CL1002 ***Programming Fundamentals Lab***	Lab 08 Nested Loops and Nd-Arrays

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2024

Author: Mr. Muhammad Aashir

AIMS AND OBJECTIVES

Aim:

To understand and implement the concepts of nested loops and N-dimensional arrays (2D and 3D arrays) in C programming.

Objectives:

- Learn how nested loops work and their application in solving problems that require repeated actions.
- Understand how to declare, initialize, and manipulate 2D and 3D arrays.
- Apply nested loops to handle complex data structures like 2D and 3D arrays.
- Solve problems involving array operations, matrix manipulations, and pattern generation using nested loops.

Introduction

In programming, loops execute a block of code multiple times. Sometimes, we need to execute multiple loops inside another loop to handle complex operations. This concept is known as **nested loops**. Nested loops are useful when dealing with multidimensional data structures like matrices (2D arrays) and 3D arrays, where each element can be accessed by iterating through each dimension.

N-dimensional arrays (Nd-arrays) allow storing data in a structured format, such as grids, matrices, or tables. A **2D array** represents data in rows and columns, making it easy to work with matrices, while a **3D array** can represent a set of matrices or tables, adding another layer of complexity.

In the easiest way, A **2D-array** is multiple **1D-arrays**. And a **3D-array** is multiple **2D-arrays**.

Section 1: Nested Loops

Explanation: A nested loop is a loop inside another loop. The inner loop runs completely every time the outer loop executes once. This structure is particularly useful for:

- Generating patterns
- Iterating over multi-dimensional arrays
- Performing repeated operations that depend on each other.

Example 1: Generating a Simple Pattern

Print a right-angled triangle pattern using nested loops.

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 1; i <= 5; i++) { // Outer loop for rows
      for (j = 1; j <= i; j++) { // Inner loop for columns
         printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Example 2: Multiplication Table

Display a multiplication table up to 5x5 using nested loops.

```
#include <stdio.h>

int main() {
   int i, j;
   for (i = 1; i <= 5; i++) {
      for (j = 1; j <= 5; j++) {
         printf("%d\t", i * j);
      }
      printf("\n");
   }
   return 0;

}
```

Output:

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
```

Example 3: Inverted Pyramid Pattern

Print an inverted right-angled pyramid pattern using nested loops.

```
#include <stdio.h>

int main() {
    int rows = 5;
    for (int i = rows; i >= 1; i--) { // Outer loop for rows
        for (int j = 1; j <= i; j++) { // Inner loop for columns
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
* * * * *
* * * *
* * *
* *
*
```

Section 2: 2D Arrays and Nested Loops

Explanation: A 2D array is like a grid or table. Each element is identified by two indices: one for the row and one for the column. Nested loops are often used to iterate over 2D arrays.

Example 1: Matrix Addition
Write a program to add two 2x2 matrices using nested loops.

```
#include <stdio.h>

int main() {
   int mat1[2][2] = {{1, 2}, {3, 4}};
   int mat2[2][2] = {{5, 6}, {7, 8}};
   int result[2][2];

   // Adding two matrices
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         result[i][j] = mat1[i][j] + mat2[i][j];
      }
   }
   // Displaying the result
   printf("Sum of two matrices:\n");
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         printf("%d ", result[i][j]);

      }
      printf("\n");
   }
   return 0;
}
```

Output:

```
6 8
10 12
```

References:

- Matrix Operations in C

Section 3: 3D Arrays and Nested Loops

Explanation: A 3D array can be visualized as multiple 2D grids stacked on top of each other. Each element is accessed using three indices. Nested loops are used to navigate through each layer, row, and column.

Example 1: Initializing a 3D Array

Write a program to initialize and display a 2x2x2 3D array.

```
#include <stdio.h>

int main() {
    int arr[2][2][2] = {
        {{1, 2}, {3, 4}},
        {{5, 6}, {7, 8}}
    };

    // Displaying the 3D array
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                printf("%d ", arr[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
    return 0;
}
```

Output:

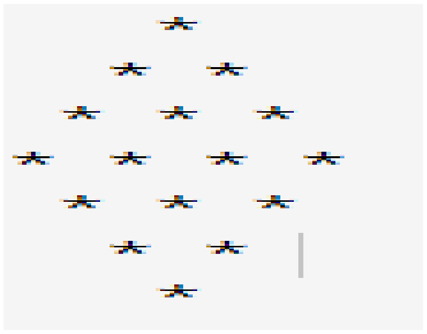
```
1 2
3 4

5 6
7 8
```

References:

- 3D Array Manipulation

More problems :’D

1. Write a C program that generates a sequence of prime numbers within a given range using nested loops.
2. Generate a pattern of odd/even numbers in a multi-dimension array of [2][n][2] where the first 2 is the different arrays for even and odd. The N is the amount of array quantity in each odd/even array and the 2 are the two even/odd values in each of them, add them in decreasing order starting from a user-specified number using nested loops.
3. Write a C program to find the saddle point(s) in each 3x3 matrix. A saddle point is an element that is the smallest in its row and the largest in its column.
4. Write a C program to multiply two matrices of size 3x3 and display the result matrix. Also subtract the resultant matrix from Matrix A as well.
5. Write a C program to generate a diamond shape pattern using nested loops. The program should take the number of rows for the upper half of the diamond as input from the user.

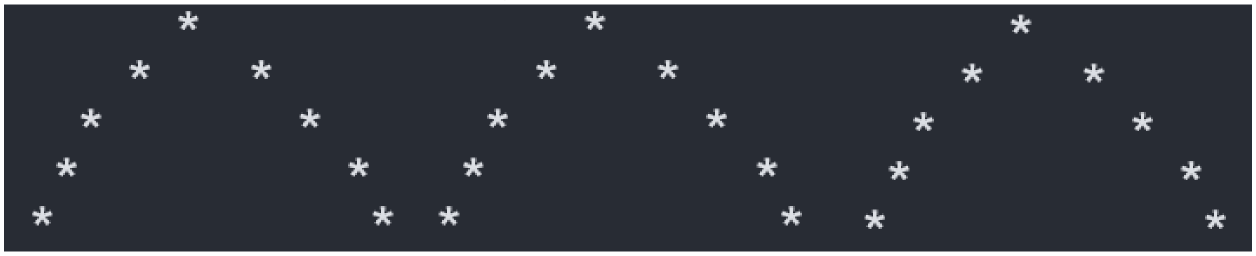


```
      *
    *   *
  *   *   *
*   *   *   *
  *   *   *
    *   *
      *
```

6. Print a pattern using nested loops

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

7. Do it for another but use nested while loops.



```
      *                *                *
    *    *           *    *           *    *
  *        *       *        *       *        *
 *           *    *           *    *           *
*             *  *             *  *             *
```