**B.DATA STRUCTURES IN R**

**1) Perform Addition, Subtraction, Multiplication and Transpose of a Matrix in R**

**Aim:** To Execute the Addition, Subtraction, Multiplication and Transpose of a Matrix in R Programming

**Procedure:**

1)Create two matrices, matrix1 and matrix2

2)Use the + operator to add the matrices element-wise. Store the result in addition\_result

3)Use the - operator to subtract one matrix from another element-wise. Store the result in subtraction\_result

4)Use the %\*% operator for matrix multiplication. Transpose matrix2 using the t() function before multiplying with matrix1. Store the result in multiplication\_result

5)Use the t() function to transpose matrix1. Store the result in transpose\_result.

**Program:**

**matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)**

**matrix2 <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)**

**addition\_result <- matrix1 + matrix2**

**subtraction\_result <- matrix1 - matrix2**

**multiplication\_result <- matrix1 %\*% t(matrix2) # Use %\*% for matrix multiplication**

**transpose\_result <- t(matrix1)**

**print("Addition Result:")**

**print(addition\_result)**

**print("Subtraction Result:")**

**print(subtraction\_result)**

**print("Multiplication Result:")**

**print(multiplication\_result)**

**print("Transpose Result:")**

**print(transpose\_result)**

**Output:**

**[1] "Addition Result:"**

**[,1] [,2] [,3]**

**[1,] 8 10 12**

**[2,] 14 16 18**

**[1] "Subtraction Result:"**

**[,1] [,2] [,3]**

**[1,] -6 -6 -6**

**[2,] -6 -6 -6**

**[1] "Multiplication Result:"**

**[,1] [,2]**

**[1,] 50 68**

**[2,] 122 167**

**[1] "Transpose Result:"**

**[,1] [,2]**

**[1,] 1 4**

**[2,] 2 5**

**[3,] 3 6**

**Result : Hence the program is verified and executed successfully in Addition, Subtraction, Multiplication and Transpose of a Matrix in R.**

**2) Find Transpose of a matrix in R and deconstruct a matrix.**

**Aim:** To Find the Transpose of a matrix in R and deconstruct a matrix.

**Procedure:**

1)Define or create a matrix in R using the matrix() function

2)Extract specific elements from the matrix

3)Use indexing with row and column numbers enclosed in square brackets [row, column] to access individual elements.

4)Use indexing with only the row number [row, ] to extract an entire row. Assign the row to a new variable.

5)Use indexing with only the column number [, column] to extract an entire column. Assign the column to a new variable.

6)Print the extracted elements, row, or column to verify the results.

**Program:**

**# Transpose of Matrix**

**mat <- matrix(1:6, nrow = 2)**

**transpose <- t(mat)**

**print(mat)**

**print(transpose)**

**Output:**

**[,1] [,2] [,3]**

**[1,] 1 3 5**

**[2,] 2 4 6**

**[,1] [,2]**

**[1,] 1 2**

**[2,] 3 4**

**[3,] 5 6**

**# Deconstruct of matrix**

**mat <- matrix(1:6, nrow = 2)**

**element\_1 <- mat[1, 1] # Extracts the element at row 1, column 1**

**element\_2 <- mat[2, 3] # Extracts the element at row 2, column 3**

**row\_1 <- mat[1, ]**

**column\_2 <- mat[, 2]**

**print(element\_1)**

**print(element\_2)**

**print(row\_1)**

**print(column\_2)**

**Output:**

**[1] 1**

**[1] 6**

**[1] 1 3 5**

**[1] 3 4**

**Result: Hence the Program is Executed and verified Successfully in Transpose of a matrix in R and deconstruct a matrix.**

**3) Perform the operation of combining matrices in R using cbind() and rbind() functions.**

**Aim :** To Perform the operation of combining matrices in R using cbind() and rbind() functions**.**

**Procedure:**

Using cbind() to Combine Matrices Horizontally:

1)Create or define multiple matrices in R using the matrix() function, specifying the elements and dimensions for each matrix.

