Unit 1.2
Language Preliminaries & .Net Basic (8-Hrs)

Language Preliminaries(C#)

- C# is a general-purpose, modern, type -safe and object-oriented programming language pronounced as "C sharp".
- C# language has its roots from the family of C languages such as C, C++ and is fully Event-driven and visual programming language.
- This language was created by Microsoft and runs on the .NET Framework.
- C# Programming language will allow developers to build a variety of secure and robust applications such
  i. Mobile applications
  ii. Desktop applications
  iii. Web applications
  iv. Web services
  v. Web sites
  vi. Games
  vii. Database applications

Basic Language Construct:C# Program Structure

- **WAP to Display hello world.**

```
using System;
namespace MyProgram
{
    class Program
    {
        //Program to Display Hello World
        static void Main(string[] args)
        {
            Console.WriteLine("Hello world");
            Console.ReadKey();
        }
    }
}
```

Explanation:

- Comments:
  - Comments are used for explaining code and are used in similar manner as in Java or C or C++.
  - Compilers ignore the comment entries and does not execute them.
  - Comments can be of single line or multiple lines.
    1. Single line Comments
    Syntax:
    // Single line comment
    2. Multi line comments:
    Syntax:
    /* Multi line comments*/
- using System:
  - using keyword is used to include the System namespace in the program. namespace declaration:
  - A namespace is a collection of classes.
  - The MyProgram namespace contains the class Program.
- class:
  - The class contains the data and methods to be used in the program.
  - Methods define the behavior of the class.
  - Class Program has only one method Main().
- static void Main():
  - static keyword tells us that this method is accessible without instantiating the class.

- void keywords:
  - It tells that this method will not return anything.
  - Main() method is the entry-point of our application.
  - In our program, Main() method specifies its behavior with the statement
    Console.WriteLine("Hello world");
- Console.WriteLine():
  - WriteLine() is a method of the Console class defined in the System namespace
- Console.ReadKey():
  - This makes the program wait for a key press and prevents the screen from running and closing quickly.
- C# is case sensitive and all statements and expressions must end with semicolon (;).

C# Identifier

- In programming languages, identifiers are used for identification purposes. Or in other words, identifiers are the user-defined name of the program components.
- In C#, an identifier can be a class name, method name, variable name or label.

Example:

```
public class Demo {
   public static void Main ()
   {
       int x;
   }
}
```

- Here the total number of identifiers present in the above example is 3 and the names of these identifiers are:
  1. Demo: Name of the class
  2. Main: Method name
- x: Variable name

Rules for Naming Identifier

- The only allowed characters for identifiers are all alphanumeric characters(**[A-Z]**, **[a-z]**, **[0-9]**), '_' (underscore). For example "a@" is not a valid C# identifier as it contain '@' – special character.
- Identifiers should not start with digits([0-9]). For example "123demo" is a not a valid in C# identifier.
- Identifiers should not contain white spaces.
- Identifiers are not allowed to use as keyword unless they include @ as a prefix. For example, **@int** is a valid identifier, but "**int**" is not because it is a keyword.
- C# identifiers are case-sensitive.
- C# identifers cannot contain more than 512 characters.
- Identifiers does not contain two consecutive underscores in its name.

C# Data Type

- Data types specify the type of data that a valid C# variable can hold.
- C# is a **strongly typed programming language** because in C#, each type of data (such as integer, character, float, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.
- Data types in C# is mainly divided into three categories
  1. Value Data Types
     - In C#, the Value Data Types will directly store the variable value in memory and it will also accept both signed and unsigned literals.
     - The derived class for these data types are **System.ValueType**.
     - Following are **different Value Data Types** in C# programming language :
       a. **Signed & Unsigned Integral Types:** There are 8 integral types which provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.

ALIAS	TYPE NAME	TYPE	SIZE(BITS)	RANGE	DEFAULT VALUE
sbyte	System.Sbyte	signed integer	8	-128 to 127	0
short	System.Int16	signed integer	16	-32768 to 32767	0
Int	System.Int32	signed integer	32	-2^{31} to $2^{31}-1$	0
long	System.Int64	signed integer	64	-2^{63} to $2^{63}-1$	0L
byte	System.byte	unsigned integer	8	0 to 255	0
ushort	System.UInt16	unsigned integer	16	0 to 65535	0
uint	System.UInt32	unsigned integer	32	0 to 2^{32}	0
ulong	System.UInt64	unsigned integer	64	0 to 2^{63}	0

b. Floating point types: There are 2 floating point data types which contain the decimal point.

