C Programming: Deck 5

Operator Precedence & Associativity

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

Topics Covered

Why Precedence & Associativity Matter

- Determine order of evaluation in expressions
- Critical for correct program behavior
- Avoid unexpected results
- Write clear, unambiguous code

Example:

- What is `2 + 3 * 4`?
- Is it `(2 + 3) * 4 = 20`? or
- Is it `2 + (3 * 4) = 14`?

Answer: 14 (multiplication has higher precedence than addition)

Key Concepts

Precedence:

- Determines which operator is evaluated first
- Higher precedence = evaluated earlier
- Similar to BODMAS/PEMDAS in mathematics

Associativity:

- Determines order when operators have same precedence
- Left-to-right: `a - b - c = (a - b) - c`
- Right-to-left: `a = b = c` means `a = (b = c)`

Parentheses:

- Override all precedence and associativity rules
- Always evaluated first

Operator Precedence Table (Part 1)

Level	Operators	Description	Assoc
1	`() [] -> .`	Parentheses, array, member	L to R

2	`++ --` (postfix)
	`++ --` (prefix)
	`+ -` (unary)
	`! ~`
	`* &`
	`sizeof`
		Postfix inc/dec
		Prefix inc/dec
		Unary plus, minus
		Logical NOT, bitwise NOT
		Dereference, address-of
		Size of
			R to L

3	`* / %`	Multiply, divide, modulus	L to R
4	`+ -` (binary)	Addition, subtraction	L to R
5	`<< >>`	Left shift, right shift	L to R

Operator Precedence Table (Part 2)

Level	Operators	Description	Assoc
6	`< <= > >=`	Relational operators	L to R
7	`== !=`	Equality operators	L to R
8	`&`	Bitwise AND	L to R
9	`^`	Bitwise XOR	L to R
10	`|`	Bitwise OR	L to R
11	`&&`	Logical AND	L to R
12	`||`	Logical OR	L to R
13	`?:`	Ternary conditional	R to L
14	`= += -= *= /= %= &= ^= |= <<= >>=`	Assignment operators	R to L
15	`,`	Comma operator	L to R

Precedence Hierarchy - Quick Reference

Remember: High to Low

1. Parentheses `()`
2. Unary: `++ -- ! ~ + - * &`
3. Arithmetic: `* / %` then `+ -`
4. Shift: `<< >>`
5. Relational: `< <= > >=` then `== !=`
6. Bitwise: `&` then `^` then `|`
7. Logical: `&&` then `||`
8. Ternary: `?:`
9. Assignment: `= += -= ...`
10. Comma: `,`

Program 1: Arithmetic Precedence

```c
#include <stdio.h>
int main() {
  int result;
  result = 10 + 5 * 2;
  printf("10 + 5 * 2 = %d\n",
         result);
  printf("Evaluation: 10+(5*2)\n");
  printf("  5*2 = 10\n");
  printf("  10+10 = 20\n\n");
  result = 20 - 10 / 2;
  printf("20 - 10 / 2 = %d\n",
         result);
  printf("Evaluation: 20-(10/2)\n");
  printf("  10/2 = 5\n");
  printf("  20-5 = 15\n");
  return 0;
}
```

Output:

```
10 + 5 * 2 = 20
Evaluation: 10+(5*2)
  5*2 = 10
  10+10 = 20
20 - 10 / 2 = 15
Evaluation: 20-(10/2)
  10/2 = 5
  20-5 = 15
```

Rule:

- *, /, % before +, -

Program 2: Multiple Operations Same Level

```c
#include <stdio.h>
int main() {
  int result;
  result = 20 / 4 * 2;
  printf("20 / 4 * 2 = %d\n",
         result);
  printf("Left to right:\n");
  printf("  20/4 = 5\n");
  printf("  5*2 = 10\n\n");
  result = 100 - 50 + 25;
  printf("100 - 50 + 25 = %d\n",
         result);
  printf("Left to right:\n");
  printf("  100-50 = 50\n");
  printf("  50+25 = 75\n");
  return 0;
}
```