2)Use the cbind() function followed by the matrices you want to combine horizontally, separated by commas.

3)Assign the result to a new variable if you want to store the combined matrix.

4)Print the combined matrix to verify the result.

**Program:**

**mat1 <- matrix(1:4, nrow = 2)**

**mat2 <- matrix(5:8, nrow = 2)**

**combined <- cbind(mat1, mat2)**

**print(combined)**

**Output:**

**[,1] [,2] [,3] [,4]**

**[1,] 1 3 5 7**

**[2,] 2 4 6 8**

**Using rbind() to Combine Matrices Vertically**

**Procedure:**

1)Create or define multiple matrices in R using the matrix() function, specifying the elements and dimensions for each matrix.

2)Use the rbind() function followed by the matrices you want to combine vertically, separated by commas.

3)Assign the result to a new variable if you want to store the combined matrix.

4)Print the combined matrix to verify the result.

**Output:**

**[,1] [,2]**

**[1,] 1 2**

**[2,] 3 4**

**[3,] 5 6**

**[4,] 7 8**

**Result:Hence the Program is Executed and Verified Successfully in the operation of combining matrices in R using cbind() and rbind() functions.**

**4) Perform array manipulation in R .**

**Aim:** To Perform the Array manipulation in R programming

**Procedure:**

1)Reshaping Arrays in R program

2)Subsetting Arrays

3)Merging Arrays

4)Transforming Arrays

**Program:**

**#Reshaping Arrays in R program**

**mat <- matrix(1:12, nrow = 3, ncol = 4)**

**reshaped\_array <- array(mat, dim = c(2, 6))**

**print(reshaped\_array)**

**Output:**

**[,1] [,2] [,3] [,4] [,5] [,6]**

**[1,] 1 4 7 10 1 4**

**[2,] 2 5 8 11 2 5**

**#Subsetting Arrays in R**

**Program:**

**mat <- matrix(1:12, nrow = 3, ncol = 4)**

**subset\_array <- mat[2, ]**

**print(subset\_array)**

**Output:**

**[1] 2 5 8 11**

**#Merging Arrays in R**

**Program:**

**# Create two matrices**

**mat1 <- matrix(1:6, nrow = 2)**

**mat2 <- matrix(7:12, nrow = 2)**

**merged\_array <- cbind(mat1, mat2)**

**print(merged\_array)**

**Output:**

**[,1] [,2] [,3] [,4] [,5] [,6]**

**[1,] 1 3 5 7 9 11**

**[2,] 2 4 6 8 10 12**

**#Transforming Arrays**

**Program:**

**# Create a matrix**

**mat <- matrix(1:12, nrow = 3, ncol = 4)**

**transformed\_array <- apply(mat, 2, sum)**

**print(transformed\_array)**

**Output:**

**[1] 6 15 24 33**

**Result : Hence the Code is Executed and Verified Successfully to Perform array manipulation in R .**

**5) Perform calculations across array elements in an array using the apply() function.**

**Aim:** To Perform calculations across array elements in an array using the apply() function in R programming.

**Procedure:**

1)Create an array my\_array using the array() function. In this example, it is a 3x4 array with elements from 1 to 12.

2)Define a custom function product() that calculates the product of elements in a vector using the prod() function.

3)Use the apply() function to apply the custom function product() to each row of the array my\_array. In this case, we specify 1 as the MARGIN argument to indicate that the function should be applied to rows.

4)The result is stored in the result variable.

