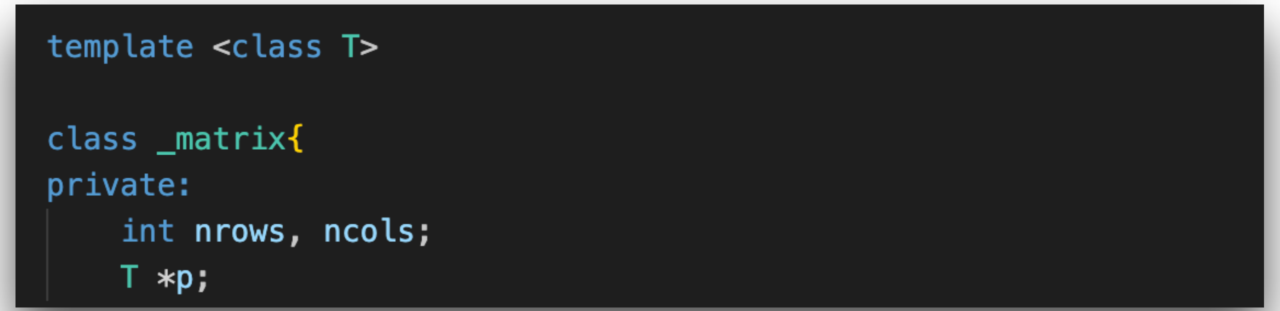
Task Report

Aryan Gholami
40030973
Course name: Principles of Programming
Professor's name: Alireza Kazemi

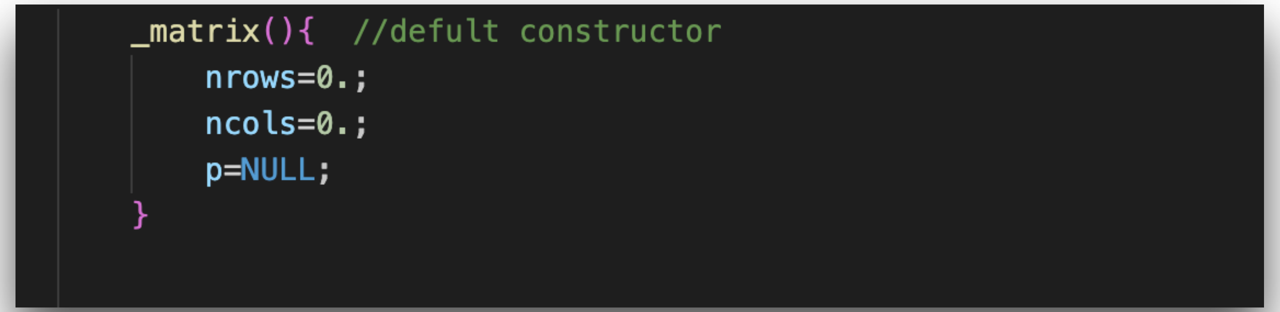
July 6th

In this program, a template has been used to support double and float data types. In the private section, variables for the number of rows and columns of the matrix have been defined, as well as a pointer to reference the data of the matrix.



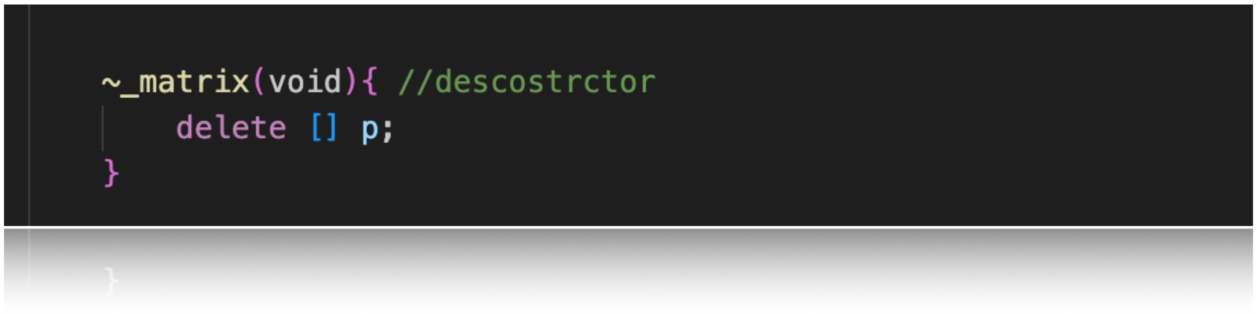
```
template <class T>

class _matrix{
private:
    int nrows, ncols;
    T *p;
```

The first function of this class is its default constructor, which defines its rows and columns as zero and points to NULL to prevent unauthorized memory manipulation by the user.



```
    _matrix(){  //defult constructor
        nrows=0.;
        ncols=0.;
        p=NULL;
    }
```

The default destructor: This member function, after the end of the program, frees the memory allocated by the program to prevent memory leaks.



