

BICTE. 465

Unit 2: Control Statements in C#

Prepared By: Bhesh Raj Pokhrel

2. Control Statements

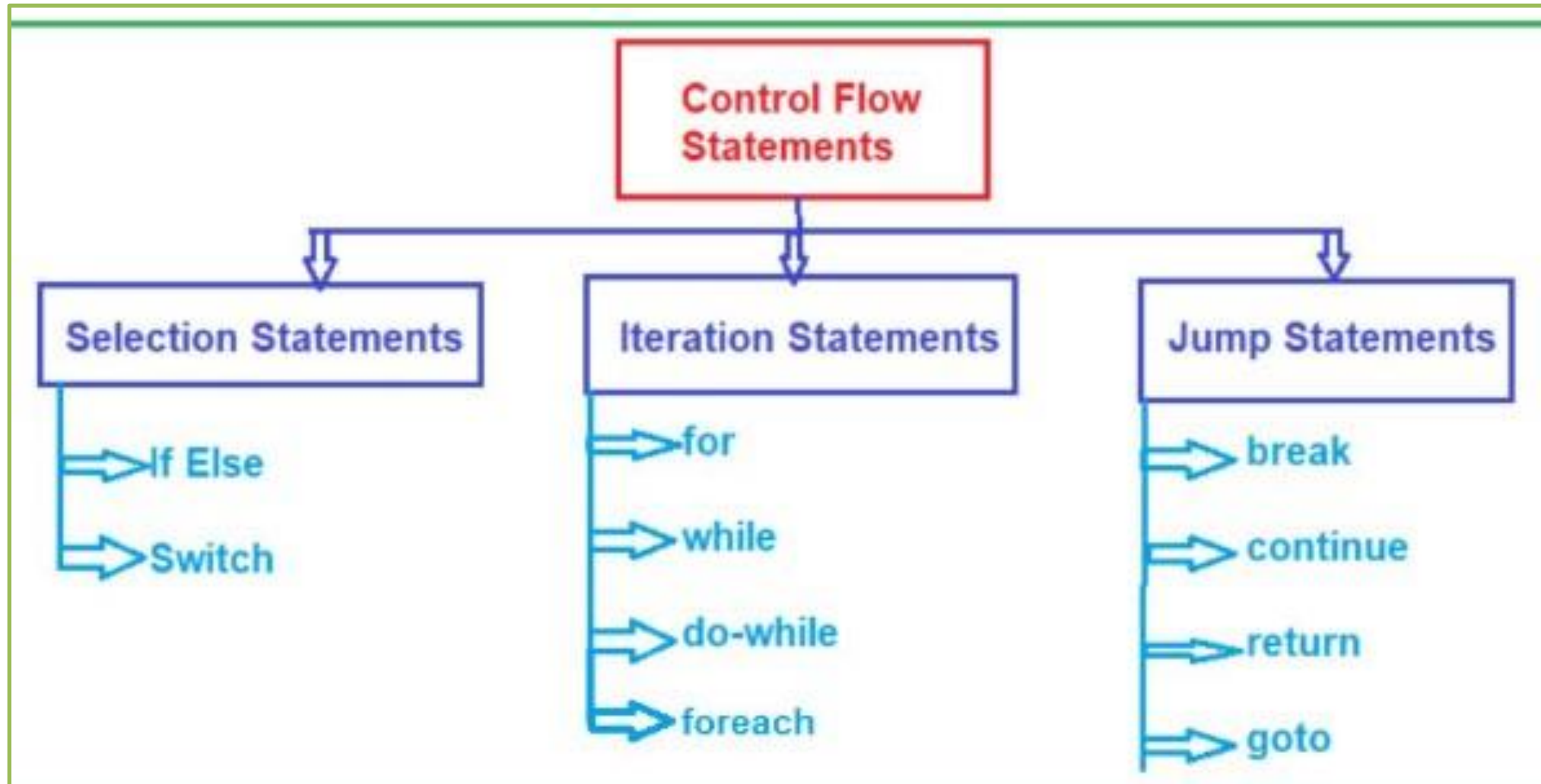
- By default, when we write statements in a program, the statements are going to be executed **sequentially from top to bottom** line by line.
- It is also possible in the programming language **to alter the execution** of the program. It means, instead of executing the statements sequentially one by one, we can change the order of execution. i.e. If we want, we can **skip** or **jump** or **repeatedly** execute some of the statements.
- **The Control Flow Statements** in C# are the statements that **Alter** the Flow of Program Execution.

Types of Control Flow Statements in C#:

- In C#, the control flow statements are divided into the following three categories:
 1. **Selection Statements or Branching Statements:** (Examples: if-else, switch case, nested if-else, if-else ladder)
 2. **Iteration Statements or Looping Statements:** (Examples: while loop, do-while loop, for-loop, and for-each loop)
 3. **Jumping Statements:** (Examples: break, continue, return, goto)

2. Control Statements

- Basically we can categorized the control statement as follows:



2.1. Selection or Branching Control Flow Statements

- It is also called as **Decision Making** Statements.
- The Selection or Branching or Decision-Making Statements in C# allow us to make a decision, **based upon** the **result** of a condition.
- If the **condition is satisfying**, we may need to execute some piece of code and if the condition is failed, we might need to execute some other piece of code.
- It executes different sections of code depending on a specific condition.

