C# Basics

1. **Keywords**
2. **Variables**
3. **Identifiers**
4. **Data Types**
5. **Coding Standards**
6. **Operators**
7. **Data Type Compatability**
8. **Control Structures**
9. **Methods**
10. **Strings**
11. **Array**

**------------------------------------------------------------------------------------------------------------------------------------------------------------**

Program:

A set of instructions that directs a computer's hardware to perform a task is called a program, or software program.

----------------------------------------------------------------------------------------------------------------------------------------------------

Programming:

The act of creating a program is known as programming.

----------------------------------------------------------------------------------------------------------------------------------------------------

Programming language:

To write programs, you need a programming language.

----------------------------------------------------------------------------------------------------------------------------------------------------

Algorithm

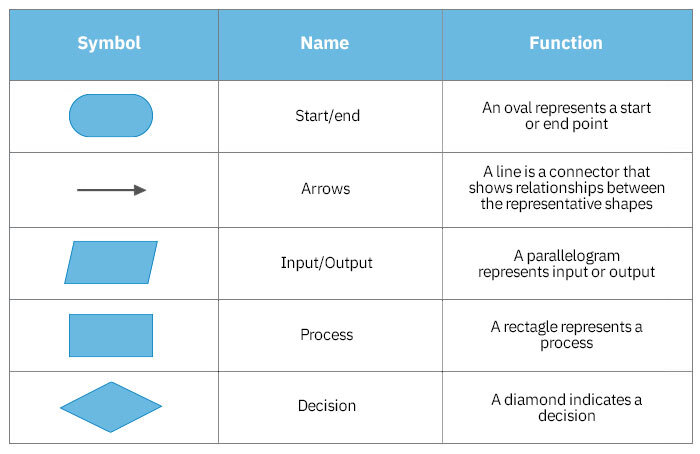
"An algorithm is a process or set of rules to be followed by the computer while solving a problem".

-----------------------------------------------------------------------------

There are primarily two ways of representing an algorithm:

**1)Pseudo-code:** Pseudo-code represents the algorithm in a way that it is in between a programming language and English statements

**2)Flowchart:** Flowchart represents the algorithm in a diagrammatic way



----------------------------------------------------------------------------------------------------------------------------------------------------

C# Introduction

--It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.

--C# has roots from the C family, and the language is close to other popular languages like C++ and Java.

--The first version was released in year 2002. The latest version, C# 10, was released in November 2021.

Syntax:

using System; //using System; means that we can use classes from the System namespace.

namespace HelloWorld

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

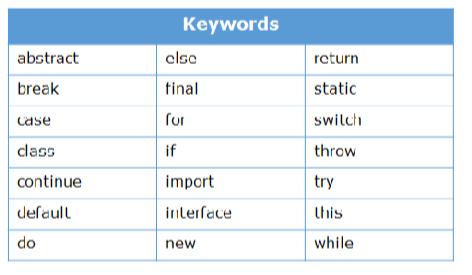
Namespace

A namespace is a container for classes.

The namespace also gives unique names to its classes so we can have the same class name in different namespaces.

--------------------------------------------------------------------------------------------------------------------------------------------------

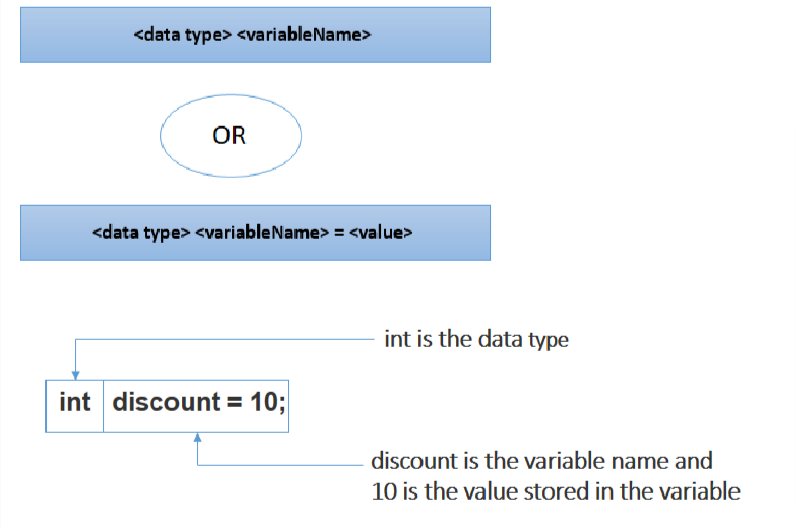
1.KEYWORDS  
Before proceeding with writing more programs, you need to know some more things about C  
sharp.  
Let's start with keywords. Every programming language has a set of keywords which are reserved  
words having a predefined meaning. Each keyword represents a specific functionality in the  
language.  
Some of the keywords in C sharp are listed below:



2.VARIABLES  
Programs are mainly written to implement a functionality. Every program needs data and a  
mechanism to store this data. Data is stored in variables. A variable is a named memory location  
which holds some value. The value stored in a variable can vary during the execution of the  
program.

Observe the below code to understand how data is stored in variables.  
In C#, it is necessary to declare a variable along with a data type before it can be used.  
static void Main(string[] args)  
{  
int discount = 10; // discount is a variable  
double totalPrice = 200; // totalPrice is a variable  
double priceAfterDiscount = totalPrice \* (1 - ((double)discount /  
100)); // priceAfterDiscount is a variable  
Console.WriteLine("Customer has paid a bill of amount: " +  
priceAfterDiscount);  
}

C Sharp(C#)  
  
**2.1 Declaring variables**  
You can declare variables in C sharp as follows:

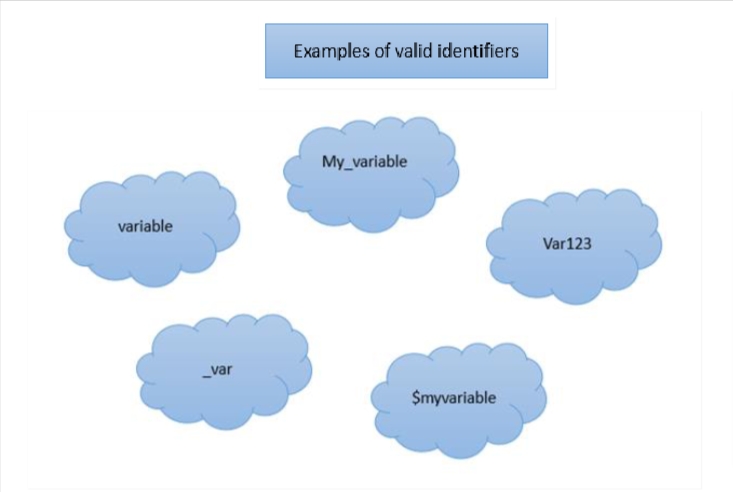


3.IDENTIFIERS **3.1 Why do we need identifiers?**

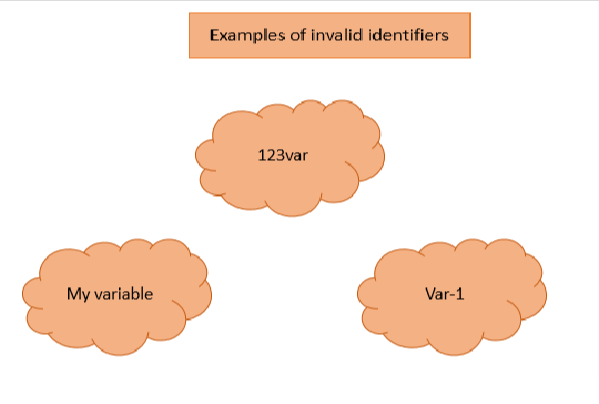


**3.2 What are identifiers?**  
Similarly, in C#, an identifier is the name given to a variable, method or class to uniquely identify  
it.  
In C# , following rules apply to the identifier name:  
• It can contain alphanumeric characters([a-z], [A-Z], [0-9]), dollar sign ($), underscore (\_)  
• It should not start with a digit ([0-9])  
• It should not have spaces  
• It should not be a C# keyword  
• It is case-sensitive  
• It has no length restrictions

**3.3 Valid Identifiers**



**3.4 Invalid Identifiers**

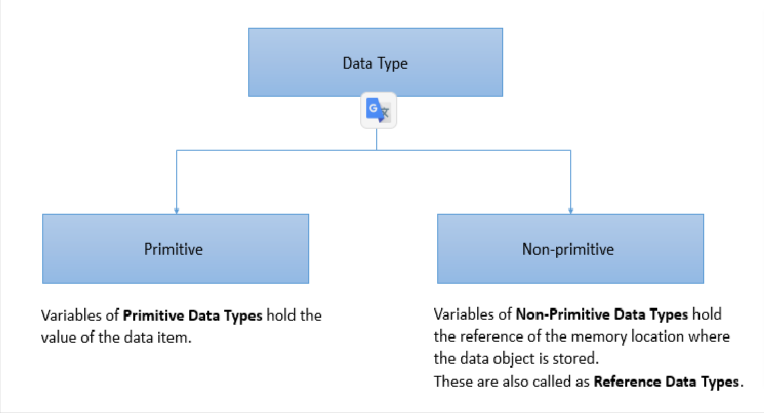


4.DATA TYPES

Data type defines the type of data that can be stored in the variable and the memory required by

it.

