MODULE - II

UNDERSTANDING DOCUMENT

PREPARED BY
PINAL PAMBHAR

Table of Contents

Operators and Expressions

Expressions

- An expression in C# is a combination of operands (variables, literals, method call) and operators that can be evaluated to a single value.
- An expression must have at least one operand but may not have any operator.

Operators

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

- Arithmetic Operators: +,-,*,/,%,++,--
- Relational Operators: ==,!=,>,<,>=,<=
- Logical Operators: &&,||,!
- Bitwise Operators: `,&,|,^,<<,>>
- Assignment Operators: =,+=,-=,*=,/=,%=,<<=,>>=,&=,^=,|=
- Miscellaneous Operators: sizeof(), typeof(), ?:, *, &, is

Arithmetic Operators: The arithmetic operators perform arithmetic operations on all the numeric type operands.

Relational Operators: Relational Operators are useful to check the relation between two operands like we can determine whether two operand values equal or not, etc.,

Logical Operators: Logical Operators are useful to perform the logical operation between two operands like AND, OR, and NOT based on our requirements.

Bitwise Operators: It will work on bits, and these are useful to perform bit-by-bit operations such as Bitwise AND (&), Bitwise OR (|), etc.

Assignment Operators: The assignment operator assigns the value of its right-hand operand to a variable.

Miscellaneous Operators: It performs some miscellaneous tasks like Returns the size of a data type, Returns the type of a class, etc.

Loop Iteration

for,foreach,while,do..while

for Loop

- The **for** loop executes a block of statements repeatedly until the specified condition returns false.
- The **for** keyword indicates a loop in C#.

Syntax

```
for (initializer; condition; iterator)
{
    //code block
}
```

foreach Loop

- The **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.
- Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.

Syntax

```
foreach (data_type var_name in collection_variable)
{
    //code block
}
```

while Loop

- The **while loop** loops through a block of code as long as a specified condition is True.

Syntax

```
while (condition)
{
    //code block
}
```

do while Loop

- The do/while loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do
{
    //code block
}
While (condition);
```

break, continue

break

- The break statement is used to terminate a loop(for, if, while, etc.) or a switch statement on a certain condition.
- And after terminating the controls will pass to the statements that are present after the break statement, if available.
- If the break statement exists in the nested loop, then it will terminate only those loops which contain the break statements.

Syntax

