Table of Topics

UNIT 2: Array Operations

Introduction to Arrays, Representation of Linear Arrays in Memory – Dynamically allocated arrays, Traversing, and other Array Operations, Multi-dimensional Arrays, Polynomials and Sparse Matrices

Overview

In this unit we will be studying about one of the simple linear data structures, *Array*. We will be learning about the creation, memory allocation for it, and various operations on it namely; traversing, insertion, deletion, searching, and sorting. We will also learn about multi-dimensional arrays, polynomials, and sparse matrices.

Objectives

In this Unit you will learn –

1. Basics of the data structure - Array
2. How an array is created and its memory allocations
3. Operations on array
4. Different types of arrays

Learning Outcomes

At the end of this Unit you would -

- ✓ Understand the static linear data structure, Array
- ✓ Will understand the use and applicability of it
- ✓ Will be able to use different operations on array
- ✓ Understand the concept of multi-dimensional arrays, and
- ✓ Know about polynomials, and sparse matrices.

Unit Prerequisites

This unit requires a prior knowledge of basic Java programming.

Pre-Unit Preparatory Material

- ➢ Refresh your knowledge on Java [if any specific link is there from previous courses, which can be given here]

2.1 Introduction

Arrays are the static linear data structure available. We can consider arrays as a closed container with a fixed memory size to hold the same type of values. Array is the simplest non-primitive linear static data structure which stores the similar values (collection of same data type values in Java) in a sequential order. The values are stored continuously in the memory. The items stored in the array are called as *elements*, and each element is identified through its *index* value. The index always starts from 0 and has the value up to *n-1* for an array of size *n*.

2.2 Creation of an Array and Accessing the Elements

Java arrays are declared, initialized, and allotted memory dynamically. We can see these operations for one-dimensional arrays in the following sub-sections.

2.2.1 Array Declaration

Arrays are declared in Java as follows.

Syntax:

```
<data-type>[ ] <arrayName>; //better way
<data-type> <arrayName>[ ]; //earlier way, less used
```

The square brackets indicate that an array of that particular data type is being formed, with the mentioned arrayName.

Example:

```
int[ ] empNo;           // an array of employee numbers of integer data type
```

Array declaration doesn’t create / allocate memory for the actual array; instead it tells the compiler that the particular array variable exists.

2.2.2 Array Instantiation

Array instantiation is the process of creating memory space for the array that is declared. Array declaration creates a reference for the array variable. The ***new*** keyword is used to allot memory for arrays in heap memory space.

Syntax:

```
<arrayName> = new <data-type>[size];
```

Example:

```
empNo = new int[10];        // allots 10 integer space for the array empNo // on
                            heap memory
```

The memory is allotted in such a way it cannot be changed in size later; hence we say arrays have static memory space, i.e., fixed size. Also, the elements of the array are stored on contiguous memory locations.

2.2.3 Array Initialization

Array initialization is the process of assigning values to the array elements. The new keyword will automatically create a null value for the elements in the array. It will be zero for numeric data types, and null for the other references.

Array can be initialized by accessing each element of the array using its index.

Syntax:

```
<arrayName>[index] = <value>;
```

Example:

```
empNo[0] = 101;
empNo[1] = 102;
```

Thus, creation of an array happens in two steps; array declaration which creates reference for the array variable, and array instantiation which creates memory space for the array elements in the heap memory. Once the array is created, values can be assigned to it through initialization.

The array declaration and instantiation can be done together too, as shown in the following example.

Example:

```
int[ ] empNo = new int[10];        // array creation in a single step
```

Also, initialization can also be carried together, as shown below.

Example:

```
int[ ] empNo = new int[ ] {101, 102, 103, 104, 105};
```

Here, the array literals are used to directly create and initialize the array. The size of the array is taken as 5 from the number of literals / elements supplied to it.

2.2.4 Traversing Array Elements

Traversing an array means visiting each and every element of the array. It can be done through *a* loop or *foreach* loop in Java. Traversing is a very important operation of any data structure as it helps to access, and work on all the elements of that particular data structure.