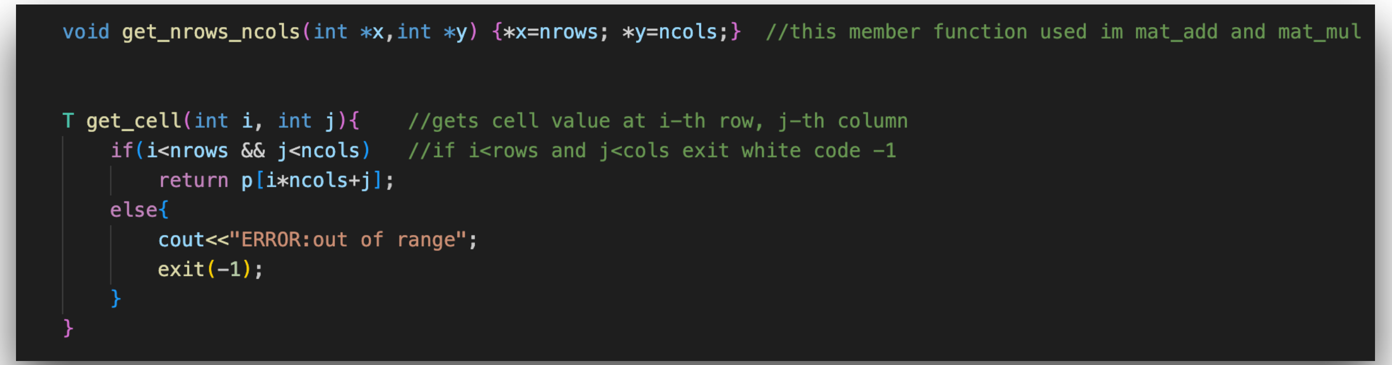
```
    ~_matrix(void){ //descostrctor
        delete [] p;
    }
```

The member function "getRowsAndColumns" receives two integer pointers and stores the numerical value of the number of rows and columns in the referenced memory.

This member function is useful in matrix multiplication and addition member functions and will be used.

The member function "get_cell" takes two integers as input, representing the row and column, and returns the value of that cell as a floating-point number (double or float).

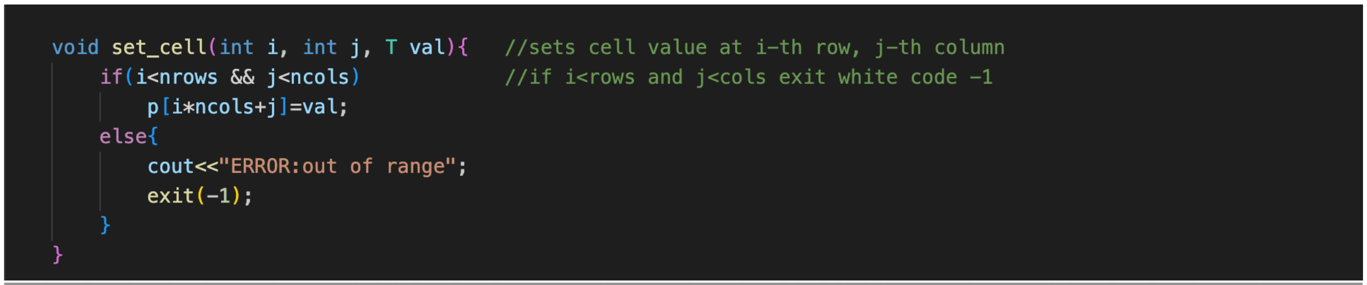
In this function, if the user enters an invalid value, the program terminates with a -1 code.



```
void get_nrows_ncols(int *x,int *y) {*x=nrows; *y=ncols;}  //this member function used im mat_add and mat_mul


T get_cell(int i, int j){    //gets cell value at i-th row, j-th column
    if(i<nrows && j<ncols)   //if i<rows and j<cols exit white code -1
        return p[i*ncols+j];
    else{
        cout<<"ERROR:out of range";
        exit(-1);
    }
}
```