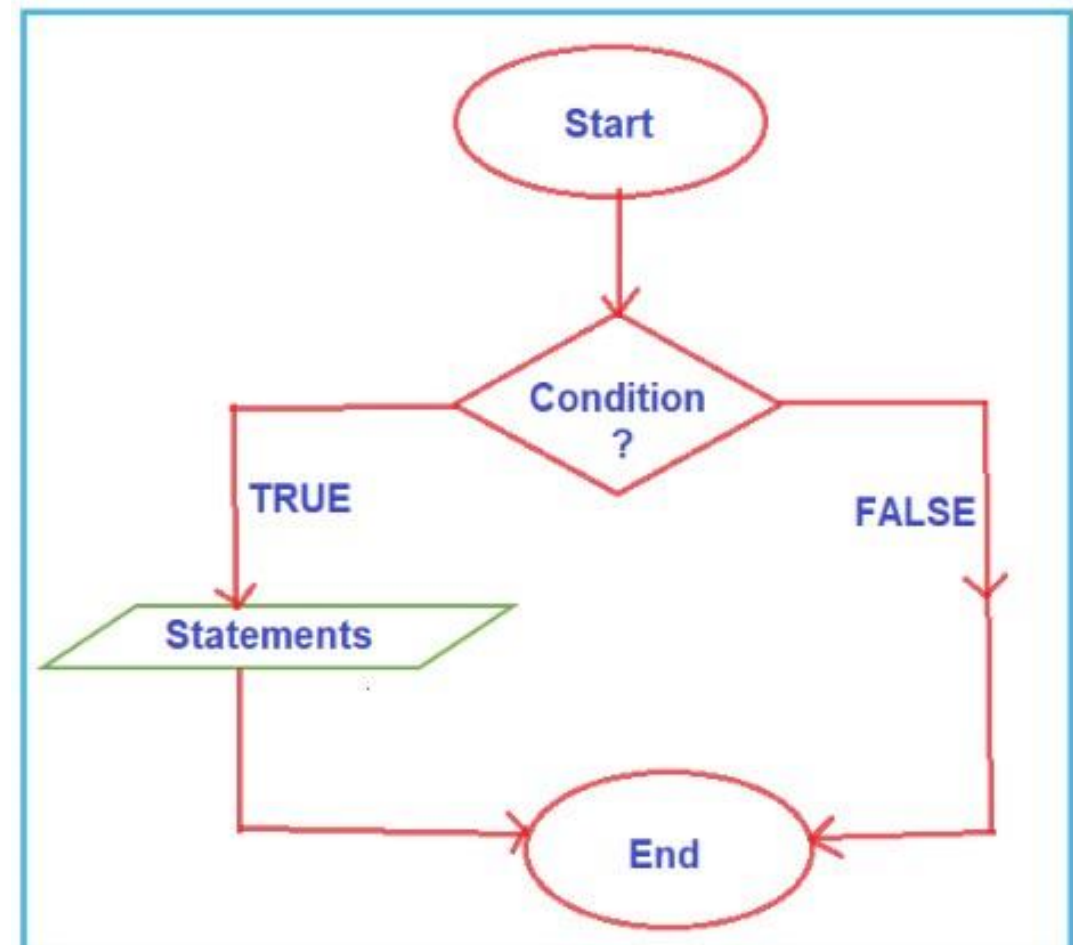
Selection statements can be divided into the following categories:

1. Simple **if** statement
2. **if .. else** statement
3. Nested **if .. else** statement
4. **else if** ladder
5. Switch Statements

2.1.1. If Statement in C#

- The **if** statement is a powerful decision-making statement and is used to control the flow of execution of statements.
- It **executes** a block of statements (one or more instructions) when the **condition in the if block is true** and when the condition is false, it **will skip** the execution of the if block.
- Using else block is **optional** in C#. Following is the syntax to use the if block in the C# language.

```
if(condition)
{
    Statements;
}
```



Example: If Statement

```
using System;

class ExampleIFStatement
{
    static void Main(string[] args)
    {
        int number;
        Console.WriteLine("Enter a Number: ");
        number = Convert.ToInt32(Console.ReadLine());
        if (number > 10)
        {
            Console.WriteLine(number+" is greater than 10 ");
            Console.WriteLine("End of if block");
        }
        Console.WriteLine("End of Main Method");
    }
}
```

```
Enter a Number: 20
20 is greater than 10
End of if block
End of Main Method
```

2.1.2. If and Else Block without Curly Braces in C#

- In the declaration of if block or else block if we **do not specify** statements using **curly braces {}**, then **only the first statement** will be considered as the if block or else block statement.

```
//Example: if else without curly braces
```

```
using System;
```

```
class IfWithoutCurlyBraces
```

```
{
```

```
    static void Main(string[] args)
```

```
{
```

```
    Console.Write("Enter a number: ");
```

```
    int number = Convert.ToInt32(Console.ReadLine());
```

```
    if (number == 10)
```

```
        Console.WriteLine("Hi"); //This Statement belongs to IF Block
```

```
    else
```

```
        Console.WriteLine("Hello"); //This Statement belongs to ELSE Block
```

