = 1; a(2,2) = 4;

Initializes matrix ‘a’ with zeros with 2 columns and 2 rows. Later overwrites element (1,1) with 1 and element (2,2) by 4.

a = [1 0;

0 4];

Mat b(a);

Initializes matrix ‘b’ by matrix ‘a’.

Mat b = a;

Initializes matrix ‘b’ by matrix ‘a’.

Don’t use destructor function with ~.

**Display elements**

**int rsize() const;**

**Returns the row size of the matrix.**

**e.g**

**Mat a(4,5);**

**int row\_size = a.rsize();**

**Now row\_size contains 4**

**int csize() const;**

**Returns the column size of the matrix.**

**e.g**

**Mat a(4,5);**

**int col\_size = a.csize();**

**Now col\_size contains 5**

**void size();**

**Displays the size of a matrix.**

**e.g.**

**Mat a(4,5);**

**a.size();**

**displays size as (4X5)**

**void show();**

**Displays the contents of a matrix**

**e.g.**

**Mat a(4,5);**

**a.show();**

**displays the contents of matrix a**

**Some Major matrix operators**

**Mat& operator= (const Mat& );**

**Operatotr= is used to copy (deep copy) one matrix to other.**

**Mat a(4,4),c;**

**Mat b=a;**

**c = b;**

**Here matrix ‘a’ is copied into matrix ‘b’. Then matrix ‘b’ is copied into matrix ‘c’.**

**double& operator()(unsigned int ,unsigned int );**

**operator()(unsigned int ,unsigned int ); is used to access/allocate an element of a matrix**

**e.g.**

**Mat a(3,3);**

**a(1,1) = 3;**

**Allocates element (1,1) by number 3.**

**a(4,4) = 1;**

**Gives error as the element (4,4) is outside the prescribed description of matrix ‘a’ since matrix ‘a’ is of size (3,3)**

**double& operator()(unsigned int );**

**operator()(unsigned int ); is used to access/allocate an element of a matrix with either only one row are one column**

**e.g.**

**Mat a(3,1);**

**a(1) = 1;**

**Allocate first element of matrix ‘a’ with value 1. Note that the size of the matrix is declared as single column matrix.**

**Setters**

**void reset(int row,int col);**

**Resets the matrix to the specified size.**

**e.g.**

**Mat a(3,3); Initialized to a 3X3 matrix**

**a.reset(2,2); Reset the matrix to 2X2 matrix (Only top left matrix will be taken after truncation)**

**a.reset(4,4); Reset the matrix to 4X4 matrix. Unknowns will be set to zeros.**

**void set(int position,double value);**

**Sets a specified value at a specified position. Note that this will work only on matrix with either only one column or one row.**

**e.g.**

**Mat a(1,3);**

**a.set(3,45.0);**

**Sets the 3rd element of matrix ‘a’ with value 45.**

**a.set(6,67.0);**

**Note 6th element is not defined. So first the program resets to a 6X1 matrix and sets 6th element to 67.**

**void set(double);**

**Sets all the elements the matrix with the specified value.**

**e.g**

**Mat a(2,2);**

**a.set(1);**

**This will imply a = [1 1;1 1]**

**void set(int row\_position, int column\_position,double value);**

**Sets a specified value at a specified position.**

**e.g.**

**Mat a;**

**a.set(3,3,23);**

**Resets the matrix to a 3X3 matrix. Then sets the element (3,3) of matrix ‘a’ with value 23. Not all the other elements in this case are to zero by default.**