2.2.4.1 Traversing Using *for* Loop

The *for* loop can be used to traverse through the elements of an array. It will visit each and every element of the array if its step value is 1 and the initial value is set as 0.

Example:

```
// for loop to access each element and initialize
for(int i = 0; i < empNo.length; i++)
        empNo[i] = 100+i+1;
```

The above code snippet starts with the index 0 of array *empNo* and for every run of for loop gets incremented to the next index, and traverses up to *length-1* of the array element. The *<arrayName>.length* in-built function is used to find the length of the array. The index values will be 1 less than the length as it starts from 0.

2.2.4.2 Traversing Using *foreach* Loop

Newer versions of Java provide the *foreach* loop, in which without introducing initial value and step size, we are able to traverse all the elements of the array.

Example:

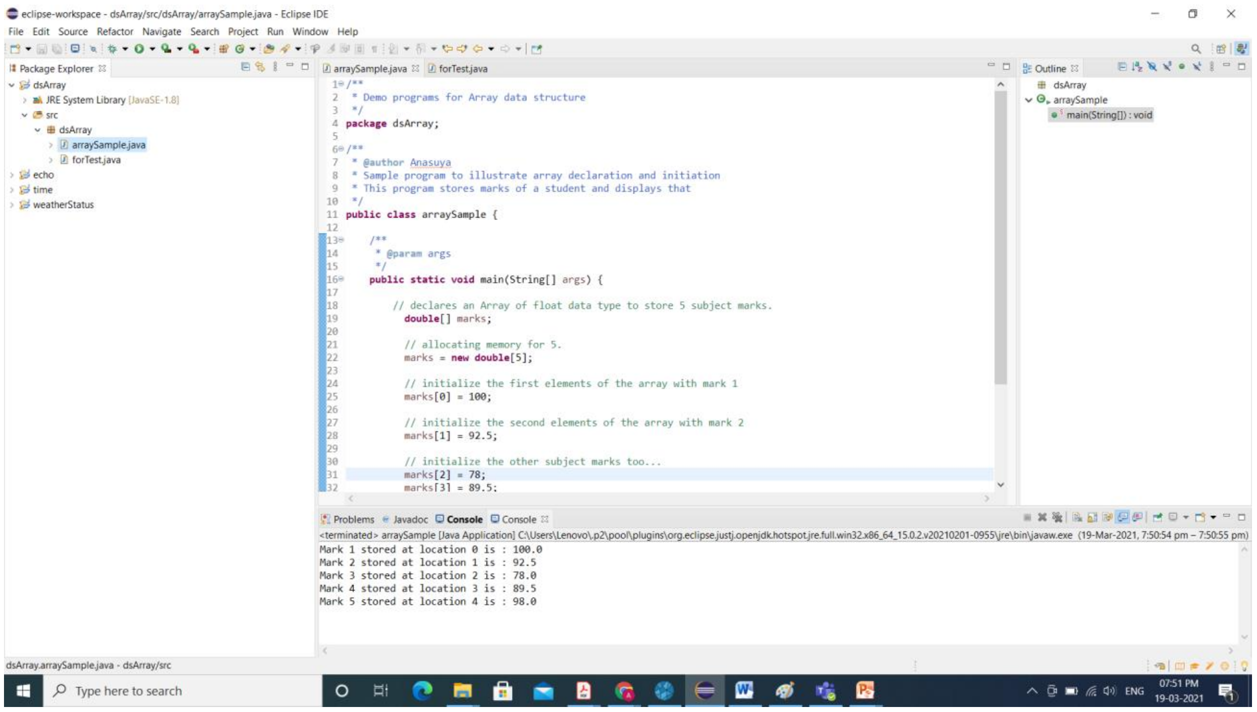
```
// access each element to display it using foreach loop
for(int element: empNo)
        System.out.println(element);
```

The above code snippet introduces just one integer variable *element*, and each and every item of the array *empNo* is visited in the order of its index.

2.2.5 Array Handling

The earlier sub-sections defined an array and showed how to traverse its elements and access elements from it. This section depicts a few programs and gives explanations of them.

