Preface

In today's technology-driven world, the ability to code is rapidly becoming an essential skill—not just for computer scientists, but for anyone who seeks to understand and shape the future. This book, **"Learn to Code – Volume 1,"** is crafted with the intention of introducing foundational programming concepts using the C language—a language that continues to power everything from embedded systems to operating systems.

The content is designed especially for beginners and students in technical disciplines who are just stepping into the world of programming. Each chapter addresses a specific problem, accompanied by detailed explanations, sample code, and walkthroughs that make learning engaging and practical. Starting from basic arithmetic operations and moving through logical constructs, recursion, array manipulations, and pattern printing, this volume ensures a gradual and structured approach to building programming logic.

The pedagogy followed here focuses on real-time problem-solving, with clear and concise narration to strengthen the learner's ability to write efficient and readable code. By solving these problems and understanding the logic behind each, readers will develop not just the technical skill, but

also the confidence to tackle more complex challenges in computer science.

It is our hope that this book serves as a stepping stone for learners to build a strong foundation in coding and logical thinking. May this be the beginning of a rewarding and empowering journey into the world of programming.

Happy Learning and Happy Coding!

Index

Problem 1

Introduction

"Hello everyone! Today, we're going to learn how to code matrix multiplication using the C programming language. Matrix multiplication is a fundamental operation in mathematics, physics, and computer science, especially in areas like graphics, AI, and engineering."

Scene 1: Problem Definition

"Let's start by defining the problem. We are given two matrices — Matrix A and Matrix B. Our goal is to multiply them to produce a third matrix, Matrix C."

"But remember: Matrix multiplication is only valid when the number of columns in Matrix A is equal to the number of rows in Matrix B."

Scene 2: Matrix Multiplication Formula

"If A is an M×N matrix and B is an N×P matrix, then their product C will be an M×P matrix."

"The value at position C[i][j] is the sum of the product of elements from row i of A and column j of B. That is:

C[i][j] = A[i][0]*B[0][j] + A[i][1]*B[1][j] + ... + A[i][n-1]*B[n-1][j];

Scene 3: Code Walkthrough

"Now let's write the code in C step by step."

Include the header:

#include <stdio.h>

Declare matrices:

"We'll use 2D arrays to store the matrices. Let's take maximum size as 10x10 for simplicity."

int A[10][10], B[10][10], C[10][10];

Input dimensions:

"We'll ask the user to input the dimensions of matrices A and B, and check if multiplication is possible."

int m, n, p, q;

printf(\"Enter rows and columns of matrix A: \");

scanf(\"%d%d\", &m, &n);

printf(\"Enter rows and columns of matrix B: \");

scanf(\"%d%d\", &p, &q);

if (n != p) {

printf(\"Matrix multiplication not possible.\\n\");

return 0;

}

Scene 4. Input elements:

"Then we take input for both matrices."

printf(\"Enter elements of matrix A:\\n\");

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

scanf(\"%d\", &A[i][j]);

printf(\"Enter elements of matrix B:\\n\");

for (int i = 0; i < p; i++)

for (int j = 0; j < q; j++)

scanf(\"%d\", &B[i][j]);

Scene 5. Multiply matrices:

"Now comes the main part — the multiplication logic."

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

for (int j = 0; j < q; j++) {

C[i][j] = 0;

for (int k = 0; k < n; k++) {

C[i][j] += A[i][k] * B[k][j];

}

}

}

This loop calculates each element of the resulting matrix C when multiplying two matrices:

Matrix A of size m × n

Matrix B of size n × q

Result: Matrix C of size m × q

for (int i = 0; i < m; i++)

Iterates over rows of Matrix A and Matrix C.

i represents the current row in Matrix A and C.

for (int j = 0; j < q; j++)

Iterates over columns of Matrix B and Matrix C.

j represents the current column in Matrix B and C.

So, at this point, you're addressing element C[i][j], which is the cell at:

Row i in result matrix C

Column j in result matrix C

C[i][j] = 0;

Initializes the element C[i][j] to zero.

You'll accumulate the sum of products into this variable.

for (int k = 0; k < n; k++)

This loop handles the dot product operation.

