PHASE 2

PREPARED BY BRIJESH KAMANI

Brijesh Kamani
FULL STACK DEVELOPER TRAINEE | RKIT

Contents

8. Operators and Expressions

8.1 incremental, ternary etc.

Incremental Operator:

- *In C# there are two main **unary** operators **'++'** and **'--'**.*
- *'++' is called the Incremental operator, which is used for the increment value of operand by 1.*
- *We can use this operator **before** or **after** the operand.*
- *If we use incremental operator **before** operand then it's called '**Prefix incremental operator**'.*
- *In prefix, incremental operator value of operand increased by 1 **before** executing a current expression.*
- *And if we use incremental operator **after** operand that it's called '**Postfix incremental operator**'.*
- *In Postfix operator value of the operand increased by 1 **after** executing the current expression.*

Ternary Operator (?:)

- **Syntax***:*

```
condition? statement 1: statement 2
```

- *Here you can see Ternary or Conditional operator contains three pasts:*
  - **Condition***: It can be any type of statement or expression which must be evaluated to be either true or false.*
  - **Statement 1***:This portion contains any statement and it is only executed if the above condition is evaluated to be true otherwise not.*
  - **Statement 2***:If the above condition is false then and only then this statement will be executed.*

9. Loop Iteration

9.1 for, foreach, while, do..while

- **while:**

  Syntax:

```
while(condition)
{
    //Statements
}

// Here, the above statements are executed repeatedly until the
specified condition is not evaluated as false.
```

- **do...while:**

  Syntax*:*

```
do
{
//Statements
}while(condition);

// Unlike while loop statements execute a minimum of 1 time it
doesn't matter if the specified condition is evaluated to be true or
not.
```

- **for**

  Syntatx:

```
for(initilizer;condition; Iterators)
{
       //statements
}
```

  - *Here,* **Initializers** *is used for initializing variables before entering into the loop. The scope of these variables is limited for Block of For Loop.*
  - **Condition** *defines how many times the loop will keep going.*
  - **Statements** *are executed until the specified condition is evaluated to false.*
  - **Iterators** *execute at each iteration of the loop, after every statement of the loop has been executed.*

- **foreach**

  Syntax:

```
foreach(data_type var_name in collection_variable)
{
     // statements
}
```

  - ***foreach*** *loop is used to* ***iterate*** *over the elements of the* ***collection****. The collection may be an* ***array*** *or a list. It executes* ***for each element*** *present in the* ***array****.*

9.2 break, continue

- **break:**

  syntax:

```
break;
```

  - *a* **break** *is used to* ***terminate*** *the* ***loop*** *or* ***switch*** *block which contains this brack statement.*
  - *We can only write break* ***within loop*** *or* ***switch*** *block;*

- **Continue:**

  Syntax*;*

```
continue;
```

  - **Continue** *statement used to* ***terminate current iteration*** *of the* ***loop*** *and* ***start next iteration*** *of the* ***loop****.*
  - *Code after continue statement will* ***not execute*** *for the* ***current iteration****.*

.

10. Understanding Arrays

10.1 Define and usage of the array

- *An* **Array** *is a* ***fix-sized sequential*** *collection of different* ***variables*** *of the* ***same types****.*
- ***Memory allocates*** *for these variables are* ***sequentially*** *so we can directly access these variables by* ***indexes****.*
- **Syntax***:*

```
<type[]> <arrayName>;
```

  - *Here* ***Type*** *is the* ***Datatype*** *of our array.*
  - *Naming rules for* ***arrayName*** *are the same as Variable names.*

- *There are* ***two ways*** *of specifying the* ***size of an Array****.*
  - *First is the set value of array elements* ***during declaration.***

    Like,

```
int[] myArray = {1,2,3,4,5};
```

  - *Here size of this variable becomes 5.*
  - *And If we don't want to set values of the array during declaration we must have to specify the size of an array like this,*

```
int[] myArray = new int[5];
```

- *Latter we can* ***access*** *and* ***reset*** *values of array elements by their* ***index****.*
- ***Index*** *of an array starts with* ***0*** *so if we want to* ***retrieve the value*** *of the* ***first element*** *of an array then we do like this,*

```
Console.Writeline(myArray[0]);       // it can print value of First
Array Element.
```

- *If we want to* ***change*** *the value of the* ***5th*** *element of an array we can do like this*

```
myArray[4] = 10;
```

- *This is a* ***One Dimensional Array****. An Array can be* ***two*** *or* ***multiple*** *Dimensional also.*

  For example,

the following declaration creates a two-dimensional array of four rows and two columns.

```
int[,] array = new int[4, 2];
```