The sample programs shown here are executed with eclipse workspace for Java. A package named *dsArray* is created to enclose all the programs of this unit. Figure 2.1 shows the eclipse workspace.



```java
/**
 * Demo programs for Array data structure
 */
package dsArray;

/**
 * @author Anasuya
 * Sample program to illustrate array declaration and initiation
 * This program stores marks of a student and displays that
 */
public class arraySample {

    /**
     * @param args
     */
    public static void main(String[] args) {

        // declares an Array of float data type to store 5 subject marks.
          double[] marks;

          // allocating memory for 5.
          marks = new double[5];

          // initialize the first elements of the array with mark 1
          marks[0] = 100;

          // initialize the second elements of the array with mark 2
          marks[1] = 92.5;

          // initialize the other subject marks too...
          marks[2] = 78;
          marks[3] = 89.5;
```

```
<terminated> arraySample [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (19-Mar-2021, 7:50:54 pm – 7:50:55 pm)
Mark 1 stored at location 0 is : 100.0
Mark 2 stored at location 1 is : 92.5
Mark 3 stored at location 2 is : 78.0
Mark 4 stored at location 3 is : 89.5
Mark 5 stored at location 4 is : 98.0
```

Figure 2.1: Eclipse-Workspace for Java

Program 2.1: Java program to illustrate array creation and accessing elements

```java
/**
 * Demo programs for Array data structure
 */
package dsArray;

/**
 * @author Anasuya
 * Sample program to illustrate array declaration and initiation
 * arraySample class stores marks of a student and displays that
 */
public class arraySample {

        /**
         * @param args
         */
        public static void main(String[] args) {

                // declares an Array of float data type to store 5 subject marks
            double[] marks;

            // allocating memory for 5
22                  marks = new double[5];
23
24                  // initialize the first elements of the array with mark 1
25                  marks[0] = 100;
26
27                  // initialize the second elements of the array with mark 2
28                  marks[1] = 92.5;
29
30                  // initialize the other subject marks too...
31                  marks[2] = 78;
32                  marks[3] = 89.5;
33                  marks[4] = 98;
34
35                  // accessing the elements of the specified array
36                  for (int i = 0; i < marks.length; i++)
37                    System.out.println("Mark " + (i+1) + " stored at location " + i +  " is : "+ marks[i]);
38
39              }
40
41      }
```

Output:

```
Mark 1 stored at location 0 is : 100.0
Mark 2 stored at location 1 is : 92.5
Mark 3 stored at location 2 is : 78.0
Mark 4 stored at location 3 is : 89.5
Mark 5 stored at location 4 is : 98.0
```

In Program 2.1, the multi-comment option is included before every function and interface to describe its operation, and the parameters included in it. Lines 1 – 3 say that the *dsArray* is the common interface for array programs from this unit. Lines 6 – 10 describe the class *arraySample*; the operations carried out in it. Lines 13 – 15 include the parameter used in *main()*, the *args* using *@param*. Single line comments are used before statements to explain their actions. It is good programming practice to include detailed comments before each method or classes or interfaces, which help in maintaining any system on a later date. In Line 19, *double[] marks;* declares an array of type *double*. In Line 22, memory is allotted to it through the *new* keyword, specifying size of 5. Lines 25, 28, 31 – 33 assign the values for array elements. The *for* loop in Lines 36 – 37 traverse through each element of the array and prints the corresponding value. Output of the program is shown below the program, which tells the index location as well as the value of the element.

Program 2.2: Java program to test *for* loop and *foreach* loop

```
package dsArray;

public class forTest {

        public static void main(String[] args) {
                // create an array
                int[] empNo = new int[5];

9                       // for loop to access each element and initialize
10                      for(int i = 0; i < empNo.length; i++)
11                              empNo[i] = 100+i+1;
12
13                      // access each element to display it using foreach loop
14                      for(int element: empNo)
15                              System.out.println(element);
16
17              }
18
19      }
```

Output:

```
101
102
103
104
105
```

Program 2.2 showcases the use of *for* loop and *foreach* loop. For simplicity, multi-comment option before each method is avoided in this program. The creation of the array is carried out in a single step as shown in Line 7. The *for* loop in Lines 10 – 11, initializes values to the *empNo*. The *foreach* loop is used to access them and print the employee numbers, which is shown in Lines 14 – 15.

