# Section 1: Basic Questions

## 1. What are the different types of loops in C#?

C# supports the following types of loops:  
- `for` loop: Used when the number of iterations is known beforehand.  
- `while` loop: Used when the condition needs to be checked before entering the loop.  
- `do-while` loop: Executes the loop body at least once before checking the condition.  
- `foreach` loop: Iterates through each element in a collection or array.

## 2. Explain the syntax and working of the for loop in C#.

The `for` loop in C# is used to execute a block of code a specified number of times. It has three parts:  
1. Initialization: Initializes the loop variable.  
2. Condition: Checks the condition before each iteration.  
3. Increment/Decrement: Updates the loop variable after each iteration.

Syntax:

for (initialization; condition; increment/decrement)  
{  
 // Code to execute  
}

Example:

for (int i = 0; i < 5; i++)  
{  
 Console.WriteLine($"Iteration: {i}");  
}

## 3. How does a while loop work?

A `while` loop executes a block of code as long as the specified condition is true.   
The condition is checked before entering the loop.

Syntax:

while (condition)  
{  
 // Code to execute  
}

Example:

int counter = 0;  
while (counter < 5)  
{  
 Console.WriteLine($"Counter: {counter}");  
 counter++;  
}

## 4. What is the difference between a while loop and a do-while loop?

| Feature | while loop | do-while loop |  
|---------------------|-------------------------------------------------|-------------------------------------------------|  
| Condition Check | Condition is checked before executing the loop. | Condition is checked after executing the loop. |  
| Execution | May not execute at all if the condition is false initially. | Executes at least once, even if the condition is false. |

Example of a do-while loop:

int counter = 0;  
do  
{  
 Console.WriteLine($"Counter: {counter}");  
 counter++;  
} while (counter < 5);

## 5. What happens if the loop condition in a while loop is initially false?

If the condition is initially false, the `while` loop will not execute its body even once.

Example:

int counter = 10;  
while (counter < 5)  
{  
 Console.WriteLine($"Counter: {counter}");  
}  
// Output: No output as the condition is false initially.

## 6. How do you use a foreach loop in C#?

The `foreach` loop is used to iterate through each element in a collection or array.

Syntax:

foreach (var item in collection)  
{  
 // Code to execute  
}

Example:

int[] numbers = { 1, 2, 3, 4, 5 };  
foreach (int number in numbers)  
{  
 Console.WriteLine(number);  
}

## 7. Can we modify elements inside a foreach loop? Why or why not?

No, you cannot modify elements inside a `foreach` loop because it operates on a read-only basis for the collection.   
If you try to modify the collection directly, it results in a compilation error.

Example (Invalid):

int[] numbers = { 1, 2, 3, 4, 5 };  
foreach (int number in numbers)  
{  
 number++; // Compilation error: Cannot modify 'number' because it is a foreach iteration variable.  
}

To modify elements, you should use a `for` loop or other methods that allow index-based manipulation.

Example (Valid Modification with for loop):

int[] numbers = { 1, 2, 3, 4, 5 };  
for (int i = 0; i < numbers.Length; i++)  
{  
 numbers[i] += 1; // Modifying elements  
}  
  
foreach (int number in numbers)  
{  
 Console.WriteLine(number); // Outputs: 2, 3, 4, 5, 6  
}

## Section 2: Intermediate Questions

### 8 .What is an infinite loop? Provide examples using for, while, and do-while.

An infinite loop is a loop that continues executing indefinitely because the termination condition is never met.  
Examples:  
   
