Days 10–21 — Intermediate to Advanced Foundation

DAY 10 – Important Loop Programs

Topics

- Understanding loop-based problem solving
- Counter-controlled loops
- Accumulator concept
- Finding and processing digits

Programs

1. Factorial
2. Sum of first N numbers
3. Multiplication table
4. Fibonacci series
5. Prime number
6. Reverse a number
7. Count digits in a number

Challenge ☆

Find whether a number is a Perfect Number.

Example:

```
6

Factors:
1 + 2 + 3 = 6

6 is a Perfect Number
```

Why Day 10?

You already learned `for`, `while`, and `do-while`. Now they should learn how to **use loops to solve mathematical problems**.

DAY 11 – Nested Loops & Patterns

Topics

- Nested loops
- Outer loop
- Inner loop
- Rows and columns
- Pattern logic

Basic Concept

```
for(i = 1; i <= 4; i++)
{
    for(j = 1; j <= i; j++)
    {
        printf("*");
    }

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

Programs

Pattern 1

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

Pattern 2

```
1
12
123
1234
```

Pattern 3

```
1
22
333
4444
```

Pattern 4

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

Challenge ☆

Print:

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

Why Day 11?

You first understand **one loop**, then **a loop inside another loop**.

DAY 12 – One-Dimensional Arrays

Now introduce a very important concept:

One variable stores one value. An array stores multiple values of the same data type.

Topics

- What is an array?
- Need for arrays
- Array declaration
- Array initialization
- Array index
- Accessing array elements
- Traversing an array
- `for` loop with arrays

Example

```
int marks[5] = {80, 75, 90, 85, 70};
```

Visual:

```
Index:      0     1     2     3     4
Marks:    | 80  | 75  | 90  | 85  | 70  |
```

Programs

1. Read and print array elements
2. Sum of array elements
3. Average of array elements
4. Find largest element
5. Find smallest element
6. Count even and odd elements
7. Search for an element

Challenge ☆

Find the second largest element in an array.

Why Day 12?

Before learning matrices, you must understand a **one-dimensional array** thoroughly.

DAY 13 – Two-Dimensional Arrays & Matrices

Topics

- Two-dimensional arrays
- Rows and columns
- Matrix representation
- Reading matrix elements
- Printing matrix elements
- Nested loops with matrices

Example

```
int a[2][3];
```

Visual:

```
        Column
          0    1    2
       +----+----+----+
Row 0  | 10 | 20 | 30 |
       +----+----+----+
Row 1  | 40 | 50 | 60 |
       +----+----+----+
```

Programs

1. Read and display matrix
2. Matrix addition
3. Matrix subtraction
4. Matrix transpose
5. Sum of diagonal elements

Advanced Program

6. Matrix multiplication

Challenge ☆

Find whether a matrix is an **Identity Matrix**.

DAY 14 – Strings Fundamentals

Now you move from numbers to **text**.

Topics

- What is a string?
- Character vs string
- Character array
- Null character `'\0'`
- String declaration
- String initialization
- Reading strings
- Displaying strings

Example

```
char name[] = "Raju";
```

Memory concept:

```
R    a    j    u   \0
↓    ↓    ↓    ↓    ↓
+----+----+----+----+----+
| R  | a  | j  | u  | \0 |
+----+----+----+----+----+
```

String Input/Output

Teach:

```
fgets()
puts()
```

Important Note

Do **not** teach:

```
gets()
```

as a normal programming technique. `gets()` is unsafe because it does not know the size of the destination array.

Programs

1. Read and print a string
2. Print each character
3. Count characters
4. Find string length manually
5. Reverse a string manually

Challenge ☆

Check whether a string is a **Palindrome**.

Example:

```
madam

Reverse = madam

Palindrome
```

DAY 15 – String Manipulation Functions

Now you are ready to use the built-in string library.

Header File

```
#include <string.h>
```

Functions

1. `strlen()`

Find string length.

```
strlen(name);
```

2. `strcpy()`

Copy one string to another.

```
strcpy(destination, source);
```

3. `strcat()`

Join two strings.

```
strcat(str1, str2);
```

4. `strcmp()`

Compare two strings.