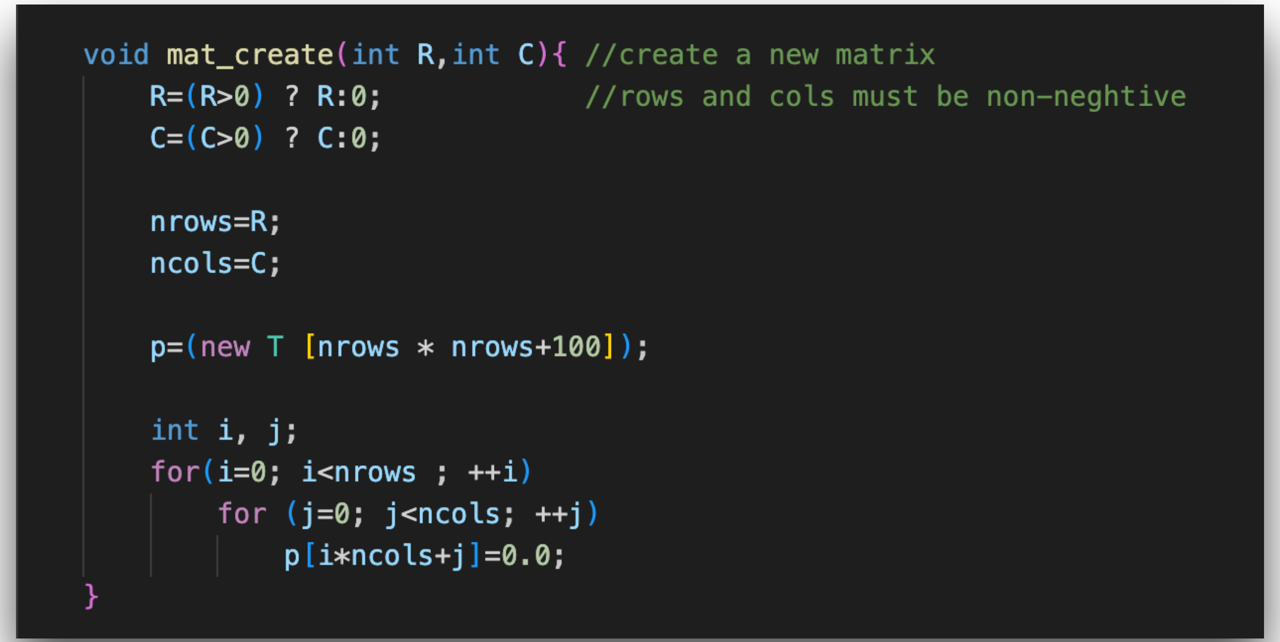
The member function "set_cell" takes two integers (representing the row and column) and one value as a float or double, and it replaces the value in the corresponding row and column.

If the user enters an invalid row or column value, the program terminates with a code of -1 to prevent unauthorized memory manipulation.



```
void set_cell(int i, int j, T val){   //sets cell value at i-th row, j-th column
    if(i<nrows && j<ncols)            //if i<rows and j<cols exit white code -1
        p[i*ncols+j]=val;
    else{
        cout<<"ERROR:out of range";
        exit(-1);
    }
}
```

The member function "mat_create" takes two non-negative integers as input, representing the row and column of a matrix. It creates memory space to store the matrix data.

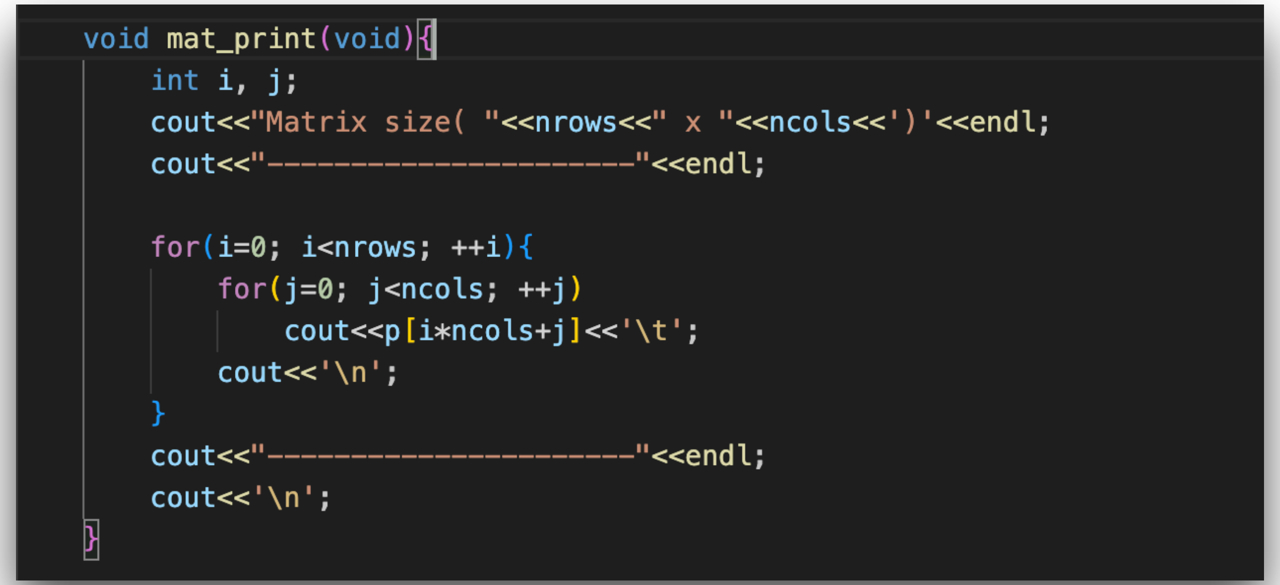


```
void mat_create(int R,int C){ //create a new matrix
    R=(R>0) ? R:0;            //rows and cols must be non-neghtive
    C=(C>0) ? C:0;

    nrows=R;
    ncols=C;

    p=(new T [nrows * nrows+100]);

    int i, j;
    for(i=0; i<nrows ; ++i)
        for (j=0; j<ncols; ++j)
            p[i*ncols+j]=0.0;
}
```

Additionally, this function allocates extra memory for a hundred float or double values to avoid memory overlap between two matrices (although I'm not sure if this is the correct solution).

