. Introduction to C Programming

- **What is C?**
  C is a general-purpose programming language created by Dennis Ritchie in 1972. It is widely used for system programming, game development, and creating applications.
- **Features of C:**
  - Simple and easy to learn
  - Structured programming language
  - Fast execution speed
  - Supports low-level (hardware-level) programming

---

2. Structure of a C Program

Every C program follows this basic structure:

```
#include <stdio.h> // Header file for input and output


int main() {      // Main function - entry point of the program
   // Code goes here
   return 0;     // Returns 0 to indicate successful execution
}
```

Example: A simple program to print "Hello, World!"

```
#include <stdio.h>


int main() {
   printf("Hello, World!\n"); // Print statement
   return 0;
}
```

Output:

Copy code

Hello, World!

---

3. Variables and Data Types

- **Variables:** Containers to store data.
- **Data Types:** Define the type of data a variable can hold.

Data Type	Description	Example
int	Integer (whole numbers)	1, 2, -100
float	Decimal numbers	3.14, -0.99
char	Single character	'A', 'z'

Example:

```
#include <stdio.h>

int main() {
   int age = 25;       // Integer variable
   float height = 5.9;  // Float variable
   char grade = 'A';    // Character variable

   printf("Age: %d\n", age);
   printf("Height: %.1f\n", height);
   printf("Grade: %c\n", grade);

   return 0;
}
```

Output:

```
makefile
Age: 25
Height: 5.9
Grade: A
```

---

4. Input and Output

- **printf** is used to print output.
- **scanf** is used to take input.

Example: Taking input from the user.

```
#include <stdio.h>
int main() {
   int number;

   printf("Enter a number: ");
   scanf("%d", &number); // Take input from the user

   printf("You entered: %d\n", number);

   return 0;
}
```

Output:

```
Enter a number: 10
You entered: 10
```

---

5. Operators

C provides various operators for performing operations.

Operator	Description	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (Remainder)	a % b

Example: Arithmetic operations.

c

```
#include <stdio.h>
int main() {
   int a = 10, b = 3;

   printf("Addition: %d\n", a + b);
   printf("Subtraction: %d\n", a - b);
   printf("Multiplication: %d\n", a * b);
   printf("Division: %d\n", a / b);
   printf("Modulus: %d\n", a % b);

   return 0;
}
```

Output:

makefile

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3
Modulus: 1
```

---

6. Control Statements

Control statements let us control the flow of the program.

1. **If-Else Statement**

```
if (condition) {
   // Code if condition is true
} else {
   // Code if condition is false
}
```

Example:

```
#include <stdio.h>

int main() {
   int num = 5;

   if (num > 0) {
      printf("Positive number\n");
   } else {
      printf("Negative number\n");
   }

   return 0;
}
```

Output:

Positive number

2. **For Loop**

```
for (initialization; condition; increment/decrement) {
   // Code to repeat
}
```

Example:

```
#include <stdio.h>

int main() {
   for (int i = 1; i <= 5; i++) {
      printf("%d\n", i);
   }

   return 0;
}
```

Output:

1

2

3

4

5

---

7. Functions

Functions are reusable blocks of code.

- **Syntax:**

```
return_type function_name(parameters) {
   // Function code
}
```

Example: Function to add two numbers.

```
#include <stdio.h>

int add(int a, int b) {  // Function definition
   return a + b;
}

int main() {
   int result = add(10, 20); // Function call
   printf("Sum: %d\n", result);

   return 0;
}
```

Output:

Sum: 30

---

8. Arrays

Arrays store multiple values of the same type.
Example:

```
#include <stdio.h>


int main() {
   int numbers[5] = {10, 20, 30, 40, 50};


   for (int i = 0; i < 5; i++) {
      printf("numbers[%d] = %d\n", i, numbers[i]);
   }


   return 0;
}
```

Output:

```
numbers[0] = 10
numbers[1] = 20
numbers[2] = 30
numbers[3] = 40
numbers[4] = 50
```

---

9. Conclusion

C programming is a powerful and versatile language that forms the foundation for learning advanced programming concepts. Start by practicing basic concepts and gradually move to advanced topics like pointers, structures, and file handling.