```
strcmp(str1, str2);
```

`strrev()`

Mention only as:

Compiler-dependent / non-standard function. Do not make it a core C requirement.

Programs

1. Find string length
2. Copy string
3. Concatenate two strings
4. Compare two strings
5. Count vowels
6. Count spaces
7. Convert/check strings using appropriate standard techniques

Challenge ☆

Create a program to determine whether **two strings are equal without using `strcmp()`.**

DAY 16 – Functions Fundamentals

Only after you have learned several programs should we introduce functions.

Topics

- What is a function?
- Why functions?
- Function declaration/prototype
- Function definition
- Function calling
- Parameters
- Return value
- `void` functions
- Functions with arguments
- Functions with return values

Level 1 – No Arguments, No Return

```
void display()
{
    printf("Hello");
}
```

Calling:

```
display();
```

Level 2 – Arguments, No Return

```
void add(int a, int b)
{
    printf("%d", a + b);
}
```

Level 3 – Arguments + Return Value

```
int add(int a, int b)
{
    return a + b;
}
```

Calling:

```
int result = add(10, 20);
```

Challenge ☆

Create functions for:

```
addition()
subtraction()
multiplication()
division()
```

and create a **calculator program using functions**.

DAY 17 – Problem Solving Using Functions

Now you apply functions to programs they already know.

Programs

1. Factorial using function
2. Prime number using function
3. Fibonacci using function
4. Largest of two numbers using function
5. Largest of three numbers using function
6. Even/Odd using function
7. Palindrome using function
8. Armstrong number using function

Challenge ☆

Create a **menu-driven mathematical program using functions**.

Example:

```
1. Factorial
2. Prime
3. Palindrome
4. Armstrong
5. Exit
```

DAY 18 – Pointers

Topics

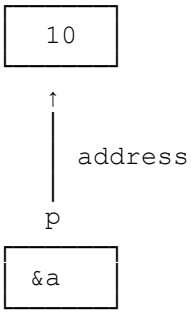
- What is a pointer?
- Address of a variable
- Address operator `&`
- Pointer declaration
- Dereferencing operator `*`
- Pointer initialization
- Pointer and variable relationship
- Pointer arithmetic

Basic Example

```
int a = 10;

int *p = &a;

printf("%d", *p);
```

Visual

```
Variable

a
```



Remember:

```
& → Address

* → Value at Address
```

Programs

1. Pointer basics
2. Print address of variable
3. Access value using pointer
4. Modify value using pointer
5. Pointer arithmetic
6. Swap two numbers using pointers

Challenge ☆

Create a function that **swaps two numbers using pointers**.

DAY 19 – Structures

After arrays, strings, functions, and pointers, you are ready for structures.

Topics

- What is a structure?
- Need for structure
- Structure declaration
- Structure variable
- Initialization
- Accessing members
- Structure with multiple data types
- Array of structures
- Structure with functions

Example

```
struct Student
{
    int roll;
    char name[30];
    float marks;
};
```

Create variable:

```
struct Student s1;
```

Access members:

```
s1.roll
s1.name
s1.marks
```

Programs

1. Student details
2. Employee details
3. Book details
4. Student marks
5. Array of you

Challenge ☆

Create a **Student Record Management Program** using structures.

DAY 20 – Unions & Preprocessor Directives

Part A – Union

Topics

- What is a union?
- Union declaration
- Union variables
- Accessing union members
- Memory allocation
- Structure vs Union

Example

```
union Data
{
    int i;
    float f;
    char ch;
};
```

Important Concept

In a structure:
`All members get separate memory.`
In a union:
`All members share the same memory.`

Structure vs Union

Structure	Union
Separate memory for members	Shared memory
Multiple members can hold values simultaneously	Normally one member's stored value is meaningful at a time
Size is generally related to all members	Size is generally based on the largest member, subject to alignment
Used for records	Used when alternative representations share storage

Part B – Preprocessor Directives

Topics

- What is a preprocessor?
- `#include`
- `#define`
- Macro
- Conditional compilation
- `#ifdef`
- `#ifndef`
- `#if`
- `#else`
- `#endif`

Example

```
#define PI 3.14159
```

Program:

```
#include <stdio.h>

#define PI 3.14159

int main()
{
    printf("%f", PI);

    return 0;
}
```

Conditional Compilation

```
#ifdef DEBUG
    printf("Debug Mode");
#endif
```

Challenge ☆

Create a program using:

```
#define
#if
#else
#endif
```

to demonstrate **conditional compilation**.

DAY 21 – Final Revision + Mini Projects + Practical Challenge

This should be the **final day**, rather than ending at Day 20.

Part 1 – Complete Revision

You revise:

```
Day 1–5
C Fundamentals

Day 6–9
Conditions + Loops

Day 10–11
Loop Problems + Nested Loops

Day 12–13
Arrays + Matrices

Day 14–15
Strings

Day 16–17
Functions

Day 18
Pointers

Day 19
Structures

Day 20
Unions + Preprocessor
```

Final Mini Projects

Give you a choice.

Project 1 – Student Management System

Use:

- `struct`
- arrays
- functions
- strings
- loops
- conditional statements

Features:

```
1. Add Student
2. Display You
3. Search Student
4. Calculate Average
5. Find Highest Marks
6. Exit
```

Project 2 – Simple ATM

Use:

- `switch`
- `if-else`
- loops
- functions

