Arrays, Vectors, Matrices

- Goal:
  - Scientific Eng.
  - Numerical computations
  - Array are very efficient way of organizing data since accessing array elements requires $O(1)$.

- Characteristics of an array:
  - Arrays are one the main data organization
  - Arrays group values and permit fast access by numeric index
  - Characteristics:
    - **Base @**: Address of the first element of the array
    - **Size**: Size of each element of the array
    - **Type**: The type of each element
    - **Dimensions**: For each dimension of the array:
      - **Upper bound** of the index range
      - **Lower bound** of the index range
    - Internal representation of an array
      - Memory cells have to be reserved for the array:
is achieved by declaration statements

- Requires a mapping or accessing function to be decoded
- uses the characteristics of the array

 o In Java:
   - An array of size n: int [10] myarray;
   - the lower bound is 0 and the upper bound is 9

**Single Dimensional Arrays: Vectors**

 o Example: Let MT be a one dimensional array: int MT [n];

| MT [0] | MT [1] | …   | MT [i] | …   | MT [n-1] |

Where is the location of MT [i] ?

 o **Mapping:**

   - Let size_of_element be the size of each element (Type of the array)
   - The mapping function is:
     
     @ of MT [0] is at Base.
     
     @ of MT [2] is at Base+ size_of_element
     
     …
     
     @ of MT [i] is at Base+(i-lower_bound)* size_of_element
- In Java, C, C++ where the lower_bound is 0:

The @ of MT[i] = base address + i * size_of_element

- **Two-dimensional arrays: Matrices**

  - Two-dimensional arrays are also called table.
  - **Requires two dimensions:**
    - Rows
    - Columns
  - **Internal Representation:**
    - How is a 2-dimensional array stored in the sequential memory?
    - **Representation:**
      - **Abstraction: User’s View**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>-10</td>
<td>-20</td>
<td>-30</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

- **Implementation: System’s View**

  - **Two common schemes:**
    - **i. Row major order:** rows are placed one after another in memory. Examples: Java, C, C++, Pascal, etc.
    - **ii. Column major order:** columns are placed one after another in memory. Example: Fortran.
- **Example:**
  
  ✓ **Row Major:**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>-20</td>
<td>-30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  ✓ **Column Major:**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>-20</td>
<td>-30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-10</td>
<td>5</td>
<td>20</td>
<td>-20</td>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>
**Abstraction Layout:**

<table>
<thead>
<tr>
<th>M[0][0]</th>
<th>...</th>
<th>M[0][j]</th>
<th>...</th>
<th>M[0][n-1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>M[i][0]</td>
<td>...</td>
<td>M[i][j]</td>
<td>...</td>
<td>M[i][n-1]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>M[n-1][0]</td>
<td>...</td>
<td>M[n-1][j]</td>
<td>...</td>
<td>M[n-1][n-1]</td>
</tr>
</tbody>
</table>

**Mapping:**

- **Row Major:** MT[i][j] is mapped to:
  \[ \text{@ of MT[i][j]} = \text{Base} + (i \times n + j) \times \text{size of element} \]

- **Column Major:** MT[i][j] is mapped to:
  \[ \text{@ of MT[i][j]} = \text{Base} + (j \times n + i) \times \text{size of element} \]
This array can be interpreted as \( p \) 2D arrays of dimensional \( n \times m \):

The address of \( a[i][j][k] \) is:

\[
\text{Base} + (i \cdot nm + j \cdot n + k) \cdot \text{size\_of\_element}
\]
• **Sparse Arrays**
  o Definition:
    ▪ Arrays with many zero elements.
  o Types:
    ▪ Arrays with one or more blocks of non-zero elements.
    ▪ Randomly Distributed non-zero elements
• **Triangular Matrices:**
  o Upper and Lower
  o Lower:
    ▪ Let MT be a one dimensional array: int MT [n][n];

\[
\begin{align*}
    MT[i][j] &= 0 \text{ if } i < j \\
    MT[i][j] &\neq 0 \text{ if } i \geq j
\end{align*}
\]

