

OBJECT-ORIENTED PROGRAMMING

LAB1



C#.NET Basics

- C#.NET is one of the Microsoft Programming Languages to work with the .NET Framework to develop different kinds of applications.
- **Different Types of Applications are Developed using C#.NET.**
 1. Windows Applications
 2. Web Applications
 3. Restful Web Services
 4. SOAP Based Services
 5. Console Applications
 6. Class Library

C#.NET Basics

What is the Visual Studio?

Visual Studio is one of the Microsoft IDE tools, This tool provides some features such as

1. Editor
2. Compiler
3. Interpreters, and many more

C#.NET Basics

What is a Console Application?

1. A console application is an application that can be run in the command prompt. For any beginner on .NET or anyone who wants to learn C# Language or anyone who wants to become an expert in C# Language, building a console application is ideally the first step to learning the C# Language.
2. The Console Applications contain a similar user interface to the Operating systems like MS-DOS, UNIX, etc.
3. The Console Application is known as the CUI application because in this application we completely work with the CUI environment.
4. These applications are similar to C or C++ applications.
5. Console applications do not provide any GUI facilities like the Mouse Pointer, Colors, Buttons, Menu Bars, etc.

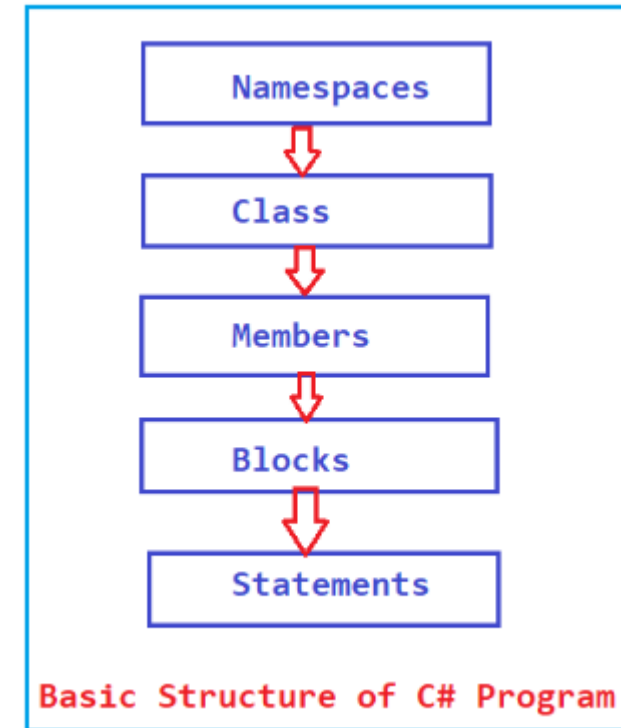
C#.NET Basics

Basic Structure of the C# Program

```
//Include classlibraries/namespaces
using System;

//Declare namespace
namespace namespaceName
{
    //Class Declaration
    class ClassName
    {
        //Data Members
        int SomeMember1;
        float SomeMember2;

        //Functions/Methods/Blocks
        static void SomeBlock()
        {
            //Statements
            Console.WriteLine("Hello World");
        }
    }
}
```



C#.NET Basics

Example to Understand the Basic Structure of a C# Program:

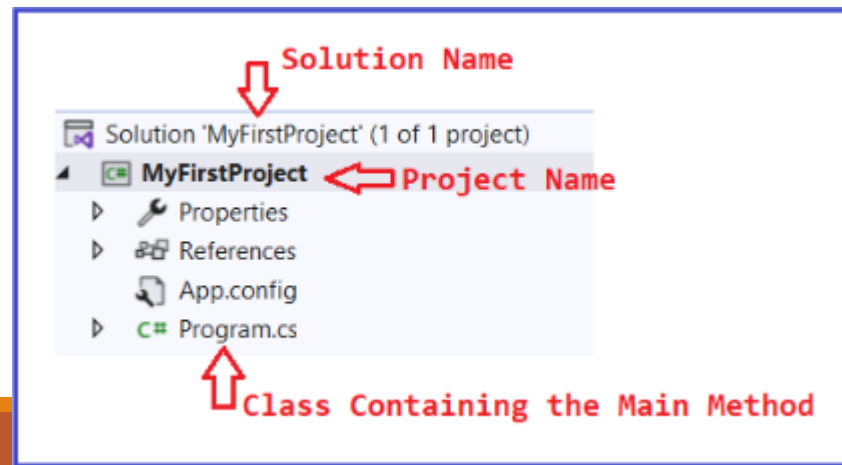
Step1: Create a New Project

Step2: choose the project type as a Console Application

Step3: MyFirstProject

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyFirstProject
{
    internal class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```



Principles of Object-Oriented Programming.