2.3 Classification of Arrays

Arrays are broadly classified into one-dimensional and multidimensional arrays. One-dimensional array has a single row of elements, while the multi-dimensional consists of separate arrays as each element. Matrix is an example of a multidimensional array. Figure 2.2 shows a one-dimensional array, whereas Figure 2.3 depicts a multi-dimensional array.

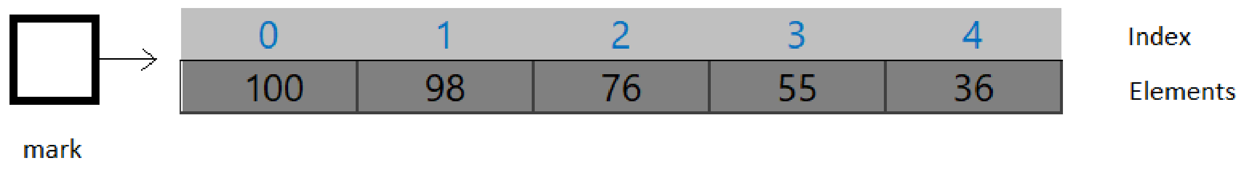


Figure 2.2: One-dimensional array

Here, *mark[ ]* is a single dimensional / one-dimensional array having 5 elements in it. The index values range from 0 to 4, and using a particular index, the corresponding element in the array can be accessed.

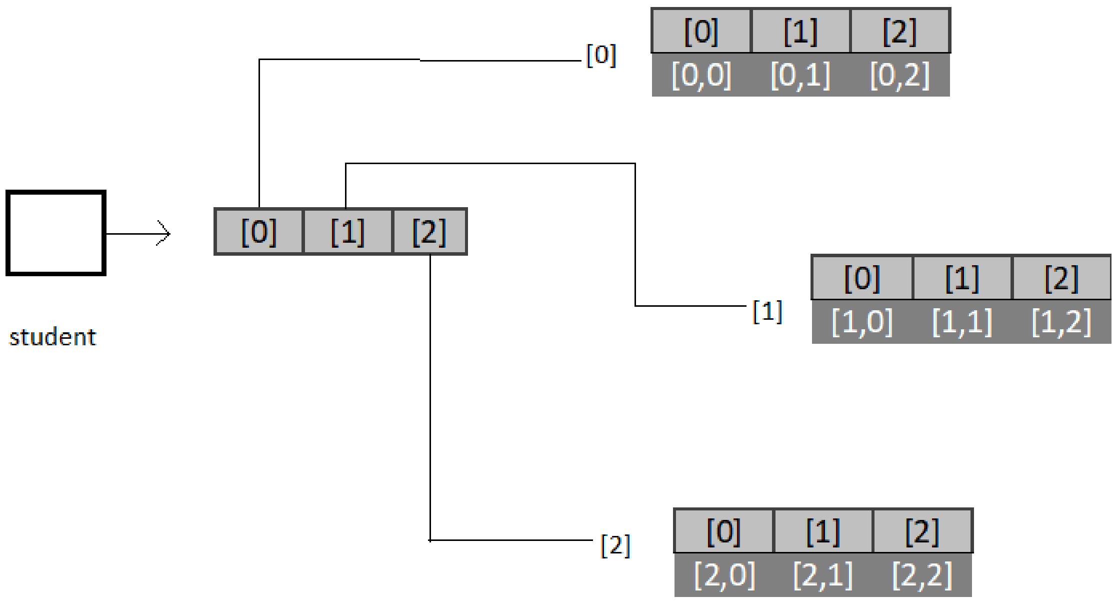


Figure 2.3: Multi-dimensional array

In a multidimensional array, each element acts as a reference to an array itself. In Figure 2.3, a multidimensional array *student* is shown. Each element of it consists of say marks of the students, thus the mark array for the corresponding student is stored on each location / index of the student array. To access the 3rd mark of the first student, we should access the index [0, 2], i.e., from *student* array 0th position [0], and in that the *mark* array is stored, and in *mark* array the 3rd position [2]; results in the two-dimensional index [0, 2]. We can imagine the student array as a table having multiple columns and rows. In general, the multi-dimensional arrays can be visualized as multi-dimensional matrices.

