Session 2: Operators & I/O in C Programming

Master the fundamental tools for computation, logic, and user interaction in C programming.

Arithmetic Operators: The Building Blocks of Computation

Arithmetic operators form the foundation of mathematical operations in C, enabling everything from simple calculations to complex algorithms.

Basic Operations

Addition (+), Subtraction (-), Multiplication (*), Division (/)

Modulus (%)

Returns the remainder after division—perfect for cyclic patterns

```
int result = 10 % 3;  // result is 1
int sum = 15 + 7;     // sum is 22
float quotient = 9.0 / 2.0;  // 4.5
```

Relational Operators: Making Comparisons

Relational operators evaluate relationships between values, returning true (1) or false (0). They're essential for decision-making in programme control flow.

1

Equality Check

== tests if two values are equal

!= tests if values are not equal

2

Magnitude

> greater than, **<** less than

>= greater or equal, **<=** less or equal

3

Practical Use

if (a >= b) checks if *a* is greater than or equal to b

Logical Operators: Combining Conditions

&& (AND)

Both conditions must be true

```
if (x > 0 && y < 10)
```

|| (OR)

At least one condition must be true

```
if (age < 18 || age > 65)
```

! (NOT)

Inverts the boolean value

```
if (!isValid)
```

Short-circuit evaluation: The && operator stops checking if the first condition is false. The || operator stops if the first condition is true. This optimises performance and prevents errors.

Bitwise Operators: Manipulating Data at the Binary Level

Bitwise operators work directly with binary representations of data, enabling low-level manipulation for optimisation and hardware control.

& (AND), | (OR), ^ (XOR)

Perform logical operations on each bit

~ (NOT)

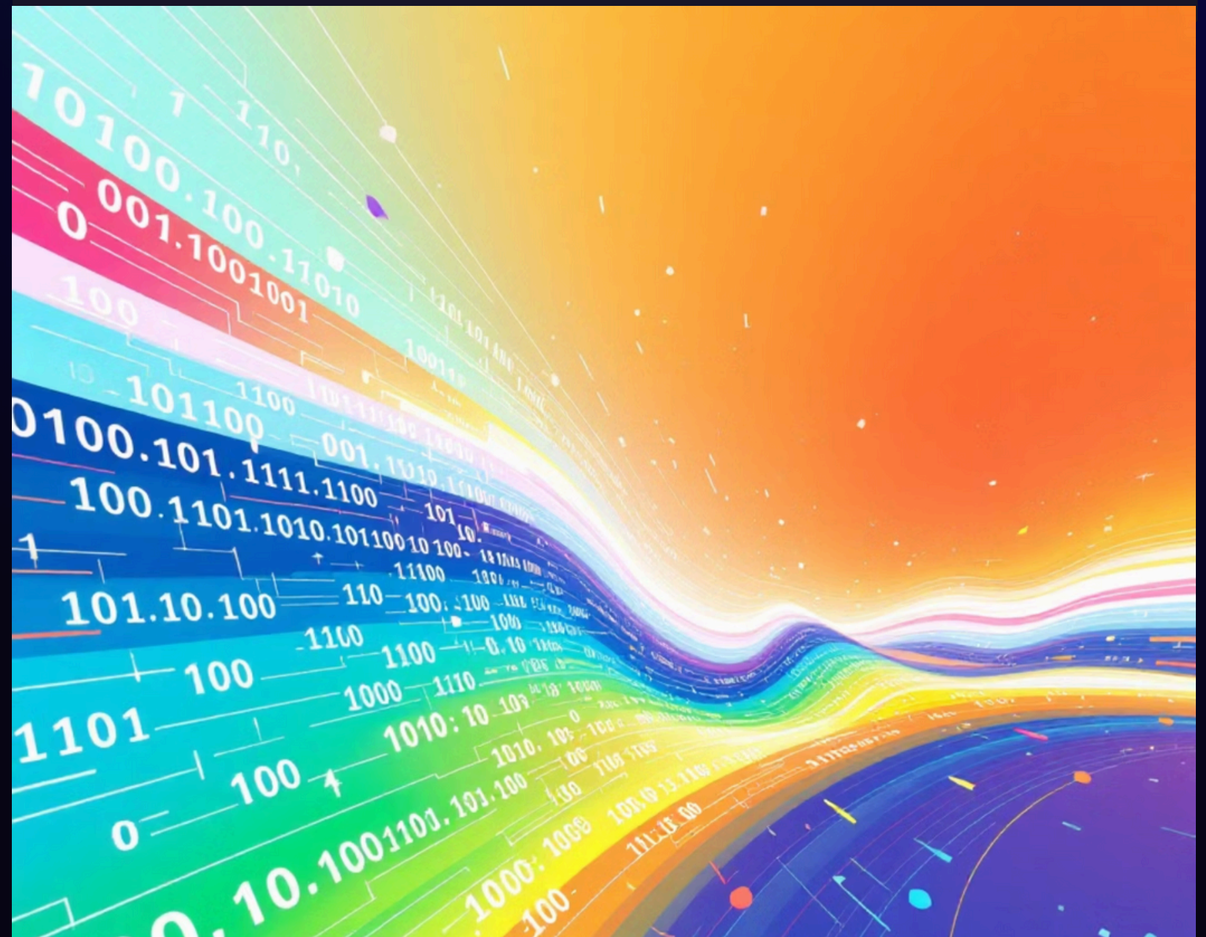
Flips all bits (one's complement)