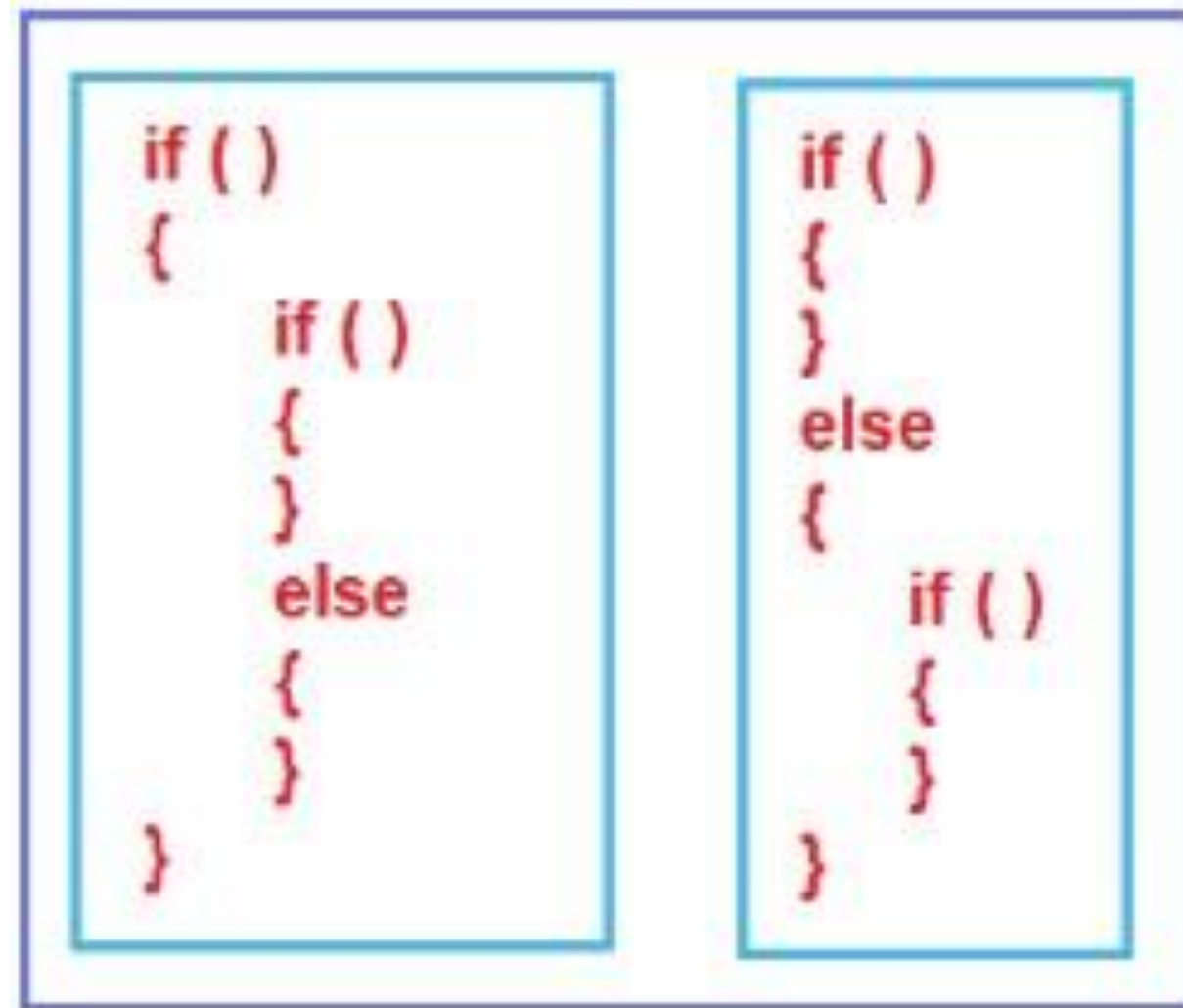
```
        Console.WriteLine("Bye");
```

```
    }
```

```
}
```

2.1.3. Nested If-Else Statements in C# :

- When an if-else statement is present **inside the body of another if or else** then this is called nested if-else.
- Nested IF-ELSE statements **are used** when we want to check for a condition only when the **previous dependent condition** is **true** or **false**.
- Depending on our logic requirements, we can **use nested if** block either inside the `if` block or inside the `else`.



Note: For every **if** condition statement, the **else block is optional**. But for every **else** block, the **if block is compulsory**. The purpose of the '**if**' statement in a program is to allow multiple execution paths for varying user inputs, making it more interactive!

e : Example

//Example : Nested If-Else statement

using System;

```
class NestedIfElseExample{
    static void Main(string[] args){
        Console.Write("Enter First Number :");
        int a=Convert.ToInt32(Console.ReadLine());
        Console.Write("Enter Second Number :");
        int b=Convert.ToInt32(Console.ReadLine());
        Console.Write("Enter Third Number :");
        int c=Convert.ToInt32(Console.ReadLine());

        int LargestNumber = 0;

        if (a > b){
            if (a > c)
                LargestNumber = a;
            else
                LargestNumber = c;
        }
        else{
            if (b > c)
                LargestNumber = b;
            else
                LargestNumber = c;
        }
        Console.WriteLine("The Largest Number is: "+LargestNumber);
    }
}
```

2.1.4. Ladder if-else statements in C#:

- In Ladder if-else statements **one of the statements** will be executed depending upon the **truth** or **false** of the conditions.
- If the **condition1** is **true** then **Statement 1** will be executed, and if **condition2** is **true** then **statement 2** will be executed, and so on.
- But if **all conditions are false**, then the last statement i.e. **else block** statement will be executed.
- The if-else statements are **executed** from **top to bottom**.
- As soon as **one** of the if conditions is true, the **statement associated** with that if block is going to be executed, and the **rest of the else-if ladder is bypassed**. If **none** of the conditions are true, then the final else statement will be executed.

```
if (condition)
    Statement 1;
else if (condition)
    Statement 2;

*
*
*
*
else
    Statement n;
```

//Example: Ladder If-Else

using System;

class LadderIfElseExample

{

static void Main(string[] args)

{

Console.Write("Enter Obtained Marks : ");

int m=Convert.ToInt32(Console.ReadLine());

if (m>=90)

{

Console.WriteLine("Grade : A");

}

else if (m>=80)

Console.WriteLine("Grade : B");

else if (m>=70)

{

Console.WriteLine("Grade : C");

}

else

Console.WriteLine("Grade : F");

}

}

2.1.5. Switch Statements in C#:

- In C#, the Switch statement is a **multi-way branch statement**. It provides an easy way to **transfer the execution** to different parts of a code **based on the value of the expression**.
- Switch case statements in C# are a **substitute for long if else statements** that compare a variable or expression to several values.
- When there are **several options** and we **have to choose only one option** from the available options depending on a single condition then we need to go for a switch statement.

