■ Operators in C - Notes

In C programming, operators are special symbols used to perform operations on variables and values. Operators are the foundation of most computations and expressions in C.

■ Types of Operators in C: 1. Arithmetic Operators 2. Relational Operators 3. Logical Operators 4. Bitwise Operators 5. Assignment Operators 6. Increment/Decrement Operators 7. Conditional (Ternary) Operator 8. Special Operators

■ Arithmetic Operators: These operators are used to perform basic mathematical operations. + Addition - Subtraction * Multiplication / Division % Modulus (remainder) Example: int a = 10, b = 3; int sum = a + b; // sum = 13 int mod = a % b; // mod = 1

■ Relational Operators: These are used to compare two values. == Equal to != Not equal to > Greater than < Less than >= Greater than or equal to <= Less than or equal to Returns 1 (true) or 0 (false).

■ Logical Operators: Used to combine multiple conditions. && Logical AND || Logical OR ! Logical NOT Example: if (a > 0 && b > 0) { ... }

■ Bitwise Operators: Operate on bits and perform bit-by-bit operations. & AND | OR ^ XOR ~ NOT << Left shift >> Right shift

■ Assignment Operators: Used to assign values to variables. = Assign += Add and assign -= Subtract and assign *= Multiply and assign /= Divide and assign %= Modulus and assign

■ Increment and Decrement Operators: ++ Increment by 1 -- Decrement by 1 int a = 5; a++; // becomes 6 ++a; // becomes 7

■ Conditional (Ternary) Operator: Syntax: condition ? expression1 : expression2; Example: int max = (a > b) ? a : b;

■ Special Operators: - sizeof: returns the size of a variable - &: address of operator - *: pointer dereferencing - ,: comma operator

■ Example Program: #include <stdio.h> int main() { int a = 5, b = 2; printf("Sum: %d\n", a + b); printf("Greater: %d\n", a > b); printf("Bitwise AND: %d\n", a & b); return 0; }

■ Output: Sum: 7 Greater: 1 Bitwise AND: 0

■ Summary: Operators enhance the expressiveness of C and allow concise coding. Understanding operator precedence and associativity is crucial in complex expressions.