C# Syntax Essentials

C# (pronounced "C-sharp") is a modern, object-oriented programming language developed by Microsoft. It's widely used for building applications on the .NET framework, including desktop, web, and mobile apps. Understanding its syntax essentials is key to writing effective C# code. This material focuses on the foundational elements of C# syntax.

---

1. Basic Program Structure

Anatomy of a C# Program

Every C# program requires a specific structure:

- **Namespace**: A way to organize code and avoid naming conflicts.
- **Class**: A container for data and methods.
- **Main Method**: The entry point of the program.

Example

```
using System;  // Importing namespace for basic functionality

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)  // Entry point
        {
            Console.WriteLine("Hello, World!");  // Output to console
        }
    }
}
```

Key Points

- `using System;` includes the `System` namespace for common classes like `Console`.
- `Main` is static and must be defined in a class; it's where execution begins.
- Semicolons (`;`) terminate statements.

---

2. Variables and Data Types

Variables

Variables store data and must be declared with a type before use.

Common Data Types

Type	Description	Example
`int`	Integer (whole number)	`int age = 25;`
`double`	Floating-point number	`double pi = 3.14;`
`string`	Text	`string name = "Alice";`
`bool`	Boolean (true/false)	`bool isActive = true;`
`char`	Single character	`char grade = 'A';`

Example

```
class Program
{
    static void Main()
    {
        int age = 30;
        double height = 5.9;
        string message = "Hello, C#!";
        bool isStudent = false;

        Console.WriteLine($"Age: {age}, Height: {height}, Message: {message},
Student: {isStudent}");
        // Output: Age: 30, Height: 5.9, Message: Hello, C#!, Student: False
    }
}
```

Key Points

- Variables are declared with `type name = value;`.
- C# supports **type inference** with `var` (e.g., `var x = 10;` infers `int`).
- String interpolation (`$"..."`) simplifies output formatting.

---

3. Operators

Types of Operators

1. **Arithmetic**: `+`, `-`, `*`, `/`, `%` (modulus)
2. **Comparison**: `==`, `!=`, `<`, `>`, `<=`, `>=`
3. **Logical**: `&&` (and), `||` (or), `!` (not)
4. **Assignment**: `=`, `+=`, `-=`, etc.

Example

```
class Program
{
    static void Main()
```

```
    {
        int a = 10;
        int b = 3;

        Console.WriteLine($"Sum: {a + b}");      // Output: Sum: 13
        Console.WriteLine($"Mod: {a % b}");      // Output: Mod: 1
        Console.WriteLine($"Equal: {a == b}");   // Output: Equal: False
        Console.WriteLine($"And: {a > 0 && b > 0}");  // Output: And: True
    }
}
```

Key Points

- Arithmetic operators work as expected; `/` with integers truncates to an integer.
- Logical operators combine conditions for control flow.

---

4. Control Structures

Conditional Statements

- `if-else`: Executes code based on a condition.
- `switch`: Selects a block of code based on a value.

Example (if-else)

```
class Program
{
    static void Main()
    {
        int number = 7;

        if (number > 0)
        {
            Console.WriteLine("Positive");
        }
        else if (number == 0)
        {
            Console.WriteLine("Zero");
        }
        else
        {
            Console.WriteLine("Negative");
        }
        // Output: Positive
    }
}
```

Example (switch)

```
class Program
{
    static void Main()
    {
        int day = 3;
        string dayName;

        switch (day)
        {
            case 1:
                dayName = "Monday";
                break;
            case 2:
                dayName = "Tuesday";
                break;
            case 3:
                dayName = "Wednesday";
                break;
            default:
                dayName = "Unknown";
                break;
        }
        Console.WriteLine(dayName);  // Output: Wednesday
    }
}
```

Loops

- `for`: Iterates a fixed number of times.
- `while`: Repeats while a condition is true.
- `foreach`: Iterates over collections (e.g., arrays).