**A = [0 0 0;0 0 0;0 0 23]**

**a.set(1,2,43); Sets element (1,2) to 43**

**A = [0 43 0;0 0 0;0 0 23]**

**void set(const Mat& );**

**Sets one matrix with other.**

**e.g.**

**Mat a(2,2),b(3,3);**

**a.set(b);**

**Matrix ‘a’ is set with matrix ‘b’.**

**void set(int,int,const Mat& );**

**Sets a matrix at the intended posion.**

**e.g.**

5 6

7 8

1 2

3 4

**Mat a(2,2),b(2,2);**

**a(1,1) = 1;**

**a(1,2) = 2;**

**a(2,1) = 3;**

**a(2,2) = 4;**

**b(1,1) = 5;**

**b(1,2) = 6;**

1 2 0

3 5 6

0 7 8

1 2

3

**b(2,1) = 7;**

**b(2,2) = 8;**

5 6

7 8

**Here a = [1 2;3 4],b= [5 6;7 8]**

**a.set(2,2,b);**

**a = [ 1 2 0;3 5 6;0 7 8]**

**Getters**

**double get(int row\_position,int col\_position);**

**Gets an element from a specified position of row and column.**

**e.g.**

**Mat a(2,2);**

**double v = a.get(1,2);**

**Gets the position at element (1,2) and returns to variable v.**

**double get(int);**

**Gets an element from a specified position. Works on only those matrices with either only one row or one column.**

**e.g.**

**Mat a(5,1);**

**double v = a.get(3);**

**Gets the position at element (3,1) and returns to variable v.**

**Mat row(int);**

**Gets a specific row of input and returns to a matrix variable.**

**e.g.**

**Mat a(2,2);**

**Mat b = a.row(2);**

**Copies b with 2nd row of matrix a;**

**Mat col(int);**

**Gets a specific column of input and returns to a matrix variable.**

**e.g.**

**Mat a(2,2);**

**Mat b = a.col(2);**

**Copies b with 2nd column of matrix a**

**Mat submat(int,int,int,int);**

**Gets sub matrix in a given matrix. Takes top-left and bottom-corner of the matrix as input to extract the sub-matrix.**

**Mat a(5,6);**

**Mat b = a.submat(3,3,4,4);**

**Matrix ‘b’ has the submatrix of ‘a’ [ element(3,3) element(3,4);**

**element(4,3) element(4,4)]**

**Mat zero();**

**Returns a zero matrix of same size of input matrix.**

**e.g.**

**Mat a(4,5);**

**Mat z = a.zero();**

**Z is zero matrix of size 4X5.**

**Mat Const(double );**

**Returns a constant matrix of specified constant and same size of input matrix.**

**e.g.**

**Mat a(4,5);**

**Mat b = a.Const(1);**

**b is 1’s matrix of size 4X5.**

**Mat eye();**

**Returns identity matrix same as input matrix, if input matrix is square matrix.**

**e.g.**

**Mat a(4,4);**

**Mat b = a.eye();**

**b is identity matrix of size 4X4.**

**Mat eyec();**

**Returns identity matrix with the column size of input matrix.**

**e.g.**

**Mat a(4,5);**

**Mat b = a.eyec();**

**b is identity matrix of size 5X5.**

**Mat eyer();**

**Returns identity matrix with the row size of input matrix.**

**e.g.**

**Mat a(4,5);**

**Mat b = a.eyer();**

**b is identity matrix of size 4X4.**

**Mat diag();**

**Returns the diagonal matrix of the input square matrix.**

**e.g.**

**Mat a(2,2);**

**Mat b = a.eye();**

**Mat c = diag(b);**

**Matric ‘c’ is row matrix with diagonal elements of b.**

**void close(); -- Closes the matrix**

**This function de allocates all the elements of the matrix. Once close call is issued nothing will be there in the matrix.**

**Mat inv(); -- Computes inverse of matrix**

**Calculates the inverse of the matrix and returns the inverse to another matrix.**

**e.g.**

**Mat b = a.inv();**

**double \*\*val;**

**One can also access the elements of the matrix using this val variable.**

**Mat a(5,5);**

**double b = a.val[0][3];**

**Access is like any other c array access.**

**Note that any errors during execution will terminate to program with relevant error description.**

**Some Major matrix operators**

**operator+**

**One can add any two matrices of same size or a matrix and single value using ‘+’ operator.**

**e.g.**

**Mat a(2,2);**

**Mat b = a + 2; // every element is added with 2**

**Mat c = a + b;**

**Mat d = 5 + c; // every element is added with 5**

**Operator-**

**One can subtract any two matrices of same size or a matrix and single value using ‘-’ operator. A matrix can also be negated using operator-.**

