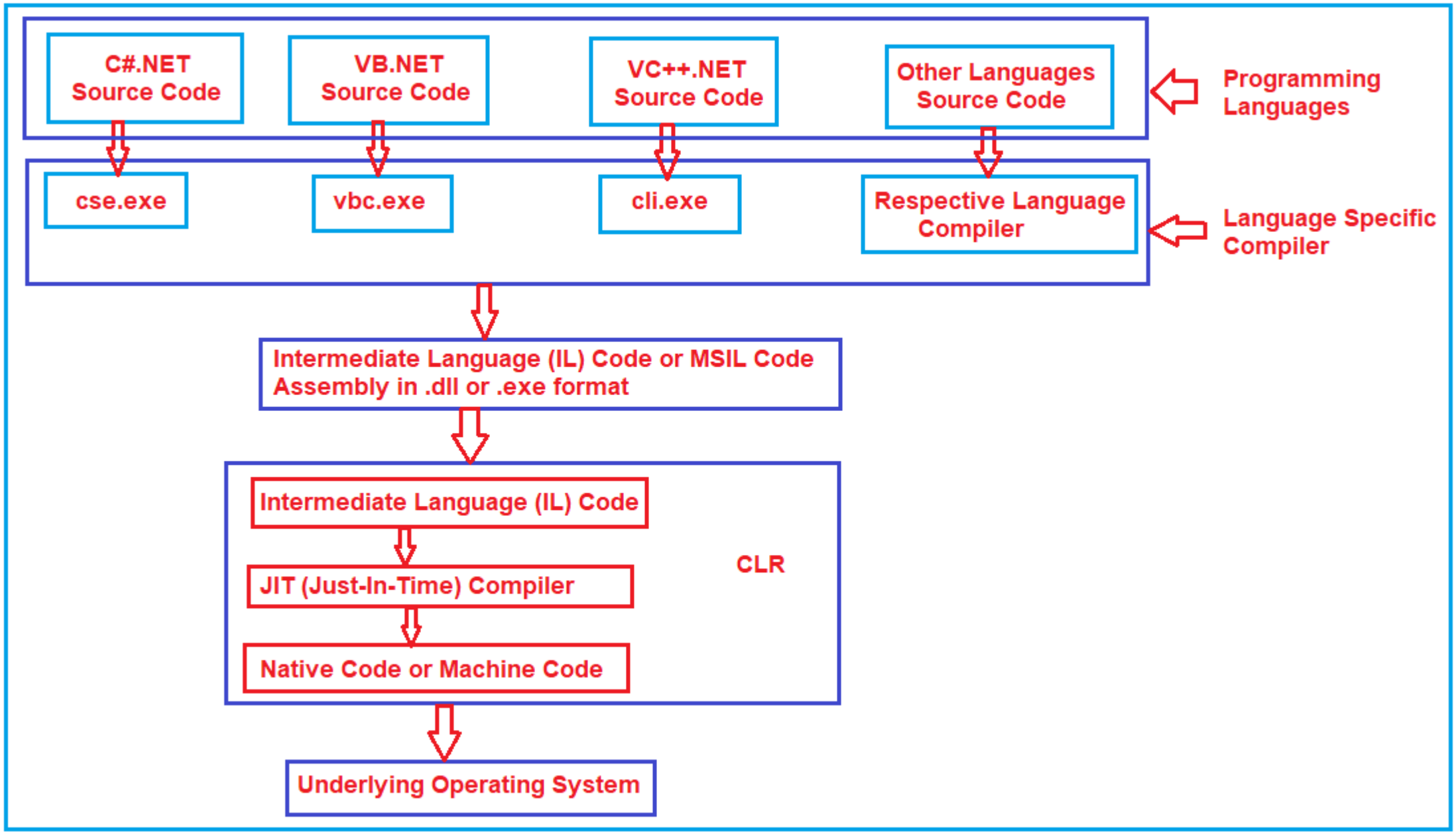
.NET Program Execution Process Flow



- What is Intermediate Language (IL) Code in .NET Framework?
- Why Partial Compiled Code or why not fully compiled Code?
- Common Language Runtime (CLR) in .NET Framework:
- Security Manager:
- JIT Compiler
- Memory Manager
- Garbage Collector
- Exception Manager
- Common Type System (CTS) in .NET Framework
- CLS (Common Language Specification) in .NET Framework

C

L

R

C# - Data Types

Value Type

✓Value type variables can be assigned a value directly. They are derived from the class **System.ValueType**.The value types directly contain data. Some examples are **int, char, and float**, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an **int** type

Reference Type

✓The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value. Example of **built-in** reference types are: **object**, **dynamic,** and **string**.

