C Programming: Deck 4

Operators

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

Topics Covered

What are Operators?

- Symbols that perform operations on operands
- Operands can be variables, constants, or expressions
- Essential for computations and logic
- Different operators for different purposes

Categories of Operators in C:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Increment/Decrement Operators
7. Special Operators

Arithmetic Operators

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

Notes:

- All work on numeric types
- Division: Integer division truncates
- Modulus: Works only with integers
- Modulus: Returns remainder of division

Program 1: Basic Arithmetic Operations

```c
#include <stdio.h>
int main() {
  int a = 20, b = 6;
  printf("a = %d, b = %d\n", a, b);
  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:

```
a = 20, b = 6
Addition: 26
Subtraction: 14
Multiplication: 120
Division: 3
Modulus: 2
```

Note:

- $20/6 = 3$ (integer division)
- $20\%6 = 2$ (remainder)

Program 2: Integer vs Float Division

```c
#include <stdio.h>
int main() {
  int a = 7, b = 2;
  float c = 7.0, d = 2.0;
  printf("Integer division:\n");
  printf("%d / %d = %d\n", a, b, a/b);
  printf("\nFloat division:\n");
  printf("%.1f / %.1f = %.2f\n",
         c, d, c/d);
  printf("\nMixed:\n");
  printf("%d / %.1f = %.2f\n",
         a, d, a/d);
  return 0;
}
```

Output:

```
Integer division:
7 / 2 = 3
Float division:
7.0 / 2.0 = 3.50
Mixed:
7 / 2.0 = 3.50
```

Key Point:

- At least one float = float result

Program 3: Modulus Operator Examples

```c
#include <stdio.h>
int main() {
  printf("Positive modulus:\n");
  printf("10 %% 3 = %d\n", 10 % 3);
  printf("15 %% 4 = %d\n", 15 % 4);
  printf("\nNegative modulus:\n");
  printf("-10 %% 3 = %d\n", -10 % 3);
  printf("10 %% -3 = %d\n", 10 % -3);
  printf("\nEven/Odd check:\n");
  printf("7 %% 2 = %d (odd)\n", 7%2);
  printf("8 %% 2 = %d (even)\n", 8%2);
  return 0;
}
```

Output:

```
Positive modulus:
10 % 3 = 1
15 % 4 = 3
Negative modulus:
-10 % 3 = -1
10 % -3 = 1
Even/Odd check:
7 % 2 = 1 (odd)
8 % 2 = 0 (even)
```

Use: Check divisibility

Increment and Decrement Operators

Operator	Name	Description
++	Increment	Increase by 1
--	Decrement	Decrease by 1

Two Forms:

- **Prefix:** `++x`, `--x` (increment/decrement first, then use)
- **Postfix:** `x++`, `x--` (use first, then increment/decrement)

Key Difference:

- Prefix: Returns new value
- Postfix: Returns old value

Program 4: Pre vs Post Increment

```c
#include <stdio.h>
int main() {
  int x = 5, y = 5;
  printf("Initial: x=%d, y=%d\n",
         x, y);
  printf("\nPrefix ++x:\n");
  printf("Value: %d\n", ++x);
  printf("After: x=%d\n", x);
  printf("\nPostfix y++:\n");
  printf("Value: %d\n", y++);
  printf("After: y=%d\n", y);
  return 0;
}
```

Output:

```
Initial: x=5, y=5
Prefix ++x:
Value: 6
After: x=6
Postfix y++:
Value: 5
After: y=6
```

Observation:

- Prefix returns 6
- Postfix returns 5
- Both end up as 6

Program 5: Increment in Expressions

```c
#include <stdio.h>
int main() {
  int a = 10, b = 10;
  int result1, result2;
  result1 = ++a + 5;
  printf("++a + 5:\n");
  printf("a=%d, result=%d\n",
         a, result1);
  result2 = b++ + 5;
  printf("\nb++ + 5:\n");
  printf("b=%d, result=%d\n",
         b, result2);
  return 0;
}
```

Output:

```
++a + 5:
a=11, result=16
b++ + 5:
b=11, result=15
```

Explanation:

- ++a: a becomes 11, then 11+5=16
- b++: uses 10, then b becomes 11
- Result: 10+5=15

Program 6: Decrement Operators