**e.g.**

**Mat a(2,2);**

**Mat b = a - 2; // every element is subtracted with 2**

**Mat c = a - b;**

**Mat d = 5 - c; // every element is negated and added with 5**

**Mat e = -d;**

**Operator\***

**One can multiply any two matrices of relevant sizes or a matrix and single value using ‘\*’ operator.**

**e.g.**

**Mat a(2,2);**

**Mat b = a\*2; // every element is multiplied with 2**

**Mat c = a\*b;**

**Mat d = 5\*c; // every element is multiplied with 5**

**Mat operator/(const Mat,double);**

**Division is only done by single valued number on a matrix.**

**e.g.**

**Mat a(2,2);**

**Mat b = a/2; // every element of matrix ‘a’ is divided by 2 and copied to b**

**Mat operator~(const Mat);**

**Operator~ used to calculate the transpose of the matrix.**

**e.g.**

**b = ~a; // ‘a’ transpose is copied to ‘b’**

**Some basic functionality**

**Mat eye(int);**

**Returns identity matrix of specified size.**

**e.g.**

**Mat a = eye(2);**

**Matrix ‘a’ has identity matrix of size 2.**

**Mat fnorm(const Mat);**

**Returns Frobineous Norm of a matrix.**

**void MATsize(const Mat );**

**Diplays the size of the matrix.**

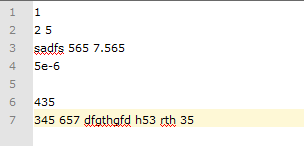
**Compilation**

**g++ youfile.cpp MAT.cpp -lm**

**Functionalities of DREAD Library:**

DREAD library is used to read text from any position in a text file. This library can read various types of data as specified by the user. This library is very useful to read input files to one’s code when data is not well structured. Following will give the various functional interfaces of dread.

For all the examples below consider the following input file



This is a ascii text file with the above mentioned inputs. Left side data in shaded region contains the line numbers. Let the file name be ***‘file.read’***.

data();

‘data’ can be used as any other data type like int, double, float, char etc. Following explains the initialisation of type ‘data’.

**data(char\*);**

Data file can be loaded while declaration using this function.

e.g.

data fp(”***file.read*** “);

here

Input is taken as char\* variable .

fp – variable file pointer to which file ‘file.read’ is loaded. Further fp will be used to access data in the file ‘file.read’

**data(string);**

Data file can be loaded while declaration using this function.

e.g.

data fp(”***file.read*** “);

here

Input is taken as string variable. #include <string> to be included to run this code

fp – variable file pointer to which file ‘file.read’ is loaded. Further fp will be used to access data in the file ‘file.read’

**~data();**

Do not use this functionality

**void load(char\*);**

On can separately load file, separately (not necessarily at initialization) using load function. Note If some other file is already loaded into this file that file will be unloaded and loaded with this file. Note here input put is a char\* variable

e.g.

data a;

a.load(‘file.read’);

a.load(‘some other file’);

**void load(string);**

On can separately load file, separately (not necessarily at initialization) using load function. Note If some other file is already loaded into this file that file will be unloaded and loaded with this file. Note here input put is a string variable. #include <string> to be included to run this code

e.g.

data a;

string str = “file.read”;

a.load(str);

**void unload();**

Unload an already loaded file. This is used to release the memory.

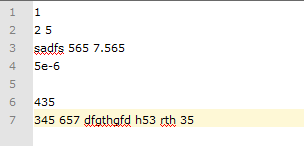
e.g.

data a(‘data.read’);

a.unload();

**Access Functions**

**int i(unsigned int row,unsigned int col);**



Returns an integer value positioned at the specified row and column in ‘file.read’. Note the delimiters used are ‘ ’ – space. ‘ ’ – tab for new columns and new-line for new row.