Pointer Type

✓Pointer type variables store the memory address of another type. Pointers in C# have the same capabilities as the pointers in C or C++.

Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0}$ to 28	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

Type Conversion

Implicit type conversion – These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.

Explicit type conversion – These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

C# Type Conversion Methods

ToByte - Converts a type to a byte.

ToChar -Converts a type to a single Unicode character, where possible.

ToDateTime - Converts a type (integer or string type) to date-time structures.

ToDecimal - Converts a floating point or integer type to a decimal type.

ToDouble - Converts a type to a double type.

ToInt32- Converts a type to a 32-bit integer.

C# - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

```
static void Main(string[] args) {
   short a;
   int b ;
   double c;

   /* actual initialization */
   a = 10;
   b = 20;
   c = a + b;
   Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
   Console.ReadLine();
}
```

Defining Variables

Syntax -<data_type> <variable_list>;

Example

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Initializing Variables

Syntax -<data_type> <variable_name> = value;

Example

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

C# - Constants

The constants refer to fixed values that the program may not alter during its execution. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant

Character Constants

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\xhh . . .	Hexadecimal number of one or more digits

Defining Constants

```
const <data_type> <constant_name> = value;
```

```
namespace DeclaringConstants {
   class Program {
      static void Main(string[] args) {
         const double pi = 3.14159;

         // constant declaration
         double r;
         Console.WriteLine("Enter Radius: ");
         r = Convert.ToDouble(Console.ReadLine());

         double areaCircle = pi * r * r;
         Console.WriteLine("Radius: {0}, Area: {1}", r, areaCircle);
         Console.ReadLine();
      }
   }
}
```

C# - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# has rich set of built-in operators and provides the

Arithmetic Operators | Relational Operators | Logical Operators | Bitwise Operators | Assignment Operators | Misc Operators

Arithmetic Operators

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

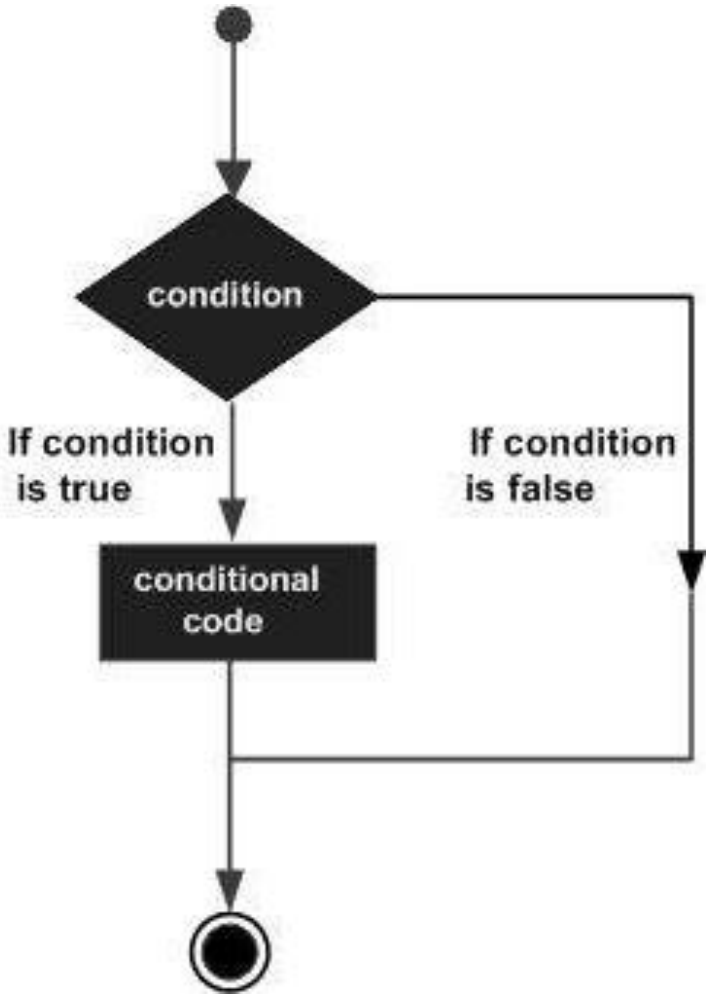
Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