There are two types of data types in C#:



**4.1 Datatypes differ in size**

Consider the example of coffee shop. You might have ordered different types of coffee like

Caramel Frappuccino Mini, Caffe Latte Tall, Caffe Mocha Grande, etc. Just like how we have

different sizes of coffee cups like mini, tall, grande etc., in Java, primitive data types have different

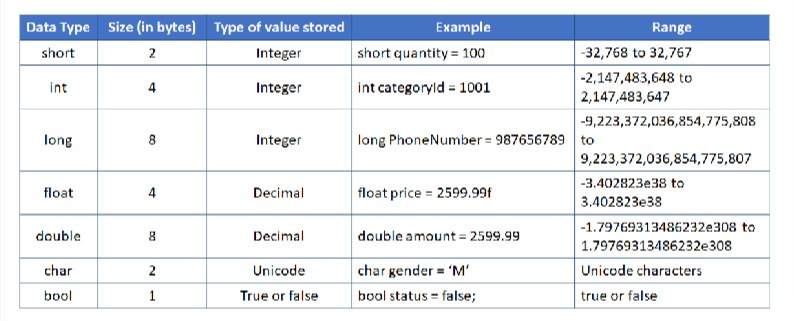
sizes.



**4.2 Primitive data types**

Primitive data types are the basic data types defined in C#. There are 7 primitive data types as

shown in the below table.



Important points about primitive data types

Here are some of the important points about primitive data types:

➢ Numeric and Boolean (true, false) values are written without quotes. E.g. int score = 85;

Boolean isQualified = true;

➢ The character value must be written in single quotes while assigning it to a character

variable. E.g. char gender = 'M';

➢ A long value is assigned to the variable, suffixed with L (uppercase letter or lower case

letter L can be used). E.g. long salary = 500000L;

➢ A float value must be suffixed with F or f while assigning to the variable. E.g. float average

= 78.6f;

➢ You will learn about different places in the program where the variables can be declared.

As of now, note that the default values are not applicable to the variables declared inside a

method. The variables declared inside a method must be initialized with a value before

printing their value or performing any operation on them. E.g.

After executing will get a error :Use of unassigned local variable ‘quantity’

**4.3 Non-primitive data types**

Non-primitive data types include classes, arrays, interfaces etc., C# provides the String data type

to store string literals. A variable of the string type can be declared and assign string literal.

Example

string name = “William Hawkings”;

Console.Writeline(name);

5.CODING STANDARDS

**5.1 Variable Names**

In C sharp, variable names should be nouns starting with lowercase letter. If it contains multiple words,

then every inner word must start with capital letter. This type of casing is called camel casing.

Some examples of variables are given below:

• mobileNumber

• name

class Demo

{

static void Main(string[] args)

{

int quantity;

float totalCost = 10 \* quantity; // error on this line

Console.WriteLine(totalCost);

}

}

C Sharp(C#)

• unitPrice

• paymentMode

• age

**5.2 Comments**

Comments are those statements which are not executed by the compiler. Comments can be used

to provide information about a variables or statement.

There are two types of comments in C sharp:

➢ Single line comment

It is used to comment only one line

Example:

➢ Multi-line comment

It is used to comment multiple lines of code

Example:

//This is a single line comment

static void Main (string[] args)

{

int age = 25; // Here age is a variable

Console.WriteLine(age);

}

//This is a Multi- line comment

static void Main (string[] args)

{ /\*

Given below is a variable age

and a print statement to print age

\*/ int age = 25;

Console.WriteLine(age);}

6.OPERATORS

**6.1 Scenario**

Observe the below code which contains various operations. Here, based on the total price, an

additional discount is offered. If the total price is greater than $500, extra 5% discount is

offered

static void Main(string[] args)

{

int discountPercentage = 10;

double totalPrice = 800;

double priceAfterDiscount = totalPrice \* (1 - ((double)discountPercentage

/ 100));

if (totalPrice > 500)

{

priceAfterDiscount = priceAfterDiscount \* (1 - ((double)5 / 100));

}

Console.WriteLine ("Customer has paid a bill of amount: " + priceAfterDiscount);

}

**6.2 Types**

Operators are the symbols used to perform specific operations. There are various operators that  
can be used for different purposes.

The operators are categorized as:

• Unary

• Arithmetic

• Relational

• Logical

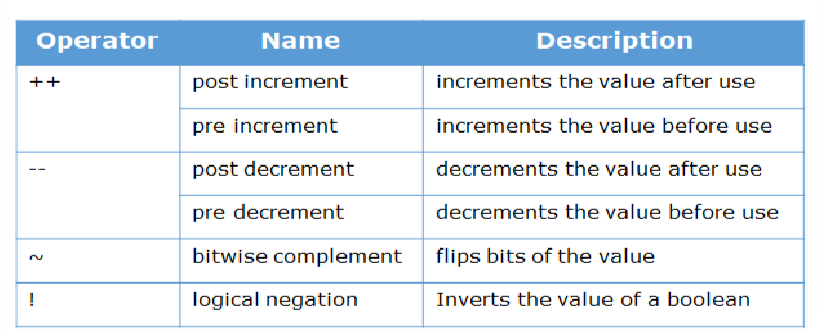
• Ternary

• Assignment

**6.2.1 Unary operators**

Unary operators act upon only one operand and perform operations such as increment,

decrement, negating an expression or inverting a Boolean value.



Example

**6.2.2 Arithmetic operators**

Arithmetic operators are used to perform basic mathematical operations like addition,

subtraction, multiplication and division.

static void Main(string[] args)

{

int numOne = 10;

int numTwo = 5;

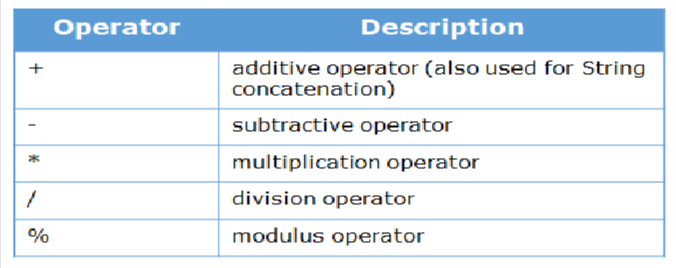
bool isTrue = true;

Console.WriteLine(numOne++ + " " + ++numOne); //Output will be 10 12

Console.WriteLine(numTwo-- + " " + --numTwo); //Output will be 5 3

Console.WriteLine(!isTrue + " " + ~numOne); //Output will be false -13

}



Example

static void Main(string args[])

{

int numOne = 10;

int numTwo = 5;

Console.WriteLine (numOne + numTwo); //Output will be 15

Console.WriteLine(numOne - numTwo); //Output will be 5

Console.WriteLine(numOne \* numTwo); //Output will be 50

Console.WriteLine(numOne / numTwo); //Output will be 2

Console.WriteLine(numOne % numTwo); //Output will be 0

}

static void Main(string[] args)

{

int numOne = 10;

int numTwo = 5;

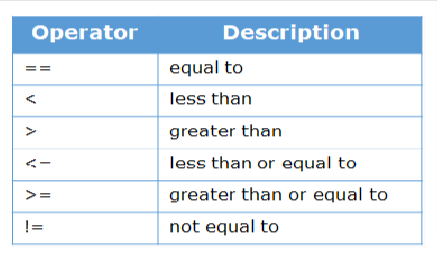
Console.WriteLine(numOne > numTwo); //Output will be true

}

**6.2.1 Relational operators**

Relational operators are used to compare two values. The result of all the relational operations is

either true or false.