Example (Loops)

```
class Program
{
    static void Main()
    {
        // For loop
        for (int i = 1; i <= 3; i++)
        {
            Console.WriteLine($"For: {i}");
        }
        // Output: For: 1, For: 2, For: 3

        // While loop
        int j = 0;
        while (j < 3)
        {
```

```
            Console.WriteLine($"While: {j}");
            j++;
        }
        // Output: While: 0, While: 1, While: 2

        // Foreach with array
        string[] fruits = { "Apple", "Banana", "Orange" };
        foreach (string fruit in fruits)
        {
            Console.WriteLine(fruit);
        }
        // Output: Apple, Banana, Orange
    }
}
```

Key Points

- Braces `{}` define code blocks; single-line statements can omit them but it's not recommended.
- `break` exits a loop or switch; `continue` skips to the next iteration.

---

5. Methods

What are Methods?

Methods are reusable blocks of code that perform specific tasks. They can take parameters and return values.

Syntax

- **Return Type**: Specifies what the method returns (e.g., `int`, `void` for no return).
- **Parameters**: Inputs to the method (optional).

Example

```
class Program
{
    // Method with return value
    static int Add(int a, int b)
    {
        return a + b;
    }

    // Method with no return value
    static void Greet(string name)
    {
        Console.WriteLine($"Hello, {name}!");
    }

    static void Main()
    {
        int sum = Add(5, 3);
```

```
        Console.WriteLine($"Sum: {sum}");  // Output: Sum: 8

        Greet("Alice");  // Output: Hello, Alice!
    }
}
```

Key Points

- Methods are declared with `returnType MethodName(parameters)`.
- `static` methods belong to the class, not an instance (required in `Main` context).
- Parameters can have default values (e.g., `int x = 0`).

---

6. Basic Input/Output

Console I/O

- **`Console.WriteLine()`**: Outputs text with a newline.
- **`Console.Write()`**: Outputs text without a newline.
- **`Console.ReadLine()`**: Reads user input as a string.

Example

```
class Program
{
    static void Main()
    {
        Console.Write("Enter your name: ");
        string name = Console.ReadLine();

        Console.WriteLine($"Welcome, {name}!");
        // Example interaction:
        // Enter your name: Bob
        // Welcome, Bob!
    }
}
```

Key Points

- Convert `ReadLine()` input to other types using methods like `int.Parse()` or `Convert.ToInt32()`.
- Use `Console.Clear()` to clear the console screen if needed.

Practical Application Example

Here's a simple program combining these concepts:

```
using System;

namespace Calculator
{
    class Program
    {
        static double Calculate(double a, double b, char operation)
        {
            switch (operation)
            {
                case '+': return a + b;
                case '-': return a - b;
                case '*': return a * b;
                case '/':
                    if (b != 0) return a / b;
                    else return double.NaN;  // Not a Number for division by zero
                default: return 0;
            }
        }

        static void Main()
        {
            Console.Write("Enter first number: ");
            double num1 = double.Parse(Console.ReadLine());

            Console.Write("Enter second number: ");
            double num2 = double.Parse(Console.ReadLine());

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

            double result = Calculate(num1, num2, op);
            Console.WriteLine($"Result: {result}");
        }
    }
}
```

Output Example

```
Enter first number: 10
Enter second number: 5
Enter operation (+, -, *, /): +
Result: 15
```

Summary Table

Concept	Description	Example

Concept	Description	Example
Program Structure	Namespace, class, Main method	`static void Main()`
Variables	Typed storage for data	`int x = 5;`
Operators	Arithmetic, comparison, logical	`x + y`, `x == y`
Control Structures	Conditionals and loops	`if`, `for`, `switch`
Methods	Reusable code blocks	`int Add(int a, int b)`
Input/Output	Console interaction	`Console.WriteLine()`

Exercises

1. Write a program that asks for a user's age and prints whether they are a minor (< 18), adult (18–65), or senior (> 65).
2. Create a method `IsEven(int number)` that returns `true` if a number is even, and use it in a loop to print all even numbers from 1 to 10.
3. Build a program that reads a list of names (until "stop" is entered) and prints them using `foreach`.