Loop statements

Before start

- Understand statements in C#
- Operators in C#
- Data types and Array

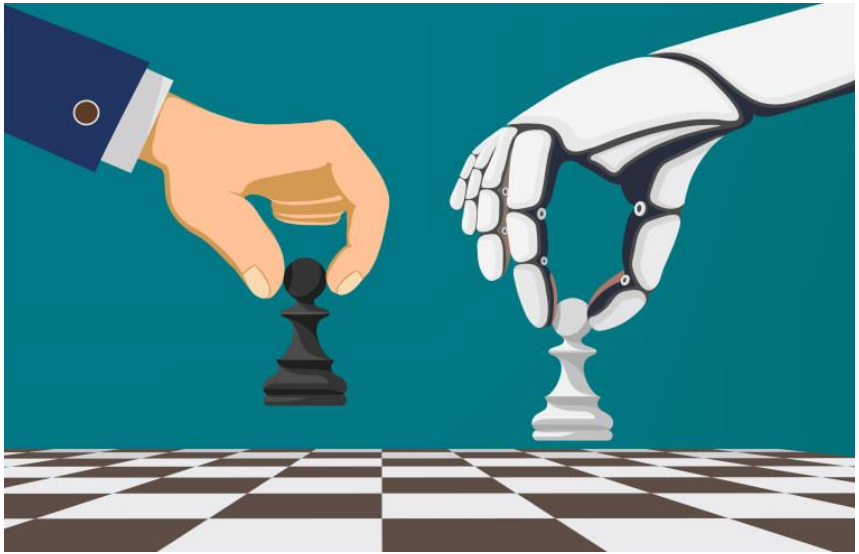
Lesson Objectives

- Loop constructs
- for loop, foreach loop
- while loop, do-while loop
- Loops comparison
- Parallel Programming with Task Parallel library

Section 1

LOOP CONSTRUCTS

Computer vs Human



Loop constructs

- Loops allow you to execute a single statement or a block of statements repetitively.
- The most common uses, loops work with a set of data such as array, collection, …
- The loop constructs are also referred to as iteration statements.

Loop constructs

- In software programming, a loop construct contains a condition that helps the compiler identify the number of times a specific block will be executed.
- If the condition is not specified, the loop continues infinitely and is termed as an infinite loop.

Loop in practice

- When developer working with loop:
  - ✓ Be consider, how many times the code looped
  - ✓ To make sure, end of loop (avoid endless loop)
  - ✓ Be consider performance and choose right approach

Section 2

for loop

for loop

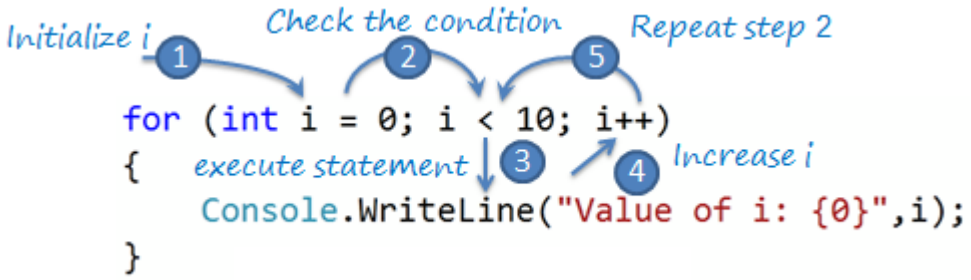
- The for loop executes a block of statements repeatedly until the specified condition returns false.
- Syntax:

```
for (variable initialization; condition; steps)
{
    //execute this code block as long as condition is satisfied;
}
```

  - ✓ variable initialization: Declare & initialize a variable here which will be used in conditional expression and steps part.
  - ✓ condition: The condition is a boolean expression which will return either true or false.
  - ✓ steps: The steps defines the incremental or decremental part

for loop

- Example: To write all number from 0 to 9:



for loop

- Step 1: declare & initialize an int type variable.
- Step 2: check the condition.
- Step 3: execute the code block if the 'if' condition returns true.
- Step 4: increment the int variable
- Step 5: eveluate the condition again and repeat the steps.

for loop

- Example: To get and print all elements in array of int:

```
for(int i=0; i< arrayOfInt.Length; i++)
{
    Console.WriteLine("arrayOfInt[{0}] = {1}", i, arrayOfInt[i]);
}
```

- Question: Why does i start from 0?

for fact

- We can use loop in any direction
- for loop can nested in for loop. But, we should consider how many times the code looped

```
var matrix = new int[10, 12];
// code to assign values to the matrix
for(int i =0; i<matrix.GetLength(0); i++)
{
    for(int j=0; j <  matrix.GetLength(1); j++)
    {
        matrix[i, j] = Math.Abs(matrix[i, j]);
    }
}
```

```
var matrix = new int[10, 12];
// code to assign values to the matrix
for(int i =0; i<matrix.GetLength(0); i++)
{
    for(int j=0; j <  matrix.GetLength(1); j++)
    {
        if(matrix[i, j] < 0)
        {
            matrix[i, j] = -matrix[i, j];
        }
    }
}
```

for fact

- Both variable initialization, steps may contains multiple expression.

