Control Structures in C#

If-Else and Switch Statements

Press Space for next page →

Today’s Agenda

- Why do we need control structures?
- If-Else statements: syntax and applications
- Switch-Case statements: when and how to use them
- Nested conditions: benefits and challenges
- Best practices for writing clean conditional code
- Hands-on coding practice

Why Do We Need Control Structures?

- Programs need to make decisions based on conditions
- Enable our code to have branching logic
- Allow for different behaviors in different scenarios
- Help create dynamic and responsive applications

If-Else Statements: Basic Syntax

```
if (condition)
{
    // Code to execute when condition is true
}
else
{
    // Code to execute when condition is false
}
```

With multiple conditions:

```
if (condition1)
{
    // Code to execute when condition1 is true
}
else if (condition2)
{
    // Code to execute when condition1 is false but condition2 is true
}
else
{
    // Code to execute when both conditions are false
}
```

If-Else Example: Weather App

```csharp
int temperature = 28;

if (temperature > 30)
{
    Console.WriteLine("It's hot outside! Remember to stay hydrated.");
}
else if (temperature > 20)
{
    Console.WriteLine("It's a pleasant day. Enjoy the weather!");
}
else if (temperature > 10)
{
    Console.WriteLine("It's a bit cool. Consider bringing a light jacket.");
}
else
{
    Console.WriteLine("It's cold outside! Bundle up!");
}
```

Output:

```
It's a pleasant day. Enjoy the weather!
```

Conditional Operators

- **Comparison Operators**:
  - `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logical Operators**:
  - `&&` (AND), `||` (OR), `!` (NOT)
- **Ternary Operator**: A shorthand for if-else
  - `condition ? valueIfTrue : valueIfFalse`

```
bool isRaining = true;
bool hasUmbrella = false;

if (isRaining && !hasUmbrella)
{
    Console.WriteLine("You'll get wet!");
}

// Same logic using ternary operator
```

Switch Statements: Syntax

```
switch (expression)
{
    case value1:
        // Code to execute if expression equals value1
        break;
    case value2:
        // Code to execute if expression equals value2
        break;
    // More cases as needed...
    default:
        // Code to execute if none of the cases match
        break;
}
```

When to use Switch instead of If-Else:

- When comparing a single variable against multiple known values
- When you have more than 3-4 conditions on the same variable
- For better readability in certain scenarios

Switch Example: Day of Week

```csharp
int dayOfWeek = 3; // 1 = Monday, 2 = Tuesday, etc.

switch (dayOfWeek)
{
    case 1:
        Console.WriteLine("It's Monday. Start of the work week!");
        break;
    case 2:
        Console.WriteLine("It's Tuesday.");
        break;
    case 3:
        Console.WriteLine("It's Wednesday. Halfway through!");
        break;
    case 4:
        Console.WriteLine("It's Thursday.");
        break;
    case 5:
        Console.WriteLine("It's Friday. Weekend is coming!");
        break;
    case 6:
    case 7:
        Console.WriteLine("It's the weekend!");
        break;
    default:
```

C# 8.0+ Switch Expressions

A more modern and concise way to write switch statements:

```csharp
int dayOfWeek = 3;

string message = dayOfWeek switch
{
    1 = > "It's Monday. Start of the work week!",
    2 = > "It's Tuesday.",
    3 = > "It's Wednesday. Halfway through!",
    4 = > "It's Thursday.",
    5 = > "It's Friday. Weekend is coming!",
    6 or 7 = > "It's the weekend!",
    _ = > "Invalid day number!"
};

Console.WriteLine(message);
```

- More concise and expression-based
- Uses `= >` for case mapping and `_` as the default case
- Can use pattern matching for more complex conditions