It runs across:

Row i of Matrix A: A[i][k]

Column j of Matrix B: B[k][j]

It multiplies matching elements and adds them up to compute C[i][j].

Scene 6. Output the result:

```
printf(\"\\nResultant Matrix C:\\n\");

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

for (int j = 0; j < q; j++) {

printf(\"%d \", C[i][j]);

}
```

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

}
```

Scene 7: Conclusion

"And that’s it! You’ve just written a full program to multiply two matrices in C."

"You can extend this further by using dynamic memory allocation or accepting floating point values. Thanks for watching, and happy coding!"

****

Problem 2

"Hi everyone! Welcome to today's session. In this video, we'll learn how to write a C program to find the factorial of a given number. This is a common example for learning loops, recursion, and control flow in programming."

Scene 1: What is a Factorial?

"Before we dive into coding, let's understand what a factorial is."

"The factorial of a number n, written as n!, is the product of all positive integers from 1 to n."

"For example:

5! = 5 × 4 × 3 × 2 × 1 = 120

"By definition:

0! = 1

"Factorial is only defined for non-negative integers."

Scene 2: Logic and Approach

"We can calculate factorial in two common ways:

Using loops (iterative method)

Using recursion

"Let's first implement it using a simple loop."

Scene 3: C Program Code (Iterative Method)

"Now, let's write the code in C."

```
#include <stdio.h>

int main() {
```

```
int n;
unsigned long long factorial = 1;

printf("Enter a number: ");
scanf("%d", &n);
// Negative number check
if (n < 0) {
printf("Factorial is not defined for negative numbers.");
} else {
for (int i = 1; i <= n; i++) {
factorial *= i;
}
printf("Factorial of %d = %llu", n, factorial);
}
return 0;
}
```

Scene 4: Explanation of the Code

"Let's go through the code step-by-step:"

#include <stdio.h> includes the standard input/output library.

We declare an integer n for user input and factorial of type unsigned long long to handle large results.

We prompt the user to enter a number.

Then, we check if the number is negative. If yes, we display an error.

Otherwise, we use a for loop that runs from 1 to n, multiplying each value into factorial.

Finally, we print the result.

long long factorial(int n)

factorial is the function name.

int n is the input parameter — the number you want the factorial of.

long long is the return type. We use long long because factorial values grow very quickly, and regular int (usually 4 bytes) may overflow. long long can hold much larger numbers (typically 8 bytes).

Recursive Concept

This function uses recursion, which means:

A function calls itself with a smaller sub-problem until it reaches a stopping condition (base case).

In our case:

n! = n × (n - 1)!

Step-by-step Logic:

Base Case:

if (n == 0 || n == 1)

return 1;

If n is 0 or 1, the factorial is defined as 1.

This is the stopping condition for the recursion — without it, the function would call itself infinitely.

Recursive Case:

else

return n * factorial(n - 1);

If n is greater than 1, the function calls itself:

factorial(n) becomes n × factorial(n-1)

It keeps calling until it reaches 1 or 0 (the base case)

Example: factorial(4)
Let's trace the function calls:

factorial(4) → 4 × factorial(3)

→ 4 × (3 × factorial(2))

→ 4 × (3 × (2 × factorial(1)))

→ 4 × (3 × (2 × 1))

→ 4 × 3 × 2 × 1 = 24

Each call waits for the result of the next until it gets to factorial(1), which returns 1, and then multiplies upward.

Scene 5: Sample Output

"Let's run the program and input 5:"

mathematica

CopyEdit

Enter a number: 5

Factorial of 5 = 120

Scene 6: Bonus – Recursive Version

"Here's a bonus version using recursion for the same task."

```
#include <stdio.h>
long long factorial(int n) {
if (n == 0 || n == 1)
return 1;
else
```

```
return n * factorial(n - 1);
}
int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);
if (num < 0)
printf("Factorial not defined for negative numbers.");
else
printf("Factorial of %d = %lld", num, factorial(num));
return 0;
}
```

Scene 7: Conclusion

"And that's how you find the factorial of a number in C - both with loops and recursion!"

"Keep experimenting with different inputs and try writing both versions yourself to strengthen your logic-building skills. Thanks for watching, and happy coding!"

*****

Problem 4