Output:

```
20 / 4 * 2 = 10
Left to right:
  20/4 = 5
  5*2 = 10

100 - 50 + 25 = 75
Left to right:
  100-50 = 50
  50+25 = 75
```

Associativity:

- Same precedence
- Evaluate left-to-right

Program 3: Complex Arithmetic Expression

```c
#include <stdio.h>
int main() {
  int result;
  result = 2+3*4-10/2+6%4;
  printf("Expression:\n");
  printf("2+3*4-10/2+6%%4\n\n");
  printf("Step 1 (*, /, %%):\n");
  printf("  3*4 = 12\n");
  printf("  10/2 = 5\n");
  printf("  6%%4 = 2\n");
  printf("  => 2+12-5+2\n\n");
  printf("Step 2 (+, -):\n");
  printf("  2+12 = 14\n");
  printf("  14-5 = 9\n");
  printf("  9+2 = 11\n\n");
  printf("Result: %d\n", result);
  return 0;
}
```

Output:

```
Expression:
2+3*4-10/2+6%4
Step 1 (*, /, %):
  3*4 = 12
  10/2 = 5
  6%4 = 2
  => 2+12-5+2
Step 2 (+, -):
  2+12 = 14
  14-5 = 9
  9+2 = 11
Result: 11
```

Program 4: Unary vs Binary Operators

```c
#include <stdio.h>
int main() {
  int a = 5, b = 3;
  int result;
  result = -a + b;
  printf("a=%d, b=%d\n", a, b);
  printf("\n-a + b:\n");
  printf("  -a = %d\n", -a);
  printf("  -5 + 3 = %d\n\n",
         result);
  result = a * -b;
  printf("a * -b:\n");
  printf("  -b = %d\n", -b);
  printf("  5 * -3 = %d\n\n",
         result);
  result = -a * -b;
  printf("-a * -b = %d\n", result);
  return 0;
}
```

Output:

```
a=5, b=3

-a + b:
  -a = -5
  -5 + 3 = -2
a * -b:
  -b = -3
  5 * -3 = -15
-a * -b = 15
```

Note:

- Unary - has higher precedence
- Evaluated before binary operators

Program 5: Increment with Other Operators

```c
#include <stdio.h>
int main() {
  int x = 5, y = 5;
  int result;
  result = ++x * 2;
  printf("++x * 2:\n");
  printf("  x becomes 6\n");
  printf("  6 * 2 = %d\n",result);
  printf("  x = %d\n\n", x);
  result = y++ * 2;
  printf("y++ * 2:\n");
  printf("  uses y=5\n");
  printf("  5 * 2 = %d\n",result);
  printf("  then y becomes %d\n",y);
  return 0;
}
```

Output:

```
++x * 2:
  x becomes 6
  6 * 2 = 12
  x = 6
y++ * 2:
  uses y=5
  5 * 2 = 10
  then y becomes 6
```

Key:

- Prefix: increment first
- Postfix: use then increment

Program 6: NOT Operator Precedence

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

Output:

```
a=5, b=0
!a + b:
  !a = 0
  0 + 0 = 0
!(a + b):
  a+b = 5
  !5 = 0
!a && b = 0
```

Precedence:

- ! higher than +
- () highest

Program 7: Relational Precedence

```c
#include <stdio.h>
int main() {
  int a=5, b=10, c=15;
  int result;
  result = a + b < c;
  printf("a=%d, b=%d, c=%d\n",
         a, b, c);
  printf("\na + b < c:\n");
  printf("  a+b = %d\n", a+b);
  printf("  15 < 15 = %d\n\n",
         result);
  result = a < b + c;
  printf("a < b + c:\n");
  printf("  b+c = %d\n", b+c);
  printf("  5 < 25 = %d\n",
         result);
  return 0;
}
```

Output:

```
a=5, b=10, c=15
a + b < c:
  a+b = 15
  15 < 15 = 0
a < b + c:
  b+c = 25
  5 < 25 = 1
```

Rule:

- Arithmetic before relational
- +, - before ¡, ¿, etc.