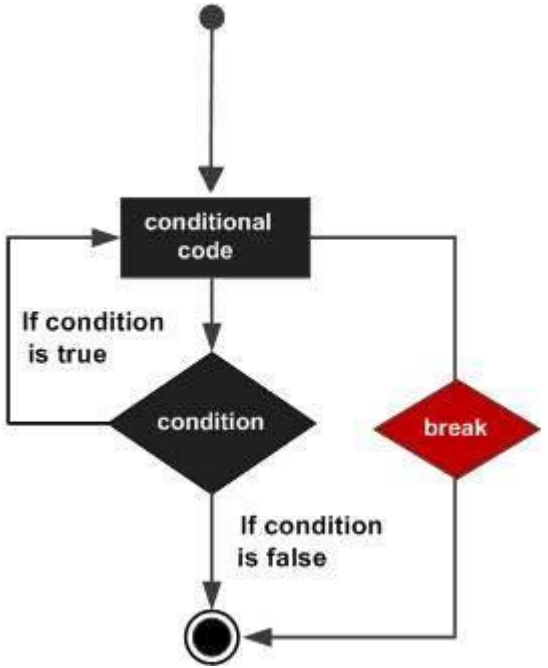
```
for (int i = 1, j = 2; i < 5 && j < 7; i++, j++)
{
    Console.WriteLine("i + j = ", i + j);
}
```

  - All 3 (variable initialization, conditionals, steps) may leave blank

```
for(; ; )
{
    // the endless loop
}
```

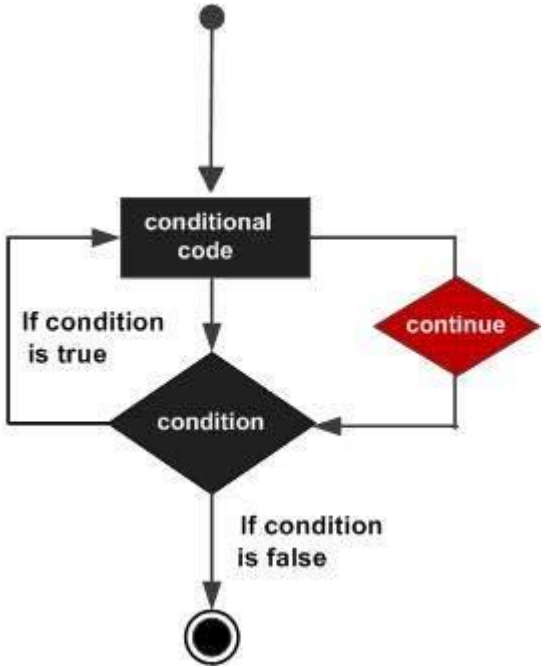
break keyword

- The break statement terminates the closest enclosing loop or switch statement in which it appears.
- Control is passed to the statement that follows the terminated statement, if any.



continue keyword

- The continue statement passes control to the next iteration of the enclosing loop statement in which it appears.
- In most cases, we can avoid continue keyword by re-order code or use conditional statements



Practice time

- Give an array of integers. Resolve these problems:
  - ✓ Get total of all elements
  - ✓ Find max/min value in array
  - ✓ Invert an array
  - ✓ ….

Section 2

foreach loop

foreach loop

- The foreach statement executes a statement or a block of statements for each element in an instance of the type
- Syntax:

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

- Example

```
foreach(int element in arrayOfInt)
{
    Console.WriteLine(element);
}
```

for VS foreach

1. Direction

for: goes in any direction

whereas

foreach: goes in default direction

for VS foreach

2. Syntax

for: complexity in its syntax

whereas

foreach: simple and natural syntax

for VS foreach

3. Performance

The foreach-loop is often less efficient than a simple for-loop

foreach loop creates a copy of the collection, so in this loop, the 'item' is not referencing to the array elements; it's just a temporary variable and you cannot assign a value to it.

