Lecture 7

The do-while Loop

The do-while loop is a post-test loop, meaning the condition is checked after the body executes.

Structure & Logic

The general syntax is:

```
do {
    // statement;
} while (expression);
```

Execution:

The body runs first, then the condition is evaluated.

Key Difference:

Unlike a while loop (which may run zero times), a do-while loop is guaranteed to execute at least once.

Best Practices (Style Note)

Braces are Mandatory:

Always use {} even for single-line statements.

Clarity:This prevents confusion, ensuring the while (expression); at the bottom isn't mistaken for the start of a standard while loop.

Practical Example:

Reversing an Integer

A do-while loop is ideal for processing numbers where you need to handle the digit 0 naturally without extra logic.

```
#include <stdio.h>
int main() {
    int val;
    // Prompt user for input
    printf("Enter a non-negative integer: ");
    scanf("%d", &val);
    printf("Reversed sequence: ");
    // The do-while loop ensures at least one iteration
    // This allows '0' to be printed correctly.
    do {
        printf("%d", val % 10); // Extract and print the last digit
        val /= 10;              // Remove the last digit from val
    } while (val != 0);
    printf("\n");
```

```
    return 0;
}
```

Key Logic Breakdown

The do-while Advantage: If the user enters 0, the condition val != 0 is false immediately. However, because it is a do-while loop, the code inside the braces executes once first, printing the 0 before checking the condition and exiting.

Simple Countdown Example

```
#include <stdio.h>
int main() {
    int count = 5;
    do {
        printf("Number: %d\n", count);
        count--; // Decrease count by 1
    } while (count > 0);
    printf("Blast off!\n");
    return 0;
}
```

Comparison between while and do while

`do-while` loop	`while` loop
Checks condition **after** running.	Checks condition **before** running.
Always runs at least **1** time.	Can run **0** times.
Menus, user input, simple counters.	Reading files, searching.

One-Dimensional (1D) Arrays in C

Introduction to Arrays:

An array is a fixed-size, sequenced collection of elements of the same data type.

Instead of declaring multiple variables like score1, score2, and score3, we use a single array variable to store all of them.

Contiguous Memory:

Array elements are stored in back-to-back memory locations.

Homogeneous:

All elements must be of the same type (e.g., all int or all float).

Declaration and Initialization

To declare an array, you must specify the type, the name, and the size (number of elements).

Syntax:

dataType arrayName[arraySize];

Examples:

int marks[5]; // Declaration without initialization

int age[5] = {18, 20, 19, 21, 22}; // Declaration with initialization

int scores[] = {90, 85, 70}; // Size is automatically 3

Accessing Elements (Indexing)

Array elements are accessed using an index (subscript).

Important:In C, array indexing starts at 0 and goes up to size - 1.

First element: age[0]

Last element (for size 5):age[4]

Example:

```c
age[0] = 25;       // Assigning value to the first element

int x = age[2];    // Accessing the third element
```

Processing Arrays (Using Loops)

Since arrays are indexed numerically, we almost always use a for loop to traverse them.

Example: Printing an Array

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    printf("Array elements: ");
    for(int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
```

Key Concepts & Common Pitfalls

A. Array Size is Fixed

Once you declare an array size (e.g., int arr[10]), you cannot change it during the program's execution.

B. Out of Bounds (The Danger Zone)

C does not check if your index is valid.

If you have an array of size 5 and try to access arr[10], the program might crash or return "garbage" data.

This is a common source of bugs.

C. Memory Calculation

The total size of an array in bytes is calculated as

Example: An int array of 5 elements on a 4-byte system takes 5 x 4 = 20 bytes.

Basic Input and Output

This program demonstrates how to use a loop to fill an array with user input and then display it.

```
#include <stdio.h>

int main() {
    int numbers[5];
    // Input: Filling the array
    printf("Enter 5 integers:\n");
    for(int i = 0; i < 5; i++) {
        printf("Element %d: ", i);
        scanf("%d", &numbers[i]);
    }
    // Output: Reading from the array
    printf("\nThe elements in the array are: ");
    for(int i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }
```

```
    return 0;
}
```

Finding the Maximum Value

A very common task is to traverse an array to find the largest number.

```
#include <stdio.h>

int main() {
    int data[] = {12, 45, 2, 89, 34, 67};
    int n = 6; // Number of elements
    int max = data[0]; // Assume first element is the largest
    for(int i = 1; i < n; i++) {
        if(data[i] > max) {
            max = data[i]; // Update max if a larger value is found
        }
    }
```

```
    printf("The maximum value in the array is:
 %d\n", max);
    return 0;
}
```

Calculating Sum and Average

This example shows how to perform mathematical operations on array data.

```
#include <stdio.h>

int main() {
    float grades[4] = {85.5, 90.0, 78.5, 92.0};
    float sum = 0.0, average;
    for(int i = 0; i < 4; i++) {
        sum += grades[i]; // Add each element to sum
    }
    average = sum / 4;
    printf("Total Sum: %.2f\n", sum);
    printf("Class Average: %.2f\n", average);
    return 0;
}
```

Pro-Tips for Arrays:

The Index Rule: Always remember that array[5] has indices from 0 to 4.

Accessing array[5] is a common error called "Off-by-one."

Initialization: If you write int arr[5] = {0};, C will initialize all elements to zero.

Size Calculation: You can find the number of elements in an array automatically using:

int length = sizeof(arr) / sizeof(arr[0]);

Illustrative Example

Suppose we have an array of integers (int):

int arr[] = {10, 20, 30, 40, 50};

In most modern systems, the size of a single int variable is 4 bytes.

Since the array contains 5 elements, the total size size of (arr) will be: 5 x 4 = 20 bytes.

The size of the first element size of (arr[0]) is the size of one int, which is 4 bytes.

Applying the Equation

length = size of (arr) / size of (arr[0])

length = 20 bytes / 4 bytes = 5 elements