In this program, memory is allocated in a row-major order. This means that memory is allocated sequentially for the elements in the first row of the matrix, followed by the elements in the second row, and so on.

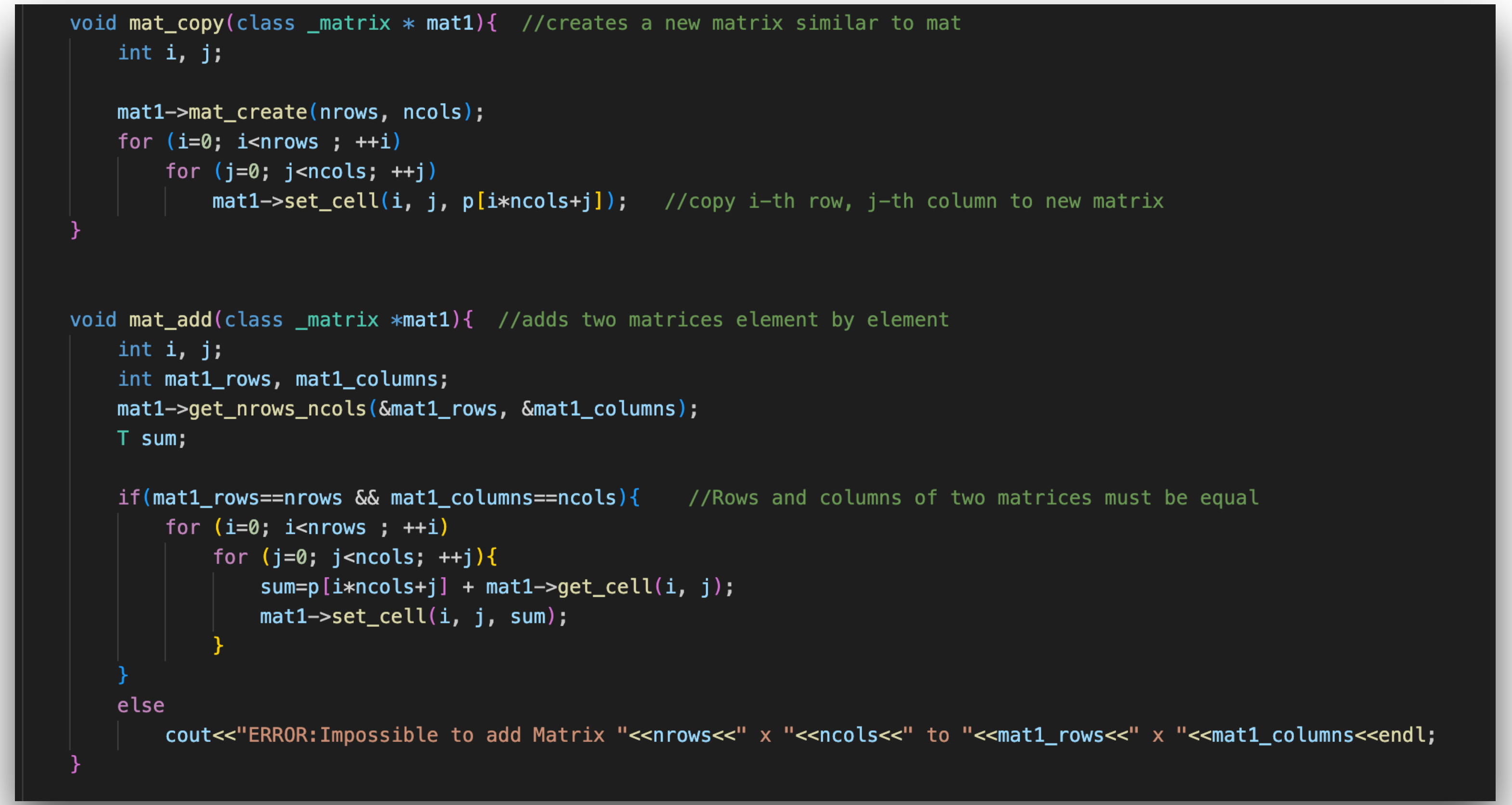
The `mat_print` member function: This function prints the data stored in the matrix in a row-by-row and column-by-column manner.



```
void mat_print(void){
    int i, j;
    cout<<"Matrix size( "<<nrows<<" x "<<ncols<<')'<<endl;
    cout<<"----------------------"<<endl;

    for(i=0; i<nrows; ++i){
        for(j=0; j<ncols; ++j)
            cout<<p[i*ncols+j]<<'\t';
        cout<<'\n';
    }
    cout<<"----------------------"<<endl;
    cout<<'\n';
}
```