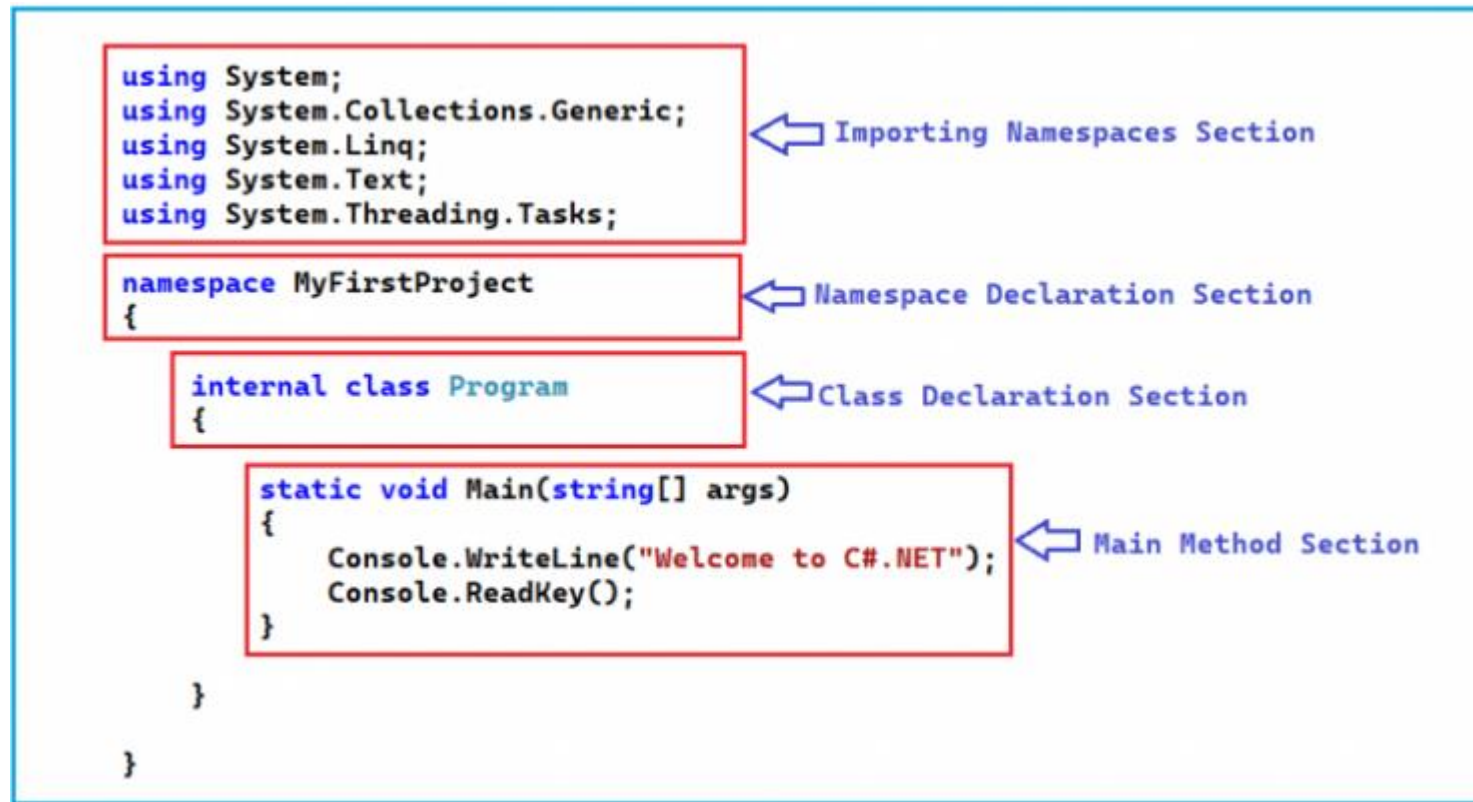
Step4 :write our code which will be used to display the message “**Welcome to C#.NET**” in the console window. To do so, modify the Main method of the Program class

```
static void Main(string[] args)
{
    Console.WriteLine("Welcome to C#.NET");
    Console.ReadKey();
}
```

Step5 :The next step is to run the .NET Application. To run any program in Visual Studio, you just need to click on the Start button or you can press **CTRL+F5** as shown in the below image

C#.NET Basics

Understanding the Code:



C#.NET Basics

Importing Namespace Section:

This section contains importing statements that are used to import the BCL (Base Class Libraries) as well as user-defined namespaces if required.

Syntax: `using NamespaceName;`

Example: `using System;`

`using System.Data;`

C#.NET Basics

Namespace Declaration Section:

Here a user-defined namespace is to be declared. In .NET applications, all classes related to the project should be declared inside some namespace. Generally, we put all the related classes under one namespace and in a project we can create multiple namespaces.