Example:

**6.2.2 Logical operators**

Logical operators are used to combine two or more relational expressions or to negate the result

of a relational expression.

static void Main(string args[])

{

int numOne = 10;

int numTwo = 5;

Console.WriteLine (numOne + numTwo); //Output will be 15

Console.WriteLine(numOne - numTwo); //Output will be 5

Console.WriteLine(numOne \* numTwo); //Output will be 50

Console.WriteLine(numOne / numTwo); //Output will be 2

Console.WriteLine(numOne % numTwo); //Output will be 0

}

static void Main(string[] args)

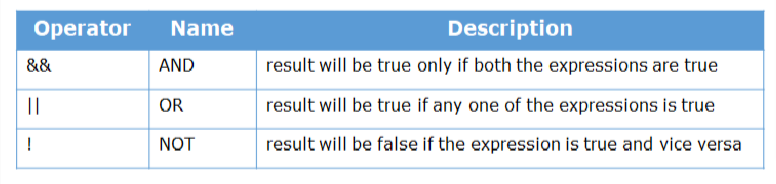
{

int numOne = 10;

int numTwo = 5;

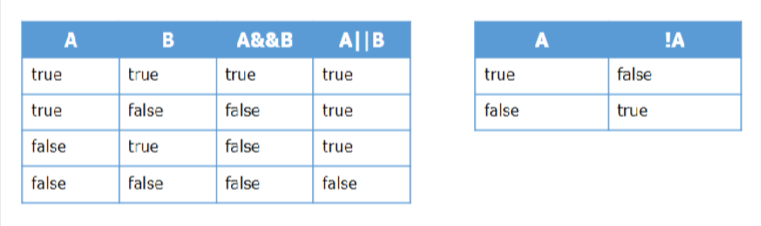
Console.WriteLine(numOne > numTwo); //Output will be true

}



Assume A and B to be two relational expressions. The below tables show the result for various

logical operators based on the value of expressions, A and B



**Example**

static void Main(string[] args)

{ int numOne = 100;

int numTwo = 20;

int numThree = 30;

Console.WriteLine (numOne > numTwo && numOne > numThree);

//Output will be true

}

**6.2.3 Ternary operator**

Ternary operator is used as a single line replacement for if-then-else statements and acts upon

three operands.

Syntax:

<condition> ? <value if condition is true> : < value if condition is false>

Example

public static void Main(string[] args)

{

int numOne = 10;

int numTwo = 5;

int min = (numOne < numTwo) ? numOne : numTwo;

Console.WriteLine (min); //Output will be 5

}

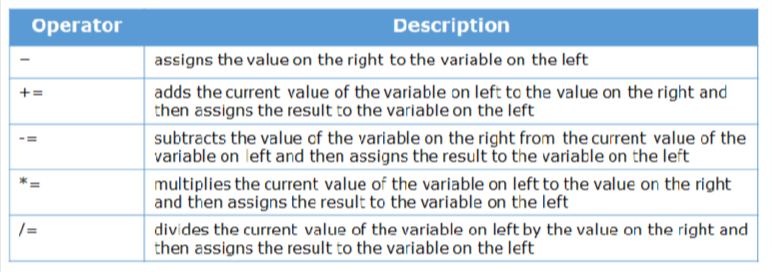
Here, first the condition (numOne < numTwo) is evaluated. The result is false and hence, min will

be assigned the value numTwo

**6.2.3 Assignment operator**

Assignment operator is used to assign the value on the right hand side to the variable on the left

hand side of the operator. Some of the assignment operators are given below:



Example

public static void Main(string[] args)

{ int numOne = 10; //The value 10 is assigned to numOne

Console.WriteLine (numOne); //Output will be 10

numOne += 5;

Console.WriteLine(numOne); //Output will be 15

numOne -= 5;

Console.WriteLine(numOne); //Output will be 10

numOne \*= 5;

Console.WriteLine(numOne); //Output will be 50

numOne /= 5;

Console.WriteLine(numOne); //Output will be 10

}

**6.2.4 Operator Precedence**

Operators also have precedence. The below table lists the operators according to the precedence

from highest to lowest. Operators with higher precedence are evaluated before the operators with

lower precedence. When an expression has two operators with the same precedence one after the

other, the expression is evaluated according to their associativity. The associativity of operators

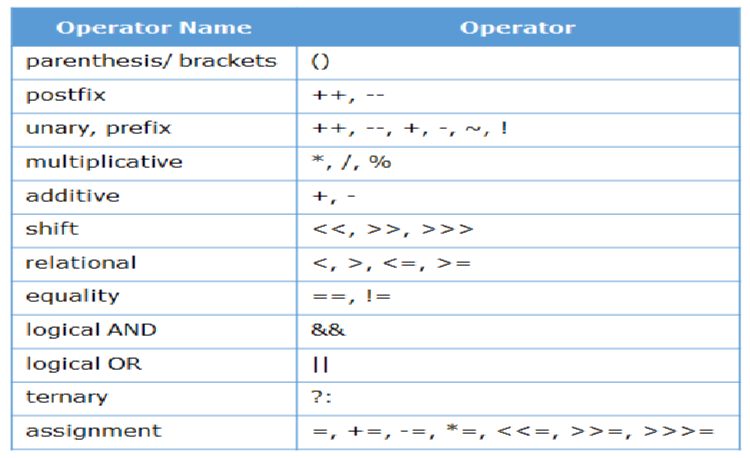
can be left-to-right, right-to-left or no associativity. For example: num1 = num2 = num3 = 39

is treated as (num1 = (num2 = (num3 = 39))), leaving all three variables with the value 39, since

the = operator has right-to-left associativity. On the other hand, 84 / 4 / 3 is treated as ((84 / 4) / 3)

since the / operator has left-to-right associativity. Some operators, like the postfix and prefix

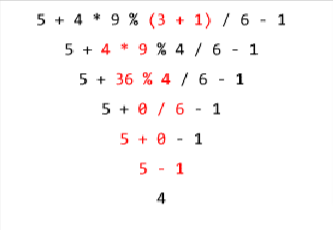
operators, are not associative: for example, the expression num++-- is invalid



**6.2.5 Evaluation of an Expression**

Have a look at the evaluation of below expression based on precedence and associativity of

operators:



7.DATA TYPE COMPATABILITY

By now, you know about keywords, variables, identifiers, operators and data types. You will now

understand the compatibility between data types. Consider that you have two vessels of different

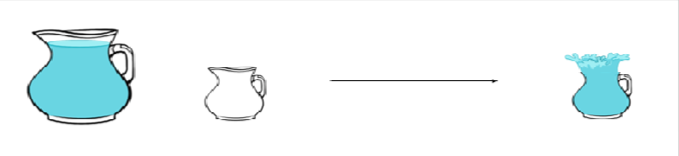
sizes. If you fill the small vessel completely with water and pour this water into the large vessel,

the water easily gets transferred.



If you try to do the same thing by pouring all the water from large vessel to smaller vessel,

the water overflows.



Similar to this, a variable will not be able to store a value if the value is more than the capacity of

the datatype of the variable.

Whenever any operation is performed containing different data types, the d ata type of result is

the largest data type in the expression.

For example, the result of (intVar + floatVar) will be a float value, the result of (intVar + longVar

+ doubleVar) will be a double value.

When you assign the value of one data type to another or when you perform operation on two

operands, their data types must be compatible with each other. If the data types are not compatible,

then the data type of an operand needs to be converted.

This conversion is of two types:

• Implicit

• Explicit

You have learnt various data types available in C# till now. Sometimes you may have to assign

value of one data type to another data type for calculations. Depending on the data type of the

source and destination variables, the data will be converted. This conversion can be either

narrowing conversion or widening conversion.

Execute the below code to understand narrowing and widening conversion of data types in C#.

static void Main(string[] args)

{

**/\* Implicit conversion from integer to double is called Widening conversion.**

Storing value of smaller datatype to variable of bigger datatype \*/

double doubleValue = 50;

Console.WriteLine("Integer to double implicit conversion: " + doubleValue);

/\*Uncomment the below statement and observe the following error:

Cannot implicitly convert type 'double' to 'int' \*/

//int intValue0 = 5.5;

**/\*Explicit conversion from double to int is called Narrowing conversion.**

Storing value of larger datatype to variable of smaller datatype.