Program 8: Multiple Relational Operators

```c
#include <stdio.h>
int main() {
  int a=5, b=10, c=5;
  int result;
  result = a < b == c < b;
  printf("a=%d, b=%d, c=%d\n",
         a, b, c);
  printf("\na<b == c<b:\n");
  printf("Step 1 (< < same):\n");
  printf("  a<b = %d\n", a<b);
  printf("  c<b = %d\n", c<b);
  printf("  => 1 == 1\n");
  printf("Step 2 (==):\n");
  printf("  1 == 1 = %d\n\n",
         result);
  printf("Result: %d\n", result);
  return 0;
}
```

Output:

```
a=5, b=10, c=5

a<b == c<b:
Step 1 (< < same):
  a<b = 1
  c<b = 1
  => 1 == 1
Step 2 (==):
  1 == 1 = 1

Result: 1
```

Note:

- < before ==

Program 9: Logical AND/OR Precedence

```c
#include <stdio.h>
int main() {
  int a=1, b=0, c=1;
  int result;
  result = a || b && c;
  printf("a=%d, b=%d, c=%d\n",
         a, b, c);
  printf("\na || b && c:\n");
  printf("Step 1 (&& first):\n");
  printf("  b && c = %d\n",b&&c);
  printf("  => a || 0\n");
  printf("Step 2 (||):\n");
  printf("  1 || 0 = %d\n\n",
         result);
  result = (a || b) && c;
  printf("(a||b) && c = %d\n",
         result);
  return 0;
}
```

Output:

```
a=1, b=0, c=1
a || b && c:
Step 1 (&& first):
  b && c = 0
  => a || 0
Step 2 (||):
  1 || 0 = 1
(a||b) && c = 1
```

Rule:

- && before ——
- () changes order

Program 10: Complex Logical Expression

```c
#include <stdio.h>
int main() {
  int a=5, b=10, c=15;
  int result;
  result = a<b && b<c || a==c;
  printf("a=%d, b=%d, c=%d\n",
         a,b,c);
  printf("\na<b && b<c || a==c\n");
  printf("Step 1 (relational):\n");
  printf("  a<b = %d\n", a<b);
  printf("  b<c = %d\n", b<c);
  printf("  a==c = %d\n", a==c);
  printf("  => 1&&1 || 0\n");
  printf("Step 2 (&&):\n");
  printf("  1&&1 = 1\n");
  printf("  => 1 || 0\n");
  printf("Step 3 (||): %d\n",
         result);
  return 0;
}
```

Output:

```
a=5, b=10, c=15
a<b && b<c || a==c
Step 1 (relational):
  a<b = 1
  b<c = 1
  a==c = 0
  => 1&&1 || 0
Step 2 (&&):
  1&&1 = 1
  => 1 || 0
Step 3 (||): 1
```

Order:

- Relational, &&, ——

Program 11: Assignment Associativity

```c
#include <stdio.h>
int main() {
  int a, b, c;
  a = b = c = 10;
  printf("a = b = c = 10\n");
  printf("Right to left:\n");
  printf("  c = 10\n");
  printf("  b = c (b=10)\n");
  printf("  a = b (a=10)\n");
  printf("Values: a=%d,b=%d,c=%d\n\n",
         a, b, c);
  a = (b = 5) + (c = 3);
  printf("a=(b=5)+(c=3)\n");
  printf("  b=5, c=3\n");
  printf("  a = 5+3 = 8\n");
  printf("Values: a=%d,b=%d,c=%d\n",
         a, b, c);
  return 0;
}
```