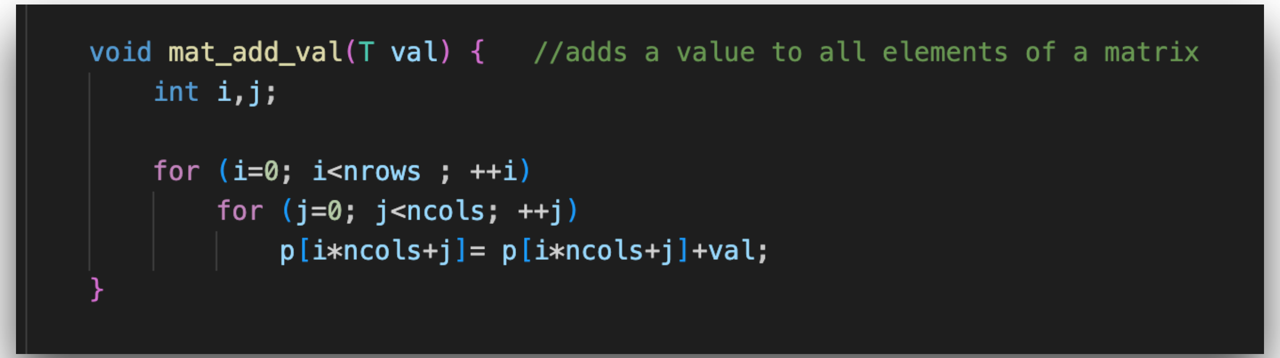
The `mat_copy` member function: This function takes the address of a matrix as input, then using the `mat_create` member function, allocates memory and copies the initial matrix values to the input address.

The `mat_add` member function: This function takes the address of a matrix as input. If matrix addition is possible, it performs the addition operation and replaces the values in the input matrix address. If matrix addition is not possible, it prints an error.



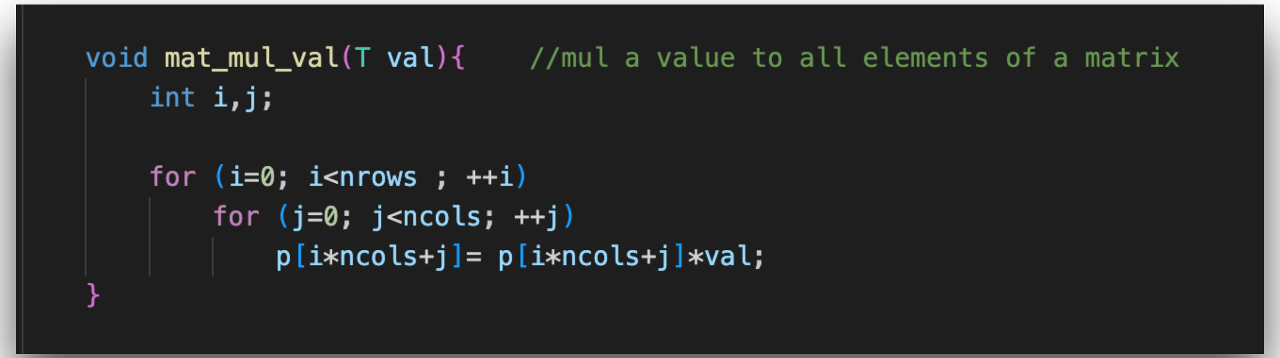
```
void mat_copy(class _matrix * mat1){  //creates a new matrix similar to mat
    int i, j;

    mat1->mat_create(nrows, ncols);
    for (i=0; i<nrows ; ++i)
        for (j=0; j<ncols; ++j)
            mat1->set_cell(i, j, p[i*ncols+j]);   //copy i-th row, j-th column to new matrix
}


void mat_add(class _matrix *mat1){  //adds two matrices element by element
    int i, j;
    int mat1_rows, mat1_columns;
    mat1->get_nrows_ncols(&mat1_rows, &mat1_columns);
    T sum;

    if(mat1_rows==nrows && mat1_columns==ncols){    //Rows and columns of two matrices must be equal
        for (i=0; i<nrows ; ++i)
            for (j=0; j<ncols; ++j){
                sum=p[i*ncols+j] + mat1->get_cell(i, j);
                mat1->set_cell(i, j, sum);
            }
    }
    else
        cout<<"ERROR:Impossible to add Matrix "<<nrows<<" x "<<ncols<<" to "<<mat1_rows<<" x "<<mat1_columns<<endl;
}
```

The `mat_add_val` member function: This function takes a float or double value as input and adds it to each element of the matrix.



