ASSIGNMENT 1

*Implement the types using **classes**. Represent the types using sequences of same-type elements stored in **vectors** (vector<> in C++).*

*Create a main program with a **menu** to demonstrate the services (all the methods) a class provides in arbitrary order (based on the selections of the user). Make the main program instantiate an object of the class. The methods of the class can be called through the menu items. Print the state of the object either after the completion of each menu item or through another menu item for printing. If there are methods or friend functions that need multiple objects (like adding two matrices, for example), the main program should make it possible to create and print these objects through the menu.*

*Implement **unit tests** which should be run automatically.*

1. Implement the chessboard matrix type which contains integers. In these matrices, every second entry is zero. The entries that can be nonzero are located like the same-colored squares on a chessboard, with indices (1, 1), (1, 3), (1, 5), ..., (2, 2), (2, 4), ... The zero entries are on the indices (1, 2), (1, 4), ..., (2, 1), (2, 3), ... Store only the entries that can be nonzero in row-major order in a sequence. Don't store the zero entries. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a shape of m by n).
2. Implement the X matrix type which contains integers. These are square matrices that can contain nonzero entries only in their two diagonals. Don't store the zero entries. Store only the entries that can be nonzero in a sequence. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a square shape).
3. Implement the N matrix type which contains integers. These are square matrices that can contain nonzero entries only in their first and last column, and in their main diagonal. Don't store the zero entries. Store only the entries that can be nonzero in a sequence. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a square shape).

```
X 0 0 0 0 0 0 0 X
X X 0 0 0 0 0 0 X
X 0 X 0 0 0 0 0 X
X 0 0 X 0 0 0 0 X
X 0 0 0 X 0 0 0 X
X 0 0 0 0 X 0 0 X
X 0 0 0 0 0 X 0 X
X 0 0 0 0 0 0 X X
X 0 0 0 0 0 0 0 X
```

4. Implement the block matrix type which contains integers. These are square matrices that can contain nonzero entries only in two blocks on their main diagonal. Let the size of the first and second blocks be b1 and b2, where $1 \leq b1,b2 \leq n-1$ and $b1+b2=n$ (in the example, $b1=4$ and $b2=5$). Don't store the zero entries. Store only the entries that can be nonzero in a sequence or two smaller matrices. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a square shape).

```
X X X X 0 0 0 0 0
X X X X 0 0 0 0 0
X X X X 0 0 0 0 0
X X X X 0 0 0 0 0
0 0 0 0 X X X X X
0 0 0 0 X X X X X
0 0 0 0 X X X X X
0 0 0 0 X X X X X
0 0 0 0 X X X X X
```

5. Implement the set type which contains integers. Represent the set as a sequence of its elements. Implement as methods: inserting an element, removing an element, returning whether the set is empty, returning whether the set contains an element, returning a random element without removing it from the set, returning the largest

element of the set (suggestion: store the largest entry and update it when the set changes), printing the set.

6. Implement the set type which contains integers. Represent the set as a sequence of its elements. Implement as methods: inserting an element, removing an element, returning whether the set is empty, returning whether the set contains an element, returning a random element without removing it from the set, returning the number of even numbers in the set (suggestion: store the number of even numbers and update it when the set changes), printing the set.
7. Implement the set type which contains integers. Represent the set as a sequence of its elements. Implement as methods: inserting an element, removing an element, returning whether the set is empty, returning whether the set contains an element, returning a random element without removing it from the set, returning the sum of the numbers in the set (suggestion: store the sum and update it when the set changes), printing the set.
8. Implement the bag type which contains integers. Represent the bag as a sequence of (element, frequency) pairs. Implement as methods: inserting an element, removing an element, returning the frequency of an element, returning the most frequent element from the bag (suggestion: store the most frequent element and update it when the bag changes), printing the bag.
9. Implement the bag type which contains integers. Represent the bag as a sequence of (element, frequency) pairs. Implement as methods: inserting an element, removing an element, returning the frequency of an element, returning the number of elements which occur only once in the bag (suggestion: store the number of these elements and update it when the bag changes), printing the bag.
10. Implement the bag type which contains integers. Represent the bag as a sequence of (element, frequency) pairs. Implement as methods: inserting an element, removing an element, returning the frequency of an element, returning the largest element in the bag (suggestion: store the largest element and update it when the bag changes), printing the bag.
11. Implement the polynomial type. Represent a polynomial as a sequence of its real-valued coefficients. Implement as methods: adding two polynomials, multiplying two polynomials, evaluating the polynomial (substituting a value to the variable).