ALIAS	TYPE NAME	SIZE(BITS)	RANGE (APROX)	DEFAULT VALUE
float	System.Single	32	$\pm1.5 \times 10^{-45}$ to $\pm3.4 \times 10^{38}$	0.0F
double	System.Double	64	$\pm5.0 \times 10^{-324}$ to $\pm1.7 \times 10^{308}$	0.0D

c. Decimal Types: The decimal type is a 128-bit data type suitable for financial and monetary calculations. It has 28-29 digit Precision. To initialize a decimal variable, use the suffix m or M. Like as, decimal x = 300.5m;. If the suffix m or M will not use then it is treated as double.

ALIAS	TYPE NAME	SIZE(BITS)	RANGE (APROX)	DEFAULT VALUE
decimal	System.Decimal	128	$\pm1.0 \times 10^{-28}$ to $\pm7.9228 \times 10^{28}$	0.0M

d. Character Types: The character types represents a UTF-16 code unit or represents the 16-bit Unicode character.

ALIAS	TYPE NAME	SIZE IN(BITS)	RANGE	DEFAULT VALUE
char	System.Char	16	U +0000 to U +ffff	'\0'

e. Boolean Types: **It has to be assigned either true or false value. Values of type bool are not converted implicitly or explicitly (with casts) to any other type. But the programmer can easily write conversion code.**

ALIAS	TYPE NAME	VALUES
bool	System.Boolean	True / False

2. Reference Data Types
   - The Reference Data Types will contain a memory address of variable value because the reference types won't store the variable value directly in memory. The built-in reference types are **string, object.**
     a. **String:** It represents a sequence of Unicode characters and its type name is **System.String**. So, string and String are equivalent.

        Example: String str="Net Centric Computing";

     b. **Object:** In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. So basically it is the base class for all the data types in C#. Before assigning values, it needs type conversion. When a variable of a value type is converted to object, it's called **boxing**. When a variable of type object is converted to a value type, it's called **unboxing**. Its type name is System.Object.

        Example:

```
object a = 1;
int b = 2;
int c = (Int32)a;//unboxing
object d = b; //boxing
```

3. Pointer Data Type
   - The Pointer Data Types will contain a memory address of the variable value. To get the pointer details we have a two symbols ampersand (&) and asterisk (*). ampersand (&): It is Known as Address Operator. It is used to determine the address of a variable. asterisk (*): It also known as Indirection Operator. It is used to access the value of an address. Syntax :

     type* identifier;

     Example :

     int* p1, p; // Valid syntax

Note: Pointer Data Types will not work on online compiler. Unsafe code requires the `unsafe' command line option to be specified and we need to go to project properties page and check under Build the checkbox Allow unsafe code. The following program demonstrates this concept.

```
using System;
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            unsafe
            {
                // declare variable
                int n = 10;
                // store variable n address
                // location in pointer variable p
                int* p = &n;
                Console.WriteLine("Value :{0}", n);
                Console.WriteLine("Address :{0}", (int)p);
                Console.ReadKey();
            }

        }
    }
}
```

```
Value :10
Address :7729720
```

C# Variable

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.
- Variable can be either numeric, string, date, character, Boolean etc.
- C# is **case-sensitive programming** language. That is lower-case letters are treated as distinct from upper-case letters. Thus the word main in a program is quite different from the word Main or the word MAIN.

Defining/Declaring Variables
Syntax:
<data_type> <variable_list>;

Example: int num1;
String name;
double a, b, c;

Initializing variable
Syntax:
Variable_Name=Value;

Example: int num; //Variable declaration
num=5; //initialization

The above two students can be combined as below
int num=5; //Declaring and Initialization

C# Constant

- A constant is a variable whose value can't be changed throughout its life time.
- Constant are declared by prefixing an initialized variable with const keyword
For example:
const double pi=3.1416;

C# Operator

- An operator is a symbol that instructs to perform some operation, or action, on one or more operands.
- An operand is something that an operator acts on.
- Let us take a simple example, consider a simple expression 4+5, here 4 and 5 are operand and + is an arithmetic operator that perform addition of two operands 4 and 5.
- In C#, all operands are expressions. C operators fall into two categories as below

i. According to number of operands

- Unary operator
The operator which requires only one operand is called unary operator. Increment operator ++, decrement operator --, are unary operator.
- Binary operator
The operator which requires two operands is called binary operator. For example, plus +, minus -, etc are binary operator.
- Ternary operator
The operator that requires three operands are called ternary operators. For example, the operator pair "?:" also called as conditional operator is a ternary operator.