- We can also create an **Array of Arrays** called **Jagged Arrays.**

  We create Jagged Arrays like this,

```
int[][] jaggedArray = new int[3][];
```

11. Defining and Calling Methods

11.1 Define method and use

- *A Method is a bunch of statements that perform some operation for a specific class/object.*
- *We just have to define a method for the specific type of option in class and later whenever we need to perform that operation we just need to call this method.*

Syntax of creating Method:-

```
[Access_specifire] <Return_type> <method_name> ([parameters , …])
{
   // Lines of code….
}
```

- *Here **Access-Specifier** represents the access area of Method.*
- ***Return_Type** represent which data type of value this method will return. If it returns nothing then its data type must be Void.*
- *Rules for providing methods name is same as Variable name.*
- *All **parameters** must be defined within parentheses. During calling We can pass data to our method using Parameters. Parameters specify how many numbers of data and which type of data method can accept after calling. The order of this data must be the same as already specified in a method declaration.*
- *Body of Method must be within **Curly bracket**. The body contains all the codes that will execute after the calling method.*

Syntax of Calling a Method:

```
   <Method_name>([Paramete rs, …]);
```

11.2 different type of parameters in method (Value type, Ref. type, optional)

Value type:

- ***Value** type parameters are such parameters where they **hold the only value** of **passed variables** during **Method Calling**.*
- *It means it just **copies** the **value** of passed variables. Any **changes** done on parameters **doesn't affect** to passed variables of the **Method Calling**.*

Reference type:

- *It means its **hold** a **memory address** of the passed variables of Method Calling. It means any **changes** done on that parameter **affect** those **variables** that **have been passed** during **Method Calling**.*

Optional type:

- *An **optional** parameter has a **default value** as part of its **definition**. If **no argument** is sent for that parameter, the **default value** is **used**.*

12. Working with strings

12.1 string class study

- *a **string** is used to **store** a **collection** of **characters** that represent **Text**. We can directly use **System**.**String class** or we can use **string keyword** which is an **alias** for String class for creating String object.*
- *The **value** of this string must be within **double quotes** (“ ”).*

Example:

```
//Initialize string with value.
string myString = "this is string Example";
```

- *Value of string is stored as a **sequential read-only** collection of **Char objects**. There is no **null-terminating character** at the end of a C# string; therefore a C# string can **contain any number** of embedded **null characters** ('\0').*
- *The **Length** property of a string **represents** the **number** of **Char** objects it contains*
- *String objects are* **immutable**. *they **cannot** be **changed** after they have been created. All of the **String methods** and C# operators that appear to **modify** a string **return** the results in a **new string object**.*

12.2 Use of various string methods

Clone()	*It is used to return a reference to this instance of String.*
Copy(String)	*It is used to create a new instance of String with the same value as a specified String.*
Compare(String, String)	*It is used to compare two specified String objects. It returns an integer that indicates their relative position in the sort order.*
Concat(String, String)	*It is used to concatenate two specified instances of String.*
Contains(String)	*It is used to return a value indicating whether a specified substring occurs within this string.*
Equals(String, String)	*It is used to determine that two specified String objects have the same value.*
IndexOf(String)	*It is used to report the zero-based index of the first occurrence of the specified string in this instance.*
Insert(Int32, String)	*It is used to return a new string in which a specified string is inserted at a specified index position.*
IsNullOrEmpty(String)	*It is used to indicate that the specified string is null or an Empty str*
PadLeft(Int32)	*It is used to return a new string that right-aligns the characters in this instance by padding them with spaces on the left.*
Remove(Int32)	*It is used to return a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.*
Replace(String, String)	*It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.*
Split(Char[])	*It is used to split a string into substrings that are based on the characters in an array.*

StartsWith(String)	*It is used to check whether the beginning of this string instance matches the specified string.*
Substring(Int32)	*It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.*
ToLower()	*It is used to convert String into lowercase.*
ToUpper()	*It is used to convert String into uppercase.*
Trim()	*It is used to remove all leading and trailing white-space characters from the current String object.*
TrimEnd(Char[])	*It Is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.*
TrimStart(Char[])	*It is used to remove all leading occurrences of a set of characters specified in an array from the current String object*

Reference:- https://www.javatpoint.com/c-sharp-strings

13. Working with datetimes

13.1 Datetime struct

- *Datetime struct is used to perform some operations like **Add**, **Subtract**, **Convert** formats of Date and time from a particular date and time.*
- *We can also **retrieve** the **current date** and time and **operate** on it.*
- *We can **retrieve** the **remaining days** or **weeks** or **years** from a Particular date using this struct.*

Example:

```
var date1 = new DateTime(2008, 5, 1, 8, 30, 52);
Console.WriteLine(date1);   // 05/01/2008 08:30:52
```