Syntax: `namespace NamespaceName {}`

Example: `namespace MyFirstProject {}`

C#.NET Basics

Class Declaration Section:

This is to declare the start-up class of the project. In every .NET Desktop application like console and windows, there should be a start-up class. In the Console Application, the start-up class name is Program.cs. A start-up class is nothing but a class that contains a Main() method from which the program execution is going to start.

Syntax:

```
class ClassName  
{  
}
```

Example:

```
class Program  
{  
}
```

C#.NET Basics

Main() Method Section:

The main() method is the entry point or starting point of the application execution. When the application starts executing, the main method will be the first block of the application to be executed. The Main method contains the main logic of the application.

C#.NET Basics

What is Console Class in C#?

The Console class is available in the System namespace. This Console class provides some methods and properties using which we can implement the user interface in a console application.

Properties of Console Class in C#:

Title, BackgroundColor, ForegroundColor, CursorSize.

Methods of Console Class in C#:

Clear(), ResetColor(), Write("string"), Beep(), WriteLine("string"), Write(variable). WriteLine(variable). Read(), ReadLine(), ReadKey().

C#.NET Basics

Data Types in C#

Why do we need Data Types in C#?

The Datatypes in C# are basically used to store the data temporarily in the computer through a program. In the real world, we have different types of data like integers, floating-point, characters, boolean, strings, etc. To store all these different kinds of data in a program to perform business-related operations, we need the data types.

C#.NET Basics

What is a Data Type in C#?

The Datatypes are something that gives information about **Size** of the memory location.

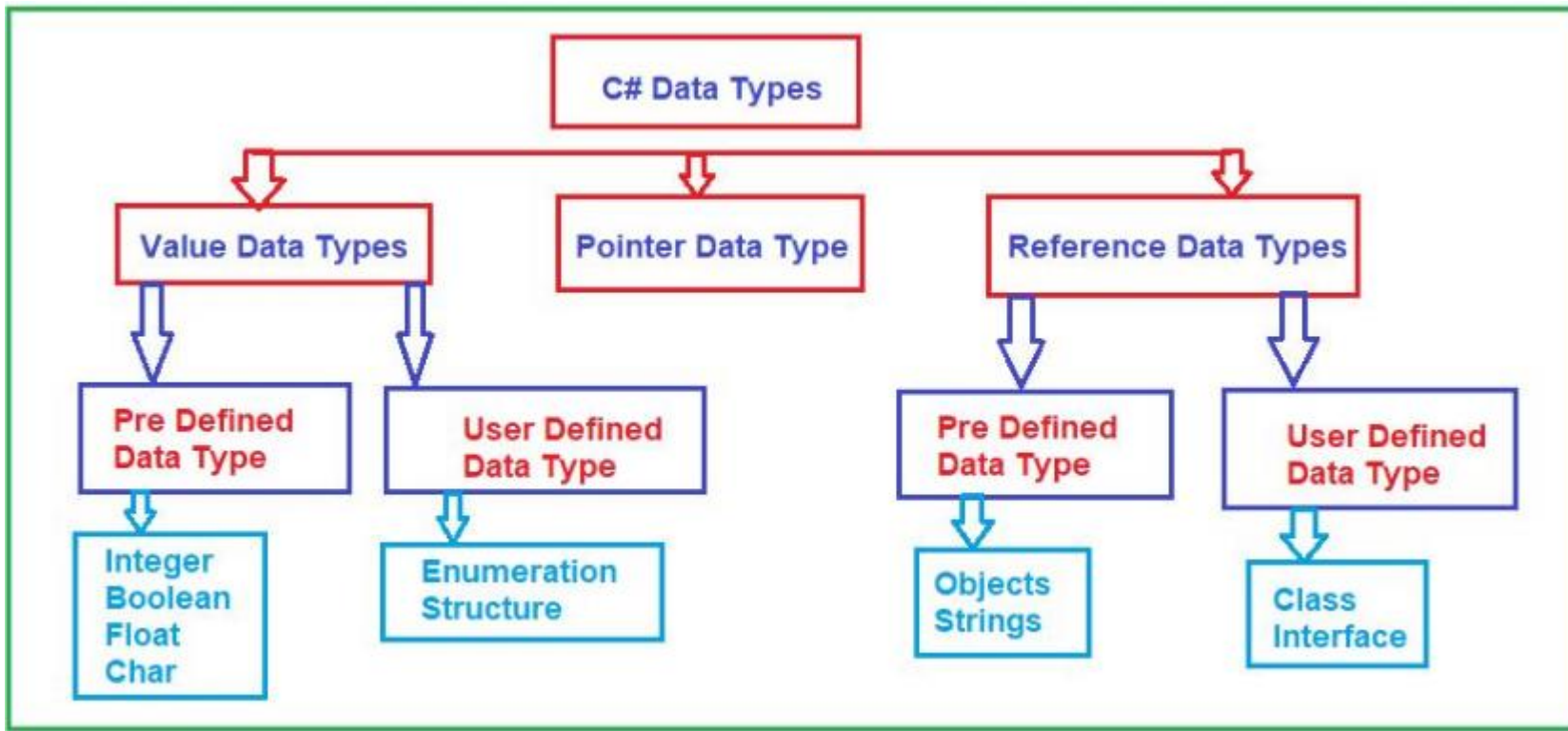
The **Range of data** that can be stored inside that memory location

Possible **Legal Operations** that can be performed on that memory location.

