CC-112L

Programming Fundamentals

Laboratory 04

Introduction to Programming, Algorithms and C

Version: 1.0.0

Release Date: 02-03-2025

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Understand and implement different **control structures** in C.
- Write C programs using **decision-making** and **looping constructs**.

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Hafiz Anzar Ahmad	anzar@pucit.edu.pk
Teacher Assistants	Manahil	Bitf21m002@pucit.edu.pk
	Maheen Fatima	Bitf22m031@pucit.edu.pk
	Rimsha Majeed	Bitf22m029@pucit.edu.pk
	Momna Muzaffar	Bcsf22m021@pucit.edu.pk
	Zainab Mehmood	Bcsf22m038@pucit.edu.pk
	Khadija tul Kubra	Bitf22m025@pucit.edu.pk
	Inam ul Haq	Bitf22m017@pucit.edu.pk
	M. Saad	Bitf23m003@pucit.edu.pk

Overview

Loops are fundamental control structures in programming that allow repetitive execution of code. Among these, the **`while`** and **`do-while`** loops are particularly useful for situations where the number of iterations is unknown beforehand. Understanding their efficient use helps programmers write clear, optimized, and effective code.

---

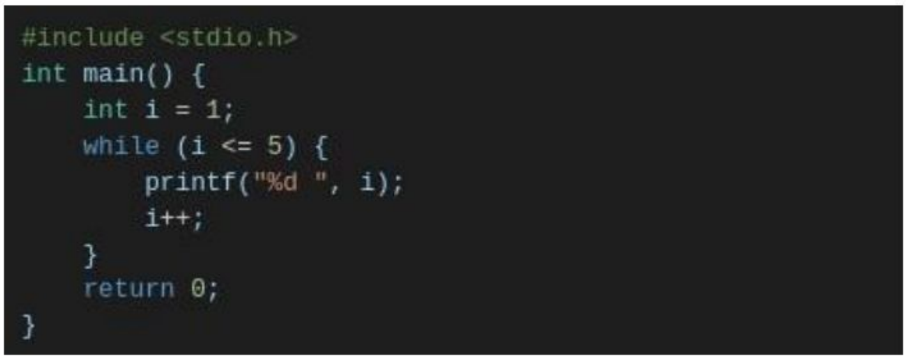
1. while Loop:

The `while` loop executes a block of code **as long as the given condition remains true**. If the condition is false initially, the loop body will **not execute at all**.

Syntax:

```
while (condition) {
    // Code inside loop
}
```

Example: Printing numbers from 1 to 5



```c
#include <stdio.h>
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

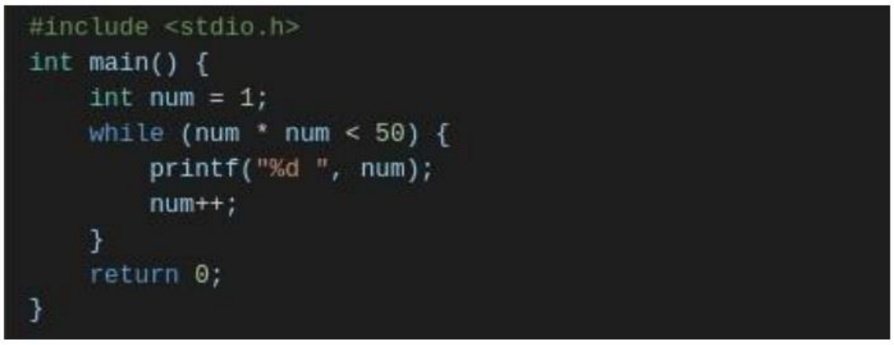
Output:

```
1 2 3 4 5
```

Use Cases:

- When the number of iterations is **unknown**.
- Checking for user input until a valid value is received.
- Processing data streams until an end condition is met.

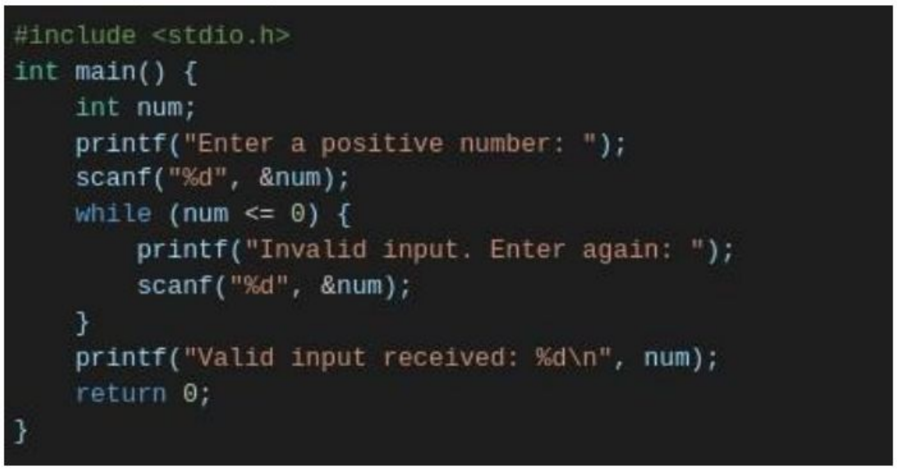
Example 1: When the number of iterations is unknown