Section 2.2 described in detail the creation, memory allocation, traversing and array handling for single-dimensional array data structure. Operations can be extended to multi-dimensional by including the different number of dimensions into different [ ]. For example, a two-dimensional array can be declared and created as follows.

2.3.1 Multi-dimensional Array Creation

The syntax for array declaration and memory allocation of a two-dimensional array is as follows.

Syntax: [Two-dimensional array declaration and memory allocation]

```
<data-type>[ ][ ] <arrayName> = new <data-type>[row_size][column_size];
```

Example:

```
int[ ][ ] student = new int[3][3];
```

In general, an N-dimensional array can be created as follows.

Syntax:

<data-type>[ ][ ]…[ ] <arrayName> = new <data-type>[size1][size2]…[sizeN];
(N-dimensions)

Similarly, all operations can be done on multi-dimensional arrays just like the single-dimensional arrays.

2.4 Array Operations

We have seen how to do array traversing in Section 2.2.4, and the other operations which can be carried out on arrays will be discussed in this section.

2.4.1 Insertion

We can change the values of elements (replace elements) from the array, but we cannot insert / add a new element to an array, as its memory is fixed.
Program 2.3 shows a sample.

Program 2.3: Java program to replace an element in an array

```java
package dsArray;

import java.util.Scanner; // Needed to be imported for using Scanner class

//Program to check insertion of elements into an array
public class arrayInsertion {

		public static void main(String[] args) {
			// Declare an array
			int[] sample1;

			// Scanner object to be created using java.util class
			Scanner sO= new Scanner(System.in);	//System.in is a standard input stream

			// Get the size of array
			System.out.print("Enter size of array: ");
			int size = sO.nextInt();

			// Dynamically allot memory to the array
			sample1 = new int[size];

			// Insert values to the elements/ assign values
			for(int i = 0; i < sample1.length; i++) {
				System.out.println("Enter input "+ (i+1) + " : ");
				sample1[i] = sO.nextInt();
			}
```

```

        // Print the array
        for(int i = 0; i < sample1.length; i++)
            System.out.println("Element "+ (i+1) + " : " + sample1[i]);

        // Replace first element of the array
        sample1[0] = sample1[0] + 50;

        // Print the updated array
        System.out.println("Updated array...");
        for(int i = 0; i < sample1.length; i++)
            System.out.println("Element "+ (i+1) + " : " + sample1[i]);

        // Close the scanner object
        sO.close();

    }

}
```

Output:

```
Enter size of array: 3
Enter input 1 :
11
Enter input 2 :
23
Enter input 3 :
12
Element 1 : 11
Element 2 : 23
Element 3 : 12
Updated array...
Element 1 : 61
Element 2 : 23
Element 3 : 12
```

2.4.2 Deletion

It is not possible to delete the memory space for an array as the memory is fixed. But to delete an element from an array we need to use any of the following methods.

1. Using another array
2. Using arrayList

Using another array

Deletion of an element using another array can also be done by two methods; first, the elements of the original array can be copied to another array in such a manner avoiding the element to be removed / deleted. And this can be achieved by creating an array of *length – 1* from the original array. Second method is to use the Java 8 streams. Here, the array is first converted into a stream, the required element is deleted and the values copied to another array. Program 2.4 shows deleting an element using the first method.