5)Print the original array my\_array and the result vector

**Program:**

**my\_array <- array(1:12, dim = c(3, 4))**

**print(my\_array)**

**product <- function(x) {**

**prod(x)}**

**result <- apply(my\_array, 1, product)**

**print(result)**

**Output:**

**> print(my\_array)**

**[,1] [,2] [,3] [,4]**

**[1,] 1 4 7 10**

**[2,] 2 5 8 11**

**[3,] 3 6 9 12**

**> print(result)**

**[1] 280 1760 11880**

**Result: Hence the Program is Executed and Verified Successfullyto Perform calculations across array elements in an array using the apply() function.**

**6) Create a data frame and print the structure of the data frame in R.**

**Aim:** To Create and execute a data frame and print the structure of the data frame in R.

**Procedure:**

1)Create a data frame using the data.frame() function. The data frame has three variables: Name, Age, and Height.

2)The values for each variable are provided as vectors.

3)Use the str() function to print the structure of the data frame df.

4)The output displays the structure of the data frame, including the variable names, data types (chr for character, num for numeric), and the number of observations (rows) in the data frame.

5) Print the result in data frame

**Programm:**

**df <- data.frame(**

**Name = c("John", "Jane", "Mike"),**

**Age = c(25, 30, 35),**

**Height = c(170, 165, 180),**

**StringsAsFactors = FALSE**

**)**

**str(df)**

**Output:**

**'data.frame': 3 obs. of 3 variables:**

**$ Name : chr "John" "Jane" "Mike"**

**$ Age : num 25 30 35**

**$ Height: num 170 165 180**

**Result : Hence the program is Executed and verified Successfully ,to Create a data frame and print the structure of the data frame in R.**

**7) Demonstrate the creation of S3 class and S4 class in R.**

**Aim:** To Demonstrate the creation of S3 class and S4 class in R.

**S3 Class:**

**Procedure:**

1)Define the class name and its representation.

2)Create an instance of the class using the structure() function.

3)Define a generic function for the class using the naming convention print.<class>.

4)Call the generic function on the object to see the result.

5)Print the result

**Program:**

**setClass("Person", representation(name = "character", age = "numeric"))**

**person <- structure(list(name = "John Doe", age = 30), class = "Person")**

**print.Person <- function(obj)**

**{**

**cat("Name:", obj$name, "\n")**

**cat("Age:", obj$age, "\n")**

**}**

**print(person)**

**Output:**

**Name: John Doe**

**Age: 30**

**S4 Class:**

**Procedure:**

1)Define the class name, slots, and prototype.

2)Define a method for the class using the setMethod() function.

3)Create an instance of the class using the new() function.

4)Call the method on the object to see the result.

5)Print the result

**Program:**

**setClass("Rectangle",**

**slots = c(length = "numeric", width = "numeric"),**

**prototype = list(length = 0, width = 0)**

**)**

**setMethod("area", "Rectangle", function(object) {**

**object@length \* object@width**

**})**

**rectangle <- new("Rectangle", length = 5, width = 3)**

**area(rectangle)**

**Output:**

**[1] 15**

**Result: Hence the code is Executed and Verified Successfully to Demonstrate the creation of S3 class and S4 class in R.**

**8) Demonstrate the creation of S3 class and S4 class in R. Also illustrate how the fields of the object can be accessed using the $ operator. Modify the Name field by reassigning the name to Paul.**

**Aim :** To find how the fields of the object can be accessed using the $ operator. Modify the Name field by reassigning the name to Paul.

**Procedure:**

1)Create an object with fields.

2)Access the fields using the $ operator.

3)Modify the name field by reassigning a new value.

4)Access the modified field to verify the change.

5)Print the result

**Program:**

**person <- list(name = "John Doe", age = 30)**

**cat("Original Name:", person$name, "\n")**

**cat("Original Age:", person$age, "\n")**

**person$name <- "Paul"**

**cat("Modified Name:", person$name, "\n")**

**Output:**

**Original Name: John Doe**

**Original Age: 30**

**Modified Name: Paul**

**Result : Hence the Program is Executed and Verified Successfully to the fields of the object can be accessed using the $ operator. Modify the Name field by reassigning the name to Paul.**