WAP to demonstrate the concept of unary, binary and ternary operators

```
using System;
namespace ConsoleApp
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            int n1 = 7, n2 = 4, n3;
            n3 = (n1 > n2) ? n1 : n2; //conditional ?: ternary operator
            Console.WriteLine("Greatest Number: " + n3);
            n1++;//increment ++ Unary operator
            Console.WriteLine("After Increment n1: " + n1);
            n3 = n1 + n2;//Plus + binary operator
            Console.WriteLine("Sum: " + n3);
            Console.ReadKey();
        }
    }
}
```

```
Greatest Number: 7
After Increment n1: 8
Sum: 12
```

ii. According to utility and action

- Arithmetic/Mathematical operators
- Increment Decrement operator
- The assignment operator
- Relational operators
- Logical operators
- Conditional operators
- Bitwise operator

1. Arithmetic / Mathematical Operators

- C# mathematical operators perform mathematical operations such as addition and subtraction. five ***binary*** *mathematical operators*.
- C's binary operators take two operands.

Operator	Symbol	Action	Example	
Addition	+	Adds two operands	x + y	4+5=9
Subtraction	-	Subtracts the second operand from the first operand	x - y	4-5=-1
Multiplication	*	Multiplies two operands	x * y	4*5=20
Division	/	Divides the first operand by the second operand	x / y	6/2=3
Modulus	%	Gives the remainder when the first operand is divided by the second operand	x % y	5%2=1

2. Increment/Decrement Operator

- The unary mathematical operators are so named because they take a single operand.
- C# has two unary mathematical operators:
  - i. The increment operator(++) is used to increase the value of operand by 1
  - ii. The decrement operator(--) is used to decrement the value of operator by 1.

Operator	Symbol	Action	Examples
Increment	++	Increments the operand by one	++x, x++
Decrement	--	Decrements the operand by one	--x, x--

The increment and decrement operators can be used only with variables, not with constants. The operation performed is to add one to or subtract one from the operand. In other words, the statements

```
++x; --y;
```

are the equivalent of these statements:

```
x = x + 1;      y = y - 1;
```

We should note from above Table that either unary operator can be placed **before its operand (prefix mode) or after its operand (postfix mode).** These two modes are not equivalent.

For example : Consider following statements:

```
x = 10;
y = x++;            // similar to y=x and then x=x+1
```

After these statements are executed, x has the value 11, and y has the value 10. The value of x was assigned to y, and then x was incremented. In contrast, the following statements result in both y and x having the value 11. x is incremented, and then its value is assigned to y.

```
x = 10;
y = ++x;        // similar to x=x+1 and then y=x
```

3. **Assignment Operator**
   - The assignment operator is the equal sign (=). Its use in programming is somewhat different from its use in regular math. If we write

     x = y;

     in a C# program, it doesn't mean "x is equal to y." Instead, it means "assign the value of y to x." In a C# assignment statement, the right side can be any expression, and the left side must be a variable name. Thus, the form is as follows:

     variable = expression;

     When executed, expression is evaluated, and the resulting value is assigned to variable.

   - There are basically five other form of **shorthand assignment operator.**
   - They are +=, -=, /=, *= and %=.
   - The use of shorthand assignment operator makes our code length shorter and easier to read.

Example:

a+=b is similar to a=a+b
a-=b is similar to a=a-b
a*=b is similar to a=a*b
a/=b is similar to a=a/b
a%=b is similar to a=a%b

4. **Relational Operator**
   - Relational operators are used to compare two similar operands, and depending on their relation, take some action.
   - C#'s relational operators are used to compare expressions, asking questions such as, "Is x greater than 100?" or "Is y equal to 0?" An expression containing a relational operator evaluates to either true (1) or false (0).

Operator	Symbol	Question Asked	Example
Equal	==	Is operand 1 equal to operand 2?	x == y
Greater than	>	Is operand 1 greater than operand 2?	x > y
Less than	<	Is operand 1 less than operand 2?	x < y
Greater than or equal to	>=	Is operand 1 greater than or equal to operand 2?	x >= y
Less than or equal to	<=	Is operand 1 less than or equal to operand 2?	x <= y
Not equal	!=	Is operand 1 not equal to operand 2?	x != y

Table: Relational operators in use.

Expression	How It Reads	What It Evaluates To
5 == 1	Is 5 equal to 1?	0 (false)
5 > 1	Is 5 greater than 1?	1 (true)
5 != 1	Is 5 not equal to 1?	1 (true)
(5 + 10) == (3 * 5)	Is (5 + 10) equal to (3 * 5)?	1 (true)

5. **Logical operator**
   - Sometimes we need to ask more than one relational question at once.