What **Types of Results** come out from an expression when these types are used inside that expression?

C#.NET Basics

What are the Different Types of Data types Available in C#?



C#.NET Basics

Type Casting in C#

Type Casting or Type Conversion in C# is the process to change one data type value into another data type.

Types of Type Casting in C#

1. Implicit Type Casting
2. Explicit Type Casting

C#.NET Basics

Implicit Conversion or Implicit Type Casting / Automatic Type Conversion in C#

The typecasting or type conversion is done from a smaller data type to a larger data.

Implicit Type Casting happens when:

1. The two data types are compatible.
2. When we assign a value of a smaller data type to a bigger data type.

C#.NET Basics

The following diagram shows the implicit types of conversion that are supported by C#:

Convert from Data Type

byte
short
int
long
float

Convert to Data Type

short, int, long, float, double
int, long, float, double
long, float, double
float, double
double

C#.NET Basics

Explicit Conversion or Explicit Type Casting in C#

If you want to convert the large data type to a small data type in C#, then you need to do the same explicitly using the cast operator. Explicit Conversion or Explicit Type Casting in C# is done by using the Cast operator

C#.NET Basics

Example

to Understand

Explicit Conversion or

Explicit Type Casting in C#

```
using System;
namespace TypeCastingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            double numDouble = 1.23;

            // Explicit Type Casting
            int numInt = (int)numDouble;

            // Value Before Conversion
            Console.WriteLine("Original double Value: " + numDouble);

            // Value After Conversion
            Console.WriteLine("Converted int Value: " + numInt);
            Console.ReadKey();
        }
    }
}
```

C#.NET Basics

Conversion with Helper Methods in C#:

For the conversion between non-compatible types like **integer** and **string**, **the** .NET Framework provided us with the Convert class, Parse method, and TryParse method.

C#.NET Basics

Convert Class Helper Methods in C#:

The Convert class provides the following methods to convert a value to a particular type. The following methods are going to convert the value irrespective of type compatibility. It means if types are compatible, then it will convert and if types are not compatible, then also it will try to convert.

C#.NET Basics

Method	Description
ToBoolean	It will convert a type to Boolean value
ToChar	It will convert a type to a character value
Byte	It will convert a value to Byte Value
ToDecimal	It will convert a value to Decimal point value
ToDouble	It will convert a type to double data type
ToInt16	It will convert a type to 16-bit integer
ToInt32	It will convert a type to 32 bit integer
ToInt64	It will convert a type to 64 bit integer
ToString	It will convert a given type to string
ToUInt16	It will convert a type to unsigned 16 bit integer
ToUInt32	It will convert a type to unsigned 32 bit integer
ToUInt64	It will convert a type to unsigned 64 bit integer

C#.NET Basics

Type Conversion using Parse() Method in C#

In C#, we can also use the built-in Parse() method to perform type conversion. So, while performing type conversion between non-compatible types like int and string, we can also use Parse() method like the Convert class helper methods. Now, if you go to the definition of built-in value data types such as int, short, long, bool, etc., then you will see that the Parse method is implemented as a static method in those built-in value data types. So, using the built-in type, we can call the Parse method.

C#.NET Basics

Using TryParse Method in C#:

C#.NET Basics

Variables in C#

- A name that is given for any computer memory location is called a variable.
- **Rules for variable declaration in C#:**
 1. A variable name must begin with a letter or underscore.
 2. Variables in C# are case sensitive
 3. They can be constructed with digits and letters.
 4. No special symbols are allowed other than underscores.
 5. sum, Height, _value, and abc123, etc. are some examples of the variable name
- The Syntax for declaring a variable in C# is as follows:
Syntax: data_type variable_name;

C#.NET Basics

- The Syntax for initializing a variable in C# is as follows:
Syntax: data_type variable_name = value;

C#.NET Basics

Types of Variables in a Class in C#:

1. Non-Static/Instance Variable
2. Static Variable
3. Constant Variable
4. Readonly Variable

C#.NET Basics

Operators in C#

- Operators in C# are symbols that are used to perform operations on operands.
- For example, $2 + 3 = 5$
- **Types of Operators in C#:**
 1. Arithmetic Operators
 2. Relational Operators
 3. Logical Operators
 4. Bitwise Operators
 5. Assignment Operators
 6. Unary Operators or
 7. Ternary Operator or Conditional Operator

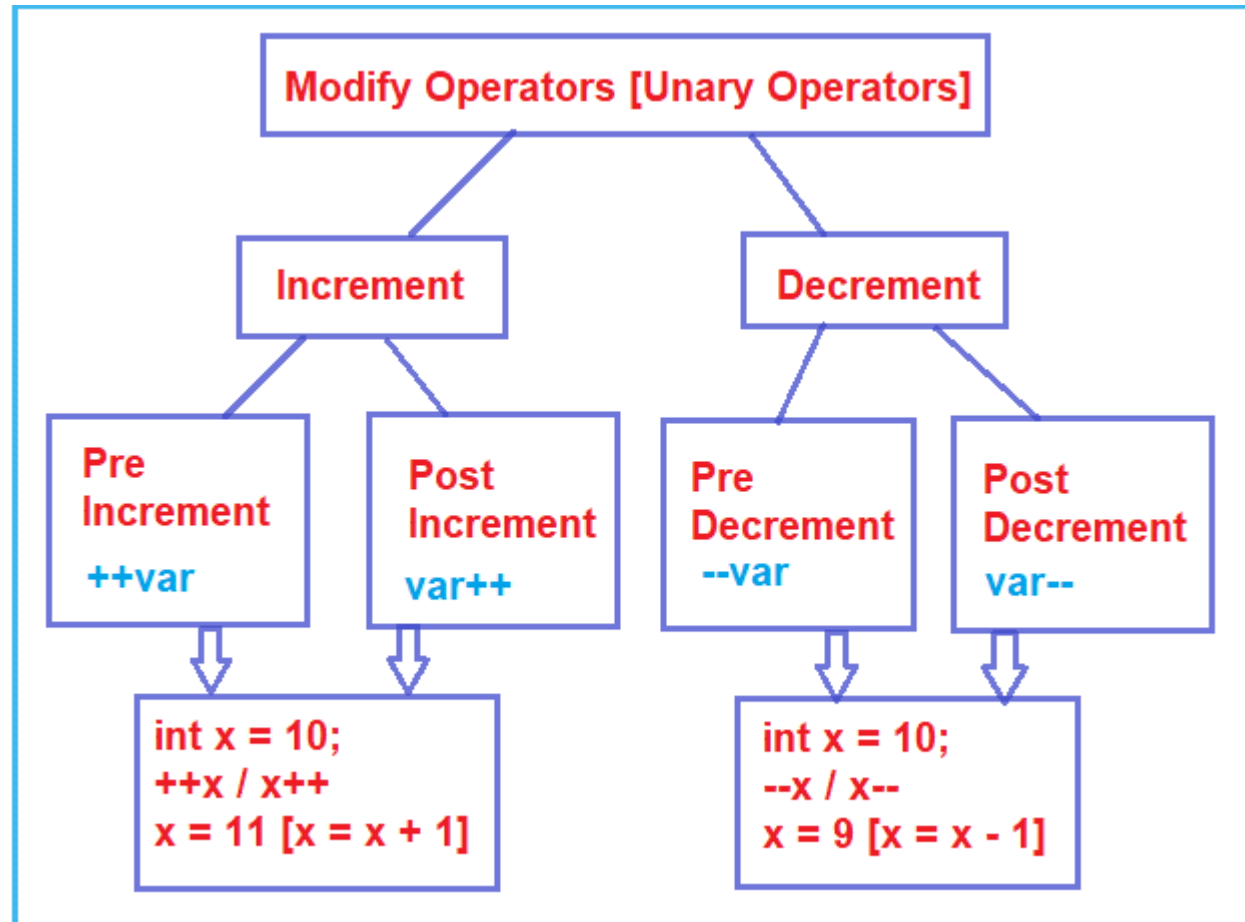
C#.NET Basics

- In C#, the Operators can also be categorized based on the Number of Operands:
 1. **Unary Operator:** The Operator that requires one operand (variable or value) to perform the operation is called Unary Operator.
 2. **Binary Operator:** Then Operator that requires two operands (variables or values) to perform the operation is called Binary Operator.
 3. **Ternary Operator:** The Operator that requires three operands (variables or values) to perform the operation is called Ternary Operator. Ternary Operator is also called Conditional Operator.