<< (left shift), >> (right shift)

Move bits left or right, effectively multiplying or dividing by powers of 2

```
// Left shift: multiply by 4
int x = 5;        // Binary: 0101
x = x << 2;       // Binary: 10100
// Result: x is now 20

// Bitwise AND for even/odd check
if (num & 1) {
    // Odd number
}
```



Assignment Operators: Shorthand for Efficiency

Assignment operators provide concise syntax for common operations, making code cleaner and reducing potential errors.

1

Basic Assignment

= assigns a value

2

Compound Operators

+=, -=, *=, /=, %= combine operation with assignment

3

Result

Fewer keystrokes, clearer intent, reduced bugs

Traditional Syntax

```
count = count + 5;
total = total - discount;
value = value * 2;
```

Shorthand Syntax

```
count += 5;
total -= discount;
value *= 2;
```

The Ternary Operator: Compact Conditional Expressions

The ternary operator offers a streamlined alternative to simple if-else statements, perfect for inline conditional assignments.

01

Evaluate Condition

Test a boolean expression

02

True Path

Execute first expression if condition is true

03

False Path

Execute second expression if condition is false

```
// Syntax: condition ? expr_if_true : expr_if_false
int max = (a > b) ? a : b;
char* status = (score >= 50) ? "Pass" : "Fail";
printf("Grade: %c\n", (marks > 90) ? 'A' : 'B');
```

Standard I/O in C: printf() for Output

The printf() function is your primary tool for displaying formatted output to the console, offering precise control over data presentation.

Format Specifiers

- **%d** or **%i** - integers
- **%f** - floating-point numbers
- **%c** - single characters
- **%s** - strings
- **%p** - pointers