Output:

```
a = b = c = 10
Right to left:
  c = 10
  b = c (b=10)
  a = b (a=10)
Values: a=10,b=10,c=10
a=(b=5)+(c=3)
  b=5, c=3
  a = 5+3 = 8
Values: a=8,b=5,c=3
```

Associativity:

- Assignment: right-to-left

Program 12: Compound Assignment Precedence

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

Output:

```
Initial: x = 10
x += 5 * 2:
  5*2 = 10 first
  x = x+10
  x = 20
x *= 3 + 2:
  3+2 = 5 first
  x = x*5
  x = 50
```

Note:

- Right side evaluated first
- Then assignment

Program 13: Arithmetic + Relational + Logical

```c
#include <stdio.h>
int main() {
  int a=5, b=10, c=15;
  int result;
  result = a+b>c && c-a<b;
  printf("a=%d, b=%d, c=%d\n",
         a,b,c);
  printf("\na+b>c && c-a<b\n");
  printf("Step 1 (arithmetic):\n");
  printf("  a+b = %d\n", a+b);
  printf("  c-a = %d\n", c-a);
  printf("  => 15>15 && 10<10\n");
  printf("Step 2 (relational):\n");
  printf("  15>15 = 0\n");
  printf("  10<10 = 0\n");
  printf("  => 0 && 0\n");
  printf("Step 3 (&&): %d\n",
         result);
  return 0;
}
```

Output:

```
a=5, b=10, c=15
a+b>c && c-a<b
Step 1 (arithmetic):
  a+b = 15
  c-a = 10
  => 15>15 && 10<10
Step 2 (relational):
  15>15 = 0
  10<10 = 0
  => 0 && 0
Step 3 (&&): 0
```

Order:

- Arithmetic $\rightarrow$ Relational $\rightarrow$ Logical

Program 14: All Operator Types

```c
#include <stdio.h>
int main() {
  int a=2, b=3, c=4;
  int result;
  result = ++a*b+c--<10&&b>a;
  printf("Initial: a=2,b=3,c=4\n");
  printf("\n++a*b+c--<10&&b>a\n");
  printf("Step 1 (++ prefix):\n");
  printf("  a becomes 3\n");
  printf("Step 2 (arithmetic):\n");
  printf("  3*3 = 9\n");
  printf("  9+4 = 13\n");
  printf("Step 3 (postfix--):\n");
  printf("  use 4, then c=3\n");
  printf("Step 4 (relational):\n");
  printf("  13<10=0, 3>3=0\n");
  printf("Step 5 (&&): %d\n",
         result);
  printf("Final: a=%d,c=%d\n",
         a,c);
  return 0;
}
```

Output:

```
Initial: a=2,b=3,c=4
++a*b+c--<10&&b>a
Step 1 (++ prefix):
  a becomes 3
Step 2 (arithmetic):
  3*3 = 9
  9+4 = 13
Step 3 (postfix--):
  use 4, then c=3
Step 4 (relational):
  13<10=0, 3>3=0
Step 5 (&&): 0
Final: a=3,c=3
```

Program 15: Ternary Operator Precedence

```c
#include <stdio.h>
int main() {
  int a=5, b=10, c=15;
  int result;
  result = a<b ? b+c : b-c;
  printf("a=%d, b=%d, c=%d\n",
         a,b,c);
  printf("\na<b ? b+c : b-c\n");
  printf("Step 1 (relational):\n");
  printf("  a<b = 1 (true)\n");
  printf("Step 2 (ternary):\n");
  printf("  true, so b+c\n");
  printf("  10+15 = %d\n\n",
         result);
  result = a+b<c ? 1 : 0;
  printf("a+b<c ? 1 : 0\n");
  printf("  a+b=15, 15<15=0\n");
  printf("  false, so 0\n");
  printf("  Result: %d\n",result);
  return 0;
}
```

