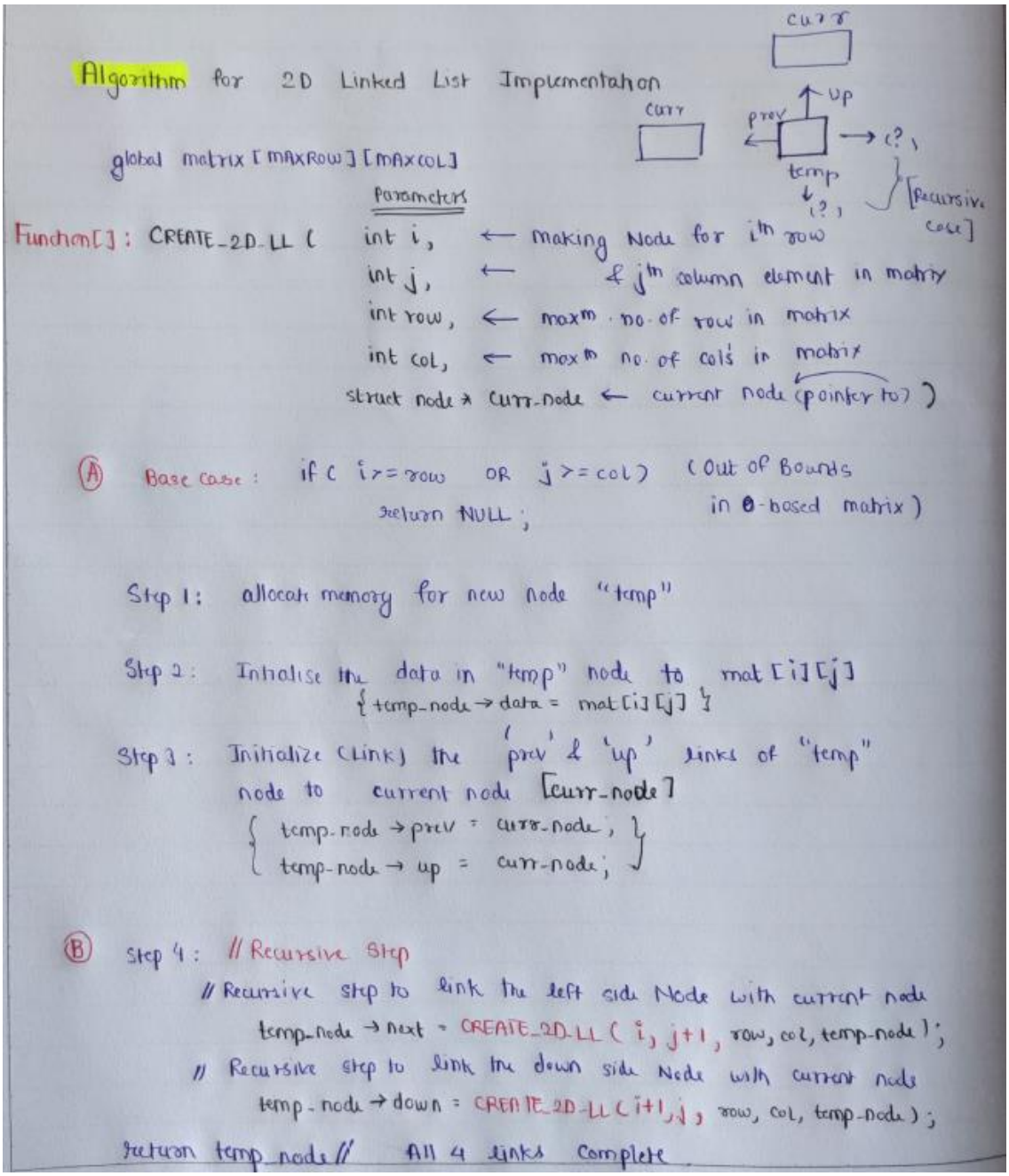
TUTORIAL XII:
2-D Matrix Linked List Implementation
U19CS012 [D-12]

Input: A 2D matrix

Aim: Convert it into a *doubly linked list* which has *four pointers*; next, previous, up and down. Each node of this list should be **connected** to its next, previous, up and down nodes respectively.

Write an **algorithm** for the above task and implement the same.