C# - Decision Making

Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



if statement

An if statement consists of a boolean expression followed by one or more statements.

if...else statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

nested if statements

You can use one if or else if statement inside another if or else if statement(s).

switch statement

A switch statement allows a variable to be tested for equality against a list of values.

nested switch statements

You can use one switch statement inside another switch statement(s).

```
switch(match expression/variable)
{
    case constant-value:
        statement(s) to be executed;
        break;
    default:
        statement(s) to be executed;
        break;
}
```

The ? : Operator

We have covered **conditional operator ? :** in previous chapter which can be used to replace **if...else** statements. It has the following general form −

```
Exp1 ? Exp2 : Exp3;
```

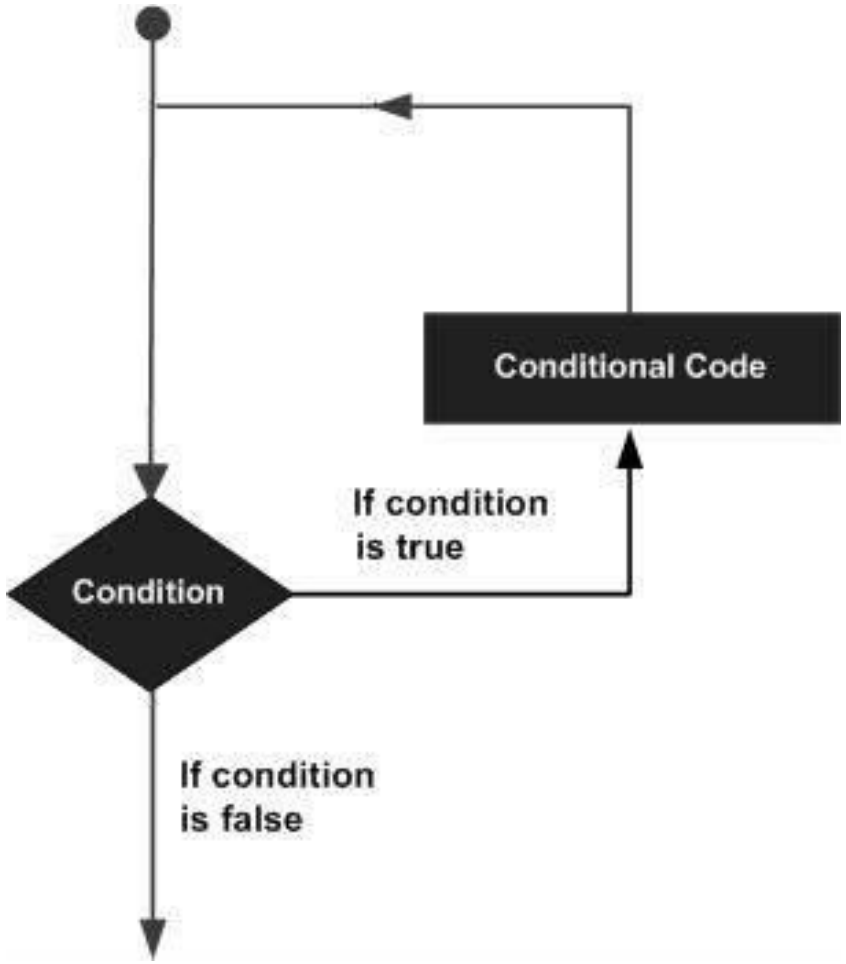
```
int x = 10, y = 100;

var result = x > y ? "x is greater than y" : "x is less than y";

Console.WriteLine(result);
```

C# Loops

There may be a situation, when you need to execute a block of code several number of times. In general, the statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.



Loop Type & Description
while loop It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop It is similar to a while statement, except that it tests the condition at the end of the loop body
nested loops You can use one or more loop inside any another while, for or do..while loop.
break statement Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.