Syntax of Switch Statements in C# Language:

- By using the **switch keyword** we can create **selection statements** with multiple blocks. And the Multiple blocks can be constructed by using the **case keyword**.
- The switch **expression** is of **integer type** such as **int, byte, or short**, or of an **enumeration** type, or of **character type**, or of **string type**.
- Data types like float, double, decimal, bool, object, or dynamic are **not supported**.
- The **expression is checked** for different **cases** and the match case will be executed.

```
switch(variable)
{
    case 1:
        //execute your code
    break;
    case n:
        //execute your code
    break;
    default:
        //execute your code
    break;
}
```

Example 1:

Points to remember:

- In C#, **duplicate** case values are **not allowed**. If we try we will get a **compilation error**.
- The **default block** in the switch statement is **optional**.
- We **need to use the break statement** inside the switch block to terminate the switch statement execution. The **break statement is mandatory**.
- **Nesting** of switch statements **is allowed**. However nested switch statements are **not recommended** by Microsoft.

```
//Example Switch-Case statement
```

```
using System;
```

```
class SwitchCaseExample
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        Console.Write("Enter a number :");
```

```
        int x = Convert.ToInt32(Console.ReadLine());
```

```
        switch (x)
```

```
        {
```

```
            case 1:
```

```
                Console.WriteLine("Choice is 1");
```

```
                break;
```

```
            case 2:
```

```
                Console.WriteLine("Choice is 2");
```

```
                break;
```

```
            case 3:
```

```
                Console.WriteLine("Choice is 3");
```

```
                break;
```

```
            default:
```

```
                Console.WriteLine("Choice other than 1, 2 and 3");
```

```
                break;
```

```
        }
```

```
    }
```

```
}
```

Example 2:

- After the end of each case block, it is **necessary to insert a break statement**.
- If we are not inserting the break statement, then we will get a **compilation error**. But we can **combine multiple case blocks** with a **single break statement** if and only if the previous case statement **does not have any code block**. For a better understanding, look at the above example.

```
//Example case block without break
using System;

class WithoutBreakExample{
    public static void Main(){
        Console.Write("Enter a vowel: ");
        char ch=Convert.ToChar(Console.Read());
        //char ch=Console.ReadKey().KeyChar;
        Console.WriteLine(); // Move to next line
        switch (ch)
        {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
                Console.WriteLine(ch+" is vowel.");
                break;
            default:
                Console.WriteLine(ch+" is consonant.");
                break;
        }
    }
}
```

2.1.6. goto case or default label :

- In C#, the goto case statement allows the program to **jump directly to another case label** within the same switch statement.
- It's useful when two or more cases need to **share the same block of code** without repeating it.
- For example, we can write goto case 2; inside case 1 to continue execution from the code written under case 2.
- Similarly, we can also use goto default; to transfer control to the **default case**.

```
//goto case or default example
using System;

class GotoCaseExample{
    static void Main(){
        Console.Write("Enter a number (1-3): ");
        int number = Convert.ToInt32(Console.ReadLine());

        switch (number){
            case 1:
                Console.WriteLine("Case 1 executed.");
                // Jump to another case
                goto case 2;
            case 2:
                Console.WriteLine("Case 2 executed.");
                break;
            case 3:
                Console.WriteLine("Case 3 executed.");
                // Jump to default case
                goto default;
            default:
                Console.WriteLine("This is the default case.");
                break;
        }
        Console.WriteLine("Program finished.");
    }
}
```

2.1.7. Fall-through in Switch Statement

- In the **absence of the break** statement in a case block, if the control moves to the next case block without any problem, it is known as '**fall-through**'.
- Fall-through is **permitted** in **C, C++ and Java**.
- But **C#** does **not permit automatic** fall-through, if the case block contains executable code. However, it is **allowed** if the case block is **empty**.
- For instance,

```
switch (m)
{
    case 1:
        X = y;
    case 2:
        X = y + m;
    default:
        X = y - m;
}
```

is an error in C#.

- If we **want two consecutive** case blocks to be executed **continuously**, we have to force the process by using the **goto** statement.

2.1.7. Fall-through in Switch Statement

- For example:

```
Switch (m)
{
    case 1:
        X = y;
        goto case 2;
    case 2:
        x=y+m
        goto default;
    default:
        X = y-m;
        break;
}
```

The **goto** mechanism enables us to jump **backward** and **forward** between cases and therefore arrange labels arbitrarily.