(A)**Algorithm:**



(B)Code:

```c
#include <stdio.h>
#include <stdlib.h>

/*
Structure Of 4-Directional Node
          up
prev <- [DATA] -> next
          down
*/

struct node
{
    int data;
    struct node *up;
    struct node *down;
    struct node *next;
    struct node *prev;
};

#define MAXROW 10000
#define MAXCOL 10000

int mat[MAXROW][MAXCOL];

// head Pointer -> head of Linked List
struct node *head = NULL;

// Creation of the 2-D Doubly Linked List
struct node *CREATE_2D_LL(int i, int j, int row, int col, struct node *curr_node);
// Display of the 2-D Doubly Linked List
void DISPLAY_2D_LL(struct node *head);

int main()
{
    // Number of Rows
    int row;
    // Number of Columns
    int col;

    printf("Enter the Number of Rows in Matrix : ");
    scanf("%d", &row);
    printf("Enter the Number of Columns in Matrix : ");
    scanf("%d", &col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            printf("a[%d][%d] = ", i, j);
```

```
            scanf("%d", &mat[i][j]);
        }
    }

    head = CREATE_2D_LL(0, 0, row, col, NULL);

    DISPLAY_2D_LL(head);

    return 0;
}

// Creation of the 2-D Doubly Linked List
struct node *CREATE_2D_LL(int i, int j, int row, int col, struct node *curr_node)
{
    // Base Case
    if (i >= row || j >= col)
    {
        return NULL;
    }

    // Create A Temporary Node
    struct node *temp_node;
    temp_node = (struct node *)malloc(sizeof(struct node));

    // Assign the Value in the Node
    temp_node->data = mat[i][j];

    // Intialise the Prev & Up Pointer of node to Current
    // Since temp_node is Linked to Current Pointer
    temp_node->prev = curr_node;
    temp_node->up = curr_node;

    // Recursive Step to Link the Left Sided Node with Current Node
    temp_node->next = CREATE_2D_LL(i, j + 1, row, col, temp_node);
    // Recursive Step to Link the Down Sided Node with Current Node
    temp_node->down = CREATE_2D_LL(i + 1, j, row, col, temp_node);

    // Return the temp_node with all its 4 Links Complete
    return temp_node;
}

// Display of the 2-D Doubly Linked List
void DISPLAY_2D_LL(struct node *head)
{
    // pointer to move right
    struct node *rPtr;

    // pointer to move down
    struct node *dPtr = head;
```

```
    rPtr = head;
    {
        printf("         ");
        while (rPtr != NULL)
        {
            printf("NULL  ");
            rPtr = rPtr->next;
        }
    }
    printf("\n");

    rPtr = head;
    // loop till struct node->right is not NULL
    printf("         ");
    while (rPtr != NULL)
    {
        printf("^     ");
        rPtr = rPtr->next;
    }
    printf("\n");

    rPtr = head;
    // loop till struct node->right is not NULL
    printf("         ");
    while (rPtr != NULL)
    {
        printf("|     ");
        rPtr = rPtr->next;
    }
    printf("\n");

    rPtr = head;
    // loop till struct node->right is not NULL
    printf("         ");
    while (rPtr != NULL)
    {
        printf("v     ");
        rPtr = rPtr->next;
    }
    printf("\n");

    int cnt = 0;
    // loop till struct node->down is not NULL
    while (dPtr != NULL)
    {

        rPtr = dPtr;
        // loop till struct node->right is not NULL
        printf("NULL <-> ");
        while (rPtr != NULL)
```

```
        {
            printf("%d <-> ", rPtr->data);
            rPtr = rPtr->next;
        }
        printf("NULL\n");

        rPtr = dPtr;
        // loop till struct node->right is not NULL
        printf("         ");
        while (rPtr != NULL)
        {
            printf("^     ");
            rPtr = rPtr->next;
        }
        printf("\n");

        rPtr = dPtr;
        // loop till struct node->right is not NULL
        printf("         ");
        while (rPtr != NULL)
        {
            printf("|     ");
            rPtr = rPtr->next;
        }
        printf("\n");

        rPtr = dPtr;
        // loop till struct node->right is not NULL
        printf("         ");
        while (rPtr != NULL)
        {
            printf("v     ");
            rPtr = rPtr->next;
        }
        printf("\n");

        dPtr = dPtr->down;
    }
    rPtr = head;
    {
        printf("         ");
        while (rPtr != NULL)
        {
            printf("NULL  ");
            rPtr = rPtr->next;
        }
    }
}
```

(C)**Test Cases:**

1.) 3 x 3 Matrix

1	2	3
4	5	6
7	8	9

```
Enter the Number of Rows in Matrix : 3
Enter the Number of Columns in Matrix : 3
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
a[2][0] = 7
          NULL  NULL  NULL
          ^     ^     ^
          |     |     |
          v     v     v
NULL <-> 1 <-> 2 <-> 3 <-> NULL
          ^     ^     ^
          |     |     |
          v     v     v
NULL <-> 4 <-> 5 <-> 6 <-> NULL
          ^     ^     ^
          |     |     |
          v     v     v
NULL <-> 7 <-> 8 <-> 9 <-> NULL
          ^     ^     ^
          |     |     |
          v     v     v
          NULL  NULL  NULL
```

2.) 3 x 4 Matrix

19	22	43	25
66	36	39	11
78	63	94	88

```
Enter the Number of Rows in Matrix : 3
Enter the Number of Columns in Matrix : 4
a[0][0] = 19
a[0][1] = 22
a[0][2] = 43
a[0][3] = 25
a[1][0] = 66
a[1][1] = 36
a[1][2] = 39
a[1][3] = 11
a[2][0] = 78
a[2][1] = 63
a[2][2] = 94
a[2][3] = 88
          NULL   NULL   NULL   NULL
          ^      ^      ^      ^
          |      |      |      |
          v      v      v      v
NULL <-> 19 <-> 22 <-> 43 <-> 25 <-> NULL
          ^      ^      ^      ^
          |      |      |      |
          v      v      v      v
NULL <-> 66 <-> 36 <-> 39 <-> 11 <-> NULL
          ^      ^      ^      ^
          |      |      |      |
          v      v      v      v
NULL <-> 78 <-> 63 <-> 94 <-> 88 <-> NULL
          ^      ^      ^      ^
          |      |      |      |
          v      v      v      v
          NULL   NULL   NULL   NULL
```