- C#'s logical operators let us combine two or more relational expressions into a single expression that evaluates to either true or false.

Operator	Symbol	Example
AND	&&	exp1 && exp2 e.g. (num>100)&&(num%2==0)
OR	||	exp1 || exp2 e.g. (num<100)||(num%2!=0)
NOT	!	!exp1 e.g. num!=5

Expression	What It Evaluates To
(exp1 && exp2)	True (1) only if both exp1 and exp2 are true; false (0) otherwise
(exp1 || exp2)	True (1) if either exp1 or exp2 is true; false (0) only if both are false
(!exp1)	False (0) if exp1 is true; true (1) if exp1 is false

Example:

Expression	What It Evaluates To
(5 == 5) && (6 != 2)	True (1), because both operands are true
(5 > 1) || (6 < 1)	True (1), because one operand is true
(2 == 1) && (5 == 5)	False (0), because one operand is false
!(5 == 4)	True (1), because the operand is false

6. Conditional operator

- The operator pair "?:" is known as conditional operator. The conditional operator is C#'s only ternary operator, meaning that it takes three operands. Its syntax is

  exp1 ? exp2 : exp3;

  If exp1 evaluates to true (that is, nonzero), the entire expression evaluates to the value of exp2. If exp1 evaluates to false (that is, zero), the entire expression evaluates as the value of exp3.

  For example, the following statement assigns the value 1 to x if exp is true and assigns 100 to x if exp is false:

  x = exp ? 1 : 100;

  Likewise, to make z equal to the larger of x and y, we can write

  z = (x > y) ? x : y;

7. Bitwise Operator

- C# has special operators for bit oriented programming known as bitwise operators.
- They are used for manipulating data in bit level. These are used to testing the bits or shifting the bits left or right.
- Bitwise operators are not applied to float and double.
- Following table shows the bitwise operators and their meanings.

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise X-OR
<<	shift left
>>	shift right
~	one's complement

Example:

1.

A=4;
B=6;
C=A&B ;
We will get 4 in variable C
How?

A= 0000 0000 0000 0100
B= 0000 0000 0000 0110
C=0000 0000 0000 0100 which is similar to 4 in decimal.
Note (Assuming integer data type is 16 bit, represent 4 and 6 in binary form and perform bitwise anding)

2.

A=4;
B=6;
C=A|B;
We will get 6 in variable C
How?
A= 0000 0000 0000 0100
B= 0000 0000 0000 0110
C= 0000 0000 0000 0110 which is similar to 6 in decimal.

3.

A=6;
B=A<<3;
The value of Variable B will be 48
How?
6 = 0000 0000 0000 0110
First shift= 0000 0000 0000 1100
Second shift = 0000 0000 0001 1000
Third shift= 0000 0000 0011 0000 which is equal to 48 in decimal

Consider another example
A=-60
B=A<<1;
The value of B will be 120.
How?
60 = 0000 0000 0011 1100
-60 = 1111 1111 1100 0011 +1
= 1111 1111 1100 0100
First shift= 1111 1111 1000 1000
= (-2^15+2^14+2^13+……2^7+2^3)
= -120

4.

A=60;
B=A>>3;
The value of B will be 7
How?
60 = 0000 0000 0011 1100
First shift =0000 0000 0001 1110
Second Shift = 0000 0000 0000 1111
Third Shift = 0000 0000 0000 0111 which is equal to 7 in decimal

Consider another example
A=-60;
B=A>>1;
The value of B will be -30

How?
60 = 0000 0000 0011 1100
-60 = 1111 11111100 0011 +1
= 1111 1111 1100 0100
First Right Shift = 1111 1111 1110 0010 (Note the sign bit must be presrerved)
= (-2^15 +2^14+2^13……+2^5 + 2^1)
= -30 in decimal.

5.

A=60;
B=~A;
The value of B will be -61.
How?
60 = 0000 0000 0011 1100
~60= 1111 1111 1100 0011
= (-2^15+ 2^14 + 2^13 +……2^6+2^1+2^0)

= -61.