2.1.7. Fall-through in Switch: Example

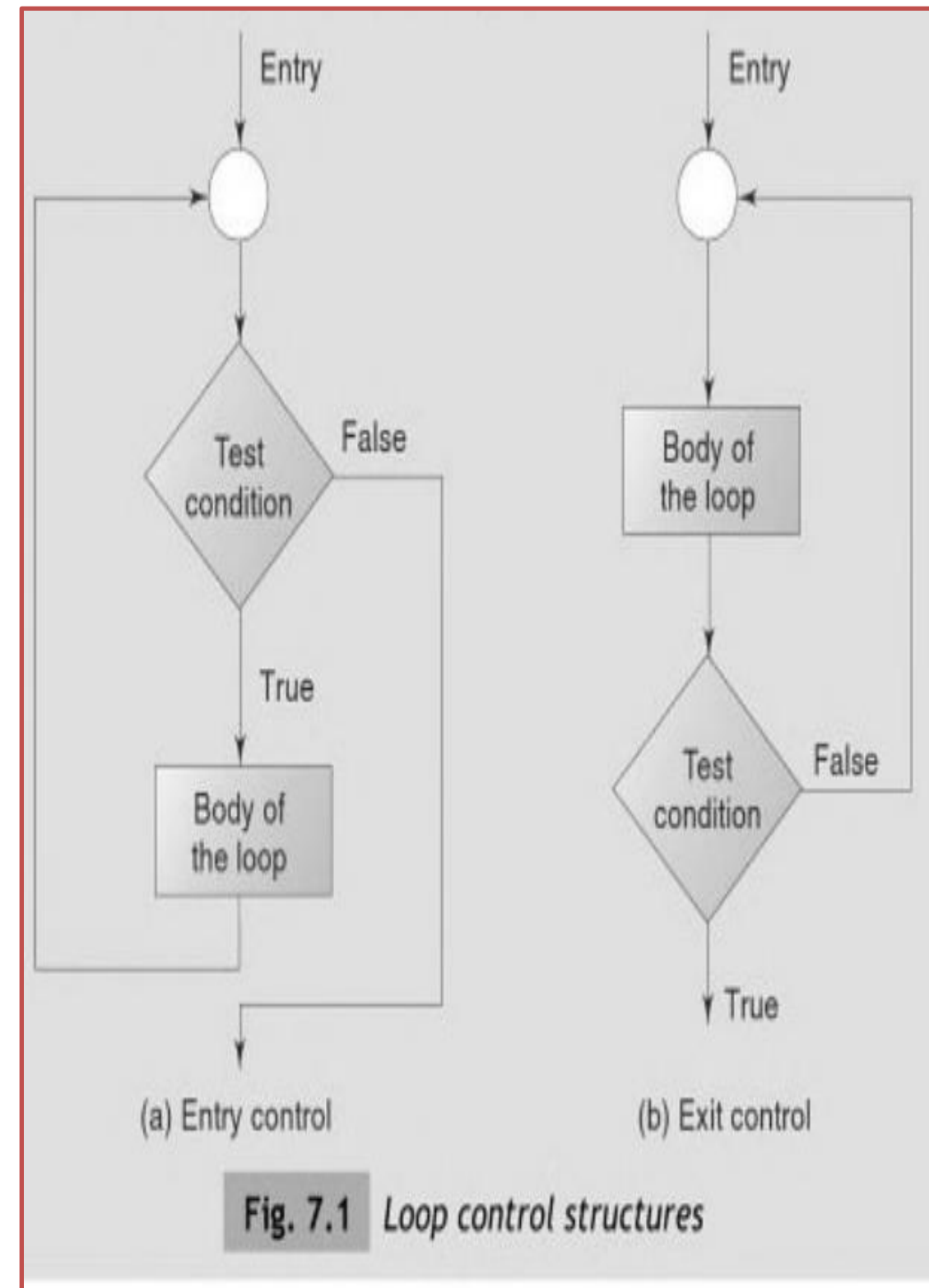
```
using System;
class SwitchFallthrough{
    public static void Main(){
        Console.WriteLine("Select your Cola Can:");
        Console.WriteLine("S=SMALL M=MEDIUM L=LARGE");
        Console.Write("Please enter your option: ");
        string s = Console.ReadLine();
        int cost = 0;
        switch(s)
        {
            case "S":
                cost +=10;
                break;
            case "M":
                cost +=15;
                goto case "S";
            case "L":
                cost +=30;
                goto case "S";
            default:
                Console.WriteLine("Invalid option. Select S, M, or L.");
                break;
        }
        if(cost!=0 )
            Console.WriteLine("Please pay {0} Rupees.", cost);
    }
}
```

2.1.7. Fall-through in Switch: Example Output

```
C:\BICTE\C#_Projects\controlStatements>SwitchFallthrough.exe  
Select your Cola Can:  
S=SMALL M=MEDIUM L=LARGE  
Please enter your option: M  
Please pay 25 Rupees.
```

3. Loop Statement:

- In looping, **a sequence of statements** are executed until **some conditions** for the termination of the loop are **satisfied**.
- A **loop statement** consists of two segments, one known as the **body of the loop** and the other known as the **control statement**.
- Depending on the **position of the control statement** in the loop, a control structure may be classified either as the **entry-controlled loop** or as the **exit-controlled loop**.
- In the **entry-controlled loop**, the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed.
- In the case of **an exit-controlled loop**, the **test is performed at the end of the body of the loop** and therefore the body is executed unconditionally for the first time.