Narrowing conversion will lead to data loss \*/

int intValue1 = (int)(5.5);

Console.WriteLine("Double to integer explicit type conversion: " + intValue1);

//Explicit conversion from double to int

int intValue2 = Convert.ToInt32(5.5);

Console.WriteLine("Double to integer explicit type conversion: " + intValue2);

string stringValue1 = "10";

/\* Uncomment the below statement and observe the following error:

Cannot implicitly convert 'str' to 'int' \*/

//int intValue3 = stringValue1;

/\* Uncomment the below statement and observe the following error:

Cannot convert 'str' to 'int' \*/

//int intValue4 = (int)(stringValue1);

/\* The above statement gives error because int and string are incompatible

datatypes\*/

int intValue5 = Convert.ToInt32(stringValue1);

Console.WriteLine("String to int explicit Conversion: " + intValue5);

}

8.CONTROL STRUCTURES

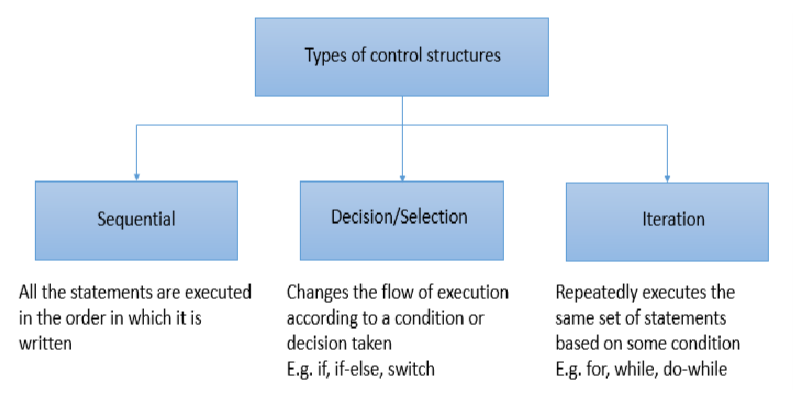
Next, you will have a look at control structures.

In a program, the instructions are usually executed line by line. Sometimes, all the statements in a

program may not be executed. There can be change in the flow of control and can be implemented

using control structures.

Remember that you have already learnt about control structures while discussing algorithms



Let us first start with selection control structure.

**8.1 IF STATEMENT**

Selection control structure is implemented using if statement.

An "if" statement contains a relational and logical expression followed by a block of statements.

Based on the result of the expression, the corresponding statements/code blocks get executed or

skipped.

Syntax:

if (<condition>) {

// Curly braces are not required if there is only one statement inside the block

<statements>;

}

The statements inside the "if" block gets executed only when the condition evaluates to true.

**8.1.1 IF -DEMO:**

While ordering food in "SwiftFood", when a customer orders food items, you need to calculate the

total cost that the customer should pay for the order. The "Regular Customers" are provided with

a 5% discount for their orders. Here, we are assuming that each food item costs $10. In the

condition, we are checking if the customer type is "Regular". If the condition is true, the statements

within the if block get executed to calculate total cost after discount.

public static void Main(string[] args)

{

String customerType = "Regular";

int quantity = 2;

int unitPrice = 10;

int totalCost = 0;

int discount = 5;

totalCost = unitPrice \* quantity;

if (customerType == "Regular")

{

totalCost = totalCost - (totalCost \* discount / 100);

Console.WriteLine ("You are a regular customer and eligible for 5%

discount");

}

Console.WriteLine("Total cost: " + totalCost);

}

**8.2 IF-ELSE STATEMENT**

An if statement can be written along with an else statement. The condition/expression given in the

if statement is checked and set of statements are executed based on the outcome of the condition.

If the condition is true, the statements written in if block get executed. If the condition is false,

then the statements inside else block get executed.

Syntax:

if (<condition>) {

<statements>;

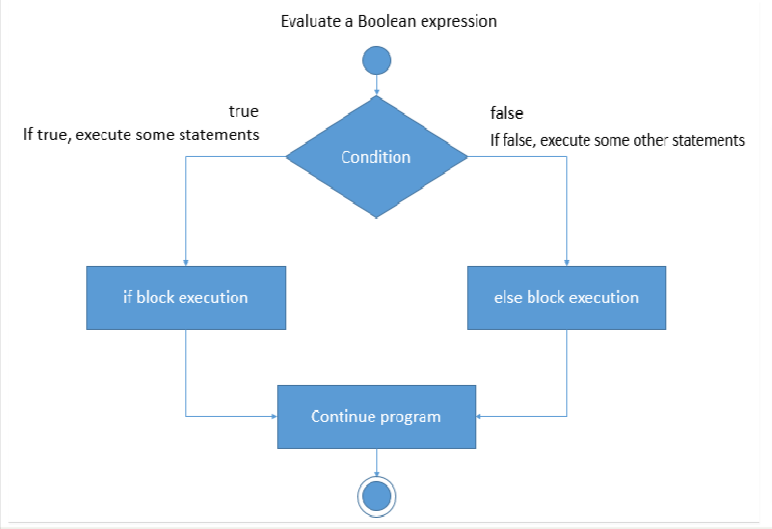
}

else {

<statements>;

}

**8.2.1 Flowchart**



**8.2.2. IF -ELSE DEMO**

In the below code, we are calculating the total cost that the customers should pay for the order. In

the previous example, we considered only Regular customers. The non-regular customers have to

pay an additional delivery charge of $5. So, first the customer type is checked. If the customer type

is Regular, then the if block gets executed. For other types of customers, the statements of

else block get executed.

public static void Main(string[] args)

{

String customerType = "Regular";

int quantity = 2;

int unitPrice = 10;

int totalCost = 0;

int discount = 5;

int deliveryCharge = 5;

totalCost = unitPrice \* quantity;

if (customerType == "Regular")

{

totalCost = totalCost - (totalCost \* discount /

100);

Console.WriteLine ("You are a regular customer and eligible for 5%

discount");

}

else

{

totalCost = totalCost + deliveryCharge;

Console.WriteLine("You need to pay an additional delivery charge of

$5");

}

Console.WriteLine ("Total cost: " + totalCost);

}

}

**8.3 IF- ELSE IF STATEMENT**

You can also have else if statements. As the name suggests, it is a combination of else and

if. Like else, it extends an if statement to execute a different set of statements in case the

original if expression evaluates to false. Then, the conditions present inside the else if blocks are

checked. Once a condition evaluates to true, remaining else if and else statements are skipped.

When all the conditions are false, the else block is executed. Coding the else block is optional.

Syntax:

if (<condition 1>) {

<statements>; }

else if (<condition 2>) {

<statements>; }

else if (<condition 3>) {

<statements>; }

else {

<statements>;}

**8.3.1 IF- ELSE IF DEMO**

When a customer places an order, the total cost varies according to the customer type. The

customer type must be either Regular or Guest. For other values, the program should display an

error message. This can be represented using if-else-if statements as shown below.

public static void Main(string[] args)

{

string customerType = "Regular";

int quantity = 3;

int unitPrice = 10;

int discount = 5;

int deliveryCharge = 5;

int totalCost = unitPrice \* quantity;

if (customerType == "Regular"){

totalCost = totalCost - (totalCost \* discount / 100);

Console.WriteLine("You are a regular customer and eligible for 5%

discount");

Console.WriteLine("The total cost to be paid is " + totalCost);}

else if (customerType == "Guest"){

totalCost = totalCost + deliveryCharge;

Console.WriteLine("You need to pay an additional delivery charge of $5");

Console.WriteLine("The total cost to be paid is" + totalCost);}

else // If there is only one statement inside a block, {} is optional

Console.WriteLine("Invalid customer type");

}

**8.4 NESTED IF STATEMENT**

When an if statement is written within another if statement, it is known as nested if statement. It

enables us to test multiple criteria. The inner if block condition gets executed only when the

condition of the outer if block evaluates to true.

