Operators & Expressions in C#

Week 3: Building the foundation for calculations and decisions

Today’s Agenda

- What are operators and expressions?
- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Operator precedence
- Real-world applications
- Hands-on coding

What Are Operators?

- Symbols that tell the compiler to perform specific operations
- Act on operands (variables, values, or expressions)
- Transform or combine values to produce new values
- Essential tools for decision-making and calculations

```
int sum = 5 + 3;         // Addition operator
bool isValid = age > = 18; // Greater than or equal operator
double average = (a + b + c) / 3; // Multiple operators
```

Arithmetic Operators

- Addition (`+`): `5 + 3` → `8`
- Subtraction (`-`): `5 - 3` → `2`
- Multiplication (`*`): `5 * 3` → `15`
- Division (`/`): `10 / 2` → `5`
- Modulus (`%`): `10 % 3` → `1`
- Increment (`++`): `i++` or `++i`
- Decrement (`--`): `i--` or `--i`

```
int sum = 10 + 5;         // 15
int difference = 10 - 5;  // 5
int product = 10 * 5;     // 50
int quotient = 10 / 5;    // 2
int remainder = 10 % 3;   // 1

int a = 5; a++;           // a becomes 6
int b = 5; ++b;           // b becomes 6
int c = 5; int d = c++;   // d = 5, c = 6
int e = 5; int f = ++e;   // f = 6, e = 6
```

Division Deep Dive

- Integer division truncates (removes decimal part)
- For floating-point division, at least one operand must be float/double
- Be careful with division by zero!

```
int result1 = 10 / 3;
// result1 = 3 (truncated)

double result2 = 10.0 / 3;
// result2 = 3.33333...

int x = 5 / 0;
// Runtime error: Division by zero!
```

Assignment Operators

- Basic assignment (`=`)
- Compound assignment:
  - `+=` , `-=` , `*=` , `/ =` , `%=`
- Shorthand for operations
- Improves code readability

```
int x = 10;  // Assign 10 to x

x += 5;      // x = x + 5
// x is now 15

x -= 3;      // x = x - 3
// x is now 12

x *= 2;      // x = x * 2
// x is now 24

x / = 4;      // x = x / 4
// x is now 6
```

Comparison Operators

- Equal to (`==`)
- Not equal to (`!=`)
- Greater than (`>`)
- Less than (`<`)
- Greater than or equal to (`>=`)
- Less than or equal to (`<=`)
- Always return a boolean result

```
int a = 5, b = 10;

bool result1 = (a == b);  // false
bool result2 = (a != b);  // true
bool result3 = (a > b);   // false
bool result4 = (a < b);   // true
bool result5 = (a >= 5);  // true
bool result6 = (b <= 5);  // false
```

! Be careful not to confuse `=` (assignment) with `==` (comparison)

Logical Operators

- AND (`&&`): Both conditions must be true
- OR (`||`): At least one condition must be true
- NOT (`!`): Inverts the boolean value
- Used to combine multiple conditions
- Short-circuit evaluation

```
bool isAdult = (age ⩾ 18);
bool hasMembership = true;

// AND example
bool canEnter = isAdult && hasMembership;
// true only if both are true
// canEnter = true if age ⩾ 18 AND has membership

// OR example
bool hasAccess = isAdmin || hasPermission;
// true if either one is true
// hasAccess = true if isAdmin OR has permission

// NOT example
bool isInvalid = !isValid;
// inverts the boolean value
// isInvalid = true if isValid is false
```

Short-Circuit Evaluation

- For `&&` : If first operand is `false` , second is never evaluated
- For `||` : If first operand is `true` , second is never evaluated
- Improves performance
- Can be used strategically in code

```
// Second condition only evaluates if user ≠ null
if (user ! = null && user.IsActive) {
    // Process active user
}

// Avoid null reference exception
int? length = text?.Length; // Modern C# null conditional
// Or traditional approach:
if (text ! = null && text.Length > 10) {
    // Process text
}
```

Operator Precedence

- Determines the order of operations
- Similar to mathematical rules (PEMDAS)
- Use parentheses to control evaluation order
- Improves code clarity

```
int result1 = 5 + 3 * 2;      // result1 = 11 (not 16)
// Multiplication happens before addition

int result2 = (5 + 3) * 2;    // result2 = 16
// Parentheses force addition to happen first

bool result3 = x > 5 && y < 10 || z = = 15;
// Evaluation order: (x > 5 && y < 10) || z = = 15
// Comparison operators → Logical AND → Logical OR
```

Precedence Chart (Simplified)

1. Parentheses ()
2. Increment/decrement (`++` , `--`)
3. Arithmetic operators (`*` , `/` , `%` then `+` , `-`)
4. Comparison operators (`<` , `>` , `<=` , `>=` , `==` , `!=`)
5. Logical operators (`&&` then `||`)
6. Assignment operators (`=` , `+=` , `-=` , etc.)

When in doubt, use parentheses to make your intent clear!

AI Code Generation Demo

- Let's see how AI can help generate expressions and calculations
- We'll ask it to create a BMI calculator
- We'll analyze the generated code together
- Understand why it works (or doesn't)

```
// Generated code example for BMI calculator
double weight = 70; // in kilograms
double height = 1.75; // in meters
double bmi = weight / (height * height);

Console.WriteLine($"Weight: {weight}kg, Height: {height}m");
Console.WriteLine($"BMI: {bmi:F2}");
Console.WriteLine($"Category: {(bmi < 18.5 ? "Underweight" :
                                bmi < 25 ? "Normal" :
                                bmi < 30 ? "Overweight" : "Obese")}");
```

Real-World Example: Discount Calculator

- Calculate final price after applying discounts
- Apply quantity discounts and coupon codes
- Use comparison and logical operators for decision making

```
// Calculate discount based on quantity and coupon
double price = 29.99;
int quantity = 3;
bool hasCoupon = true;

double subtotal = price * quantity;
double discount = 0;

// Quantity discount
if (quantity > = 5) {
    discount += subtotal * 0.1; // 10% off for 5+ items
}
else if (quantity > = 3) {
    discount += subtotal * 0.05; // 5% off for 3-4 items
}

// Additional coupon discount
if (hasCoupon) {
    discount += subtotal * 0.15; // 15% off with coupon
```

Guided Coding Session

Let's write code for a grade calculator that:

1. Takes test scores as input (0-100)
2. Calculates the average
3. Assigns a letter grade based on the average:
   - A: 90-100
   - B: 80-89
   - C: 70-79
   - D: 60-69
   - F: Below 60
4. Displays if the student passed or failed (60+ to pass)

Student Coding Task

Create a simple temperature converter that:

1. Takes a temperature value and its unit (C or F) as input
2. Converts it to the other unit using the appropriate formula:
   - C to F: (C × 9/5) + 32
   - F to C: (F - 32) × 5/9
3. Displays the result
4. Indicates if water would freeze or boil at that temperature

Common Mistakes

- Using `=` instead of `==` in conditions
- Integer division truncation
- Incorrect operator precedence
- Missing parentheses
- Logical errors in complex boolean expressions
- Off-by-one errors when using increment/decrement

Key Takeaways

- Operators are the building blocks for calculations and decisions
- Pay attention to operator precedence
- Use parentheses when in doubt
- Integer division behaves differently from floating-point division
- Logical operators allow for complex decision-making
- Always consider edge cases in your calculations

Next Week: Control Structures

- We'll explore if-else statements
- Learn about switch-case
- Make our programs truly interactive
- Implement decision-making logic

Questions?

Thank you for your attention!

Let’s continue with a hands-on exercise.