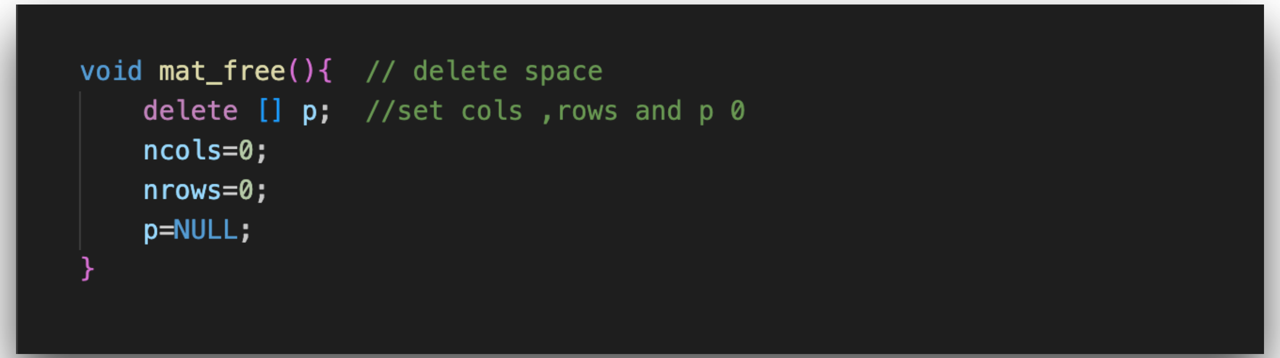
```
void mat_add_val(T val) {   //adds a value to all elements of a matrix
    int i,j;

    for (i=0; i<nrows ; ++i)
        for (j=0; j<ncols; ++j)
            p[i*ncols+j]= p[i*ncols+j]+val;
}
```

The `mat_mul_val` member function: This function takes a double or float value as input and multiplies it with each element of the matrix (scalar multiplication).



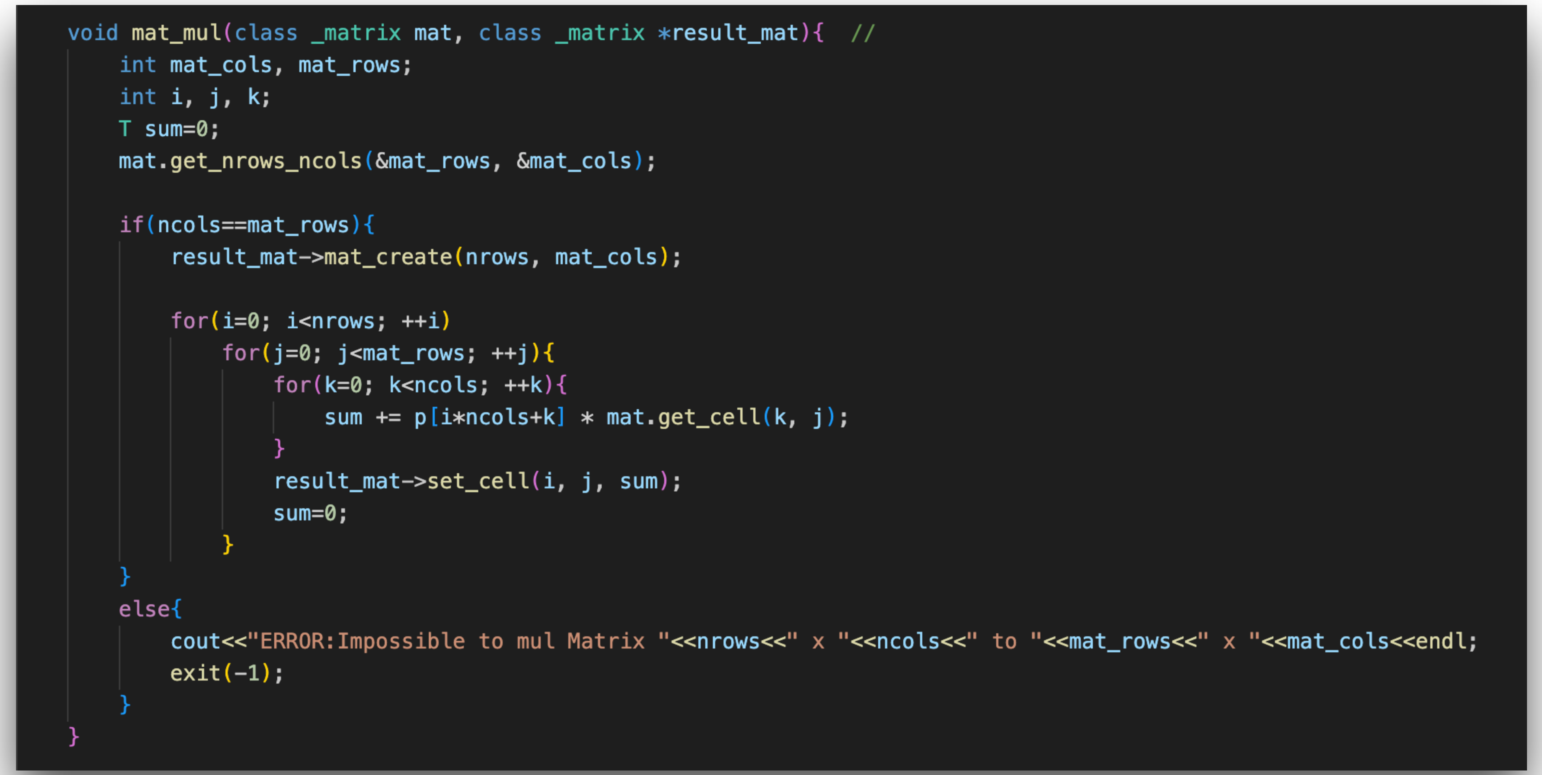
```
void mat_mul_val(T val){    //mul a value to all elements of a matrix
    int i,j;

    for (i=0; i<nrows ; ++i)
        for (j=0; j<ncols; ++j)
            p[i*ncols+j]= p[i*ncols+j]*val;
}
```

