Fundamentals of Programming

Lab Manual 6

Nested Loops in C Programming

SCHOOL OF INTERDISCIPLINARY ENGINEERING AND SCIENCES NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY (NUST)

Objective:

The objective of this lab on nested loops in C programming should be to familiarize students with the concept of nested loops.

Prerequisites:

Before starting this lab, you should have a basic understanding of:

- C programming language
- Loop structures (e.g., for, while, do-while)
- Basic control flow statements (e.g., if, else)

Introduction:

Nested loops in C are loops that are placed within the body of another loop. This allows for multiple iterations of inner loops for each iteration of the outer loop. Nested loops are commonly used for tasks that involve working with two-dimensional arrays, matrix operations, generating combinations, permutations, and more.

Code Example:

Here's an example of nested loops in C:

```
#include <stdio.h>

int main() {

    int rows = 3;

    int columns = 4;

    // Outer loop controls rows

    for (int i = 0; i < rows; i++) {

        // Inner loop controls columns

        for (int j = 0; j < columns; j++) {
```

```
            printf("(%d, %d) ", i, j);
        }
        printf("\n"); // Move to the next line after each row is printed
    }
    return 0;
}
```

In this example, we have an outer loop iterating over the rows and an inner loop iterating over the columns. Each time the inner loop runs to completion (iterates over all columns), the outer loop advances and the inner loop starts again for the next row.

Output:

(0, 0) (0, 1) (0, 2) (0, 3)

(1, 0) (1, 1) (1, 2) (1, 3)

(2, 0) (2, 1) (2, 2) (2, 3)

This example prints out the indices of a 3x4 matrix, demonstrating the concept of nested loops. Each pair of indices (i, j) corresponds to a cell in the matrix.

1. Single Nested Loop:

A single nested loop consists of one loop contained within another. Typically, the outer loop controls the iteration over rows, while the inner loop controls the iteration over columns. This structure is commonly used for tasks involving two-dimensional data, such as matrices, grids, or tables.

This is the simplest form of nested loop, where one loop is nested within another. Useful for tasks that require iterating over two dimensions of data.

Example:

```
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        // Do something with (i, j)  }}
```

Code Example:

In this example, we'll write a C program to print a multiplication table using a single nested loop.

```
#include <stdio.h>

int main() {
    // Define the size of the multiplication table
    int rows = 10;
    int columns = 10;
    // Outer loop controls the rows
    for (int i = 1; i <= rows; i++) {
        // Inner loop controls the columns
        for (int j = 1; j <= columns; j++) {
            // Multiply the current row number (i) with the current column number (j)
            int result = i * j;
            // Print the result with formatting
            printf("%3d ", result); // Adjust the width as needed
        }
```

```
        // Move to the next line after printing all columns for a row
    printf("\n");
    }
    return 0;
}
```

Output:

This program will generate the multiplication table from 1 to 10, where each row represents the multiplication of the row number with numbers from 1 to 10.

```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
```

2. Multiple Nested Loops:

Multiple nested loops involve having more than one loop within another loop. Each inner loop executes completely for each iteration of the outer loop. This allows iterating over multi-dimensional data structures or performing repetitive tasks with more complex patterns.

Multiple loops can be nested within each other to handle multi-dimensional data or perform complex repetitive tasks.

Each nested loop adds another dimension to the iteration process.

Example:

```
for (int i = 0; i < dim1; i++) {
   for (int j = 0; j < dim2; j++) {
      for (int k = 0; k < dim3; k++) {
         // Do something with (i, j, k)}}}
```

Code Example:

In this example, we'll create a program that prints a pattern of stars resembling a right-angled triangle using multiple nested loops.

```
#include <stdio.h>

int main() {
   int rows = 5; // Number of rows in the triangle


   // Outer loop for rows
   for (int i = 1; i <= rows; i++) {
      // Inner loop for printing stars
```

```
        for (int j = 1; j <= i; j++) {
            printf("* ");
        }
        printf("\n"); // Move to the next line after each row
    }
    return 0;
}
```

Explanation of the Code:

- We define the number of rows in our triangle (rows = 5 in this example).
- We use an outer loop to iterate over each row of the triangle. The loop variable i represents the current row number.
- Inside the outer loop, we use an inner loop to print the stars for each row. The inner loop iterates from 1 to the current row number (i) and prints a star for each iteration.
- After printing the stars for each row, we move to the next line (printf("\n")) to start printing the next row.

Output:

```
*

* *

* * *

* * * *

* * * * *
```

3. Loop Nesting with Different Incremental Steps:

Nested loops can have different incremental steps, allowing for more flexible iterations.

Useful when dealing with irregular patterns or when you need to control the iteration more precisely.

Example:

```
for (int i = 0; i < rows; i++) {
   for (int j = 0; j < columns; j += 2) {
      // Do something with (i, j)
   }
}
```

4. Loop Nesting with Conditional Break or Continue:

Loop nesting with a conditional break or continue involves incorporating conditions within nested loops that allow for the early termination of the inner loop or skipping certain iterations based on specific conditions. This is useful when you need to control the flow of execution within nested loops based on certain criteria.

Inner loops can have conditions that cause them to break or continue independently of the outer loop.

Useful when you need to skip certain iterations based on specific conditions.

Example:

```
for (int i = 0; i < rows; i++) {
   for (int j = 0; j < columns; j++) {
```

```
        if (some_condition) {
            // Do something and break inner loop
            break;  }}}
```

Code Example:

```
#include <stdio.h>
int main() {
    int rows = 5;
    int columns = 5;

    // Outer loop for rows
    for (int i = 0; i < rows; i++) {
        // Inner loop for columns
        for (int j = 0; j < columns; j++) {
            // Check if row number is greater than column number
            if (i > j) {
                // If true, print a space and continue to the next iteration
                printf("  ");
                continue; // Skip printing stars in this iteration
            }
            // Otherwise, print a star
```

```
            printf("* ");
        }
        printf("\n"); // Move to the next line after each row
    }
    return 0;
}
```

Explanation of the Code:

- We define the number of rows and columns for our pattern (both set to 5 in this example).
- We use an outer loop to iterate over each row of the pattern.
- Inside the outer loop, we use an inner loop to iterate over each column of the pattern.
- Within the inner loop, we check if the current row number (i) is greater than the current column number (j). If it is, we print a space and continue to the next iteration of the inner loop.
- If the condition is not met (i.e., the current row number is not greater than the current column number), we print a star.
- After printing all the characters for each row, we move to the next line using printf("\n").

Output:

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

Lab Task:

1. Write a program that prints the following using a for loop:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

2. Write a program that creates the following:

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

3. Write a program that creates the following: