Objective:

- It should help help you understand the use of class templates.

Challenge – 1: *Generic Matrix using Generic Array* (2, 1, 2, 2, 3, .5, .5, 1, 1.5, 2, 3, 4, 2, 4, 3)

Consider the following interface for `class Matrix` which you need to implement as class template. While implementing any function for the `class Matrix` or while defining its data members you must not use primitive arrays if you need to use array anywhere in your code then it must only be the generic `class Array` discussed in class whose interface is given below.

```
class Matrix
{
    //Decide the data members yourself
public:
     1.  Matrix();
     2.  Matrix(int r, int c);
     3.  Provide Copy Constructor, assignment operator and destructor in your class
         if needed.
     4.  //Operator [] for your class both const/non-const version. Decide the
         prototype yourself.
     5.  int getRows() const;
     6.  int getColumns() const;
     7.  void print() const;
     8.  operator == to check whether calling and received objects are equal or not?
         Decide the prototype yourself.
     9.  Transpose function which returns the transpose of calling object without the
         *this object. Decide the prototype yourself.
    10.  Provide isSymmetric function which checks whether calling object is
         symmetric or not without chaging the *this object. Decide the prototype
         yourself. If Aᵗ = A, then return true otherwise false.
    11.  void resize(int newRow, int newCol);
         resize the matrix according to new row and column. Make sure that the old
         matrix elements should be preserved in the new resized matrix if possible.
         If new row or column value is zero or negative then make the matrix
         rows and columns set to zero.
    12.  Provide function to add calling and received object (matrices). Your
         function should return addition result without doing any change in calling
         and received object.
    13.  Just like add function provide a function to multiply two matrices.
};
```

```
template<typename T>
class Array
{
    T * data;
    int  capacity;
    int isValidIndex(int) const
public:
    Array(int = 0);
    ~Array();
    Array(const Array<T> &);
    Array<T> & operator = (const Array<T> &);

    T & operator [] (int index)
    const T & operator [] (int) const
    int getCapacity() const
    void reSize ( int newCap )
};
```