Syntax:

if (<condition 1>) {

if (<condition 2>) {

<statements>;

}

else {

<statements>;

}

}

**8.4.1 Nested If Demo**

In the below code, the total cost for an order is calculated using nested if blocks. The assumption

over here is that each food item costs $10. The Regular customers are provided with a 5% discount

for their orders whereas the Guests need to pay an additional delivery charge of $5. First, the

customer type is checked. If the customer type is Regular, then the if block gets executed. If the

customer type is Guests, then else if block gets executed. Also, for regular customers, if the total

cost exceeds $20, a special gift voucher will be provided to the customers. This condition is

checked within the outer if block. If customer type is invalid, then the statements of else block

get executed.

public static void Main(string[] args)

{

String customerType = "Guest";

int quantity = 2;

int unitPrice = 10;

int totalCost = 0;

int discount = 5;

int deliveryCharge = 5;

totalCost = (unitPrice \* quantity);

if (customerType == "Regular")

{

totalCost = totalCost - (totalCost \* discount / 100);

Console.WriteLine("You are a regular customer and have

got 5% discount");

Console.WriteLine("The total cost to be paid is " +

totalCost);

if (totalCost >= 20)

{

Console.WriteLine("You have got a gift

voucher!!!!");

}

}

else if (customerType == "Guest")

{

totalCost = totalCost + deliveryCharge;

Console.WriteLine("You need to pay an additional

delivery charge of $5");

Console.WriteLine("The total cost to be paid is " +

totalCost);

}

else

{

Console.WriteLine("Invalid selection");

}

}

**8.5. SWITCH-CASE**

Similar to if else statements, switch statement is also a selection control structure.The

switch statement enables to select a block from a set of options. It allows the flow of execution to

be switched according to a value.

Syntax:

switch (expression or variable) {

case value1: <statements>;

break;

case value2: <statements>;

break;

default: <statements>;

break;}

During execution, the result of expression or variable written in the switch statement is compared

with the constant values of cases one by one. When a match is found, the set of statements present in

that case are executed until a break statement is encountered or till the end of switch block,

whichever occurs first. In the absence of break statement, the flow of control falls through subsequent

cases and executes the statements of all those cases until it reaches a break statement or end of switch

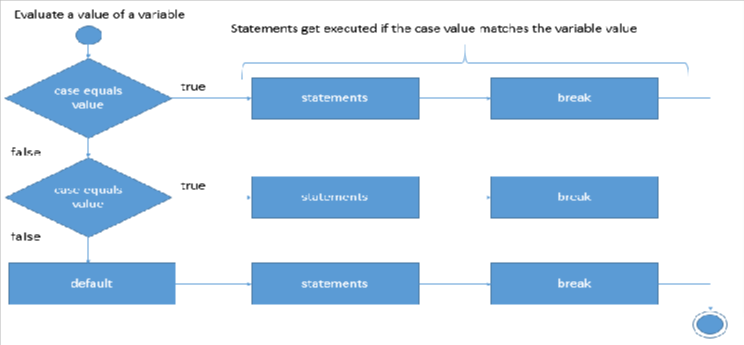
block.

The switch block can have a special case called default. The default case is executed when none of

the cases match with the value of expression/variable. default is optional. If none of the cases match

and if there is no default statement, the control comes out of switch block without executing any case.

**8.5.1 Flowchart**



**8.5.2 Switch Case Demo**

In the below example, the food item ordered gets displayed. If the ordered food is anything other than

Burger, Pizza and Sandwich, then a message, 'Invalid selection' is displayed from default case.

public static void Main(string[] args)

{

String orderedFood = "Pizza";

switch (orderedFood)

{

case "Burger":

Console.WriteLine("You have ordered Burger. Unit price:

$10");

break;

case "Pizza":

Console.WriteLine("You have ordered Pizza. Unit price:

$15");

break;

case "Sandwich":

Console.WriteLine("You have ordered Sandwich. Unit

price: $8");

break;

default:

Console.WriteLine("Invalid selection");

break;

}

}

You will now learn iteration control structures in detail.

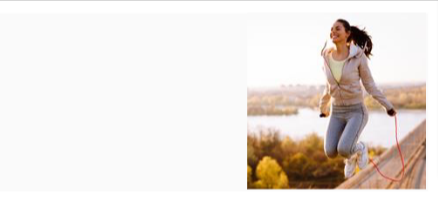
Iteration control structures are used to execute a set of statements repeatedly. To terminate the

repetition, a condition is required.

Let us consider that the girl in the image should jump over the rope repetitively until she is tired.

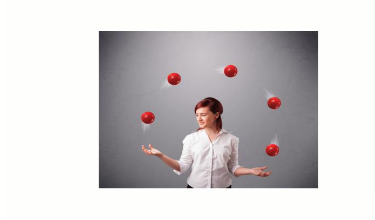
That means, a repeated action has to be performed as long as a condition (getting tired) is met.

Also, the number of times the girl is going to jump over the rope cannot be estimated before.



Consider another scenario. Here, the girl should keep juggling until she misses a catch. It is not known

at the beginning of juggling when she will miss a catch.



**8.6 WHILE**

Similarly, in programming, when you want to repeatedly execute the statements as long as a condition

is met, you can use the iteration control structure called while loop. When the condition becomes

false, the while loop terminates and control goes to the statement written after the while loop. The

while loop is used when the number of iterations are not known. In case of while loop, the condition

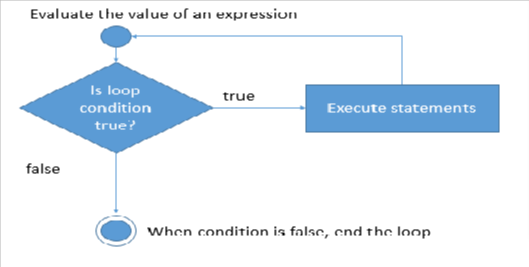
is tested before entering the while loop block and hence it is known as an entry-controlled loop.

Syntax:

while (<condition>) {

<statements>; }

**8.6.1 FLOWCHART**



**8.6.2 WHILE- DEMO**

As discussed before in pseudo-code, let us assume that an order should be placed for multiple food

items and that the number of food items to be ordered is not known at the beginning. If the user wants

to order an additional food item, the process of accepting the food item, quantity and calculation of

the total amount is repeated whenever the order is placed.

In the below code, wantToAddFoodItem is initialized to 'Y'. The condition of while loop evaluates

to true, so, the statements inside the loop get executed, i.e., the total cost of the order is calculated. The

customer is then asked if he/she wants to add one more food item. If the customer provides the input

as 'Y', then the condition evaluates to true and the statements inside the loop are executed again. If

the input is 'N', the flow of control goes out of the loop.

Execute the below code in Visual Studio and observe the output by providing the input.

Sample Output:

public static void Main(string[] args)

{

// Create a Scanner object

int totalCost = 0;

char wantToAddFoodItem = 'Y';

int unitPrice = 10;

int quantity = 1;

while (wantToAddFoodItem == 'Y')

{

totalCost = totalCost + (quantity \* unitPrice);

Console.WriteLine("Order placed successfully");

Console.WriteLine("Total cost: " + totalCost);

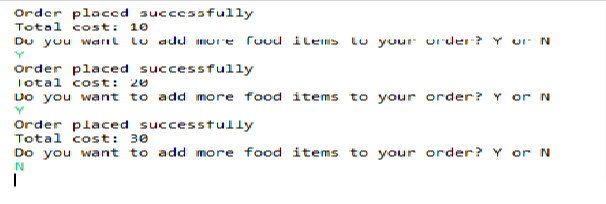
Console.WriteLine("Do you want to add more food items to your

order? Y or N");

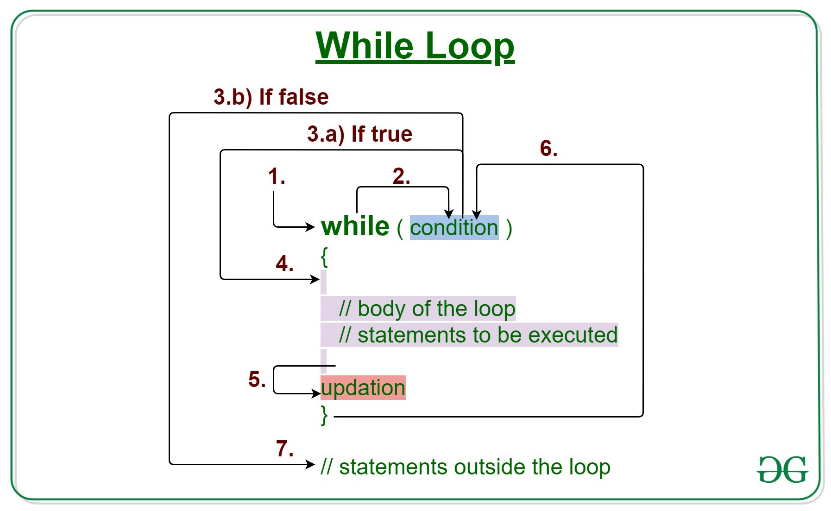
wantToAddFoodItem = Convert.ToChar(Console.ReadLine());

}

}



**8.7 DO-WHILE**



The next loop that you will see is do-while.