```
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
```

Project 3 – Electricity Bill Calculator

Use:

- input/output
- conditions
- functions
- calculations

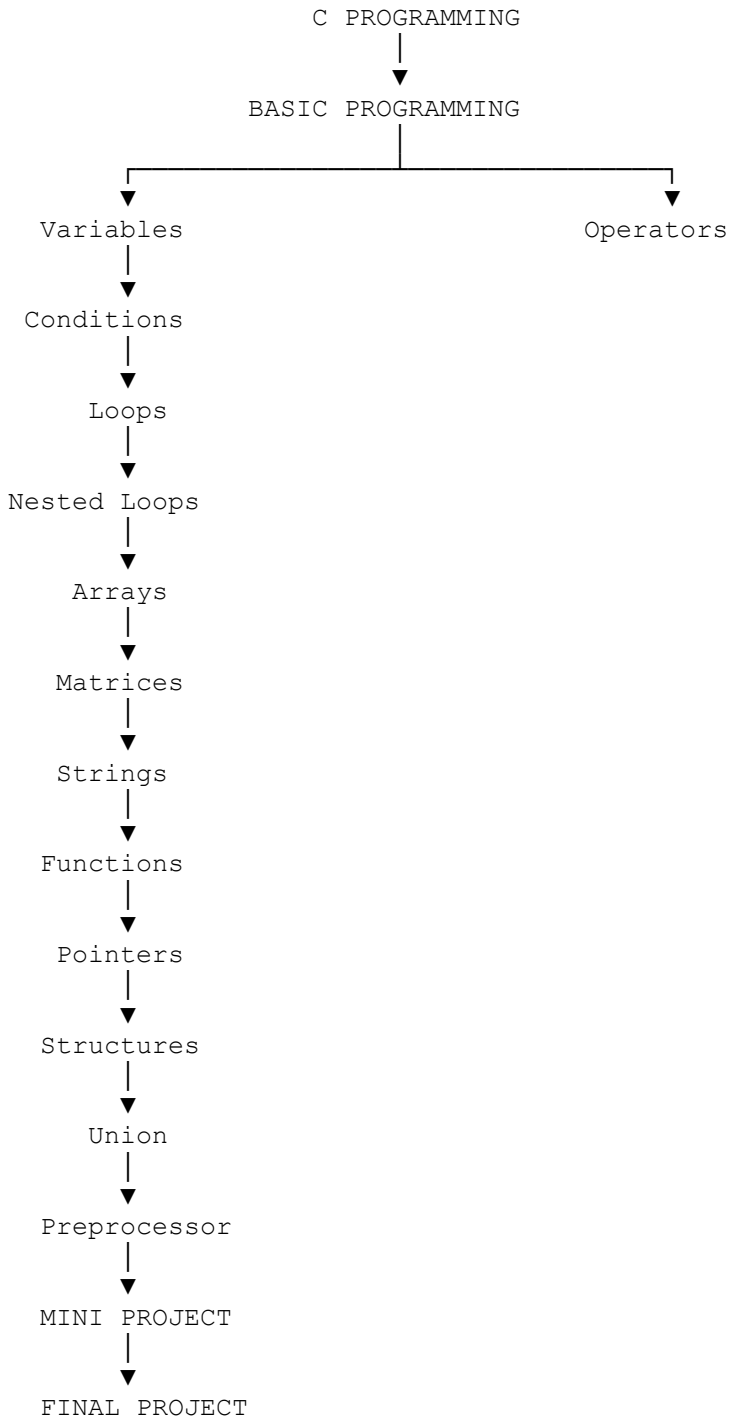
Project 4 – Employee Record System

Use:

- structures
- arrays
- functions
- strings

🎯 Final 21-Day Skill Progression

This is the important part of your course design:



Final Corrected Syllabus at a Glance

Day	Main Topic	Student Level
10	Loop Problem Solving	Beginner → Intermediate
11	Nested Loops & Patterns	Intermediate
12	1-D Arrays	Intermediate
13	2-D Arrays & Matrices	Intermediate
14	String Fundamentals	Intermediate
15	String Functions	Intermediate
16	Functions Fundamentals	Intermediate
17	Problem Solving Using Functions	Intermediate
18	Pointers	Intermediate → Advanced
19	Structures	Advanced Foundation
20	Unions & Preprocessor	Advanced Foundation
21	Revision + Mini Projects + Final Challenge	**Practical Mastery**

One important teaching principle

Don't make **Day 18–20** just theory. By that stage, you should be solving programs with the new concept **and combining it with everything they learned earlier**.

For example:

Structure + Array + String + Function + Loop

is much more valuable at the end of the course than learning each topic in isolation.

That will make your 21-day challenge feel like a genuine progression from **"I don't know C"** → **"I can write and understand C programs."**