```c
#include <stdio.h>
int main() {
  int x = 10, y = 10;
  printf("Initial: x=%d, y=%d\n",
         x, y);
  printf("Prefix --x: %d\n", --x);
  printf("After: x=%d\n", x);
  printf("Postfix y--: %d\n", y--);
  printf("After: y=%d\n", y);
  printf("\nMultiple ops:\n");
  printf("--x: %d\n", --x);
  printf("--x: %d\n", --x);
  return 0;
}
```

Output:

```
Initial: x=10, y=10
Prefix --x: 9
After: x=9
Postfix y--: 10
After: y=9
Multiple ops:
--x: 8
--x: 7
```

Relational Operators

Operator	Name	Example
==	Equal to	`a == b`
!=	Not equal to	`a != b`
>	Greater than	`a > b`
<	Less than	`a < b`
>=	Greater than or equal	`a >= b`
<=	Less than or equal	`a <= b`

Return Value:

- 1 if condition is true
- 0 if condition is false

Important: == (comparison) vs = (assignment)

Program 7: Relational Operators

```c
#include <stdio.h>
int main() {
  int a = 10, b = 20;
  printf("a = %d, b = %d\n", a, b);
  printf("\nRelational Results:\n");
  printf("a == b: %d\n", a == b);
  printf("a != b: %d\n", a != b);
  printf("a > b: %d\n", a > b);
  printf("a < b: %d\n", a < b);
  printf("a >= b: %d\n", a >= b);
  printf("a <= b: %d\n", a <= b);
  return 0;
}
```

Output:

```
a = 10, b = 20
Relational Results:
a == b: 0
a != b: 1
a > b: 0
a < b: 1
a >= b: 0
a <= b: 1
```

Note:

- 1 = true, 0 = false

Program 8: Using Relational in Conditions

```c
#include <stdio.h>
int main() {
  int age = 18;
  printf("Age: %d\n", age);
  if (age >= 18) {
    printf("Adult\n");
  } else {
    printf("Minor\n");
  }
  int marks = 85;
  printf("\nMarks: %d\n", marks);
  if (marks >= 90) {
    printf("Grade: A\n");
  } else if (marks >= 75) {
    printf("Grade: B\n");
  } else {
    printf("Grade: C\n");
  }
  return 0;
}
```

Output:

```
Age: 18
Adult
Marks: 85
Grade: B
```

Logical Operators

Operator	Name	Description
&&	Logical AND	Both conditions true
||	Logical OR	At least one true
!	Logical NOT	Negation

Truth Tables:

AND (&&):

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

OR (——):

A	B	A——B
0	0	0
0	1	1
1	0	1
1	1	1

NOT (!):

A	!A
0	1
1	0

Program 9: Logical AND Operator

```c
#include <stdio.h>
int main() {
  int a = 1, b = 0;
  printf("a=%d, b=%d\n", a, b);
  printf("\nLogical AND:\n");
  printf("a && a: %d\n", a && a);
  printf("a && b: %d\n", a && b);
  printf("b && b: %d\n", b && b);
  printf("\nPractical use:\n");
  int age = 25;
  int hasLicense = 1;
  if (age >= 18 && hasLicense) {
    printf("Can drive\n");
  }
  return 0;
}
```

Output:

```
a=1, b=0
Logical AND:
a && a: 1
a && b: 0
b && b: 0
Practical use:
Can drive
```

Note:

- Both must be true

Program 10: Logical OR Operator

```c
#include <stdio.h>
int main() {
  int a = 1, b = 0;
  printf("a=%d, b=%d\n", a, b);
  printf("\nLogical OR:\n");
  printf("a || a: %d\n", a || a);
  printf("a || b: %d\n", a || b);
  printf("b || b: %d\n", b || b);
  printf("\nPractical use:\n");
  char grade = 'A';
  if (grade=='A' || grade=='B') {
    printf("Pass with honors\n");
  }
  return 0;
}
```

Output:

```
a=1, b=0
Logical OR:
a || a: 1
a || b: 1
b || b: 0
Practical use:
Pass with honors
```

Note:

- At least one must be true

Program 11: Logical NOT Operator