Program 2.4: Java program to delete an element from an array

```java
package dsArray;

public class arrayDeletion {

		public static void main(String[] args) {
			// Create an array
			int[] sample2 = new int[3];

			// Insert element values
			sample2[0] = 12;
			sample2[1] = 32;
			sample2[2] = 11;

			// Print the array
			for(int i = 0; i < sample2.length; i++)
				System.out.println("Element "+ (i+1) + " : " + sample2[i]);

			// Print the array length
			System.out.println("Length of original array: " + sample2.length);

			// To delete element 2
			int rm_index = 2;

	    // display index
	    System.out.println("Element to be removed at index: " + rm_index);

	    // if array is empty or index is out of bounds, removal is not possible
	    if (sample2 == null || rm_index >= sample2.length)
	        System.out.println("No removal operation can be performed!!");

	    // Create a another array of size one less than original array
	    int[] proxyArray = new int[sample2.length - 1];

	    // copy all the elements in the original to proxy array except the one at index
	    for (int i = 0, k = 0; i < sample2.length; i++) {

	        // check if index is crossed, continue without copying
	        if (i == rm_index) {
	            continue;
	        }

	        // else copy the element
	        proxyArray[k++] = sample2[i];
	    }

	   // Print the copied proxy array
	   System.out.println("Array after removal operation: ");
	   for(int i = 0; i < proxyArray.length; i++)
				System.out.println("Element "+ (i+1) + " : " + proxyArray[i]);

			// Print the array length
			System.out.println("Length of new array: " + proxyArray.length);
```

```

            }

    }
```

Output:

```
Element 1 : 12
Element 2 : 32
Element 3 : 11
Length of original array: 3
Element to be removed at index: 2
Array after removal operation:
Element 1 : 12
Element 2 : 32
Length of new array: 2
```

Using arrayList

Array List is a dynamic data type that belongs to List and showcases the characteristics of an array. It has a method called *remove()*, which can be directly used to remove an element from an array. We will be learning about it in later units.

2.4.3 Searching and Sorting

Elements in an array can be sorted and searched efficiently. There are different algorithms to do them in a better manner. We will be learning about the searching and sorting algorithms in later units.

2.4.4 Other Operations

Arrays can be used to perform various operations. The *java.utils.array* class can be imported in programs to include various inbuilt methods on arrays. Readers are used to refer to the Java documentation for details on the list of various inbuilt functions. Polynomial operations and also checking for sparse matrices can also be carried out through arrays.

2.4.4.1 Polynomial Operations using Arrays

The coefficients of the polynomials can be stored as separate arrays, and all arithmetic operations can be carried out on them considering as normal arrays declared with primitive data types.

2.4.4.2 Sparse Matrix

Arrays can be used to create a sparse matrix, and also to check if the given matrix is a sparse matrix or not. A sparse matrix is one who’s non-zero elements are very few compared to the total elements, i.e., elements are sparse. Program 2.5 creates a sparse matrix, shows how to check if a matrix is sparse, and lists the non-zero elements by their respective rows and columns.

Program 2.5: Java program using sparse matrix

```java
package dsArray;

public class sparseMatrix {

		public static void main(String[] args) {
			// Creating a sparse matrix
			int sparseMatrix[][]
		= {
		    {0, 0, 3, 0, 4},
		    {0, 0, 5, 7, 0},
		    {0, 0, 0, 0, 0},
		    {0, 2, 6, 0, 0}
		};

		// Calculates number of rows and columns present in given matrix
	int rows = sparseMatrix.length;
	int cols = sparseMatrix[0].length;

		int count = 0;
	System.out.println("Original Matrix...");
		for (int i = 0; i < rows; i++) {
	    for (int j = 0; j < cols; j++) {
	        System.out.printf("%d ", sparseMatrix[i][j]);
	        if (sparseMatrix[i][j] != 0) {
	            count++;
	        }
	    }
	    System.out.printf("\n");
	}

	//Calculates the size of array
	int size = rows * cols;

	// Checking for sparse matrix
	if(count < (size/2))
	    System.out.println("\nGiven matrix is a sparse matrix");
	else
	    System.out.println("\nGiven matrix is not a sparse matrix");

	// To find the non-zero values and positions on sparseMatrix
	// Corresponding row, column, and the non-zero value are stored in compactMatrix
	// Number of columns in compactMatrix (count) must be equal to number of
	// non - zero elements in sparseMatrix
	int compactMatrix[][] = new int[3][count];

	// Making of new matrix
	int k = 0;
	for (int i = 0; i < rows; i++) {
	    for (int j = 0; j < cols; j++) {
	        if (sparseMatrix[i][j] != 0) {
```

```
                    compactMatrix[0][k] = i;
                    compactMatrix[1][k] = j;
                    compactMatrix[2][k] = sparseMatrix[i][j];
                    k++;
                }
            }
        }

        System.out.println("\nRows, columns, and the corresponding non-zero values are:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < count; j++) {
                System.out.printf("%d ", compactMatrix[i][j]);
            }
            System.out.printf("\n");
        }

    }

}
```

