

**Names:** Elysée NIYIBIZI

**REG No:** 2305000921

**Department:** BSC (Computer Science)  
**MODULE:** DSA (Data Srtucture and Algorithm

**Date:** 8 February 8, 2025

# ASSIGNMENT 1

**Question 1.**

#### ****(a) Algorithm to Display All Elements of the Matrix****

### The following algorithm prints the elements row by row, accessing each element via its indices. ****Algorithm****

1. **Start**
2. **For** each row i from 0 to rows - 1
   1. **For** each column j from 0 to columns - 1
      1. Print MatrixA[i][j] with a space.
   2. Print a new line to move to the next row.
3. **End**

**Implementation in Python**

````

# Define the matrix A

matrix\_A = [

[1, 2, 3, 4, 5, 6, 7, 8],

[9, 10, 11, 12, 13, 14, 15, 16],

[17, 18, 19, 20, 21, 22, 23, 24],

[25, 26, 27, 28, 29, 30, 31, 32],

[33, 34, 35, 36, 37, 38, 39, 40],

[41, 42, 43, 44, 45, 46, 47, 48],

[49, 50, 51, 52, 53, 54, 55, 56],

[57, 58, 59, 60, 61, 62, 63, 64]

]

# Display the matrix row by row

for row in matrix\_A:

for element in row:

print(element, end=" ")

print() # Move to the next line after each row

```

#### ****(b) Software Application for Student Grades****

We are developing a software application that tracks student grades. The grades are stored in a **2D array**, where:

* Each **row** represents a student.
* Each **column** represents a subject.

The array has:

* **10 rows** (for 10 students).
* **8 columns** (for 8 subjects).

### ****Algorithm to Store and Display Student Grades****

#### ****(i) Store the Grades****

1. **Start**
2. Create a **2D array** grades with 10 rows and 8 columns.
3. Take input for each student’s grades (loop through each row and each column).
4. **End**

#### ****(ii) Display the Grades****

1. **Start**
2. **For** each student i from 0 to 9
   1. Print "Student i+1 Grades:"
   2. **For** each subject j from 0 to 7
      1. Print grades[i][j] with a space.
   3. Print a new line to move to the next student.
3. **End**

**Implementation in Python**

**```**

**# Number of students and subjects**

**num\_students = 10**

**num\_subjects = 8**

**# Initialize the 2D array for student grades (Example data)**

**grades = [**

**[85, 90, 78, 92, 88, 76, 95, 89],**

**[80, 85, 88, 90, 92, 94, 78, 86],**

**[75, 80, 85, 88, 90, 76, 85, 82],**

**[90, 95, 92, 88, 85, 80, 78, 90],**

**[88, 86, 85, 90, 92, 94, 80, 85],**

**[76, 78, 80, 85, 88, 90, 92, 94],**

**[95, 92, 90, 85, 80, 78, 75, 88],**

**[82, 85, 88, 90, 92, 94, 76, 80],**

**[85, 88, 90, 92, 94, 76, 80, 85],**

**[78, 80, 85, 88, 90, 92, 94, 76]**

**]**

**# Display the student grades**

**print("Student Grades:\n")**

**for i in range(num\_students):**

**print(f"Student {i + 1} Grades:", end=" ")**

**for j in range(num\_subjects):**

**print(grades[i][j], end=" ")**

**print() # Move to the next line after each student**

**```**

**Question 2.**

### ****Deleting a specific Node from the List****

Imagine you have a train with multiple coaches (train cars). Each coach is linked to the next one. You want to remove a specific coach without breaking the train. That’s exactly what this algorithm does in a **linked list**!

### ****Example Linked List (Train)****

Suppose we have a linked list like this:

```

[10] → [20] → [30] → [40] → NULL

```

Each box is a node with a value, and NULL means the end of the list.

Now, let's say we want to **delete the node with value 30**.

### ****Steps to Delete a Node****

#### ****Step 1: Check if the list is empty****

If the train has no coaches (no nodes), we cannot delete anything. The algorithm stops.

#### ****Step 2: Initialize pointers****

We set two pointers:

* temp1 starts at head (first node).
* temp2 is set to NULL.

```

temp1 → [10] → [20] → [30] → [40] → NULL

temp2 → NULL

```

#### ****Step 3: Find the node to delete****

We move temp1 one step at a time to find 30.  
Meanwhile, temp2 follows behind temp1.

```

1st iteration: temp1 = 10, temp2 = NULL

2nd iteration: temp1 = 20, temp2 = 10

3rd iteration: temp1 = 30, temp2 = 20 (FOUND!)

```

#### ****Step 4: If the node is not found****

If we reached the end (NULL) and didn’t find 30, we print:

````

**"Given node not found in the list! Deletion not possible!"**

```

(But in our case, we found 30, so we move to the next step.)

#### ****Step 5: Delete the node****

There are **three possible cases** when deleting a node:

##### ****Case 1: Deleting the first node (head)****

If the node to delete is the first one, we just move head to the next node.

```

[10] → [20] → [30] → [40] → NULL (Delete 10)

```

New head:

```

[20] → [30] → [40] → NULL

```

(Since we are deleting 30, this does not apply here.)

##### ****Case 2: Deleting the last node****

If the node is the last one, we set temp2->next to NULL.

```

[10] → [20] → [30] → [40] → NULL (Delete 40)

```

New list:

```

[10] → [20] → [30] → NULL

```

(Again, this does not apply here.)

##### ****Case 3: Deleting a middle node (like 30)****

This is **our case**!  
We update temp2->next to **skip** the node 30 and link directly to 40.

Before deletion:

```

[10] → [20] → [30] → [40] → NULL

```

After deletion:

```

[10] → [20] → [40] → NULL

```

Now, 30 is removed, and the train is still connected!

#### ****Step 6: End****

The algorithm exits, and the list is now updated.

**Final Linked List After Deletion**

**```**

[10] → [20] → [40] → NULL

**```**

Node **30** is successfully deleted!

- END! -