- **WAP to find the area and perimeter of rectangle.**

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter length and breadth of Rectangel : ");
            double length = Convert.ToDouble(Console.ReadLine());
            double breadth = Convert.ToDouble(Console.ReadLine());
            double area = length * breadth;
            double perimeter = 2 * (length + breadth);
            Console.WriteLine("Area Of Rectangle : " + area);
            Console.WriteLine("Perimeter of Rectangle : " + perimeter);
            Console.ReadKey();
        }
    }
}
```

```
Enter length and breadth of Rectangel :
4
5
Area Of Rectangle : 20
Perimeter of Rectangle : 18
```

Control Structure: Flow Control

- The statement that alter the flow of execution of program is called control statement.
- Basically a C# program is a sequence of instructions which are normally executed sequentially in the order which they appear, however in some case we have to take certain action based on the outcome of condition i.e. true or false, perform repeated actions or skip some statements.
- In those case, we need control structure.
- There are two types of control structure.

i. Decision making statement

- Decision making statements are those statements that control the flow of execution of programs. While writing a program, there are many case when we need to change the order of execution of statements based on certain condition or repeat some set of statements until certain condition is satisfied. This involves a kind of decision making to see whether a particular condition has occurred or not and then redirect the computer to execute certain statements accordingly.

a. If statement

Syntax:

```
if (condition)
{
        Statement(s) block;
}
Remaining Statements(s);
```

b. If else statement

Syntax:

```
if (condition)
{
        True statement(s) block;
}
else
{
        False statements(s) block;
}
Statement-X
Remaining Statement(s)
```

c. If else if ladder statement

Syntax:

```
if (condition1)
{
        statement(s) block 1;
}
else if (condition 2)
{
        statements(s) block 2;
}
else if (condition 3)
{
        Statement(s) block 3;
}
else
{
        Default statement(s);
}
Next Statement(s)
```

d. Switch statement

Syntax:

```
switch(Expression)
{
        case case1:
                Statement(s);
                break;
        case case2:
```

```
            Statement(s);
            break;
        ……….
        ……….
        default:
            statement(s);
            break;
    }
```

- **WAP to input name, roll and marks in three different subject of a student. Display the entire detail in console along with percentage and division if student is pass otherwise display fail information. Pass Mark for each Subject is 40.**

```
using System;
namespace MyProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter Name : ");
            string nam = Console.ReadLine();
            Console.WriteLine("Enter Roll : ");
            int roll = Convert.ToInt16(Console.ReadLine());
            Console.WriteLine("Enter Marks In Three Subjects : ");
            double m1 = Convert.ToDouble(Console.ReadLine());
            double m2 = Convert.ToDouble(Console.ReadLine());
            double m3 = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("Name : " + nam);
            Console.WriteLine("Roll : " + roll);
            String result = "";
            if (m1 >= 40 && m2 >= 40 && m3 >= 40)
            {
                double tot = m1 + m2 + m3;
                double pct = tot / 3;
                string div = "";
                if (pct >= 80)
                {
                    div = "Distinction";
                }
                else if (pct >= 60)
                {
                    div = "First Division";
                }
                else if (pct >= 50)
                {
                    div = "Second Division";
                }
                else if (pct >= 40)
                {
                    div = "Third Division";
                }
                Console.WriteLine("Total Obtained Mark: " + tot);
                Console.WriteLine("Division : " + div);
                result = "Pass";
            }
            else
            {
                result = "Fail";
            }
            Console.WriteLine("Result : "+result);
            Console.ReadKey();

        }
```

```
        }
    }
```

Output:

- **Write a menu based program to find the sum, difference, product and quotient of two numbers entered by user.**

```
using System;
namespace SwitchDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Clear();
            Console.WriteLine("Choose the option 1-6");
            Console.WriteLine("1: Addition");
            Console.WriteLine("2: Subtraction");
            Console.WriteLine("3: Multiplication");
            Console.WriteLine("4: Division");
            Console.WriteLine("5: Exit");
            int ch =Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter two numbers");
            double num1 = Convert.ToDouble(Console.ReadLine());
            double num2 = Convert.ToDouble(Console.ReadLine());
            double res = 0;
            switch (ch)
            {
                case 1:
                    res = num1 + num2;
                    break;
                case 2:
                    res = num1 - num2;
                    break;
                case 3:
                    res = num1 * num2;
                    break;
                case 4:
                    res = num1 / num2;
                    break;
                case 5:
                    System.Environment.Exit(0);
```

```
                    break;
                default:
                    break;
            }


            Console.WriteLine("Result : " + res);
            Console.ReadKey();
            Main(null);
        }
    }
}
```