```
break;
```

FlowChart:

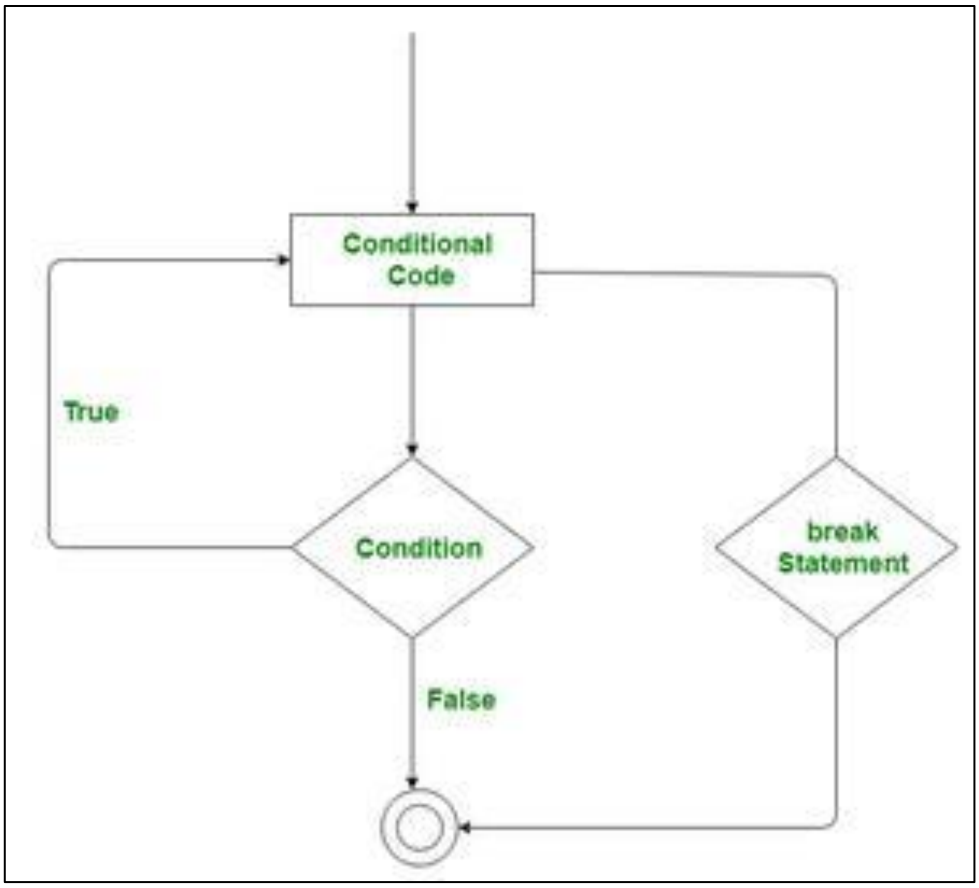


Figure 1.1 break

continue

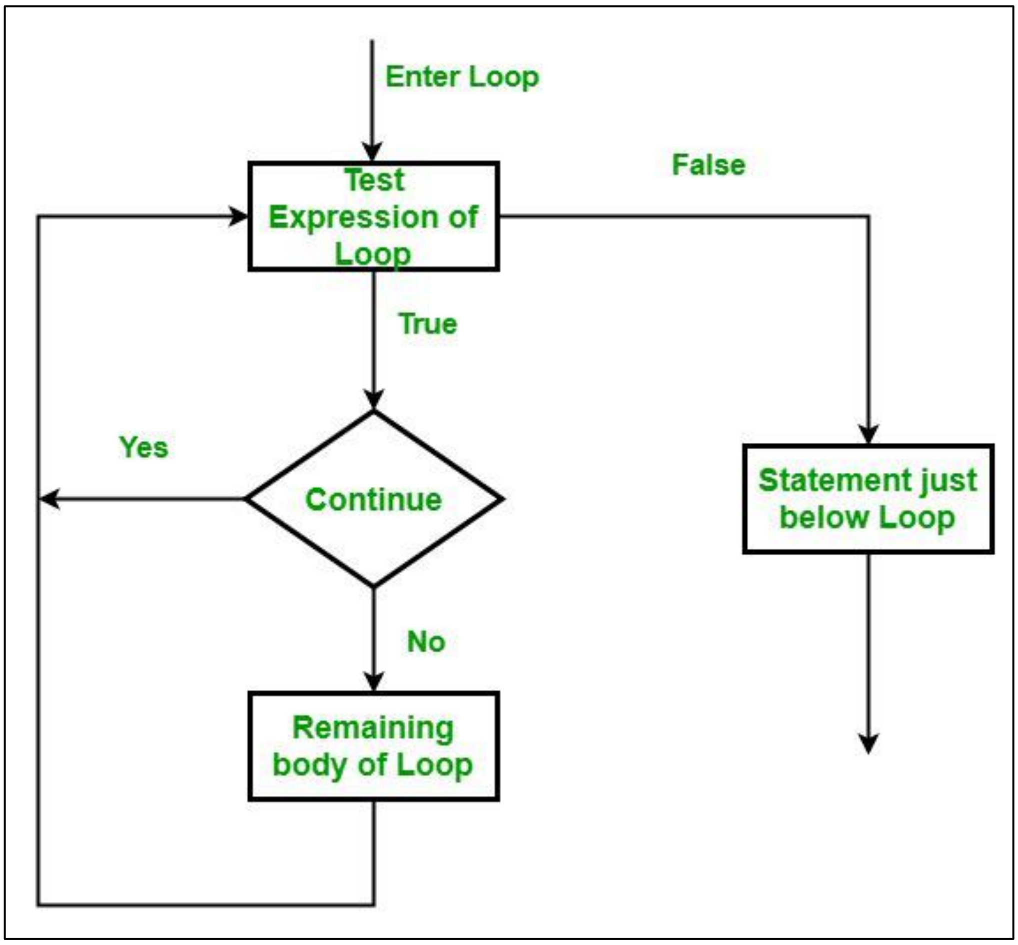
- The continue statement is used to skip over the execution part of the loop(do, while, for, or foreach) on a certain condition, after that, it transfers the control to the beginning of the loop.
- It skips its given statements and continues with the next iteration of the loop.

- Or in other words, the continue statement is used to transfer control to the next iteration of the enclosing statement(while, do, for, or foreach) in which it appears.

Syntax

```
continue;
```

FlowChart:



Understanding Arrays

- An array is a group of like-typed variables that are referred to by a common name.
- And each data item is called an element of the array.
- The data types of the elements may be any valid data type like char, int, float, etc. and the elements are stored in a contiguous location.
- The **length** of the array specifies the number of elements present in the array.
- The variables in the array are ordered and each has an index beginning from 0.
- C# array is an object of base type **System.Array**.

Array Declaration:

Syntax

```
< Data Type > [ ] < Name_Array >
```

Here,
< Data Type >: It defines the element type of the array.
[ ]: It defines the size of the array.
< Name_Array >: It is the Name of the array.

Note: Only Declaration of an array doesn't allocate memory to the array. For that array must be initialized.

Array Initialization:

Syntax

```
< Data Type > [ ] < Name_Array > = new < datatype > [size];
```

Here,

- **New** will allocate memory to an array according to its size.
- Type specifies the type of data being allocated, size specifies the number of elements in the array, and Name_Array is the name of an array variable.

Types of Array:

There are mainly three types of array are available.

1. One Dimensional Array
2. Multidimensional Array
3. Jagged Array

One Dimensional Array

- One dimensional array contains only one row for storing the values.
- All values of this array are stored contiguously starting from 0 to the array size.
- For example, declaring a single-dimensional array of 5 integers :

example

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

Multidimensional Array

- The multi-dimensional array contains more than one row to store the values.
- It is also known as a **Rectangular Array** because it's each row length is the same.
- It can be a **2D-array** or **3D-array** or more.
- To storing and accessing the values of the array, one required the nested loop.
- The multi-dimensional array declaration, initialization, and access are as follows :

Syntax