Output:

```
a=5, b=10, c=15
a<b ? b+c : b-c
Step 1 (relational):
  a<b = 1 (true)
Step 2 (ternary):
  true, so b+c
  10+15 = 25
a+b<c ? 1 : 0
  a+b=15, 15<15=0
  false, so 0
  Result: 0
```

Note:

- Ternary low precedence

Program 16: Bitwise Precedence

```c
#include <stdio.h>
int main() {
  int a=5, b=3, c=2;
  int result;
  result = a & b | c;
  printf("a=%d, b=%d, c=%d\n",
         a,b,c);
  printf("\na & b | c\n");
  printf("Step 1 (& first):\n");
  printf("  a&b = 5&3 = %d\n",
         a&b);
  printf("  => 1 | 2\n");
  printf("Step 2 (|):\n");
  printf("  1|2 = %d\n\n",result);
  result = a | b & c;
  printf("a | b & c\n");
  printf("  b&c = %d\n", b&c);
  printf("  a|2 = %d\n", result);
  return 0;
}
```

Output:

```
a=5, b=3, c=2
a & b | c
Step 1 (& first):
  a&b = 5&3 = 1
  => 1 | 2
Step 2 (|):
  1|2 = 3
a | b & c
  b&c = 2
  a|2 = 7
```

Order:

- & before ^ before —

Program 17: Shift Operators

```c
#include <stdio.h>
int main() {
  int a=8, b=2;
  int result;
  result = a << 1 + b;
  printf("a=%d, b=%d\n", a, b);
  printf("\na << 1 + b\n");
  printf("Step 1 (+ first):\n");
  printf("  1+b = %d\n", 1+b);
  printf("  => a << 3\n");
  printf("Step 2 (<<):\n");
  printf("  8<<3 = %d\n\n",result);
  result = (a << 1) + b;
  printf("(a<<1) + b\n");
  printf("  a<<1 = %d\n", a<<1);
  printf("  16+2 = %d\n", result);
  return 0;
}
```

Output:

```
a=8, b=2
a << 1 + b
Step 1 (+ first):
  1+b = 3
  => a << 3
Step 2 (<<):
  8<<3 = 64
(a<<1) + b
  a<<1 = 16
  16+2 = 18
```

Note:

- Arithmetic before shift

Program 18: Power of Parentheses

```c
#include <stdio.h>
int main() {
  int a=2, b=3, c=4;
  int r1, r2, r3, r4;
  r1 = a + b * c;
  r2 = (a + b) * c;
  r3 = a * b + c;
  r4 = a * (b + c);
  printf("a=%d, b=%d, c=%d\n\n",
         a,b,c);
  printf("a+b*c = %d\n", r1);
  printf("(a+b)*c = %d\n", r2);
  printf("a*b+c = %d\n", r3);
  printf("a*(b+c) = %d\n", r4);
  return 0;
}
```

Output:

```
a=2, b=3, c=4
a+b*c = 14
(a+b)*c = 20
a*b+c = 10
a*(b+c) = 14
```

Explanation:

- 2+3*4 = 2+12 = 14
- (2+3)*4 = 5*4 = 20
- 2*3+4 = 6+4 = 10
- 2*(3+4) = 2*7 = 14

Program 19: Nested Parentheses

```c
#include <stdio.h>
int main() {
  int a=2, b=3, c=4, d=5;
  int result;
  result = ((a+b)*(c+d))/2;
  printf("a=%d,b=%d,c=%d,d=%d\n",
         a,b,c,d);
  printf("\n((a+b)*(c+d))/2\n");
  printf("Step 1 (innermost):\n");
  printf("  a+b = %d\n", a+b);
  printf("  c+d = %d\n", c+d);
  printf("Step 2 (multiply):\n");
  printf("  5*9 = %d\n", 5*9);
  printf("Step 3 (divide):\n");
  printf("  45/2 = %d\n\n",result);
  printf("Result: %d\n", result);
  return 0;
}
```