Output:

```
Original Matrix...
0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0

Given matrix is a sparse matrix

Rows, columns, and the corresponding non-zero values are:
0 0 1 1 3 3
2 4 2 3 1 2
3 4 5 7 2 6
```

2.5 Conclusion

In this unit we have learned about the linear static data structure – array. We have seen how to declare and dynamically allocate memory to it while executing the program. Arrays are static data structures, so once created we cannot insert a new element by increasing the memory or delete an element to reduce the memory allotted. We have learned about the classification of arrays and different operations on them.

Case Study

Consider the following text given in quotes: “An array is a collection of items stored at contiguous memory locations”. Store it in an array by getting the input data and then find the following. Write a program and show the below.

Questions:

1. What is the length of the given string?
2. Display the number of times each character is present on the array

Self-Assessment Questions
Part A

1. Which of the below is used to allocate memory in arrays?
   a. new
   b. malloc
   c. create
   d. allocate

2. Which of the following is not true? The array data structure is
   a. static
   b. linear
   c. dynamic memory allocation
   d. allows inserting new elements

3. Which one is not the correct way of array creation?
   a. int[] sample = new int[5];
   b. int sample[] = new int[5];
   c. int sample = new int[];
   d. int[] sample = {5,7,10,11};

4. How can we describe an array in the best possible way?
   a. The Array shows a hierarchical structure.
   b. Arrays are immutable.
   c. Container that stores the elements of similar types
   d. The Array is not a data structure

5. Which of the following is the correct way of declaring an array?
   a. Int marks[10];
   b. int marks;
   c. marks{20};
   d. array marks[10];

6. Which of the following is the advantage of the array data structure?
   a. Elements of mixed data types can be stored.
   b. Easier to access the elements in an array
   c. Index of the first element starts from 1.
   d. Elements of an array cannot be sorted

7. What is the output of the following code snippet?

```
int[] testArray = {3,5,8,13};
```

```
System.out.println("Array elements: ");
for(int i = 0; i < testArray.length; i++)
        System.out.print(testArray[i]+" ");
```

a. Array elements:
   3 5 8 13
b. Array elements:
   35813
c. 3 5 8 13
d. 35813

8. What is the output of the following code snippet?

```
int arr[] = new int[3];
System.out.println(arr);
```

   a. [I@41a4555e (garbage value)
   b. 000
   c. null null null
   d. 0 0 0

9. Which data structure is linear?
   a. Array
   b. Tree
   c. Graph
   d. String

10. Which of these is necessary while creating an array?
    a. Row
    b. Column
    c. Both a and b
    d. None

Part B

1. Define an array and describe how it can be initialized.
2. How traversing is carried out in an array? Show with example.
3. Differentiate single-dimensional and multi-dimensional arrays
4. Describe how an array element can be deleted?
5. How will you insert values and display them in a two-dimensional array?
6. Write a program to add two polynomials
7. What is a sparse matrix? How can it be created and verified?
8. Explain the various array operations with sample code.

Post-Unit Reading Material

- Data Structures & Algorithms in Java by Robert Lafore

- Data Structures and Algorithms in Java™, Sixth Edition, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser

Topics for discussion forums

1. Discuss how elements can be inserted into an array.
2. Discuss the various array operations as asked in Question 8 of Part B.

Anasuya Threse Innocent