```
// creates a two-dimensional array of
// four rows and two columns.
int[, ] intarray = new int[4, 2];
```

Jagged Array

- An array whose elements are arrays is known as Jagged arrays it means "**array of arrays**".
- The jagged array elements may be of different dimensions and sizes.

Defining and Calling Methods

Define method and use

- Methods are generally the block of codes or statements in a program.
- This gives the user the ability to reuse the same code which ultimately saves the excessive use of memory, acts as a time saver, and more importantly, it provides better readability of code.
- A method is a collection of statements that perform some specific task and return the result to the caller.
- A method can also perform some specific task without returning anything.

Method Declaration

Method declaration means the way to construct a method including its naming.

Syntax

```
<Access_Modifier> <return_type> <method_name>([<param_list>])
{
   // Body
}
```

different types of parameters in the method

Value Type Parameters:

- It is a normal value parameter in a method or you can say the passing of value types by value.
- So when the variables are passed as value types they contain the data or value, not any reference.
- If you will make any changes in the value type parameter then it will not reflect the original value stored as an argument.

Reference Type Parameters:

- The ref is a keyword that is used for passing the value types by reference.
- Or we can say that if any changes made in this argument in the method will reflect in that variable when the control return to the calling method.
- The ref parameter does not pass the property.
- In ref parameters, the parameters must initialize before it passes to ref.
- The passing of value through the ref parameter is useful when the called method also needs to change the value of the passed parameter.

Default or Optional Type Parameters:

- As the name suggests optional parameters are not compulsory parameters, they are optional.
- It helps to exclude arguments for some parameters.
- Or we can say in optional parameters, it is not necessary to pass all the parameters in the method.
- Here, every optional parameter contains a default value which is part of its definition.
- If we do not pass any arguments to the optional parameters, then it takes its default value.
- The optional parameters are always defined at the end of the parameter list.

Working with strings

string class study

- A **string** is a sequence of Unicode characters or an array of characters.
- The range of Unicode characters will be **U+0000 to U+FFFF**. The array of characters has also termed the text.
- So the string is the representation of the text. A string is represented by a class **System.String**.

Characteristics of String Class:

- With the help of the length property, it provides the total number of characters present in the given string.
- String objects can include a null character which counts as the part of the string's length.
- It allows empty strings. Empty strings are the valid instance of String objects that contain zero characters.
- It also supports searching strings, comparison of strings, testing of equality, modifying the string, normalization of string, copying of strings, etc.

Properties:

Property	Description
Char[Int32]	Gets the char object at a specified position in the current String object.
Length	Gets the number of characters in the current String object.

Use of various string methods

There are a variety of methods in the String class. Some of them are shown below.

Method	Description
Compare()	Use to compare the two string object.
Concat()	Concatenates one or more instances of String, or the String representations of the values of one or more instances of Object.

Copy(String)	Creates a new instance of String with the same value as a specified String.
EndsWith()	Determines whether the end of this string instance matches a specified string.
Equals()	Determines whether two String objects have the same value.
GetHashCode()	Returns the hash code for this string.
IsNullOrEmpty(String)	Indicates whether the specified string is null or an empty string.
IndexOf()	Reports the zero-based index of the first occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance.

Datetime class study

- We used the DateTime when there is a need to work with the dates and times.
- We can format the date and time in different formats by the properties and methods of the DateTime.
- DateTime helps developers to find out more information about Date and Time like getting month, day, year, weekday.
- It also helps to find date differences, add several days to a date, etc.

DateTime Constructor

- It initializes a new instance of DateTime object.
- At the time of object creation we need to pass required parameters like year, month, day, etc.

Ex:

```
// 2021 is year, 10 is month, 14 is day
DateTime date1 = new DateTime(2021, 10, 14);
Console.WriteLine(date1.ToString()); // 10/14/2021 10:00:00 AM
```

DateTime Fields

DateTime object contains two static read-only fields called as MaxValue and Minvalue.

- MinValue – It provides smallest possible value of DateTime.
- MaxValue – It provides smallest possible value of DateTime.

DateTime Properties

- It contains properties like Day, Month, Year, Hour, Minute, Second, DayOfWeek and others in a DateTime object.

DateTime Methods

- DateTime contains a variety of methods which help to manipulate DateTime Object.
- Some of these methods are given below.

Method	Description
Add(TimeSpan)	Returns a new DateTime that adds the value of the specified TimeSpan to the value of this instance..
IsLeapYear(Int32)	Returns an indication whether the specified year is a leap year.
Subtract(DateTime)	Returns a new TimeSpan that subtracts the specified date and time from the value of this instance.
GetDateTimeFormats()	Converts the value of this instance to all the string representations supported by the standard date and time format specifiers.
ToUniversalTime()	Converts the value of the current DateTime object to Coordinated Universal Time (UTC).