```
Choose the option 1-6
1: Addition
2: Subtraction
3: Multiplication
4: Division
5: Exit
1
Enter two numbers
5
6
Result : 11
```

ii. Looping or repeating statement

- Looping is the process of executing a sequence of statements until some conditions for termination of the loop are satisfied. A loop program basically consists of two parts, one is body of loop and other is control statements. The control statement tests certain condition and then directs the repeated execution of the statements contained in the body of loop.
- The looping process involves the following steps.
  1. Initialization of condition variable.
  2. Execution of the statements in the loop.
  3. Test for a specified value of the condition variable for execution of the loop.
  4. Updating the condition variable.
- Types
  1. **For loop**

  Syntax:

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

WAP to print all the even numbers from 1 to 200 using for loop.

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            for(int i=2;i<=200;i=i+2)
            {
                Console.Write(i + "\t");
            }
            Console.ReadKey();
        }
    }
}
```

```
2       4       6       8       10      12      14      16      18      20      22      24      26      28      30
        32      34      36      38      40      42      44      46      48      50      52      54      56      58
        60      62      64      66      68      70      72      74      76      78      80      82      84      86
        88      90      92      94      96      98      100     102     104     106     108     110     112     114
        116     118     120     122     124     126     128     130     132     134     136     138     140     142
        144     146     148     150     152     154     156     158     160     162     164     166     168     170
        172     174     176     178     180     182     184     186     188     190     192     194     196     198
        200
```

2. While loop

Syntax:

```
while(test condition)
{
        Looping statement(s);
}
Remaining statement(s);
```

WAP to print all the even numbers from 1 to 200 using while loop.

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            int num = 2;
            while(num<=200)
            {
                Console.Write(num + "\t");
                num = num + 2;
            }
            Console.ReadKey();
        }
    }
}
```

```
2       4       6       8       10      12      14      16      18      20      22      24      26      28      30
        32      34      36      38      40      42      44      46      48      50      52      54      56      58
        60      62      64      66      68      70      72      74      76      78      80      82      84      86
        88      90      92      94      96      98      100     102     104     106     108     110     112     114
        116     118     120     122     124     126     128     130     132     134     136     138     140     142
        144     146     148     150     152     154     156     158     160     162     164     166     168     170
        172     174     176     178     180     182     184     186     188     190     192     194     196     198
        200
```

WAP to check if the input number is palindrome or not.

```
using System;
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a number : ");
            int num = Convert.ToInt32( Console.ReadLine());
            int tmp = num;
            int rev = 0;
            while(num!=0)
            {
                int ext = num % 10;
                rev = rev * 10 + ext;
                num = num / 10;
            }
            if(rev==tmp)
            {
```

```
                Console.WriteLine("The entered number is Palindrome");
            }
            else
            {
                Console.WriteLine("The entered number is Not Palindrome");
            }
            Console.ReadKey();
        }
    }
}
```

3. **Do while loop**
   Syntax:

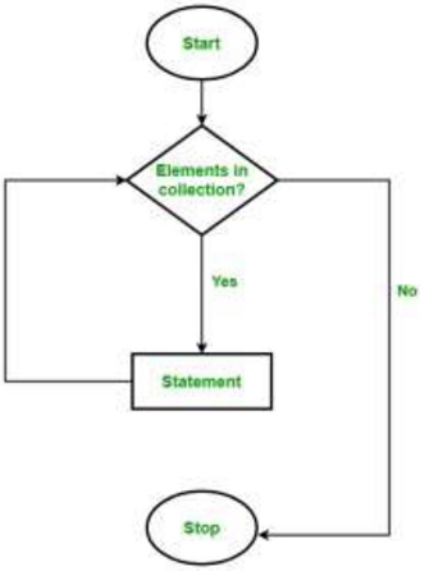
```
do
{
        Looping statement(s);
}while(test condition);
Remaining statement(s);
```

WAP to print all the even numbers from 1 to 200 using do-while loop.

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            int num = 2;
            do
            {
                Console.Write(num + "\t");
                num = num + 2;
            } while (num <= 200);
            Console.ReadKey();
        }
    }
}
```

```
2       4       6       8       10      12      14      16      18      20      22      24      26      28      30
        32      34      36      38      40      42      44      46      48      50      52      54      56      58
        60      62      64      66      68      70      72      74      76      78      80      82      84      86
        88      90      92      94      96      98      100     102     104     106     108     110     112     114
        116     118     120     122     124     126     128     130     132     134     136     138     140     142
        144     146     148     150     152     154     156     158     160     162     164     166     168     170
        172     174     176     178     180     182     184     186     188     190     192     194     196     198
        200
```

4. **For each loop**
   - The foreach loop iterates through each item in a collection, hence called foreach loop.
   - The collection may be an array or a list
   - It executes for each element present in the array or a collection.
   - It is necessary to enclose the statements of foreach loop in curly braces {}.
   - Suppose we need to access each element of an array then instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
   - In the loop body, you can use the loop variable you created rather than using an indexed array element.