3. Loop Statement:

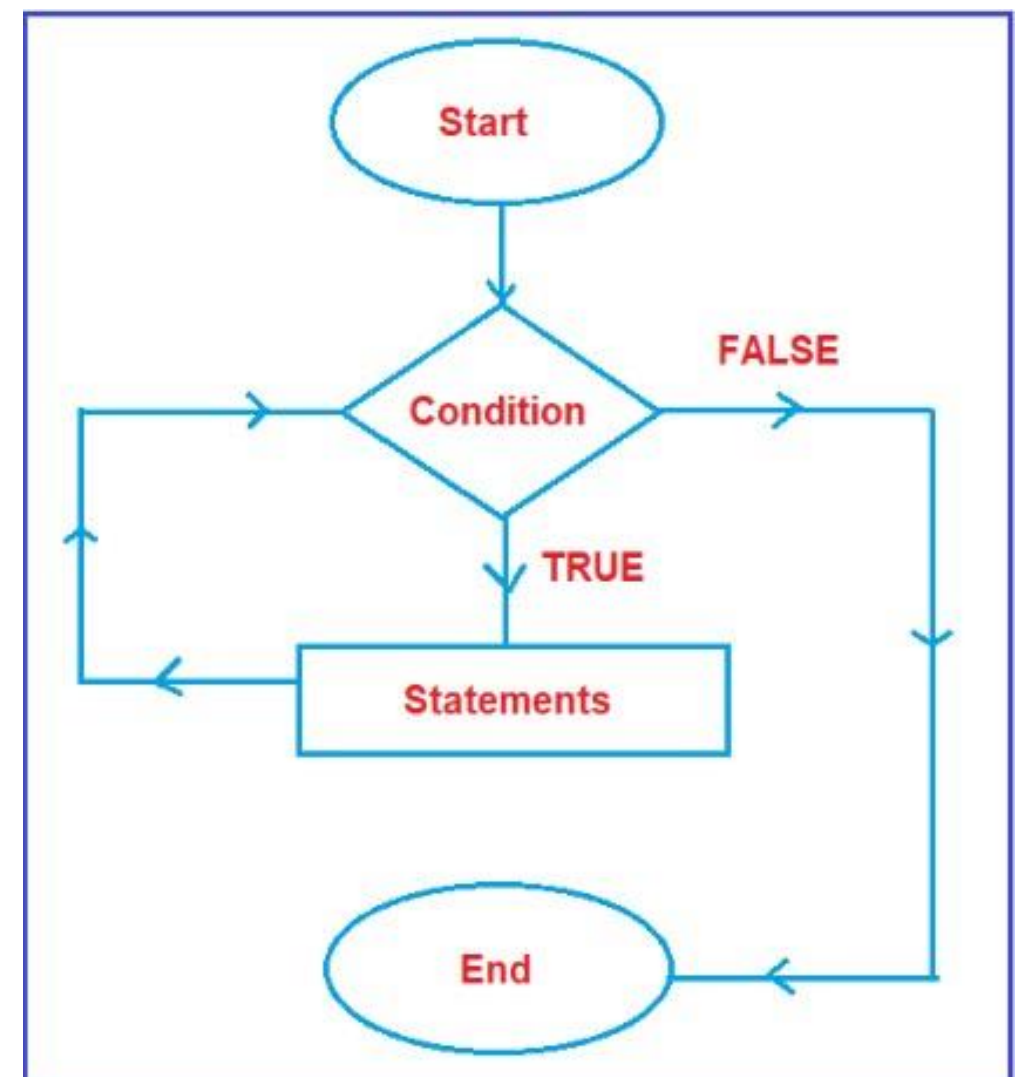
- A looping process, in general, would include the **following four steps**:
 - I. **Setting** and **initialization** of a counter.
 - II. **Execution** of the statements in the loop.
 - III. **Test** for a specified condition for execution of the loop.
 - IV. **Incrementing** the counter.
- The **test** may be either to determine whether the **loop has been repeated** the specified number of times or to determine whether a particular condition has been met with.
- The C# language provides four types of loop operations.
 - a) The **while** statement
 - b) The **do** statement
 - c) The **for** statement
 - d) The **foreach** statement
- These statements are known as ***iteration or looping*** statements.

3.1. The While Statement:

- The **simplest** of all the looping structures in C# is the **while** statement. The basic format of the **while** statement is
- **while** is an **entry-controlled loop** statement. The *test condition* is evaluated and if the condition is true, then the body of the loop is executed.
- After execution of the body, the test condition is **once again evaluated** and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop.
- The **body of the loop** may have **one or more statements**. The **braces** are **needed** only if the body contains two or more statements. However, it is a **good practice** to use braces even if the body has only one statement.

Syntax:

```
initialization;  
while(test condition)  
{  
    //Body of the loop  
}
```



3.1. While Loop Example

```
using System;

class WhileLoopTest
{
    static void Main(string[] args)
    {
        int x = 1; //Initialization
        while (x <= 5) //condition
        {
            Console.WriteLine("Value of x:" + x);
            x++; //increment
        }
    }
}
```

```
Value of x:1
Value of x:2
Value of x:3
Value of x:4
Value of x:5
```

