Chapter 2

**2.1 Introduce the control statements:**

Control statements in C# are fundamental constructs that enable programmers to control the flow of execution in their programs. These statements allow you to make decisions, repeat actions, and perform different operations based on certain conditions. They are essential for creating dynamic and flexible programs that can respond to various scenarios.

**2.2 if, if-else, and if-else ladder:**

* **The if statement** is one of the most basic control statements in C#. It allows you to execute a block of code if a specified condition is true. Here's the syntax:

if (condition)

{

// Code to be executed if the condition is true

}

**For example:**

int age = 18;

if (age >= 18)

{

Console.WriteLine("You are an adult.");

}

In this example, the code inside the if block will be executed only if the condition age >= 18 evaluates to true.

* **The if-else statement** extends the functionality of the if statement by providing an alternative block of code to be executed when the condition is false. Here's the syntax:

if (condition)

{

// Code to be executed if the condition is true

}else

{

// Code to be executed if the condition is false

}

**For example:**

int age = 16;

if (age >= 18)

{

Console.WriteLine("You are an adult.");

}else

{

Console.WriteLine("You are a minor.");

}

In this case, if the condition age >= 18 is false, the code inside the else block will be executed instead.

* **The if-else ladder** is used when you have multiple conditions to be checked. It allows you to specify different blocks of code for different conditions. Here's the syntax:

if (condition1)

{

// Code to be executed if condition1 is true

}else if (condition2)

{

// Code to be executed if condition1 is false and condition2 is true

}else

{

// Code to be executed if all conditions are false

}

**For example:**

int score = 85;if (score >= 90)

{

Console.WriteLine("Excellent!");

}else if (score >= 80)

{

Console.WriteLine("Good!");

}else if (score >= 70)

{

Console.WriteLine("Average.");

}else

{

Console.WriteLine("Needs improvement.");

}

In this example, different messages will be displayed based on the value of the score variable.

**2.3 switch statement and its functions:**

The switch statement provides an alternative way to perform different actions based on the value of a variable or an expression. It simplifies code that would otherwise require multiple if-else statements. The switch statement consists of multiple case labels and an optional default label.

**Here's the syntax:**

switch (expression)

{

case value1:

// Code to be executed if expression matches value1

break;

case value2:

// Code to be executed if expression matches value2

break;

// Add more cases as needed

default:

// Code to be executed if expression doesn't match any case

break;

}

**For example:**

int day = 3;

switch (day)

{

case 1:

Console.WriteLine("Monday");

break;

case 2:

Console.WriteLine("Tuesday");

break;

case 3:

Console.WriteLine("Wednesday");

break;

// Add more cases for other days

default:

Console.WriteLine("Invalid day");

break;

}

In this example, the output will be "Wednesday" since the value of the day variable is 3.

**2.4 for loop and its usage:**

The for loop is used to execute a block of code repeatedly for a specified number of times. It provides a convenient way to iterate over a range of values or a collection. The for loop consists of three parts: initialization, condition, and iteration.

**Here's the syntax:**

for (initialization; condition; iteration)

{

// Code to be executed in each iteration

}

**For example:**

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Value of i: " + i);

}

In this example, the for loop will execute five times, printing the value of the variable i from 1 to 5.

The initialization part is executed only once at the beginning of the loop. The condition is checked before each iteration, and if it evaluates to true, the code inside the loop is executed. After each iteration, the iteration part is executed to update the loop variable. If the condition becomes false, the loop terminates.

**2.5 do-while loop and its usage:**

The do-while loop is similar to the while loop but with a slight difference in the execution order. It executes a block of code at least once before checking the condition. If the condition is true, the loop continues to execute.

**Here's the syntax:**

do

{

// Code to be executed

} while (condition);

**For example:**

int i = 1;

do{

Console.WriteLine("Value of i: " + i);

i++;

} while (i <= 5);

In this example, the code inside the do block will be executed first, and then the condition i <= 5 will be checked. If the condition is true, the loop will continue executing. If the condition is false, the loop will terminate.

The key difference between a do-while loop and a while loop is that the do-while loop guarantees at least one execution of the loop body, regardless of the condition.

**2.6 while loop and its usage:**

The while loop is used to execute a block of code repeatedly as long as a specified condition is true. It is suitable when the number of iterations is not known beforehand.

**Here's the syntax:**

while (condition)

{

// Code to be executed

}

**For example:**

int i = 1;

while (i <= 5)

{

Console.WriteLine("Value of i: " + i);

i++;

}

In this example, the code inside the while block will be executed only if the condition i <= 5 is true. If the condition is false initially, the code inside the while block will not execute.

The while loop checks the condition before each iteration. If the condition is true, the code inside the loop is executed. If the condition becomes false, the loop terminates.

**2.7 Loop Control Statements and their features:**

In addition to the basic control statements, C# provides loop control statements that allow you to modify the behavior of loops:

* **break statement:** It is used to terminate the innermost loop in which it appears and transfer control to the statement immediately following the loop. The break statement is commonly used to exit a loop prematurely based on certain conditions.

**For example:**

for (int i = 1; i <= 10; i++)

{

if (i == 5)

break; // Exit the loop when i equals 5

Console.WriteLine("Value of i: " + i);

}

In this example, the loop will terminate when the value of i becomes 5.

* **continue statement:** It is used to skip the remaining code inside the loop for the current iteration and proceed to the next iteration. The continue statement is commonly used to skip specific iterations based on certain conditions.

**For example:**

for (int i = 1; i <= 5; i++)

{

if (i == 3)

continue; // Skip the current iteration when i equals 3

Console.WriteLine("Value of i: " + i);

}

In this example, the value of i will not be printed when it is equal to 3.

* **goto statement:** It is used to transfer control to a labeled statement within the same method. The goto statement allows you to jump to a specific section of code based on certain conditions.

For example:

int age = 16;

if (age < 18)

goto underage; // Jump to the 'underage' label

else

Console.WriteLine("You are an adult.");

underage:

Console.WriteLine("You are underage.");

In this example, the program will jump to the label "underage" if the age is less than 18.

It's important to note that the use of the goto statement should be limited and used with caution, as it can make code harder to read and understand.

These loop control statements provide additional flexibility and control over the flow of execution within loops. Understanding and using them effectively can help you create more efficient and flexible programs.