Syntax:

```
foreach (datat_type item in collection)
{
   //Statements
}
```

Array

- An array is a set of related items of same data type that share common name.
- Arrays of any type like int, float, String etc. can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index.
- Types
    1. One-dimensional Array
    2. Two dimensional Array

One dimensional Array

- One dimensional array can store a list of items of same time.
- The general form of 1D array declaration is

  type [] variablename;

  Example:

  int [] student; //declares an array variable with name "student" of type integer.
- A **new** operator is used to allocate memory for an array as below

  int []student;

  student=new int[10];

  This example allocates a 10-element array of integers and links them to student.

  The above two statement also can be combined to one statement as

  int [] student=new int[10];

- **WAP to input name of 10 students and display using**
  i. For loop

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            String[] name = new String[10]; //One Dimensional Array Initialization
            Console.WriteLine("Enter name of 10 students");
            for(int i=0;i<10;i++)
```

```
            {
                name[i] = Console.ReadLine();
            }
            Console.WriteLine("Entered Name are");
            for(int i=0;i<10;i++)
            {
                Console.WriteLine(name[i]);
            }
            Console.ReadKey();
        }
    }
}
```

```
Enter name of 10 students
ram
hari
shiva
rita
seeta
gita
bhesh
prabin
bidur
shiva
Entered Name are
ram
hari
shiva
rita
seeta
gita
bhesh
prabin
bidur
shiva
```

ii. For Each Loop

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            String[] name = new String[10];
            Console.WriteLine("Enter name of 10 students");
            for(int i=0;i<10;i++)
            {
                name[i] = Console.ReadLine();
            }
            Console.WriteLine("Entered Name are");
            foreach(string s in name)
            {
                Console.WriteLine(s);
            }
            Console.ReadKey();
        }
    }
}
```

```
Enter name of 10 students
ram
hari
shiva
rita
seeta
gita
bhesh
prabin
bidur
shiva
Entered Name are
ram
hari
shiva
rita
seeta
gita
bhesh
prabin
bidur
shiva
```

Multi-dimensional Array

- An array with two or more dimensions is called a multi-dimensional array.
- The general form for 2D array declaration is

  type variablename[,];

  Example:

  int [,] student; //declares 2D array variable with name "student" of type integer.
- A **new** operator is used to allocate memory for an array as below

  int [,] student;

  student=new int[5,4];

  This example allocates a 2D array of integers with 5 rows and 4 columns and links them to student.

  The above two statement also can be combined to one statement as

  int [,]student=new int[5,4];

- **WAP to input name and roll number of 10 students and display them.**

```
using System;
namespace Array2D
{
    class Program
    {
        static void Main(string[] args)
        {
            String[,] data = new String[10,2];   //Declaring 2D array of size (10,2)
            Console.WriteLine("Enter Name and Roll of 10 students");
            for(int i=0;i<10;i++)
            {
                for(int j=0;j<2;j++)
                {
                    data[i,j] = Console.ReadLine();
                }
            }
            Console.WriteLine("Entered Name and Roll : ");
            for (int i = 0; i < 10; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    Console.Write(data[i, j]);
                    Console.Write("\t");
                }
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

String

- String is the representation of the text.
- In C#, *string* is a sequence of Unicode characters or array of characters.
- A string is represented by class *System.String*. The *"string" keyword* is an alias for System.String class and instead of writing System.String one can use *String* which is a shorthand for *System.String* class. So we can say string and String both can be used as an alias of System.String class. So string is an *object* of System.String class.
  Example:
  string s1 = "bhesh"; // creating the string using string keyword
  String s2 = "bhesh"; // creating the string using String class
- The keyword string is used for the declaration but System.String is used for accessing static string methods.
- Some static methods of String class are
  - Contains(): The C# Contains method checks whether specified character or string is exists or not in the string value.
  - Endswith(): This EndsWith Method checks whether specified character is the last character of string or not.
  - StarsWith(): It checks whether the first character of string is same as specified character.
  - IndexOf(): Returns the index position of first occurrence of specified character.
  - LastIndexOf(): Returns the index position of last occurrence of specified character.
  - ToLower(): Converts String into lower case based on rules of the current culture.
  - ToUpper(): Converts String into Upper case based on rules of the current culture.
  - Length(): It is a string property that returns length of string.
  - Replace(): This method replaces the character.
  - Substring(): This method returns substring.
  - Trim() : It removes extra whitespaces from beginning and ending of string.

- **WAP to input full name and display the first name only.**