When the loop has to be executed at least once before the condition is checked, do-while loop is used.

After the first execution, the loop then gets repeated as long as the condition is true. In case of do-

while loop, the condition is tested after executing the code block. Hence, it is called an exit-controlled

loop.

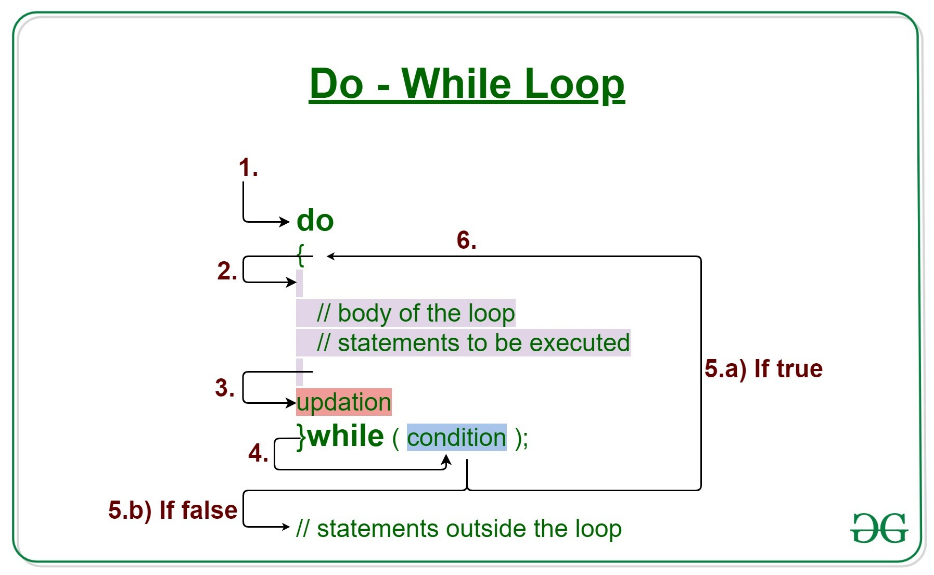
Syntax:

do {

<statements>;

} while (<condition>);

**8.7.1 DO-WHILE DEMO**



Observe the below code. The variable wantToAddFoodItem is initialized to 'N'. In this case, first the

code block is executed and then the condition is tested. Hence, the cost is calculated first and then the

condition is evaluated. When the condition is false, the loop terminates.

public static void Main(string[] args)

{

int totalCost = 0;

char wantToAddFoodItem = 'Y';

int unitPrice = 10;

int quantity = 1;

do

{

totalCost = totalCost + (quantity \* unitPrice);

Console.WriteLine("Order placed successfully");

Console.WriteLine("Total cost: " + totalCost);

Console.WriteLine("Do you want to add more food items to your order? Y

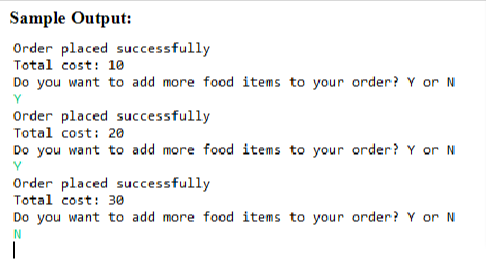
or N");

wantToAddFoodItem = Convert.ToChar(Console.ReadLine());

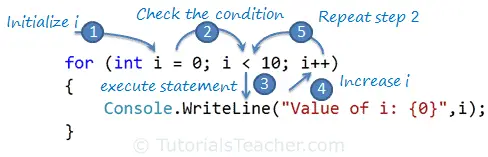
}

while (wantToAddFoodItem == 'Y');

}



**8.7 FOR LOOP**



Consider 800 meters running event on track. If the track is 400 meters, how many laps the

participants should take to complete the race? The participants will start the race from a start line

and should take two laps to finish the race. The two laps can be taken as two iterations. In this

case, the number of iterations are known before starting the race. When the number of iterations

are known, for loop can be used.

Syntax:

for (<initialization>; <condition>; <increment/decrement>)

{

<statements>;

}

Initialization: It is used for initializing the variables used for checking the condition. It is executed

only once and gets executed when the loop starts.

Condition: It is used for checking the condition to decide whether the loop should be terminated

or executed. If the condition is true, the body of the loop is executed, else the loop terminates.

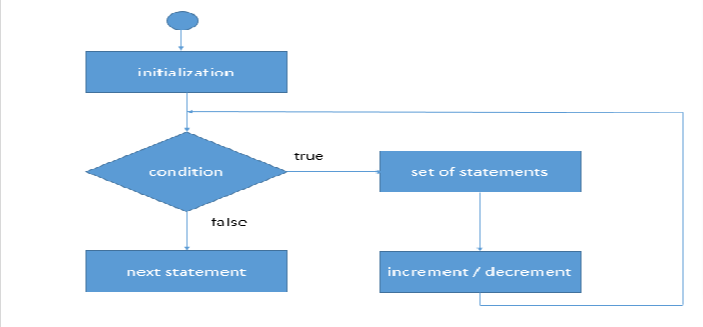
Increment / Decrement: It increments or decrements the value of the variable used for checking

the condition after every iteration of the loop.

All the three parts of for loop are optional. For instance, the for loop can also be written

as for(;;) where none of the parts are provided. This for loop will result in infinite loop.

**8.7.1 FLOWCHART**



**8.7.2 FOR LOOP DEMO**

Let us consider that the customer wants to place order for multiple food items and the number of food

items to be ordered is already decided. In the code given below, we are assuming that the customer

wants to order 3 food items.

public static void Main(string[] args)

{

int totalCost = 0;

int unitPrice = 10;

Console.WriteLine("Enter the number of food items to be

ordered");

int noFoodItemsToBeOrdered =

Convert.ToInt32(Console.ReadLine());

for (int counter = 1; counter <=

noFoodItemsToBeOrdered; counter++)

{

Console.WriteLine("Enter the food item");

String foodItem =

Convert.ToString(Console.ReadLine());

Console.WriteLine("Enter the quantity");

int quantity =

Convert.ToInt32(Console.ReadLine());

Console.WriteLine("You have ordered: " +

foodItem);

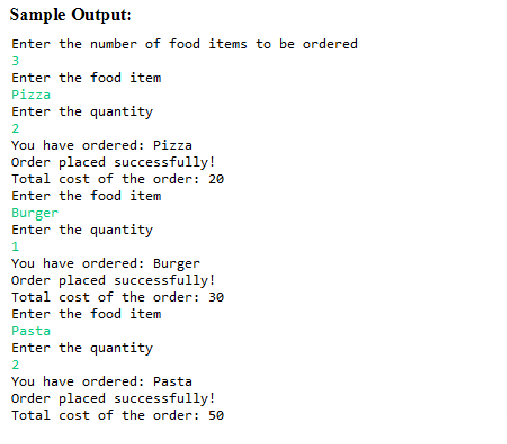
totalCost += unitPrice \* quantity;

Console.WriteLine("Order placed successfully!");

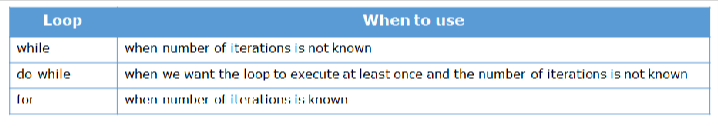
Console.WriteLine("Total cost of the order: " +

totalCost);

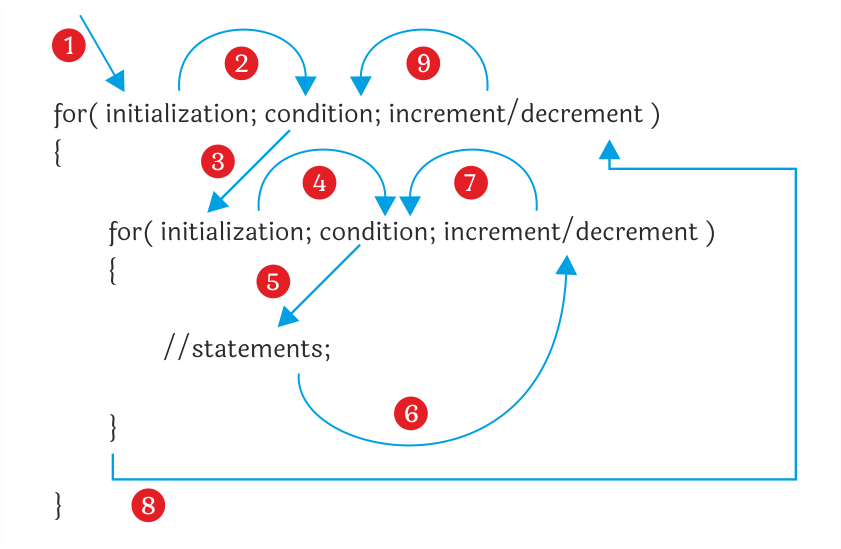
}}



**8.7.3 WHEN TO USE WHICH LOOP?**



**8.8 NESTED LOOP**



A nested loop is a loop within another loop, an inner loop within the body of an outer one.