```c
#include <stdio.h>
int main() {
    int num = 1;
    while (num * num < 50) {
        printf("%d ", num);
        num++;
    }
    return 0;
}
```

Output:
```
1 4 9 16 25 36 49
```

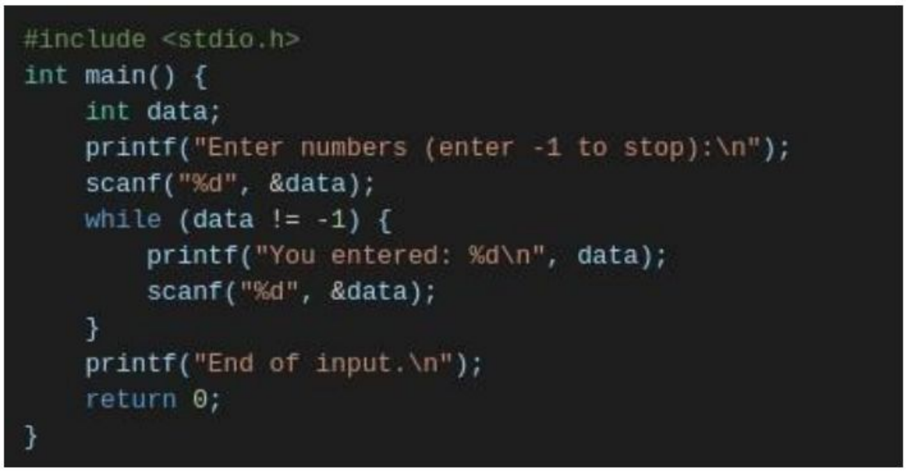
Example 2: Checking for user input until a valid value is received



```c
#include <stdio.h>
int main() {
    int num;
    printf("Enter a positive number: ");
    scanf("%d", &num);
    while (num <= 0) {
        printf("Invalid input. Enter again: ");
        scanf("%d", &num);
    }
    printf("Valid input received: %d\n", num);
    return 0;
}
```

Output:
```
1 4 9 16 25 36 49
```

Example 3: Processing data streams until an end condition is met



```c
#include <stdio.h>
int main() {
    int data;
    printf("Enter numbers (enter -1 to stop):\n");
    scanf("%d", &data);
    while (data != -1) {
        printf("You entered: %d\n", data);
        scanf("%d", &data);
    }
    printf("End of input.\n");
    return 0;
}
```

Output:
```
Enter numbers (enter -1 to stop):
5
You entered: 5
6
You entered: 6
-1
End of input.
```

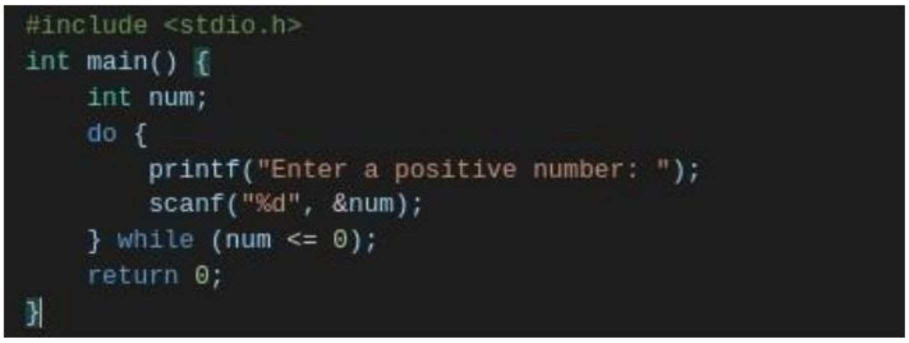
2. do-while Loop

The `do-while` loop **executes at least once**, regardless of the condition, because the condition is checked **after** the first iteration.

Syntax:

```
do {
    // Code inside loop
} while (condition);
```

Example: Getting user input until a positive number is entered