Output:

```
a=2,b=3,c=4,d=5
((a+b)*(c+d))/2
Step 1 (innermost):
  a+b = 5
  c+d = 9
Step 2 (multiply):
  5*9 = 45
Step 3 (divide):
  45/2 = 22
Result: 22
```

Rule:

- Innermost () first

Program 20: Common Mistake - Assignment in Condition

```c
#include <stdio.h>
int main() {
  int x = 5;
  printf("x = %d\n\n", x);
  printf("Wrong: if (x = 0)\n");
  if (x = 0) {
    printf("  This won't print\n");
  } else {
    printf("  x assigned 0\n");
  }
  printf("  x is now: %d\n\n", x);
  x = 5;
  printf("Correct: if (x == 0)\n");
  if (x == 0) {
    printf("  Won't print\n");
  } else {
    printf("  x still: %d\n", x);
  }
  return 0;
}
```

Output:

```
x = 5
Wrong: if (x = 0)
  x assigned 0
  x is now: 0
Correct: if (x == 0)
  x still: 5
```

Warning:

- = assigns (returns 0)
- == compares
- Big difference!

Program 21: Mistake - Logical vs Bitwise

```c
#include <stdio.h>
int main() {
  int a = 5, b = 3;
  printf("a=%d, b=%d\n\n", a, b);
  printf("Bitwise & (wrong):\n");
  if (a>0 & b>0) {
    printf("  Works but wrong!\n");
  }
  printf("  a>0 & b>0 = %d\n\n",
         a>0 & b>0);
  printf("Logical && (correct):\n");
  if (a>0 && b>0) {
    printf("  Both positive\n");
  }
  printf("  a>0 && b>0 = %d\n",
         a>0 && b>0);
  return 0;
}
```

Output:

```
a=5, b=3
Bitwise & (wrong):
  Works but wrong!
  a>0 & b>0 = 1
Logical && (correct):
  Both positive
  a>0 && b>0 = 1
```

Note:

- & is bitwise
- && is logical
- Always use && for conditions

Quick Reference - Precedence Order

High to Low:

1. `()` `[]` `->` `.`
2. `!` `~` `++` `--` (unary)
3. `*` `/` `%`
4. `+` `-` (binary)
5. `<<` `>>`
6. `<` `<=` `>` `>=`
7. `==` `!=`
8. `&` (bitwise AND)
9. `^` (bitwise XOR)
10. `|` (bitwise OR)
11. `&&` (logical AND)
12. `||` (logical OR)
13. `?:` (ternary)
14. `=` `+=` `-=` etc.
15. `,` (comma)

Associativity Rules

Left to Right:

- Most operators: `+ - * / % < > == != && ||`
- Evaluated from left side first

Right to Left:

- Unary operators: `! ~ ++ -- + - * &`
- Assignment: `= += -= *= /=`
- Ternary: `?:`
- Evaluated from right side first

Best Practices

1. **Use parentheses** for clarity
2. **Don't rely on precedence** for complex expressions
3. **One operation per line** for readability
4. **Use == not =** in conditions
5. **Use && not &** for logical operations
6. **Avoid side effects** in complex expressions
7. **Comment** non-obvious precedence
8. **Break down** complex expressions

Common Mistakes to Avoid

1. Using $=$ instead of $==$ in conditions
2. Mixing & and && (bitwise vs logical)
3. Forgetting integer division truncates
4. Not using parentheses in complex expressions
5. Assuming left-to-right for all operators
6. Using multiple $++$ $--$ in same expression
7. Relying on precedence instead of clarity

Try These!

1. Evaluate: `5 + 3 * 2 - 4 / 2`
2. Evaluate: `10 > 5 && 3 < 7 || 2 == 2`
3. What's wrong: `if (x = 10)`
4. Trace: `int x=5; y = ++x * 2 + x++;`
5. Rewrite with (): `a + b * c / d - e`
6. Evaluate: `2 << 3 + 1`

End of Deck 5

Questions? Next: Deck 6 - Conditional Statements