Let us print the following pattern:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

1 2 3 4 5 6 7

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 10

We can create the above pattern using the nested loop

public static void Main(string[] args)

{

int i, j, k;

for (i = 1; i <= 7; i++)

{

for (j = 1; j <= i; ++j)

Console.Write(j);

for (k = 7 - i; k >= 1; k--) ;

Console.WriteLine("");

}

Console.ReadLine();

}

**8.9 BREAK**

break statement is used to terminate a loop. After terminating the loop, the next statement

following the loop gets executed. In case of break statement written in nested loops, the inner most

loop gets terminated and the flow of control continues with the statements of outer loop. break

statement is also used to terminate the execution of a switch case, as already discussed.

**8.9.1 BREAK DEMO**

public static void Main(string[] args)

{

int totalCost = 0;

int unitPrice = 10;

Console.WriteLine("Enter the max amount you can pay");

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

Console.WriteLine("Enter the number of food items to be ordered");

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

for (int counter = 1; counter <= noFoodItemsToBeOrdered; counter++)

{

Console.WriteLine("Enter the food item");

string foodItem = Convert.ToString(Console.ReadLine());

Console.WriteLine("Enter the quantity");

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

totalCost += unitPrice \* quantity;

if (totalCost > maxAmountCustomerCanPay)

{

Console.WriteLine("Sorry! Total cost is crossing your

max amount limit.");

break;

}

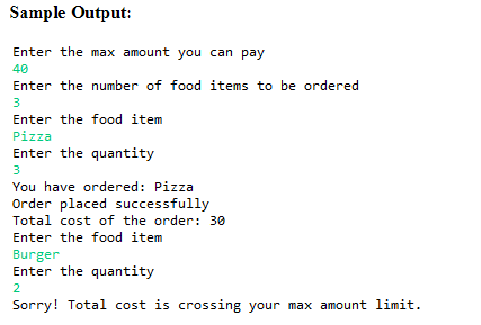
Console.WriteLine("You have ordered: " + foodItem);

Console.WriteLine("Order placed successfully");

Console.WriteLine("Total cost of the order: " + totalCost);

}

}



**8.10 CONTINUE**

continue statement is used to skip the current iteration of a loop and continue with the next

iteration. In case of while and do-while loops, continue statement skips the remaining code of the

loop and passes the control to check the loop condition. Whereas in case of for loop, the control

goes to the increment section and then the condition is checked.

**8.10.1 CONTINUE – DEMO**

In the below code, when the value of counter is 3, continue statement is encountered which skips

the current iteration and continues with the next iteration.

int totalCost = 0;

int unitPrice = 10;

Console.WriteLine("Enter the number of food items to be ordered");

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

for (int counter = 1; counter <= noFoodItemsToBeOrdered; counter++)

{

if (counter == 3)

continue;

Console.WriteLine("Enter the food item");

string foodItem = Convert.ToString(Console.ReadLine());

Console.WriteLine("Enter the quantity");

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

Console.WriteLine("You have ordered: " + foodItem);

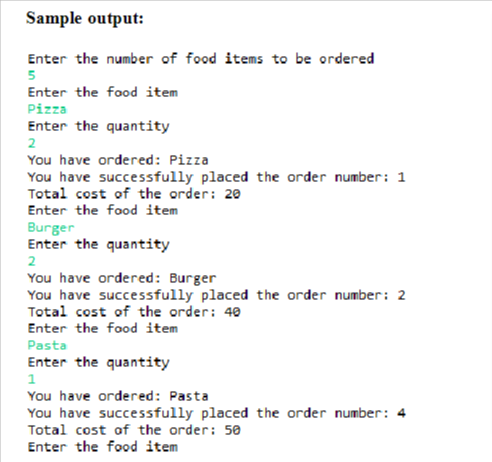
Console.WriteLine("You have successfully placed the order

number: " + counter);

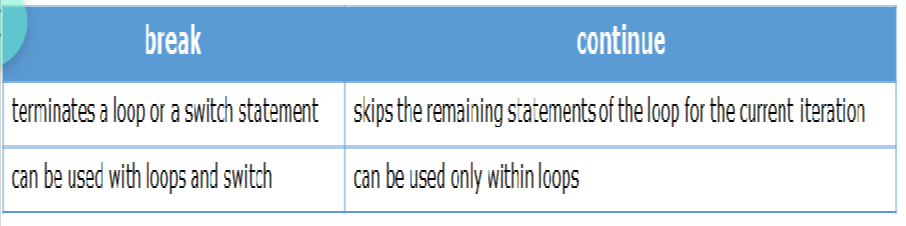
totalCost += unitPrice \* quantity;

Console.WriteLine("Total cost of the order: " + totalCost);

} }



**8.11 COMPARISON BETWEEN BREAK AND CONTINUE**



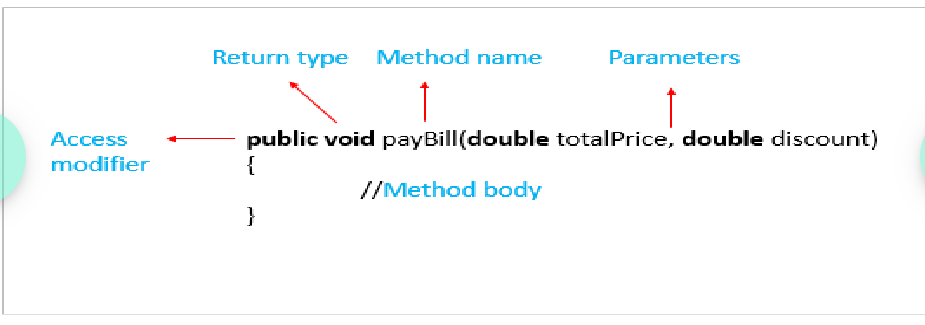
9. METHODS

You will now learn methods in detail.

A method is a set of statements which represents the behavior of a class. It is used to implement

a specific functionality. In Java, every method must be part of some class.

Syntax:



Access modifier – defines the access type of the method. You will learn more about this later in

the course.

Return type – the data type of the value returned by the method or void if nothing is returned

Method name – name of the method

Parameters – comma separated input values passed to the method, should be () if no parameters

are passed

Method body – the code that defines the functionality of the method

**9.1 METHOD – DEMO**

You have already seen how to create a method as shown below.

public String customerId;

public String customerName;

public long contactNumber;

public String address;

public void displayCustomerDetails()

{

Console.WriteLine("Displaying customer details \n\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("Customer Id : " + customerId);

Console.WriteLine("Customer Name : " + customerName);

Console.WriteLine("Contact Number : " + contactNumber);

Console.WriteLine("Address : " + address);

Console.WriteLine();

}

You have also seen how to invoke a method as shown below.

public static void Main(string[] args)

{

Class1 customer = new Class1();

customer.customerId = "C101";

customer.customerName = "Stephen Abram";

customer.contactNumber = 7856341287L;

customer.address = "D089, St. Louis Street, Springfield, 62729";

customer.displayCustomerDetails();

}

**9.2. PASSING PARAMETERS TO A METHOD**

You will next see how to pass parameters or arguments to a method.