3.1. While Loop Example

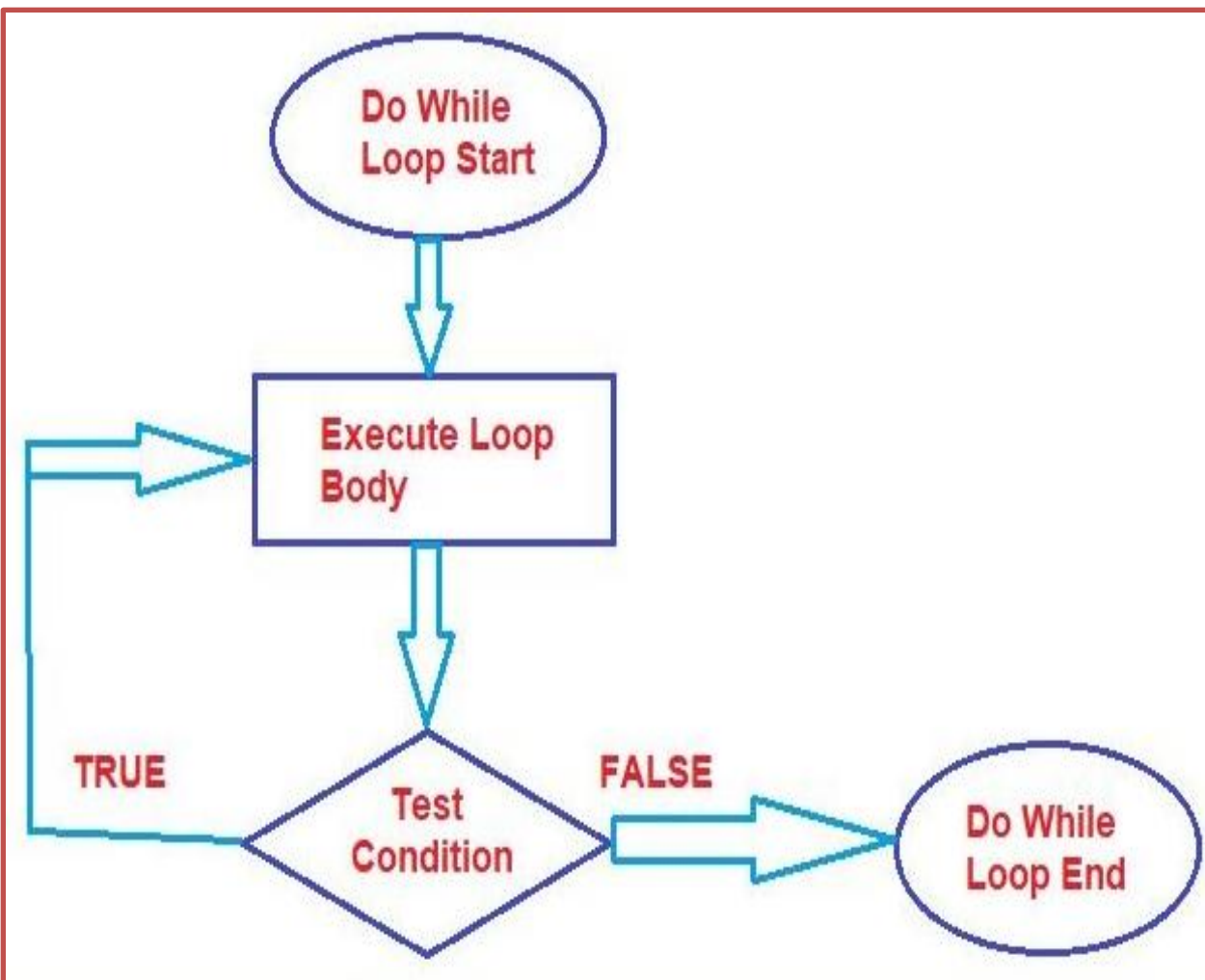
```
//While Loop Example
using System;
class WhileTest{
    public static void Main(){
        int n = 1;
        while (n<=10)
        {
            if (n%2==0){
                n++;
            }
            else{
                Console.Write(" "+n);
                n++;
            }
        }
    }
}
```

3.2. Do...While Statement:

- The do-while loop is a **post-tested loop** or **exit-controlled loop** i.e. first it will execute the loop body and then it will be going to test the condition.
- That means we need to use the do-while loop where we need to execute the loop body **at least once**.

Syntax:

```
initialization;  
do  
{  
    //Body of the loop  
}  
while (test condition);
```



```
using System;  
  
class DoWhileTest1  
{  
    static void Main(string[] args)  
    {  
        int number = 1; //Initialization  
        do  
        {  
            Console.WriteLine("number : "+number);  
            number++; //increment  
        } while (number < 5); //Condition  
    }  
}
```

3.2. Do-While Loop Example

```
//Do While Example
class DoWhileTest{
    public static void Main(){
        int row, column, y;
        row= 1;
        System.Console.WriteLine("Multiplication Table \n");
        do
        {
            column = 1;
            do
            {
                y = row*column;
                System.Console.Write(""+ y);
                column=column + 1;
            }
            while (column<=3);
            System.Console.WriteLine("\n");
            row=row+ 1;
        }
        while(row<=3);
    }
}
```

3.3. FOR Loop Statement

- **for** is another *entry-controlled loop* that provides a more concise loop-control structure. The general form of the for loop is

```
Syntax:  
for (initialization ; test  
    condition; increment)  
{  
    //Body of the loop  
}
```

The execution of the for statement is as follows:

1. **Initialization of the control variables** is done first, using assignment operator such as $i = 1$ and $\text{count} = 0$. The variables **i** and **count** are known as **loop-control variables**.
2. The value of the control variable is tested using the **test condition**. The test condition is a **relational expression**, such as $i < 10$, which determines when the loop will exit.
3. When the body of the loop is executed, the control is **transferred back to the for statement** after evaluating the last statement in the loop.
 - Now, the **control variable is incremented** and the new value of the control variable is again tested to check whether it satisfies the loop condition.
 - If the condition is satisfied, the body of the loop is again executed.
 - This process continues till the value of the control variable fails to satisfy the test condition.