C#.NET Basics

	Operator	Type
Binary Operator →	+, -, *, /, %	Arithmetic Operators
	<, <=, >, >=, ==, !=	Relational Operators
	&&, , !	Logical Operators
	&, , <<, >>, ~, ^	Bitwise Operators
	=, +=, -=, *=, /=, %=	Assignment Operators
Unary Operator →	++, --	Unary Operators
Ternary Operator →	?:	Ternary Operator or Conditional Operator

C#.NET Basics



C#.NET Basics

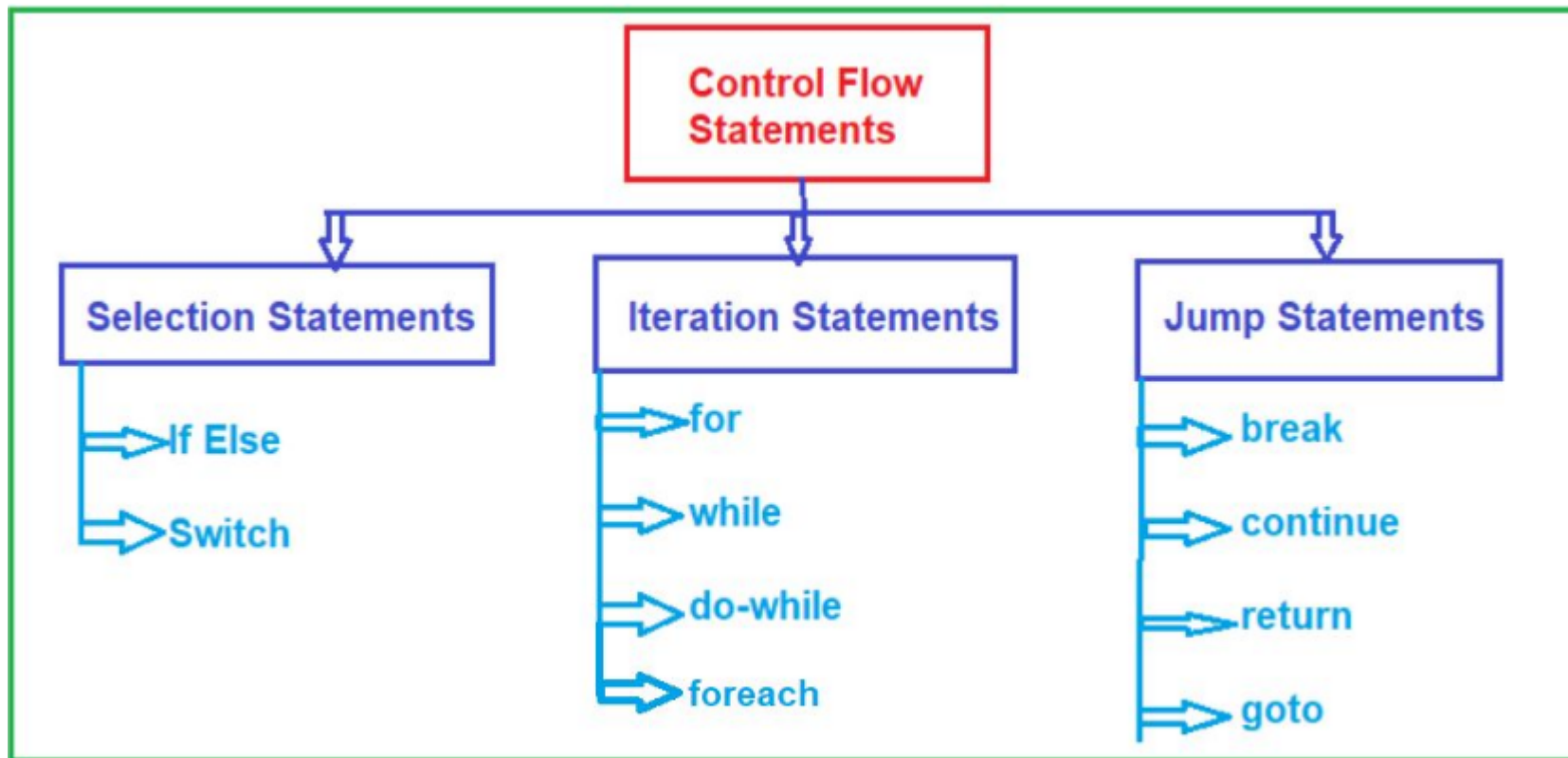
Types of Control Flow Statements in C#:

In C#, the control flow statements are divided into the following three categories:

- I. Selection Statements or Branching Statements: (Examples: if-else, switch case, nested if-else, if-else ladder)
- II. Iteration Statements or Looping Statements: (Examples: while loop, do-while loop, for-loop, and foreach loop)
- III. Jumping Statements: (Examples: break, continue, return, goto)

C#.NET Basics

Control Flow Statements in C#



C#.NET Basics

I. Selection Statements or Branching Statements

If-Else Statements in C#

```
if(condition)
{
    Statements;
}
```

If Statements

```
If (Condition)
{
    if statements;
}
else
{
    else statements;
}
```

