Chapter2 : Control statements

Control statements, also known as flow control or branching statements, are fundamental constructs in programming languages that allow developers to control the flow of execution in a program. They enable you to make decisions, execute code conditionally, and repeat code blocks based on certain conditions. Control statements are essential for implementing logic, handling different scenarios, and creating dynamic, responsive programs.

Control statements are powerful tools for creating flexible and interactive programs. They help in designing algorithms, handling user input, controlling the order of operations, and responding to changing conditions during runtime. Proper use of control statements is crucial for writing efficient and maintainable code in any programming language

- **The primary types of control statements include:**

Conditional statements: Conditional statements, such as "if-else" and "switch" statements, allow you to execute specific blocks of code based on the evaluation of certain conditions. For example, you can execute one block of code if a condition is true and another block if it is false.

Loop statements: Loop statements, including "while," "for," and "do-while" loops, enable you to repeat a block of code multiple times. They continue executing the specified code until a certain condition becomes false, or a specific number of iterations is reached.

Jump statements: Jump statements, like "break," "continue," and "return," alter the normal flow of execution within a program. They allow you to transfer control from one part of the program to another, or return from a function.

Conditional Statement :

Conditional statements are programming constructs that allow developers to make decisions based on certain conditions. They enable the program to execute different blocks of code depending on whether a given condition is true or false. Conditional statements are essential for

implementing logic and branching in a program, making it possible to control the flow of execution dynamically.

The most common types of conditional statements are:

1. **If statement:** The "if" statement is the simplest form of a conditional statement. It evaluates a Boolean expression, and if the condition is true, it executes a specific block of code.

Here's the syntax:

```
if (condition)
{
   // Code to be executed if the condition is true
}
```

2. **If-else statement:** The "if-else" statement extends the "if" statement by providing an alternative block of code to execute when the condition is false.

Here's the syntax:

```
if (condition)
{
   // Code block to execute if the condition is true
}
else
{
   // Code block to execute if the condition is false
}
```

3. **else if**: The "else if" is a combination of two control keywords used in the "if-else" construct to handle multiple conditions. It allows you to test additional conditions after the initial "if" statement, providing a way to chain multiple condition checks together.

With the "else if" construct, the program will sequentially evaluate each condition until it finds one that is true. When a true condition is found, the corresponding code block is executed, and the rest of the "else if" and "else" blocks are skipped.

Syntax:

```
if (condition1)
{
   // Code block to execute if condition1 is true
}
else if (condition2)
{
   // Code block to execute if condition2 is true
}
else
{
   // Code block to execute if all conditions are false
}
```

- Nested if , else if ladder

Loop Statement :

The most common types of loop statements are:

1. **While Loop:** The "while" loop is a control flow statement that allows you to execute a block of code repeatedly as long as a specified condition is true. The condition is evaluated before each iteration, and if it is true, the loop body is executed. If the condition is false initially, the loop body may not execute at all.

The "while" loop is commonly used when the exact number of iterations is not known beforehand, and the loop continues until a specific condition becomes false. However, it is essential to ensure that the condition eventually becomes false; otherwise, you might end up with an infinite loop.. It is used for tasks like searching, filtering, and other repetitive operations that depend on certain conditions.

Syntax:

```
while(condition)
{
   // Code to be executed if the condition is true
}
```

When using the while loop, initialization should be done before the loop starts, and increment or decrement steps should be inside the loop.

```
int i = 0; // initialization
while (i < 10) // condition
{
   i++; // increment-decrement
}
```

2. **For Loop:** The "for" loop is a compact control statement used for executing a block of code for a fixed number of iterations. It combines the loop initialization, condition, and iterator (loop increment or decrement) in a single line.The "for" loop is commonly used when you know the exact number of iterations needed or when iterating over arrays, lists, or collections. "For" loops are efficient and readable when you need to repeat a specific block of code a specific number of times.

The for loop contains the following three optional sections, separated by a semicolon:

- **Initializer:** The initializer section is used to initialize a variable that will be local to a for loop and cannot be accessed outside loop.

- **Condition:** The condition is a boolean expression that will return either true or false. If an expression evaluates to true, then it will execute the loop again; otherwise, the loop is exited.

- **Iterator:** The iterator defines the incremental or decremental of the loop variable.

Syntax:

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

3. **Do While Loop:** The “do while loop” is the similar as while loop and It guarantees that the loop body is executed at least once, even if the condition is initially false.

Syntax:

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

Similarly, while using the do while loop, initialization should be done before the loop starts, and increment or decrement steps should be inside the do section. It is commonly used when you need user input validation or to repeat an action based on some condition while ensuring the block of code is executed at least once.

```
int i = 0; // initialization
do{
   i++; // increment-decrement
}
while (i < 10) // condition
```

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 behaviour 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.

- **Return Statement**: It is used to exit a method or function and return a value. The program execution will jump back to the calling code after the return statement is encountered.

✧ **Switch Statement:** The switch statement in C# is a powerful control flow statement that allows you to execute different blocks of code based on the value of an expression. It is often used as a cleaner and more efficient alternative to long chains of if-else statements.

The syntax of the switch statement is as follows:

```
switch (expression)
{
    case value1:
        // Code block to execute if the expression matches value1
        break;
    case value2:
        // Code block to execute if the expression matches value2
        break;
    // more case blocks can be added here...
    default:
        // Code block to execute if the expression doesn't match any case
        break;
```

```
}
```

The functions of different parts of the switch statement:

- expression: The expression is a value or variable whose value is compared against the different case values. It must evaluate to an integral type (e.g., int, char, enum) or a string.

- case value: Each case label represents a specific value that the expression is compared against. If the expression matches a case value, the corresponding code block will be executed.

- code block: When the expression matches a case value, the code block associated with that case label is executed. It can include one or more statements.

- break: The "break" statement is used to exit the switch statement once a matching case is found. Without the "break," the execution would "fall through" to the next case, executing subsequent code blocks as well.

- default: The "default" label is optional and is used when the expression does not match any of the case values. If there is no "default" label, and no matching case is found, the switch statement will do nothing.

Q. Write an if-else statement that checks if a variable number is even. If it is, print "The number is even". Otherwise, print "The number is odd".
Ans:

```
using System;

class Program
{
  static void Main(string[] args)
   {
      int number = 7; // You can replace this value with any other number for testing.

      if (number % 2 == 0)
      {
         Console.WriteLine("The number is even");
      }
      else
      {
         Console.WriteLine("The number is odd");
      }
   }
}
```

Q. Write a nested if-else statement that checks if a variable grade is greater than or equal to 90. If it is, print "A". If it is greater than or equal to 80, print "B". If it is greater than or equal to 70, print "C". If it is greater than or equal to 60, print "D". Otherwise, print "F".