```c
#include <stdio.h>
int main() {
  int a = 1, b = 0;
  printf("a=%d, b=%d\n", a, b);
  printf("\nLogical NOT:\n");
  printf("!a: %d\n", !a);
  printf("!b: %d\n", !b);
  printf("!!a: %d\n", !!a);
  printf("\nPractical use:\n");
  int isRaining = 0;
  if (!isRaining) {
    printf("Go outside\n");
  }
  return 0;
}
```

Output:

```
a=1, b=0
Logical NOT:
!a: 0
!b: 1
!!a: 1
Practical use:
Go outside
```

Note:

- Reverses truth value
- !! returns to original

Program 12: Combined Logical Operators

```c
#include <stdio.h>
int main() {
  int age = 25;
  int citizen = 1;
  int criminalRecord = 0;
  printf("Age: %d\n", age);
  printf("Citizen: %d\n", citizen);
  printf("Criminal: %d\n\n",
         criminalRecord);
  if (age>=18 && citizen &&
      !criminalRecord) {
    printf("Eligible to vote\n");
  }
  return 0;
}
```

Output:

```
Age: 25
Citizen: 1
Criminal: 0
Eligible to vote
```

Explanation:

- All conditions must be true
- age$\geq$18: true
- citizen: true
- !criminalRecord: true

Assignment Operators

Operator	Example	Equivalent to
=	`a = 5`	Simple assignment
+=	`a += 5`	`a = a + 5`
-=	`a -= 5`	`a = a - 5`
*=	`a *= 5`	`a = a * 5`
/=	`a /= 5`	`a = a / 5`
%=	`a %= 5`	`a = a % 5`

Benefits:

- Shorter syntax
- More readable
- Commonly used in loops and counters

Program 13: Compound Assignment Operators

```c
#include <stdio.h>
int main() {
  int x = 10;
  printf("Initial: x = %d\n", x);
  x += 5;
  printf("After x+=5: %d\n", x);
  x -= 3;
  printf("After x-=3: %d\n", x);
  x *= 2;
  printf("After x*=2: %d\n", x);
  x /= 4;
  printf("After x/=4: %d\n", x);
  x %= 5;
  printf("After x%%=5: %d\n", x);
  return 0;
}
```

Output:

```
Initial: x = 10
After x+=5: 15
After x-=3: 12
After x*=2: 24
After x/=4: 6
After x%=5: 1
```

Trace:

- 10+5=15
- 15-3=12
- 12*2=24
- 24/4=6
- 6%5=1

Bitwise Operators

Operator	Name	Description
&	Bitwise AND	Bit-by-bit AND
|	Bitwise OR	Bit-by-bit OR
^	Bitwise XOR	Bit-by-bit XOR
~	Bitwise NOT	Bit complement
<<	Left shift	Shift bits left
>>	Right shift	Shift bits right

Used for:

- Low-level programming
- Bit manipulation
- Flags and masks
- Performance optimization

Program 14: Bitwise AND, OR, XOR

```c
#include <stdio.h>
int main() {
  int a = 12;  // 1100 binary
  int b = 10;  // 1010 binary
  printf("a = %d (binary: 1100)\n",a);
  printf("b = %d (binary: 1010)\n",b);
  printf("\nBitwise operations:\n");
  printf("a & b = %d\n", a & b);
  printf("a | b = %d\n", a | b);
  printf("a ^ b = %d\n", a ^ b);
  printf("~a = %d\n", ~a);
  return 0;
}
```

Output:

```
a = 12 (binary: 1100)
b = 10 (binary: 1010)
Bitwise operations:
a & b = 8
a | b = 14
a ^ b = 6
~a = -13
```

Binary:

- AND: $1000 = 8$
- OR: $1110 = 14$
- XOR: $0110 = 6$

Program 15: Bit Shift Operators

```c
#include <stdio.h>
int main() {
   int x = 5;  // 0101 binary
   printf("x = %d (binary: 0101)\n",x);
   printf("\nLeft shift:\n");
   printf("x << 1 = %d\n", x << 1);
   printf("x << 2 = %d\n", x << 2);
   printf("\nRight shift:\n");
   int y = 20; // 10100 binary
   printf("y = %d\n", y);
   printf("y >> 1 = %d\n", y >> 1);
   printf("y >> 2 = %d\n", y >> 2);
   return 0;
}
```

Output:

```
x = 5 (binary: 0101)
Left shift:
x << 1 = 10
x << 2 = 20
Right shift:
y = 20
y >> 1 = 10
y >> 2 = 5
```

Pattern:

- Left shift: multiply by 2
- Right shift: divide by 2

Program 16: Practical Bitwise - Even/Odd

```c
#include <stdio.h>
int main() {
  int nums[] = {5, 8, 13, 20, 7};
  int i;
  printf("Check even/odd using &:\n");
  for (i = 0; i < 5; i++) {
    if (nums[i] & 1) {
      printf("%d is odd\n", nums[i]);
    } else {
      printf("%d is even\n",nums[i]);
    }
  }
  return 0;
}
```

Output:

```
Check even/odd using &:
5 is odd
8 is even
13 is odd
20 is even
7 is odd
```

Trick:

- LSB of odd numbers is 1
- n & 1 checks last bit

Special Operators

Operator	Name	Example
`sizeof`	Size of	`sizeof(int)`
`? :`	Ternary/Conditional	`a > b ? a : b`
`,`	Comma	`a = 1, b = 2`
`&`	Address of	`&x`
`*`	Pointer dereference	`*ptr`

Note: We'll focus on sizeof and ternary operator here

Program 17: sizeof Operator

```c
#include <stdio.h>
int main() {
  int x = 10;
  float f = 3.14;
  char c = 'A';
  int arr[10];
  printf("sizeof data types:\n");
  printf("int: %lu bytes\n",
         sizeof(int));
  printf("float: %lu bytes\n",
         sizeof(float));
  printf("char: %lu bytes\n",
         sizeof(char));
  printf("\nsizeof variables:\n");
  printf("x: %lu bytes\n", sizeof(x));
  printf("arr: %lu bytes\n",
         sizeof(arr));
  return 0;
}
```

Output:

```
sizeof data types:
int: 4 bytes
float: 4 bytes
char: 1 bytes
sizeof variables:
x: 4 bytes
arr: 40 bytes
```

Note:

- Array: $10 \times 4 = 40$ bytes

Ternary/Conditional Operator

Syntax:

```
condition ?  expression1 :  expression2
```

How it works:

- If condition is true, returns expression1
- If condition is false, returns expression2
- Compact alternative to if-else

Example:

- `max = (a > b) ?  a :  b;`
- If a ¿ b, max = a, else max = b

Program 18: Ternary Operator

```c
#include <stdio.h>
int main() {
  int a = 10, b = 20;
  int max, min;
  max = (a > b) ? a : b;
  min = (a < b) ? a : b;
  printf("a=%d, b=%d\n", a, b);
  printf("Max: %d\n", max);
  printf("Min: %d\n", min);
  int num = 7;
  printf("\n%d is %s\n", num,
         (num%2==0) ? "even":"odd");
  int age = 20;
  printf("Age %d: %s\n", age,
         (age>=18) ? "Adult":"Minor");
  return 0;
}
```

Output:

```
a=10, b=20
Max: 20
Min: 10
7 is odd
Age 20: Adult
```

Uses:

- Finding max/min
- Simple conditions
- Inline decisions

Program 19: Nested Ternary

```c
#include <stdio.h>
int main() {
  int marks = 85;
  char grade;
  grade = (marks >= 90) ? 'A' :
          (marks >= 75) ? 'B' :
          (marks >= 60) ? 'C' :
          (marks >= 50) ? 'D' : 'F';
  printf("Marks: %d\n", marks);
  printf("Grade: %c\n", grade);
  int x = 5, y = 10, z = 3;
  int largest;
  largest = (x>y) ?
            ((x>z)?x:z) :
            ((y>z)?y:z);
  printf("\nLargest of %d,%d,%d: %d\n",
         x, y, z, largest);
  return 0;
}
```

Output:

```
Marks: 85
Grade: B
Largest of 5,10,3: 10
```

Warning:

- Can get complex
- Use parentheses
- Consider if-else for clarity

Operator Precedence (Partial List)

Priority	Operators	Associativity
1 (Highest)	`() [] -> .`	Left to Right
2	`! ~ ++ -- + - * & sizeof`	Right to Left
3	`* / %`	Left to Right
4	`+ -`	Left to Right
5	`<< >>`	Left to Right
6	`< <= > >=`	Left to Right
7	`== !=`	Left to Right
8	`&`	Left to Right
9	`^`	Left to Right
10	`|`	Left to Right
11	`&&`	Left to Right
12	`||`	Left to Right
13	`? :`	Right to Left
14 (Lowest)	`= += -= *= /= %=`	Right to Left

Program 20: Precedence Example

```c
#include <stdio.h>
int main() {
  int result;
  result = 2 + 3 * 4;
  printf("2+3*4 = %d\n", result);
  result = (2 + 3) * 4;
  printf("(2+3)*4 = %d\n", result);
  result = 10 - 3 + 2;
  printf("10-3+2 = %d\n", result);
  result = 10 / 2 * 3;
  printf("10/2*3 = %d\n", result);
  result = 10 < 20 && 5 > 3;
  printf("10<20 && 5>3 = %d\n",
         result);
  return 0;
}
```

Output:

```
2+3*4 = 14
(2+3)*4 = 20
10-3+2 = 9
10/2*3 = 15
10<20 && 5>3 = 1
```

Explanation:

- * before +
- () highest precedence
- Left-to-right: 10-3=7, 7+2=9
- Left-to-right: 10/2=5, 5*3=15

Program 21: Complex Expression

```c
#include <stdio.h>
int main() {
  int a=5, b=10, c=15;
  int result;
  result = a + b * c / 5 - 2;
  printf("a=%d, b=%d, c=%d\n",
         a, b, c);
  printf("\na+b*c/5-2\n");
  printf("Step by step:\n");
  printf("b*c = %d\n", b*c);
  printf("150/5 = %d\n", 150/5);
  printf("a+30 = %d\n", a+30);
  printf("35-2 = %d\n", 35-2);
  printf("\nFinal result: %d\n",
         result);
  return 0;
}
```

Output:

```
a=5, b=10, c=15
a+b*c/5-2
Step by step:
b*c = 150
150/5 = 30
a+30 = 35
35-2 = 33
Final result: 33
```

Order:

- *, / (left-to-right)
- +, - (left-to-right)

Operators Summary

- **Arithmetic:** +, -, *, /, %
- **Relational:** ==, !=, ¡, ¿, ¡=, ¿=
- **Logical:** &&, ——, !
- **Bitwise:** &, —, ^, ~, ¡¡, ¿¿
- **Assignment:** =, +=, -=, *=, /=, %=
- **Inc/Dec:** ++, –
- **Special:** sizeof, ? :

Best Practices

1. Use parentheses for clarity
2. Understand precedence and associativity
3. Be careful with $++$ and $--$ in expressions
4. Use compound assignments for readability
5. Remember integer division truncates
6. Check types before using modulus
7. Use relational operators in conditions
8. Ternary for simple conditions only
9. Comment complex expressions

Common Mistakes

1. Using $=$ instead of $==$ in conditions
   - `if (x = 5)` assigns, doesn't compare!
2. Confusing && (logical) with & (bitwise)
3. Forgetting integer division
   - `5/2` is 2, not 2.5
4. Modulus with floats (not allowed)
5. Complex nested ternary operators
6. Not using parentheses in complex expressions

Try These!

1. Write expressions using all arithmetic operators
2. Swap two numbers using compound assignment
3. Check if a number is divisible by both 3 and 5
4. Find maximum of three numbers using ternary
5. Use bitwise operators to multiply/divide by powers of 2
6. Write complex expression and trace evaluation order

Sample: Max of Three Using Ternary

```c
#include <stdio.h>
int main() {
  int a=15, b=25, c=20;
  int max;
  max = (a>b) ?
        ((a>c) ? a : c) :
        ((b>c) ? b : c);
  printf("Numbers: %d,%d,%d\n",
         a, b, c);
  printf("Maximum: %d\n", max);
  return 0;
}
```

Output:

```
Numbers: 15,25,20
Maximum: 25
```

End of Deck 4

Questions? Next: Deck 5 - Operator Precedence & Associativity

(Detailed)