foreach takes much time as compared to the 'for' loop because internally, it uses extra memory space, as well as, it uses GetEnumarator() and Next() methods of IEnumerables.

for VS foreach

```
static void Forloop()
{
    int[] list = { 1, 2, 3, 4, 5 };
    int len = list.Length;
    for (int i = 0; i < len; i++)
    {
        Console.WriteLine("FOR ITEM: " + ++list[i]);
    }
}
```

```
for items : 2
for items : 3
for items : 4
for items : 5
for items : 6
```

```
0 references
static void ForeachLoop()
{
    int[] list = { 1, 2, 3, 4, 5 };

    foreach (int value in list)
    {
        Console.WriteLine("FOREACH ITEM: " + ++value);
    }
}
```

(local variable) int value

Cannot assign to 'value' because it is a 'foreach iteration variable'

Time to discuss

- Why developers still use foreach?
- Can we set for/foreach loop endless?

Resolve problem

- Find and remove elements in a collection
  - ✓ Give list of products with: Name, Manufacture date, Expired Date.
  - ✓ Remove all products that Expired Date is in the past
- Should you use for OR foreach?
- Why and How?

Section 3

while loop

while loop

- The while loop is used to execute a block of code repetitively as long as the condition of the loop remains true.
- The while loop consists of the while statement, which begins with the while keyword followed by a boolean condition.
- If the condition evaluates to true, the block of statements after the while statement is executed.
- After each iteration, the control is transferred back to the while statement and the condition is checked again for another round of execution.
- When the condition is evaluated to false, the block of statements following the while statement is ignored and the statement appearing after the block is executed by the compiler.

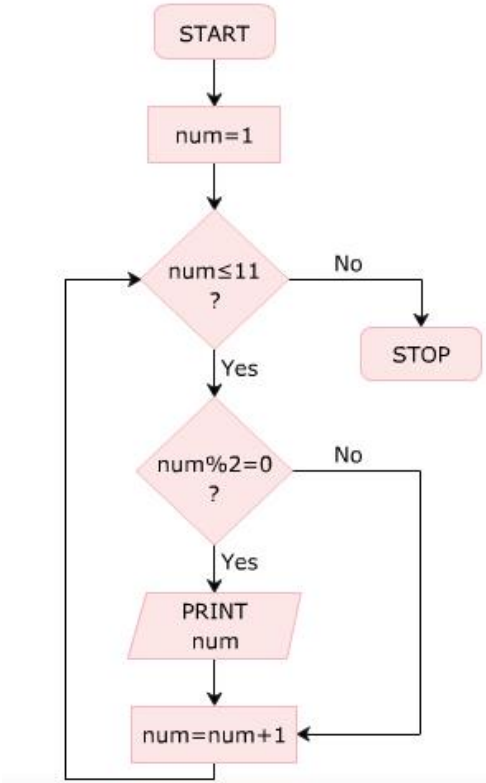
while loop

- Syntax:

```
while (condition)
{
    // body statements;
}
```

- condition: Specifies the boolean expression.

Example




```
int num = 1;
Console.WriteLine("Even Numbers");
while (num <= 11)
{
    if ((num % 2) == 0)
    {
        Console.WriteLine(num);
    }
    num = num + 1;
}
```

Endless while loop

- Forget to change the factor of condition
- Condition never false

```
int num = 1;
Console.WriteLine("Even Numbers");
while (num <= 11)
{
    if ((num % 2) == 0)
    {
        Console.WriteLine(num);
    }
    else
    {
        num += 1;
    }
}
```

do-while loop

- The do-while loop is similar to the while loop; however, it is always executed at least once without the condition being checked.
- The loop starts with the do keyword and is followed by a block of executable statements.
- The statements in the do-while loop are executed as long as the specified condition remains true.

for VS while

1. Situation

for: when you know how many times the code/block needs to be in loop.
E.g. Fine a phone number in a contacts

whereas

while: it is unsure how many times the code should be in loop
E.g. Read all lines of the text file.

for VS while

2. Conditional

if statement should be include statement to change input of conditional operator.
It usually is not depend on result of body

while statement should be change input of conditional operator in code body.
It usually is depend on result of body

for VS while

3. Endless situation

Can be: when we don't change input of conditional operator OR we re-set value of it

VS

Should be: when we don't change input of conditional operator OR we re-set value of it

for VS while

4. Best practice

With while statement, always set condition to end loop.

```
bool isEndOfFile = false;
//// Set max line to read: 10k lines only
int maxLine = 10 * 1000;
int lineCount = 0;
while(!isEndOfFile && lineCount < maxLine)
{
    var lineContent = ReadLine(out isEndOfFile);
    lineCount += 1;
}
```