The `mat_free` member function: This function deallocates the memory allocated for the matrix, sets the row and column values to zero, and assigns the pointer to NULL to prevent any unauthorized memory access.



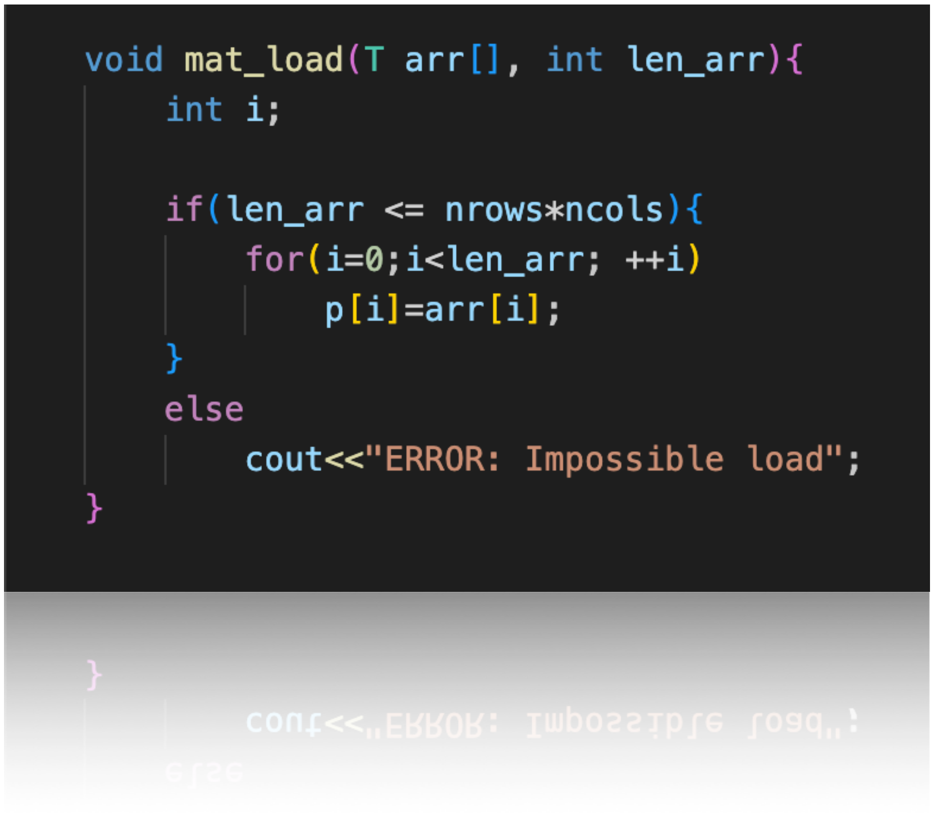
```
void mat_free(){  // delete space
    delete [] p;  //set cols ,rows and p 0
    ncols=0;
    nrows=0;
    p=NULL;
}
```

Matrix Multiplication Function: This function multiplies two matrices if possible, stores the result in a third matrix obtained using the mat_create function, and returns it. If the multiplication is not possible, the program exits with a code of -1.