```
using System;
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter full Name :");
            string str = Console.ReadLine();
            int indexoffirstspace = str.IndexOf(" ");
            string firstnameonly = str.Substring(0, indexoffirstspace);
            Console.WriteLine("First Name : "+firstnameonly);
            Console.ReadKey();
        }
    }
}
```

```
Enter full Name :
Bhesh Bdr Thapa
First Name : Bhesh
```

- **WAP to count the number of vowels, consonants, digits, space and words in a sentence entered by user.**

```
using System;
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a Sentence :");
            string str = Console.ReadLine();
            str = str.ToUpper();
            int cdigit=0, cvoweil=0 , cconsonan=0 , cspace=0 , cword =0;
            foreach (char ext in str)
            {
```

```
                if(ext == 'A' || ext == 'E' || ext == 'I' || ext == 'O' || ext == 'U')
                {
                    cvoweil++;
                }
                else if (ext >= 'A' && ext <= 'Z')
                {
                    cconsonan++;
                }
                else if(ext>='0' && ext<='9')
                {
                    cdigit++;
                }
                else if(ext==' ')
                {
                    cspace++;
                }
            }
            cword = cspace + 1;
            Console.WriteLine($"Total Number of Consonants Character : {cconsonan}");
            Console.WriteLine($"Total Number of Vowels Character : {cvoweil}");
            Console.WriteLine($"Total Number of Digits Character : {cdigit}");
            Console.WriteLine($"Total Number of Space Character : {cspace}");
            Console.WriteLine($"Total Number of Words : {cword}");
            Console.ReadKey();
        }
    }
}
```

```
Enter a Sentence :
I am From Hemja Ward No. 25.
Total Number of Consonants Character : 11
Total Number of Vowels Character : 7
Total Number of Digits Character : 2
Total Number of Space Character : 6
Total Number of Words : 7
```

- **WAP to input a string and check if it is palindrome or not.**

```
using System;
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a String :");
            string str = Console.ReadLine();
            String rev = "";
            foreach (char ext in str)
            {
                rev = ext + rev;
            }
            if(rev==str)
            {
                Console.WriteLine("The entered string is palindrome");
            }
            else
            {
                Console.WriteLine("The entered string is not palindrome");
            }
            Console.ReadKey();
        }
    }
}
```

```
Enter a String :
liril
The entered string is palindrome
```

Functions: Library Function and User Defined Function

- A function is a named, independent section of code that performs a specific task and optionally returns a value to the calling program.
- A function is a group of statements that together perform a task. Every c# program has at least one function, which is main ().
- A function is named. Each function has a unique name. By using that name in another part of the program, we can execute the statements contained in the function. This is known as calling the function. A function can be called from within another function.
- There are two types of function: Pre-defined and user-defined.

1. **Pre-defined / Library / Built-in functions:** These functions are library function which are simply called providing necessary arguments, if any, by the programmer to do some specific task as defined by that function.
2. **User-defined functions** are the functions that are written by the programmers themselves and form the part of the source code. Those are compiled with other functions.

- **WAP to calculate to find the n^{th} power of a given number.**

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter a number");
            double num= Convert.ToInt16(Console.ReadLine());
            Console.Write("Enter the power to be raised");
            double pow = Convert.ToInt16(Console.ReadLine());
            double res = Math.Pow(num, pow);
            Console.WriteLine("Result : " + res);
            Console.ReadKey();
        }
    }
}
```

In the above Pogram, pow() is a mathematical library function defined as static member in class Math.

```
Enter a number2
Enter the power to be raised2
Result : 4
```

- **WAP to find the area of circle using function double area(double).**

```
using System;
namespace Area
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter readius of circle: ");
            double rad = Convert.ToDouble(Console.ReadLine());
            double res=calculatearea(rad);
            Console.WriteLine("Area of circle : " + res);
            Console.ReadKey();
        }

        //Function Should Be Static As we Are Accessing this Function From Static Context i.e.
          static Main

        static double calculatearea(double rad)
        {
```

```
            const double pi = 3.1416; //Constant Declaration
            double res = pi * rad * rad;
            return (res);
        }
    }
}
```

In the above program calculatearea() is userdefined function.

```
Enter readius of circle:
1
Area of circle : 3.1416
```