```
#include <stdio.h>
int main() {
    int num;
    do {
        printf("Enter a positive number: ");
        scanf("%d", &num);
    } while (num <= 0);
    return 0;
}
```

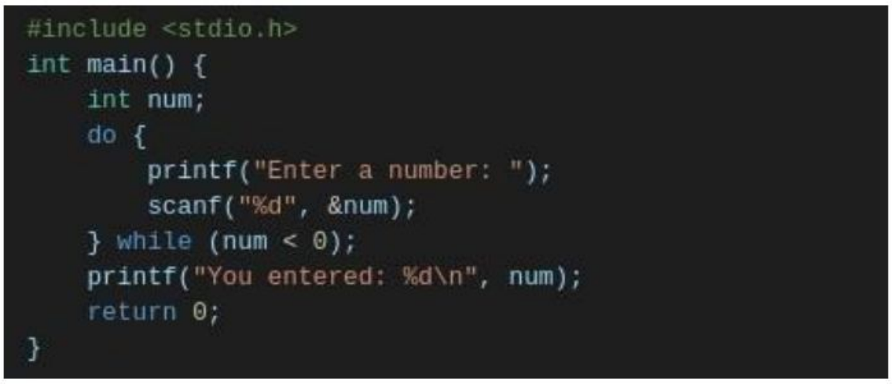
Output:

```
Enter a positive number: -3
Enter a positive number: -8
Enter a positive number: 0
Enter a positive number: 4
```

Use Cases:

- Ensuring the loop body executes at least **once**.
- Menu-driven programs where user input is required before checking conditions.
- Validating input without an initial condition check.

Example 1: Ensuring the loop body executes at least once

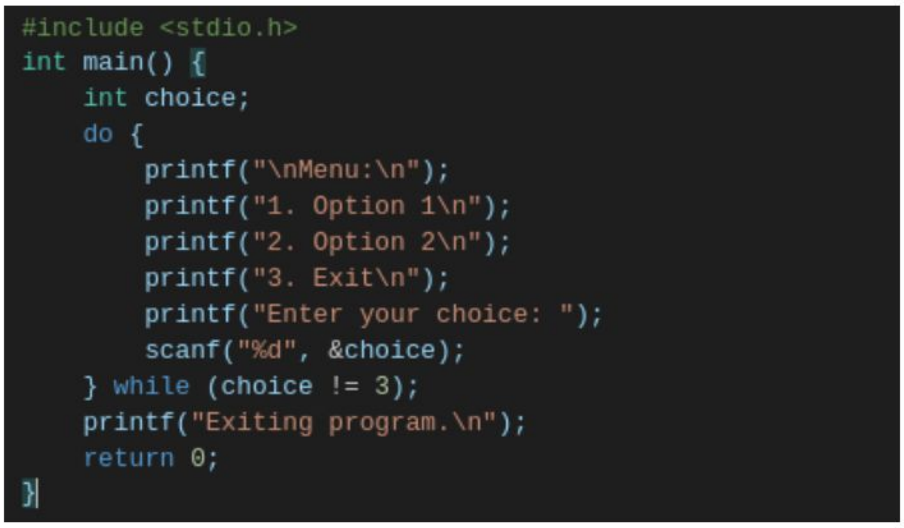


```
#include <stdio.h>
int main() {
    int num;
    do {
        printf("Enter a number: ");
        scanf("%d", &num);
    } while (num < 0);
    printf("You entered: %d\n", num);
    return 0;
}
```

Output:

```
Enter a number: -3
Enter a number: -8
Enter a number: 0
You entered : 0
```

Example 2: Menu-driven programs where user input is required before checking conditions



```c
#include <stdio.h>
int main() {
    int choice;
    do {
        printf("\nMenu:\n");
        printf("1. Option 1\n");
        printf("2. Option 2\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
    } while (choice != 3);
    printf("Exiting program.\n");
    return 0;
}
```

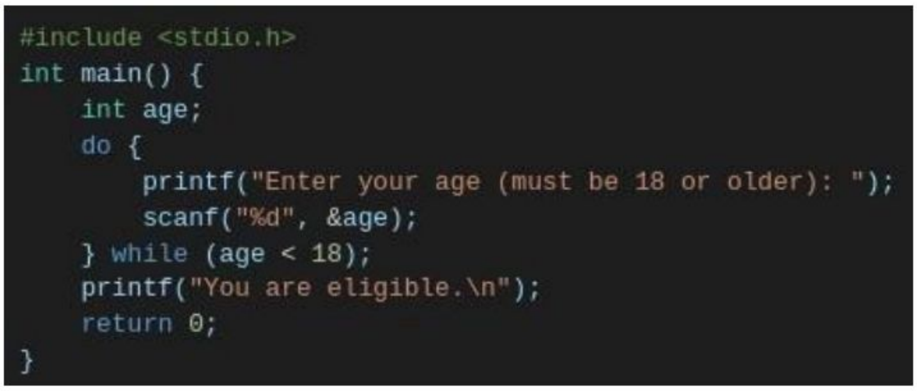
Output:

