# Basic C# Syntax

## Introduction

This summary explores the fundamentals of writing simple C# programs, focusing on core concepts such as syntax, variables, control structures, and methods. These elements form the foundation of all C# applications and are essential for writing functional, reusable code.

### Program Structure

A C# program starts with a class definition and a Main method that serves as the program's entry point. For example:

class Program

```
{

  static void Main(string[] args)

  {
    Console.WriteLine("Hello, World!");
  }
}
```

The `Console.WriteLine()` method outputs text to the console, illustrating basic syntax.

### Variables and Data Types

Variables in C# hold data and are declared with specific types. Examples include:

```
int age = 25;
string name = "John";
var count = 10;
```

The `var` keyword allows for type inference, where the compiler determines the variable's type based on its assigned value.

### Control Structures

Control structures manage the flow of a program:

#### If-Else Statements

Execute code based on conditions:

```
if (age >= 18)
{
  Console.WriteLine("You're an adult.");
}
else
{
  Console.WriteLine("You're a minor.");
}
```

### Loops

Repeat actions, such as printing numbers:

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

### Methods

Methods allow for reusable blocks of code. For example, an addition method can take two inputs and return their sum:

```
public static int Add(int a, int b)
{
  return a + b;
}
```

Incorporating methods into classes organizes functionality and enhances code reusability.

## Practical Example: Simple Calculator

A basic calculator program might look like this:

public class Calculator

```
{
  static void Main(string[] args)
  {
    int num1 = 5;
    int num2 = 10;
    int result = Add(num1, num2);
    Console.WriteLine("The sum is: " + result);
```

```
  }
  public static int Add(int a, int b)
  {
    return a + b;
  }
}
```

This program sums two integers and prints the result to the console.

#### Handling User Input

C# can also handle user input via the Console.ReadLine() method. For example:

```
Console.WriteLine("Enter your name:");
string name = Console.ReadLine();
Console.WriteLine("Hello, " + name + "!");
```

This allows programs to interact with users dynamically.

# Conclusion

You can build simple but effective C# programs, such as calculators and interactive user applications by mastering these core concepts- syntax, variables, control structures, and methods. These basics lay the groundwork for more advanced programming in C#.