Practice Time

- Give a very long array of integers. Almost elements in the array are positive numbers. Use while array to find first negative number from the array.
- Example:
  - ✓ Input: [1, 2, 3, 5, 9, 12, 15, 4, 5, **-7**, 6, 42, 53, 8, -2, …….]
  - ✓ Output: -7

PARALLEL PROGRAMMING WITH TASK PARALLEL LIBRARY

Task Parallel Library

- A set of public types and APIs in the System.Threading and System.Threading.Tasks namespaces
- Make developers more productive by simplifying the process of adding parallelism and concurrency to applications

Parallel For loop

- Standard C# for loop is going to run using a single thread whereas, Parallel For loop is going to execute using multiple threads.
  - ✓ Consider application performance
  - ✓ Consider server resource (CPU)

Syntax

A sequential for loop in C#:

```
int n = 10;
for (int i = 0; i <= n; i++)
{
    // ...
};
```

A parallel for loop in C#:

```
int n = 10;
Parallel.For(0, n, i =>
{
    // ...
});
```

Example

```
Console.WriteLine("C# For Loop");
int number = 10;
for (int count = 0; count < number; count++)
{
    //Thread.CurrentThread.ManagedThreadId returns an integer that
    //represents a unique identifier for the current managed thread.
    Console.WriteLine($"value of count = {count}, thread = {Thread.CurrentThread.ManagedThreadId}");
    //Sleep the loop for 10 miliseconds
    Thread.Sleep(10);
}
Console.WriteLine();
Console.WriteLine("Parallel For Loop");
Parallel.For(0, number, count =>
{
    Console.WriteLine($"value of count = {count}, thread = {Thread.CurrentThread.ManagedThreadId}");
    //Sleep the loop for 10 miliseconds
    Thread.Sleep(10);
});
Console.ReadLine();
```

Explanation

Single thread, sequential order

```
C# For Loop
value of count = 0, thread = 1
value of count = 1, thread = 1
value of count = 2, thread = 1
value of count = 3, thread = 1
value of count = 4, thread = 1
value of count = 5, thread = 1
value of count = 6, thread = 1
value of count = 7, thread = 1
value of count = 8, thread = 1
value of count = 9, thread = 1
```

Multiple threads, NOT sequential

```
Parallel For Loop
value of count = 0, thread = 1
value of count = 1, thread = 1
value of count = 2, thread = 3
value of count = 4, thread = 5
value of count = 6, thread = 4
value of count = 8, thread = 6
value of count = 3, thread = 1
value of count = 7, thread = 3
value of count = 5, thread = 5
value of count = 9, thread = 4
```

Parallel For loop

- Standard C# for loop is iterated in sequential order whereas, in case of Parallel For loop, the order of the iteration is not going to be in sequential order.
  - ✓ Make sure your business is not depended on the sequential

Parallel foreach loop

- A sequential Foreach loop in C#:

```
List<int> integerList = Enumerable.Range(1, 10).ToList();
foreach (int i in integerList)
{
    //...
};
```

- A parallel Foreach loop in C#:

```
List<int> integerList = Enumerable.Range(1, 10).ToList();
Parallel.ForEach(integerList, i =>
{
    //...
});
```

Example

```
DateTime StartDateTime = DateTime.Now;
Console.WriteLine(@"Parallel foreach method start at : {0}", StartDateTime);
List<int> integerList = Enumerable.Range(1, 10).ToList();
var random = new Random();
Parallel.ForEach(integerList, i =>
{
    int total = random.Next(500, 1000);
    Thread.Sleep(total);
    Console.WriteLine("{0} - {1}", i, total);
});

DateTime EndDateTime = DateTime.Now;
Console.WriteLine(@"Parallel foreach method end at : {0}", EndDateTime);
TimeSpan span = EndDateTime - StartDateTime;
int ms = (int)span.TotalMilliseconds;
Console.WriteLine(@"Time Taken by Parallel foreach method in miliseconds {0}", ms);
Console.WriteLine("Press any key to exist");
Console.ReadLine();
```

Explanation

```
Parallel foreach method start at : 11/13/2020 3:27:14 PM
1 - 646
4 - 647
8 - 709
9 - 830
2 - 864
3 - 875
6 - 963
7 - 989
5 - 994
10 - 591
Parallel foreach method end at : 11/13/2020 3:27:15 PM
Time Taken by Parallel foreach method in miliseconds 1263
Press any key to exist
```

Practice time

Lesson Summary

Thank you