Nested Conditions

```csharp
bool isWeekend = true;
bool isRaining = true;

if (isWeekend)
{
    if (isRaining)
    {
        Console.WriteLine("It's the weekend but it's raining. " +
                          "Maybe stay in and watch a movie?");
    }
    else
    {
        Console.WriteLine("It's the weekend and the weather is nice! " +
                          "Perhaps go for a hike?");
    }
}
else
{
    if (isRaining)
    {
        Console.WriteLine("It's a workday and it's raining. " +
                          "Don't forget your umbrella!");
    }
    else
```

Nested Conditions: Challenges

- Can become difficult to read and maintain
- Increases code complexity
- May lead to the "arrow anti-pattern" or "pyramid of doom"
- Makes debugging more challenging

```
// Deep nesting - hard to read!
if (condition1)
{
    if (condition2)
    {
        if (condition3)
        {
            if (condition4)
            {
                // Deeply nested code
            }
        }
    }
}
```

Simplifying Nested Conditions

- Use compound conditions with logical operators
- Early returns (guard clauses)
- Extract complex conditions into well-named boolean variables
- Consider breaking up into separate methods

```
// Instead of deep nesting, use compound conditions
if (condition1 && condition2 && condition3 && condition4)
{
    // Code that would have been deeply nested
}

// Or use early returns (guard clauses)
if (!condition1) return;
if (!condition2) return;
// Now we know both conditions are true

// Extract complex conditions into named variables
bool isEligibleForDiscount = age > 65 || isStudent || isMilitary;
bool hasCompletedProfile = !string.IsNullOrEmpty(name) && emailVerified;
if (isEligibleForDiscount && hasCompletedProfile) { ... }

```

AI Exploration

Let’s see how AI can help us generate conditional logic:

```
Prompt: "Create a C# function that determines what grade (A, B, C, D, or F)
a student gets based on their score (0-100). Use both if-else and
switch versions."
```

We'll analyze the AI-generated code together and discuss:

- Is the code correct and complete?
- Which approach (if-else vs switch) is more appropriate here?
- How could we improve the code?

Guided Coding Session

Let's create a simple calculator program together:

```csharp
using System;

class SimpleCalculator
{
    static void Main()
    {
        Console.WriteLine("Simple Calculator");
        Console.WriteLine("────────────────");

        // Get first number
        Console.Write("Enter first number: ");
        double num1 = Convert.ToDouble(Console.ReadLine());

        // Get operation
        Console.Write("Enter operation (+, -, *, /): ");
        char operation = Console.ReadLine()[0];

        // Get second number
        Console.Write("Enter second number: ");
        double num2 = Convert.ToDouble(Console.ReadLine());

        // Calculate and display result
        CalculateAndDisplay(num1, operation, num2);
```

Student Coding Tasks

1. **Complete the Calculator**: Finish the `CalculateAndDisplay` method from our guided session.
2. **Create a Mini Quiz**: Write a program that asks multiple-choice questions and provides feedback based on answers.
3. **Challenge**: Design a "Choose Your Own Adventure" game with at least 5 decision points using if-else or switch statements.

Hint: Start simple and test each part of your code before moving on. Use clear variable names that indicate what the variable represents.

Debugging & Discussion

Common issues with conditional statements:

- Using `=` (assignment) instead of `= =` (comparison)
- Forgetting to include `break` statements in switch cases
- Overcomplicating conditions
- Off-by-one errors in boundary conditions
- Unintended fall-through in switch statements

Pro Tip: When debugging conditional logic, add temporary Console.WriteLine statements to check the values of variables and which branches are being executed.

Best Practices for Conditional Logic

- Keep conditions simple and readable
- Use meaningful variable names for boolean flags
- Consider extracting complex conditions to functions
- Be consistent with your brace style
- Watch out for accidental assignments in conditions
- Think about all possible scenarios (including edge cases)
- Consider the default case (what if no conditions match?)

Up Next: Week 5

- We'll explore **Loops**: for, while, and do-while
- Learn how to automate repetitive tasks
- Understand loop control mechanisms like break and continue
- Develop skills to avoid common loop pitfalls

Thank You!

Any Questions?

Don't forget to complete this week's coding tasks!