<table>
<thead>
<tr>
<th>MT [0][0]</th>
<th>…</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MT [i][0]</td>
<td>MT [i][j]</td>
<td>…</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MT [n-1][0]</td>
<td>MT [n-1][j]</td>
<td>MT [n-1][n]</td>
<td></td>
</tr>
</tbody>
</table>
• **Implementation:**
  
  - Store only the lower half
  - Map MT into M’ using **row-major scheme**
  - MT[i][j] is located in M’[1+\(\frac{i}{2}\)+j]

• **Applications:**
  
  - Symmetric Matrices:
    - M is a symmetric matrix iff M[i][j] = M[j][i] for i,j
  - Mapping:
    - If \(i \geq j\) then
      
      use the mapping function of the lower triangular matrix
    
    else
      
      interchange i and j in the mapping function of the lower triangular matrix
• Randomly Distributed non-zero elements
  
  - Characteristics of an element $A[i][j]$ of a sparse matrix $A$:
    - row and column positions: $i$ and $j$
    - The value of the entry: $value$
  
  - Implementation:
    - Store the non-zero elements in an array $DA$ s.t. the first entry contains the following information:
      - First dimension
      - Second dimension
      - Max. number of elements: $\text{maxnonzero}$
      - $\text{int } DA[\text{maxnonzero}][3]$;
• Examples:

```java
public class Matrix {
    private int rows=0, columns=0;
    private double[][] data;

    // create r-by-c matrix of 0's
    public Matrix(int r, int c) {
        this.rows = r;
        this.columns = c;
        data = new double[r][c];
    }

    // create matrix based on 2d array
    public Matrix(double[][] d) {
        rows = d.length;
        columns = d[0].length;
        if ( rows > 0 && columns > 0 ){
            data = new double[rows][columns];
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < columns; j++)
                    data[i][j] = d[i][j];
        } else
            System.exit(0);
    }

    // Accessor method, get the number of rows in the Matrix.
    public int getRows() {
        return rows;
    }

    // Accessor method, get the number of columns in the Matrix.
    public int getColumns() {
        return columns;
    }

    // Accessor method, get an element of the Matrix.
    public double getEntry(int r, int c) {
        r = checkRowIndex(r);
        c = checkColumnIndex(c);
        return data[r][c];
    }

    // Mutator: set an element of the Matrix.
    public Matrix setEntry(int r, int c, int val) {
        r = checkRowIndex(r);
        c = checkColumnIndex(c);
        data[r][c] = val;
        return this;
    }

    // Matrix addition
    public Matrix add(Matrix b) {
        AddcheckSize(b);
        Matrix result = new Matrix(b.getRows(), b.getColumns());
        for (int r = 0; r < rows; r++)
            for (int c = 0; c < columns; c++)
                result.data[r][c] = data[r][c] + b.data[r][c];
        return result;
    }
}
```
Arrays, A. Bellaachia Page 11

```java
// Matrix Subtraction
public Matrix sub(Matrix b) {
    //Implement this method
}

// Matrix Transpose
public Matrix transpose(Matrix b) {
    //Implement this method
}

// Make sure the row parameter r is valid (i.e., 1 <= r <= rows), then performs
private int checkRowIndex(int r) {
    if ( r < 1 || r > rows )
        System.out.println("Row index "+ r + " out of range");
    return r - 1;
}

// Make sure the row parameter r is valid (i.e., 1 <= r <= rows), then performs
private int checkColumnIndex(int c) {
    if ( c < 1 || c > columns )
        System.out.println("Row index "+ c + " out of range");
    return c - 1;
}

// Make sure the matrices have the same dimensions.
private void AddcheckSize(Matrix b) {
    if ( rows != b.rows || columns != b.columns ){
        System.out.println("You cannot add two matrices of different sizes");
        System.exit(0);
    }
}

// print matrix to standard output
public void printMatrix() {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++)
            System.out.printf("%9.4f ", data[i][j]);
        System.out.println();
    }
}
```

○ Programming Assignment:
  - Design and implement the missing operations in the Matrix ADT:
    - Subtraction
    - Multiplication
    - Transpose
  - Test your implementation.