A method can accept data as arguments or parameters. Observe the below payBill method which

accepts totalPrice and discountPercentage as parameters and calculates the final price to be paid.

public string customerName;

public string customerId { get; internal set; }

public long contactNumber { get; internal set; }

public string address { get; internal set; }

public double payBill(double totalPrice, double discountPercentage)

{

Console.WriteLine("Calculating final amount to be paid......");

double priceAfterDiscount = totalPrice \* (1 - (discountPercentage /

100));

Console.WriteLine("Hi " + customerName

+ ", your final bill amount after discount is: "

+ (int)(priceAfterDiscount \* 100) / 100.0);

}

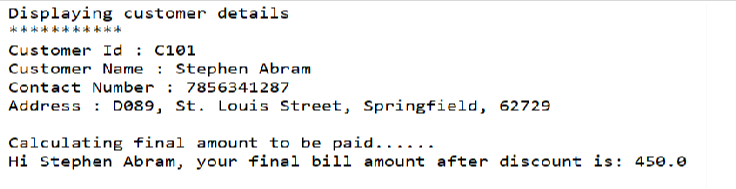
The values for the arguments need to be passed while calling the method is as shown below in the

code. The arguments passed while making the method call are known as actual parameters and

the arguments present in the method header are known as formal parameters.

public static void Main(string[] args)  
{  
CLASS1 customer = new CLASS1();  
customer.customerId = "C101";  
customer.customerName = "Stephen Abram";  
customer.contactNumber = 7856341287L;  
customer.address = "D089, St. Louis Street, Springfield, 62729";  
customer.payBill(500, 10);}

**Output:**

****

**9.3 RETURNING VALUES FROM METHOD**

The result of a method can be given back by means of a return value by using return statement as

shown in the below method. A method can return only one value at any point of time.

public double payBill(double totalPrice, double discountPercentage)

{

Console.WriteLine("Calculating final amount to be paid......");

double priceAfterDiscount = totalPrice \* (1 - (discountPercentage /

100));

return priceAfterDiscount;

}

The datatype of the return value must match the return type mentioned in the method header. If

the method does not return any value, void should be mentioned as the return type in

the method header.

**9.4 LOCAL VARIABLES**

You can also declare variables inside a method. These variables are known as local variables and

the scope of these variables is only within the method, i.e., they cannot be accessed outside the

method. Formal parameters are also local variables.

Observer the below payBill method.The variable discountPercentage and the parameter totalPrice

are local variables.

public double discountPercentage { get; internal set; }

public double payBill(double totalPrice, double discountPercentage)

{

discountPercentage = 10;

Console.WriteLine("Calculating final amount to be paid.....");

double priceAfterDiscount = totalPrice \* (1 - (discountPercentage / 100));

return priceAfterDiscount;

}

**9.5 PARAMETER PASSING TECHNIQUES**

You will now understand the different parameter passing techniques.

**9.5.1 PASS BY VALUE**

Whenever a value of a primitive data type is passed, the values are copied from the actual

parameters to the formal parameters. This kind of parameter passing is known as pass by value.

class CLASS1

{

public void changeValue(int value1, int value2)

{

value1 = value1 + value2;

value2 = value1 - value2;

Console.WriteLine(value1 + " " + value2);

}

}

Public static void Main(string[] args)

{

CLASS1 demo = new CLASS1();

int x = 20;

int y = 10;

Console.WriteLine(x + " " + y);

demo.changeValue(x, y);

Console.WriteLine("After calling changeValue function");

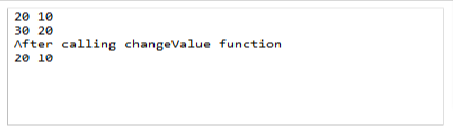
Console.WriteLine(x + " " + y);

}

In pass by value, both the actual and formal parameters point to different memory locations and

the values are copied in both the memory locations.

**Example**



You can see that the values are changed only inside the method. This is because any changes made inside

the method will be reflected only in the memory locations of the formal arguments and not in the memory

locations of the actual arguments.

**9.5.2 PASS BY REFERENCE**

When an object is passed as a parameter, the formal and the actual parameters both refer to the

same object and hence the same memory location. Therefore, the changes made inside the method

to the formal parameters are reflected in the actual parameters also. This kind of parameter passing

is known as pass by reference.

**Example:**

Observe the below code.

class CLASS1

{

public void changeValue(ref int value1, ref int value2)

{

value1 = value1 + value2;

value2 = value1 - value2;

}

}

public static void Main(string[] args)

{

CLASS1 demo = new CLASS1();

int x = 20;

int y = 10;

Console.WriteLine(x + " " + y);

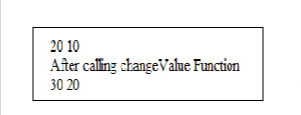
demo.changeValue(ref x, ref y);

Console.WriteLine("After calling changeValue function");

Console.WriteLine(x + " " + y);

}

**Output:**



10. STRINGS

Next, you will learn about Strings.

You have already seen that String is used to store a sequence of characters. String is a predefined

class in Java and comes with many methods which can help us perform various operations on it.

Example



There are many methods in strings. Some of them are explained below.

**10.1 String Length**

A string in C# is actually an object, which contain properties and methods that can perform certain

operations on strings. For example, the length of a string can be found with the Length property:

string name = “William Hawkings”;

Console.WriteLine(name.Length);

**10.2 String Concatenation**

The + operator can be used between strings to combine them. This is called concatenation

**Example**

public static void Main(string[] args)

{

string name1 = "William";

string name2 = "Hawkings";

string name3 = name1 + name2;

Console.WriteLine(name3);

}

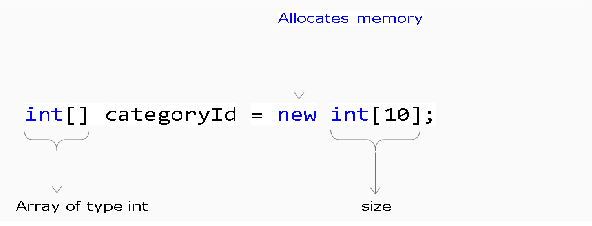
11.ARRAY

An array is a group of homogeneous data. The keyword 'new' is used to allocate memory space

based on size.

The size of an array can't be changed after declaration.

Following is the syntax to declare an array in C# programming language.



**Overview:**

➢ An array can be Single-Dimensional, Multidimensional or Jagged.

➢ The number of dimensions and the length of each dimension are established when the array

instance is created. These values can't be changed during the lifetime of the instance.

➢ The default values of numeric array elements are set to zero, and reference elements are set to

null.

➢ A jagged array is an array of arrays, and therefore its elements are reference types and are

initialized to null.

➢ Arrays are zero indexed: an array with n elements is indexed from 0 to n-1.

➢ Array elements can be of any type, including an array type.

➢ Array types are reference types derived from the abstract base type Array. Since this type

implements IEnumerable and IEnumerable<T>, you can use foreach iteration on all arrays in

C#.

**Example**

The following example creates single-dimensional, multidimensional, and jagged arrays:

class Program

{

public static void Main(string[] args)

{

// Declare a single-dimensional array of 5 integers.

int[] array1 = new int[5];

// Alternative syntax.

int[] array3 = { 1, 2, 3, 4, 5, 6 };

// Declare and set array element values.

int[] array2 = new int[] { 1, 3, 5, 7, 9 };

// Declare a two dimensional array.

int[,] multiDimensionalArray1 = new int[2, 3];

// Declare and set array element values.

int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

// Declare a jagged array.

int[][] jaggedArray = new int[6][];

// Set the values of the first array in the jagged array structure.

jaggedArray[0] = new int[4] { 1, 2, 3, 4 };

} }

**Array:**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

**1) 1-D Array:**

// Declare a single-dimensional array of 5 integers.

int[] array1 = new int[5];

// Declare and set array element values.

int[] array2 = new int[] { 1, 3, 5, 7, 9 };

// Alternative syntax.

int[] array3 = { 1, 2, 3, 4, 5, 6 };

**2) 2-D Array:**

// Declare a two dimensional array.

int[,] multiDimensionalArray1 = new int[2, 3];

// Declare and set array element values.

int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

**3) Jagged Array:**

A jagged array is an array of array. Jagged arrays store arrays instead of literal values.

// Declare a jagged array.

int[][] jaggedArray = new int[6][];

// Set the values of the first array in the jagged array structure.

jaggedArray[0] = new int[4] { 1, 2, 3, 4 };