If-Else Statements

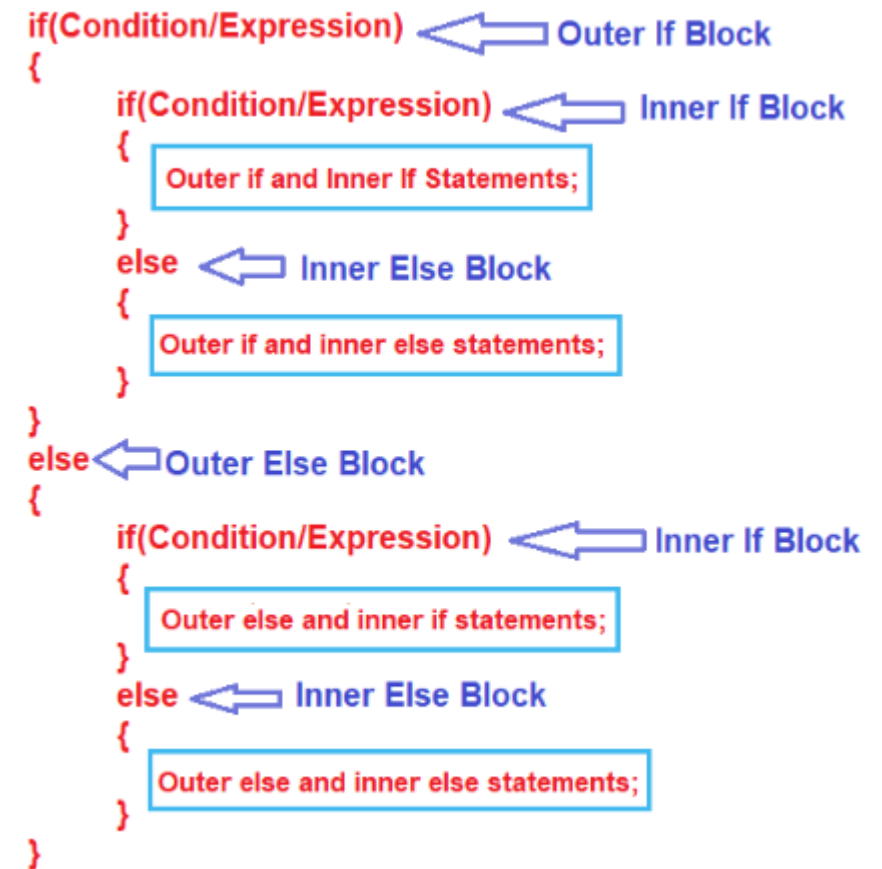
```
if (condition)
    Statement 1;
else if (condition)
    Statement 2;
.
.
.
else
    Statement n;
```

Ladder If-Else Statements

C#.NET Basics

Nested If-Else Statements in C# Language:

```
if(Condition/Expression) ← Outer If Block
{
    if(Condition/Expression) ← Inner If Block
    {
        Outer if and Inner If Statements;
    }
    else ← Inner Else Block
    {
        Outer if and inner else statements;
    }
}
else ← Outer Else Block
{
    if(Condition/Expression) ← Inner If Block
    {
        Outer else and inner if statements;
    }
    else ← Inner Else Block
    {
        Outer else and inner else statements;
    }
}
```

The diagram illustrates the structure of nested if-else statements in C#. It shows an outer if-else block. The 'if' branch contains an inner if-else block. The 'else' branch of the outer block also contains an inner if-else block. Blue arrows point from text labels to the corresponding keywords in the code: 'Outer If Block' points to the first 'if', 'Inner If Block' points to the nested 'if', 'Inner Else Block' points to the nested 'else', 'Outer Else Block' points to the outer 'else', and another 'Inner If Block' points to the second nested 'if'. Red boxes highlight the statement lines within each of the four nested blocks: 'Outer if and Inner If Statements;', 'Outer if and inner else statements;', 'Outer else and inner if statements;', and 'Outer else and inner else statements;'.

C#.NET Basics

Switch Statements in C#

- Syntax of Switch Statements in C# Language:
- The switch statement in C# only works with:
 1. Primitive data types: bool, char, and integral type
 2. Enumerated Types (Enum)
 3. String Class
 4. Nullable types of the above data types

```
switch(variable)
{
    case 1:
        //execute your code
        break;
    case n:
        //execute your code
        break;
    default:
        //execute your code
        break;
}
```

C#.NET Basics

II. Iteration Statements or Looping Statements:

Types of Loops in C#

1. For loop
2. For Each Loop
3. While loop
4. Do while loop

C#.NET Basics

1. While Loop in C# Language:

Syntax:

```
while (Condition)
{
    Statements;
}
```

2. Do while loop in C#

```
do
{
    statements;
}
while(condition);
```


C#.NET Basics

3. For Loop in C#:

The **for loop** allows the execution of instructions for a specific amount of time. It has four stages.