```
Menu:
1. Option 1
2. Option 2
3. Exit
Enter your choice: 1

Menu:
1. Option 1
2. Option 2
3. Exit
Enter your choice: 2

Menu:
1. Option 1
2. Option 2
3. Exit
Enter your choice: 3
Exiting program.
```

Example 3: Validating input without an initial condition check



```c
#include <stdio.h>
int main() {
    int age;
    do {
        printf("Enter your age (must be 18 or older): ");
        scanf("%d", &age);
    } while (age < 18);
    printf("You are eligible.\n");
    return 0;
}
```

Output:

```
Enter your age (must be 18 or older): 16
Enter your age (must be 18 or older): 17
Enter your age (must be 18 or older): 18
You are eligible.
```

Comparison and Efficient Use

Feature	while Loop	do-while Loop
Condition Check	Before execution	After execution
Guaranteed Execution	No	Yes, at least once
Usage	When zero iterations are possible	When at least one execution is needed

Efficiency Considerations:

- **Avoid infinite loops:** Always ensure the loop condition eventually becomes false.
- **Optimize condition checks:** Repeated calculations in the condition can slow down execution.
- **Use do-while for validation tasks:** When input must be taken at least once, `do-while` is preferred.
- **Use while for unknown iteration needs:** Ideal for reading files, processing dynamic data, or waiting for an event.

PRE-LAB TASKS

Concepts Used: Nested Loops, Conditional Statements, Do-While Loop

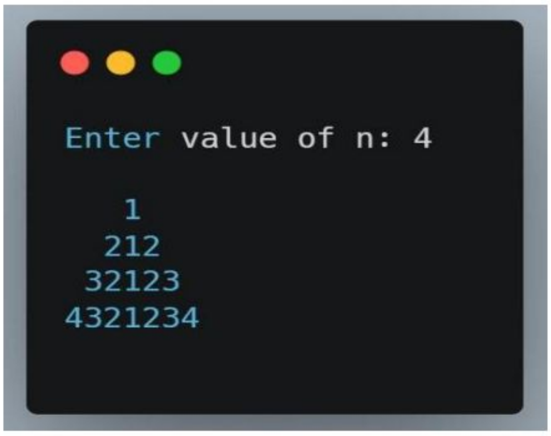
TASK 01:

Diamond Upper Half Pattern

Write a program to print the **upper half of a diamond number pattern**:

Execution Flow:

1. Ask the user for the number of rows.
2. Use nested loops to generate the pattern.
3. Print the numbers in the given sequence with proper spacing.



```
Enter value of n: 4

   1
  212
 32123
4321234
```

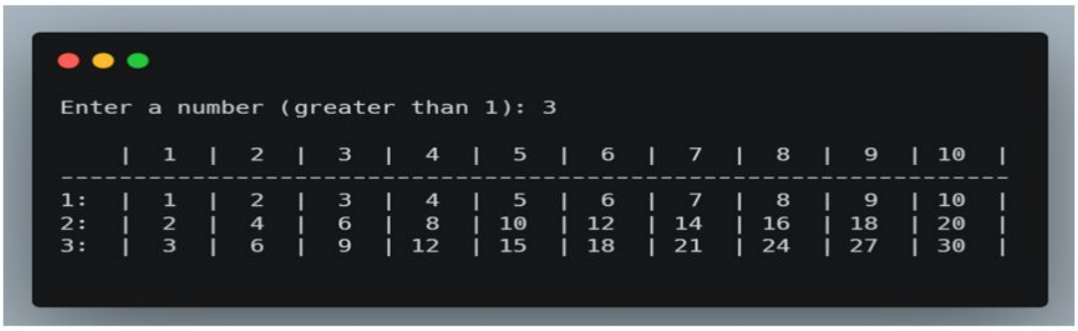
Task 02

Generate Multiplication Tables up to N

✓ Take input n from the user.
✓ Ensure n > 1 (valid input).
✓ Display multiplication tables from 1 to n for numbers 1 to n.

Execution Flow:

1. Ask the user to enter a number n.
2. Validate that n > 1.
3. Print multiplication tables from 1 to n.



```
Enter a number (greater than 1): 3

    |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  | 10  |
-----------------------------------------------------------------
1:  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  | 10  |
2:  |  2  |  4  |  6  |  8  | 10  | 12  | 14  | 16  | 18  | 20  |
3:  |  3  |  6  |  9  | 12  | 15  | 18  | 21  | 24  | 27  | 30  |
```