For Loop:  
```  
for (;;) {   
Console.WriteLine("This is an infinite loop.");   
}  
  
  
While Loop:   
  
while (true) { Console.WriteLine("This is an infinite loop."); }  
  
Do-While Loop:   
  
do {   
Console.WriteLine("This is an infinite loop.");   
} while (true);

### 9. How does the break statement work inside loops?

The `break` statement immediately terminates the loop and transfers control to the statement following the loop.  
  
Example:  
```  
for (int i = 0; i < 10; i++)   
{  
 if (i == 5) break;   
 Console.WriteLine(i);  
}  
```

### 10. What is the role of the continue statement in loops?

The `continue` statement skips the current iteration and moves to the next iteration.  
  
Example:  
```  
for (int i = 0; i < 10; i++)   
{  
 if (i % 2 == 0) continue;   
 Console.WriteLine(i);  
}  
```

### 11. How can you exit multiple nested loops at once?

Using a labeled `break` statement:  
  
outerLoop:  
for (int i = 0; i < 3; i++)   
{  
 for (int j = 0; j < 3; j++) {  
 if (i == 1 && j == 1) break outerLoop;   
 Console.WriteLine($"i={i}, j={j}");  
 }  
}

### 12 .What is the difference between break and return inside a loop?

- `break` exits only the loop.  
- `return` exits the entire method.  
Example:  
void TestMethod()   
{  
 for (int i = 0; i < 10; i++) {  
 if (i == 5) return;   
 Console.WriteLine(i);  
 }  
}

### 13 .How do you iterate through an array using loops?

Using a `for` loop:  
  
int[] numbers = {1, 2, 3, 4, 5};  
for (int i = 0; i < numbers.Length; i++)   
{  
 Console.WriteLine(numbers[i]);  
}  
  
Using `foreach`:  
  
foreach (int num in numbers)   
{  
 Console.WriteLine(num);  
}

### 14 . Write a C# program to find the largest and smallest number in an array using loops.

Program to find the largest and smallest number in an array:  
  
int[] arr = { 3, 5, 7, 2, 8 };  
int min = arr[0], max = arr[0];  
  
foreach (int num in arr)   
{  
 if (num < min) min = num;  
 if (num > max) max = num;  
}  
Console.WriteLine($"Smallest: {min}, Largest: {max}");

## Section 3: Advanced Questions

### 15. Can a for loop run indefinitely? If yes, how?

Yes, a `for` loop can run indefinitely by omitting the conditions:  
  
for (;;) {  
 Console.WriteLine("Infinite loop");  
 }

### 16 How do you implement a loop using recursion instead of traditional looping constructs?

Recursion can replace loops in scenarios where iteration is needed:  
  
void RecursiveLoop(int n) {  
 if (n == 0) return;  
 Console.WriteLine(n);  
 RecursiveLoop(n - 1);  
}  
RecursiveLoop(5);

### 17 .What is the impact of using goto inside loops? Is it recommended? Explain with an example.

Using `goto` can disrupt structured programming and is generally discouraged.  
Example:   
int i = 0;  
start:  
if (i < 5) {  
 Console.WriteLine(i++);  
 goto start;  
}  
//Avoid `goto` for better readability.

### 18 .How does a foreach loop work internally in C#?

A `foreach` loop internally uses `IEnumerator` to iterate:  
IEnumerable<int> numbers = new List<int> { 1, 2, 3 };  
IEnumerator<int> enumerator = numbers.GetEnumerator();  
while (enumerator.MoveNext())   
{  
 Console.WriteLine(enumerator.Current);  
}  
```

### 19. Can a foreach loop be replaced with a for loop? If yes, in what cases?

Yes, a `foreach` loop can be replaced with a `for` loop when indexing is needed:  
  
string[] names = { "Alice", "Bob", "Charlie" };  
for (int i = 0; i < names.Length; i++)  
 Console.WriteLine(names[i]);

### 20 .How do you optimize performance in loops when working with large datasets?

Optimizing loops for large datasets:  
- Use \*\*Parallel.ForEach\*\* for multi-threading.  
- Avoid \*\*unnecessary computations\*\* inside loops.  
- Use \*\*LINQ\*\* for optimized queries.

### 21 .What are the best practices for writing efficient loops in C#?

Best practices for efficient loops in C#:  
- \*\*Minimize expensive operations\*\*.  
- \*\*Use preallocated lists\*\*.  
- \*\*Utilize break conditions early\*\*.  
- \*\*Avoid modifying collections inside loops\*\*.

### 22. How does the Parallel.ForEach loop differ from a normal foreach loop? Provide an example.

`Parallel.ForEach` runs tasks concurrently for improved performance:  
  
Parallel.ForEach(Enumerable.Range(1, 10), num => {  
 Console.WriteLine($"Processing {num} on thread {Thread.CurrentThread.ManagedThreadId}");  
});

## Bonus Challenge

### Write a C# program that processes a list of tasks using both normal foreach and Parallel.ForEach loops. Compare and explain the output. How does parallel processing improve performance in this scenario?

C# Program  
List<int> data = Enumerable.Range(1, 10).ToList();  
  
Console.WriteLine("Using normal foreach:");  
foreach (var item in data)  
 Console.WriteLine($"Processing {item}");  
  
Console.WriteLine("Using Parallel.ForEach:");  
Parallel.ForEach(data, item => {  
 Console.WriteLine($"Processing {item} on thread {Thread.CurrentThread.ManagedThreadId}");  
});  
```  
  
// Comparison and Performance Impact  
- \*\*Normal `foreach`\*\* runs sequentially.  
- \*\*`Parallel.ForEach`\*\* improves performance by distributing tasks across CPU cores.