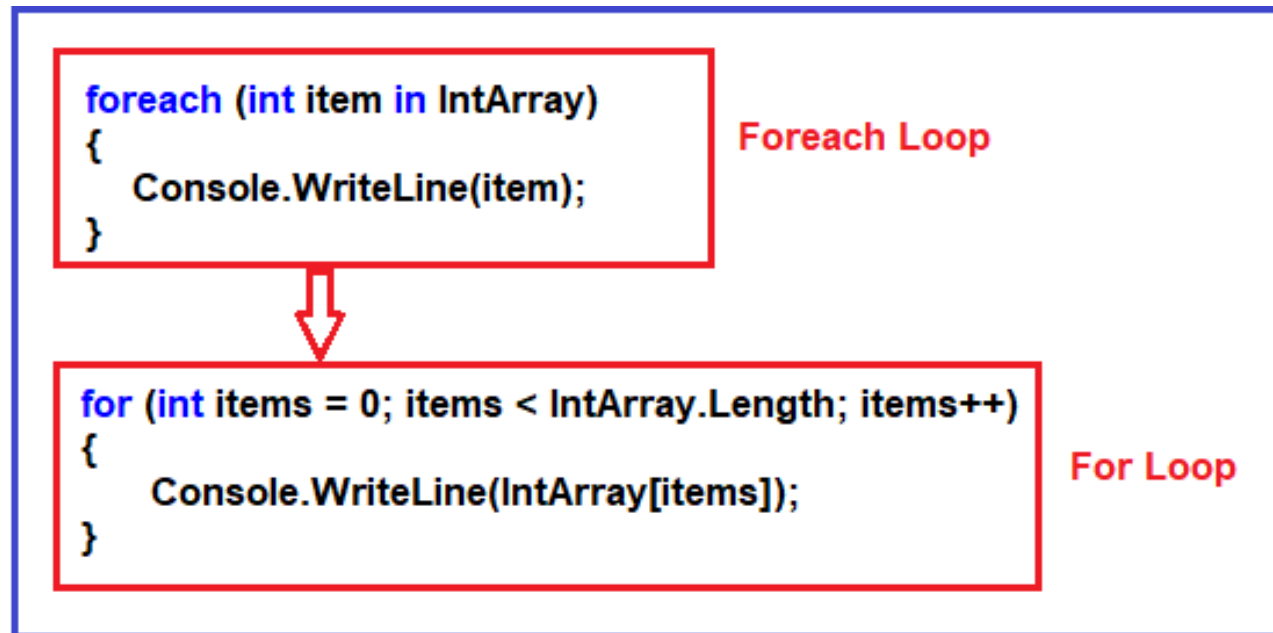
- Loop initialization
- Condition evaluation
- Execution of instruction
- Increment/Decrement

```
for (initialization; condition; increment/decrement)
{
    // c# instructions
}
```

C#.NET Basics

4. Foreach Loop in C#

The foreach loop in C# is used to iterate over the elements of a collection. Here, the collection may be an array or a list or a dictionary, etc



C#.NET Basics

III. Jumping Statements


The Jump Statements in C# are used to transfer control from one point or location or statement to another point or location or statement in the program.

1. **break**
2. **continue**
3. **goto**
4. **return** (In the Function section we will discuss the return statement)
5. **throw** (In the Exception Handling section we will discuss the throw statement)

C#.NET Basics


1. break

```
int i = 1;
while (i < 10)
{
    //Statements
    if(i == 5)
    {
        break;
    }
    //Statements
}
//Statements
```




Using break in While Loop

```
int i = 1;
do
{
    //Statements
    if (i == 5)
    {
        break;
    }
    //Statements
} while (i > 10);
//Statements
```



Using break in Do While Loop

```
for (int i = 0; i < 10; i++)
{
    //Statements
    if (i == 5)
    {
        break;
    }
    //Statements
}
//Statements
```



Using break in for loop

C#.NET Basics

2. continue

```
int i = 1;
while(i <= 5)
{
    //Statements
    if(i == 3)
    {
        continue;
    }
    //Statements
}
```

Using continue in While Loop

```
int i = 1;
do
{
    //Statements
    if (i == 3)
    {
        continue;
    }
    //Statements
}
while (i <= 5);
```

Using continue in do While Loop

```
for (int i = 1; i < 5; i++)
{
    //Statements
    if (i == 3)
    {
        continue;
    }
    //Statements
}
```

Using continue in for Loop

C#.NET Basics

3. goto

```
goto label;  
..  
.  
label: statement;
```