Consider the file as mentioned above.

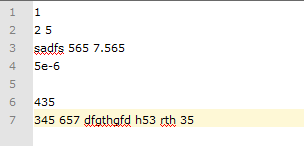
e.g.

data a(‘file.read’);

int num = a.i(2,2); // num has 5

Here integer variable ‘num’ will be populated with number 5, positioned at second row second column. If no data is present at that place a junk number/zero will be replayed based on compiler.

**float f(unsigned int,unsigned int);**



Returns a float value positioned at the specified row and column in ‘file.read’. Note the delimiters used are ‘ ’ – space. ‘ ’ – tab for new columns and new-line for new row.

Consider the file as mentioned above.

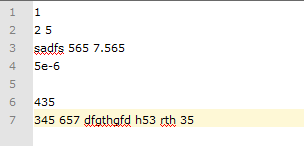
e.g.

data a(‘file.read’);

float num = a.f(3,2); // num has 7.565

Here float variable ‘num’ will be populated with number 7.565, positioned at 3rd row 2nd column. If no data is present at that place a junk number/zero will be replayed based on compiler.

**double d(unsigned int,unsigned int);**



Returns a double value positioned at the specified row and column in ‘file.read’. Note the delimiters used are ‘ ’ – space. ‘ ’ – tab for new columns and new-line for new row.

Consider the file as mentioned above.

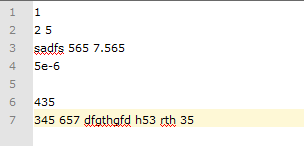
e.g.

data a(‘file.read’);

double num = a.d(4,1); // num has 5e-6

Here double variable ‘num’ will be populated with number 5e-6, positioned at 4th row 1st column. If no data is present at that place a junk number/zero will be replayed based on compiler.

**long l(unsigned int,unsigned int);**



Returns a long value positioned at the specified row and column in ‘file.read’. Note the delimiters used are ‘ ’ – space. ‘ ’ – tab for new columns and new-line for new row.

Consider the file as mentioned above.

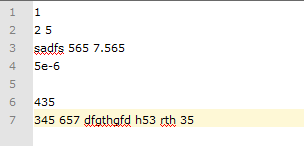
e.g.

data a(‘file.read’);

int num = a.i(2,2); // num has 5

Here long variable ‘num’ will be populated with number 5, positioned at 2nd row 2nd column. If no data is present at that place a junk number/zero will be replayed based on compiler.

**string s(unsigned int,unsigned int);**



Returns a string value positioned at the specified row and column in ‘file.read’. Note the delimiters used are ‘ ’ – space. ‘ ’ – tab for new columns and new-line for new row.

Consider the file as mentioned above.

e.g.

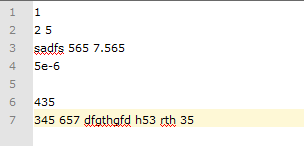
data a(‘file.read’);

string str = a.i(3,1); // str has ‘sadfs’

Here string variable ‘str’ will be populated with ‘sadfs’, positioned at 3rd row 1st column. If no data is present at that place a junk number/zero will be replayed based on compiler.

**int length();**

Gives the length of number of lines in a file that is populated with data.



e.g.

data a(‘file.read’);

int len = a.length(); // len is 7

Variable ‘len’ will be populated with number of lines in the file.

In the current situation len is 7.

**int NumOfColInRow(int row);**

Gives the number of colums in a particular line. Takes line number for input.

e.g.

data a(‘file.read’);

int len = a.NumOfColInRow(7); // len is 6

Variable ‘len’ will be populated with number of columns in row 7.

In the current situation len is 6 since there are 6 columns in line 7.

**Compilation**

**gcc –c dread.c**

**gcc –c isnum.c**

**g++ yourFile.cpp dataAdv.cpp dread.o isnum.o –lm –o executable**

**Include “dreadAdv.h” in your youtFile.cpp file.**

**This concludes the library interfaces. For any queries contact the author. Use these libraries judicially. Wish you good luck.**