3.4. Additional Features of the for Loop

- **More than one variable** can be initialized at a time in the for statement. Notice that the initialization section has two parts $p = 1$ and $n = 0$ separated by a *comma*.
- Like the initialization section, the **increment section may also** have more than one part.
- The third feature is that the test condition may have **any compound relation**.
- Another unique aspect of the **for** loop is that **one or more sections can be omitted**, if necessary. Consider the following statements:
- It is also permissible **to use expressions** in the **assignment** statements of the **initialization** and **increment sections**.

```
for (x=(m+n)/2; x>0; x=x/2)
{
//loop with expression on initialization
section
}
```

```
for (p=1, n=0; n<17; ++n)
{
//loop with two initial value
}
```

```
for (n=1, m=50; n<=m; n=n+1, m=m-1)
{
//loop with two incremental value
}
```

```
sum = 0;
for (i=1; i < 20 && sum < 100; ++i)
{
//loop with compound relation
}
```

```
m = 5;
for (;m!=100;)
{
//loop with two section omitted
    System.Console.WriteLine(m);
    m = m+5;
}
```

3.4. For Loop Example

```
//Additional feature of for loop
using System;
class ForLoopExatraFeature{
    public static void Main(){
        int num1=0, num2=10;
        for(;;((num1<=1)&& (num2>=0));num2--,num1++){
            Console.WriteLine("First Number {0} is {1} than Second Number {2}\t",
                               num1,(num1>num2? '>': '<'),num2);
        }
    }
}
```

```
C:\BICTE\C#_Projects\controlStatements>ForLoopExatraFeature.exe
First Number 0 is < than Second Number 10
First Number 1 is < than Second Number 9
```

3.5. THE FOREACH STATEMENT

- The for-each statement is similar to the for statement but implemented differently. It enables us to **iterate the elements** in **arrays and collection** classes such as List and HashTable. The general form of the for-each statement is:

```
foreach (type variable in expression)
{
    //Body of the loop
}
```

- The *type* and *variable* declare the **iteration variable**. During execution, the iteration variable **represents** the array **element** (or collection element in case of collections) for which an iteration is currently being performed. **in** is a **keyword** and the **expression** must be an **array** or **collection** type .
- The advantage of **foreach** over the **for** statement is that it automatically detects the boundaries of the collection being iterated over.

Example

```
//PRINTING ARRAY VALUES USING FOREACH STATEMENT
```

```
using System;  
class ForeachTest{  
    public static void Main(){  
        int[] arrayInt={11,22,33,44};  
        foreach (int m in arrayInt){  
            Console.Write("  "+ m);  
        }  
        Console.WriteLine();  
    }  
}
```

```
C:\BICTE\C#_Projects\controlStatements>ForeachTest.exe
```

```
11  22  33  44
```

3.7. Jumps In Loops

Jumping Out of a Loop

- An *early exit* from a loop can be accomplished by using the ***break*** and ***goto*** statements.
- We have already seen the use of the ***break*** in the switch statement. These statements can also be used within *while*, *do* or *for loops* for an **early exit**.
- When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

3.8. Skipping a Part of a Loop

Skipping a Part of a Loop

- During the loop operations, it may be necessary to **skip a part of the body** of the loop under certain conditions.
- Like the **break** statement, C# supports another similar statement called the **continue** statement.
- However, unlike **break** which causes the loop to be terminated, the **continue** statement, as the name implies, causes the loop to continue with the next iteration after skipping any statements in between.
- The continue statement tells the compiler. "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".

Example:

```
//USE OF CONTINUE AND BREAK STATEMENTS
```

```
using System;
class ContinueBreak{
    public static void Main(){
        int n = 10;
        while (n<200)
        {
            if(n<50)
            {
                Console. Write(" " + n);
                n= n + 10;
                continue;
            }
        }
    }
}
```

```
        if (n==50)
        {
            Console.WriteLine();
            n= n + 10;
            continue;
        }
        if (n>90)
            break;
        Console.Write(" " +n);
        n=n+10;
    }
    Console.WriteLine();
}
}
```

```
C:\BICTE\C#_Projects\controlStatements>ContinueBreak.exe
10 20 30 40
60 70 80 90
```

3.9. Labeled Jumps

- We have seen that the **break** will enable us to **come out of only the nearest loop** and the **continue** will enable us to **restart the current loop**.
- If we want to **jump a set of nested loops** or to continue a loop that is outside the current one, we may have to use the concept **labeling** and the **goto** statement.
- A **label** is any valid C# variable name ending with a colon. We can use labels anywhere in a program and use the **goto** statement to transfer the control to the statement marked by the label.
- We can use a **goto** statement and a label to transfer control out of a nested loop as shown in the code below:

3.9. Labeled Jumps

```
for ( int i = 0; i < 10; i++)  
{  
    while ( x < 100 )  
    {  
        y=i*x;  
        if (y > 500)  
            goto out;  
    }  
}  
out:
```

- Here, the label **out** is outside the **for** loop and therefore the statement.

goto out;

causes the execution to break out of both the loops. Note that a simple **break** cannot achieve this.

3.9. Labeled Jumps Example

//USE OF GOTO AND LABEL STATEMENTS

```
using System;
class GotoLabel{
public static void Main(){
    for(int i=1;i<100;i++)
    {
        Console.WriteLine(" ");
        if(i>=10)
            break;
        for(int j=1;j<100;j++)
        {
            Console.Write("*");
            if(j==i)
                goto loop1;
        }
        loop1:continue;
    }
    Console.WriteLine("Termination by BREAK");
}
}
```

3.9. Labeled Jumps Example Output

```
C:\BICTE\C#_Projects\controlStatements>csc GotoLabel.cs
```

```
C:\BICTE\C#_Projects\controlStatements>GotoLabel.exe
```

```
*
**
***
****
*****
*****
*****
*****
*****
*****
Termination by BREAK
```

The End