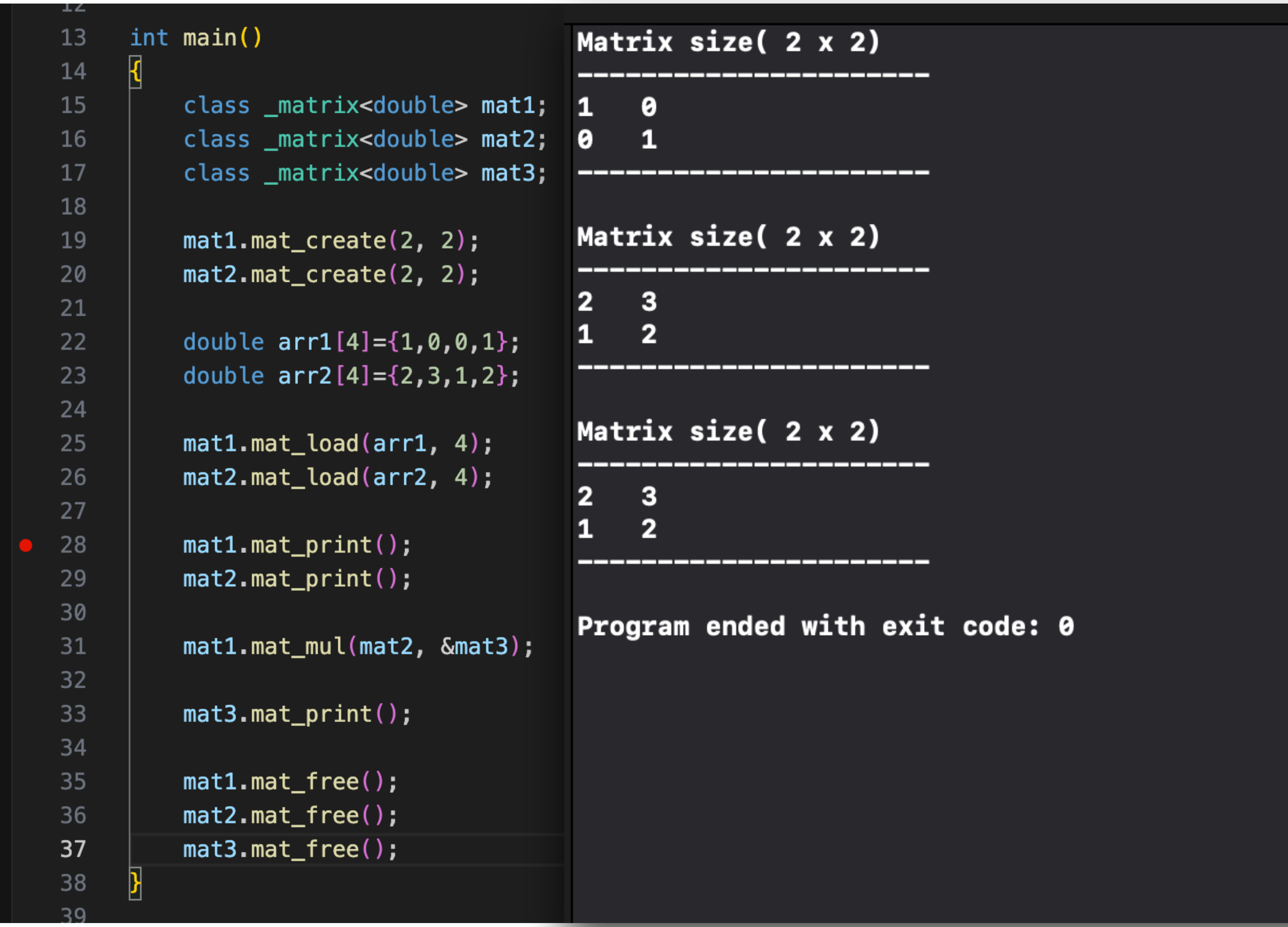
```
void mat_mul(class _matrix mat, class _matrix *result_mat){  //
    int mat_cols, mat_rows;
    int i, j, k;
    T sum=0;
    mat.get_nrows_ncols(&mat_rows, &mat_cols);

    if(ncols==mat_rows){
        result_mat->mat_create(nrows, mat_cols);

        for(i=0; i<nrows; ++i)
            for(j=0; j<mat_rows; ++j){
                for(k=0; k<ncols; ++k){
                    sum += p[i*ncols+k] * mat.get_cell(k, j);
                }
                result_mat->set_cell(i, j, sum);
                sum=0;
            }
    }
    else{
        cout<<"ERROR:Impossible to mul Matrix "<<nrows<<" x "<<ncols<<" to "<<mat_rows<<" x "<<mat_cols<<endl;
        exit(-1);
    }
}
```

The mat_load function: This function takes an array of floats or doubles along with its length and, if the length value is less than or equal to the memory allocated for the matrix, it copies the elements into the memory allocated for the matrix.



```
void mat_load(T arr[], int len_arr){
    int i;

    if(len_arr <= nrows*ncols){
        for(i=0;i<len_arr; ++i)
            p[i]=arr[i];
    }
    else
        cout<<"ERROR: Impossible load";
}
```

The code and a sample execution are provided in the two images below.



```cpp
int main()
{
    class _matrix<double> mat1;
    class _matrix<double> mat2;
    class _matrix<double> mat3;

    mat1.mat_create(2, 2);
    mat2.mat_create(2, 2);

    double arr1[4]={1,0,0,1};
    double arr2[4]={2,3,1,2};

    mat1.mat_load(arr1, 4);
    mat2.mat_load(arr2, 4);

    mat1.mat_print();
    mat2.mat_print();

    mat1.mat_mul(mat2, &mat3);

    mat3.mat_print();

    mat1.mat_free();
    mat2.mat_free();
    mat3.mat_free();
}
```

```
Matrix size( 2 x 2)
----------------------
1   0
0   1
----------------------

Matrix size( 2 x 2)
----------------------
2   3
1   2
----------------------

Matrix size( 2 x 2)
----------------------
2   3
1   2
----------------------

Program ended with exit code: 0
```