Precision Control

%10d - minimum width of 10

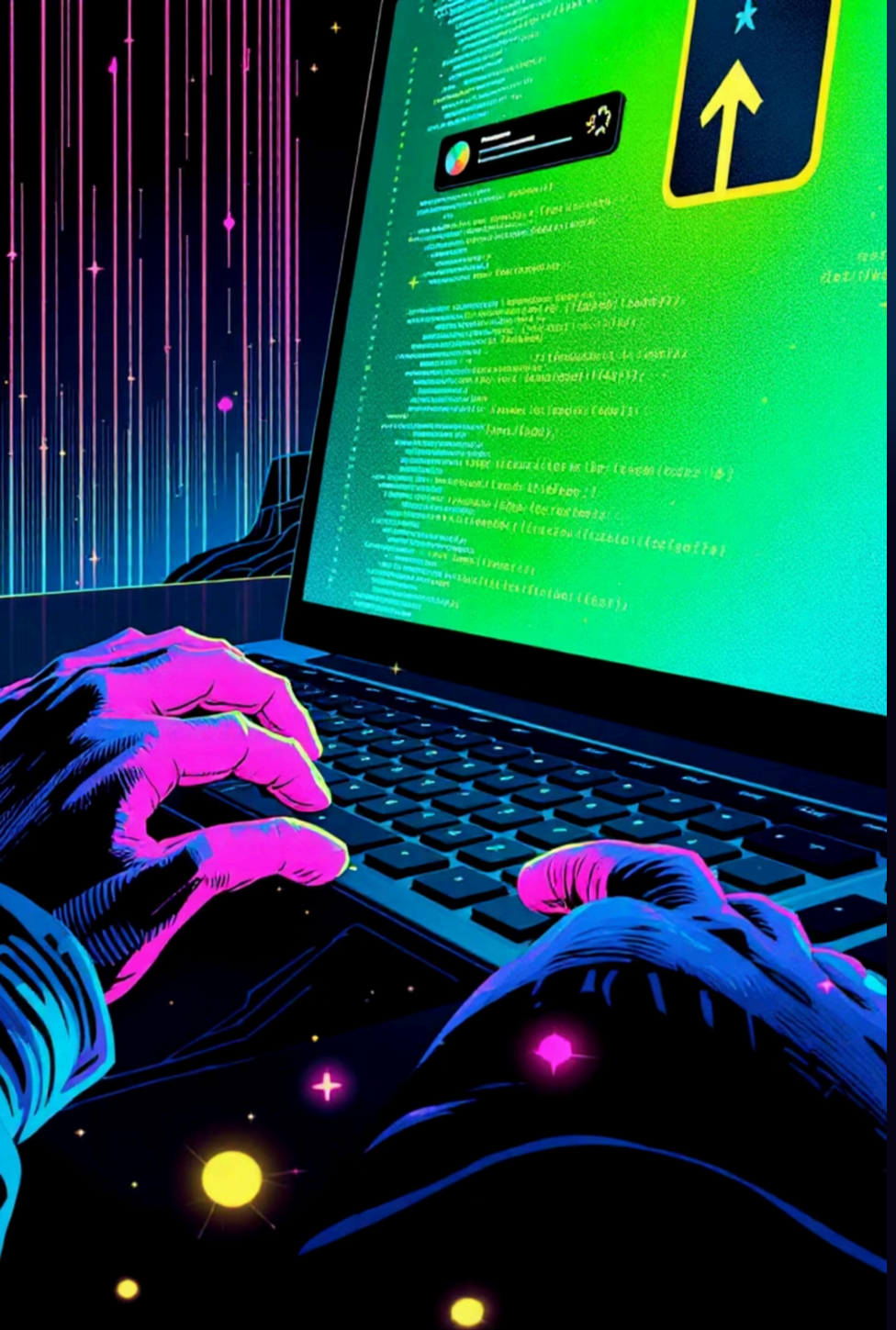
%.2f - 2 decimal places

%10.2f - width 10, 2 decimals

```
int score = 87;
float average = 87.456;
char grade = 'B';

printf("Score: %d\n", score);
printf("Average: %.2f\n", average);
printf("Grade: %c\n", grade);

// Output:
// Score: 87
// Average: 87.46
// Grade: B
```



Standard I/O in C: scanf() for Input

The scanf() function reads formatted input from the user, enabling interactive programmes. Proper usage requires attention to address operators and format specifiers.

Critical: Use Address Operator

Always use **&** before variable names to pass their memory addresses (except for strings/arrays)

```
scanf("%d", &age);  // Correct ✓
scanf("%d", age);   // Wrong ✗
```

Multiple Inputs

Read several values in one statement with space-separated format specifiers

```
scanf("%d %f", &age, &height);
```

Common Pitfall

Whitespace characters (space, tab, newline) can cause issues with %c. Use a space before %c to skip whitespace:

```
scanf(" %c", &choice);
```

Summary & Takeaway

Operators Enable Power

From arithmetic to bitwise manipulation, operators provide the tools for data transformation and decision-making logic.

I/O Enables Interaction

printf() and scanf() form the foundation of user communication in C programmes.

Mastery Through Practice

Combine operators with I/O functions to build real-world applications—practice is essential for proficiency.