**C# Keywords**

C# contains reserved words that have special meaning for the compiler. These reserved words are called "keywords". Keywords cannot be used as an identifier (name of a variable, class, interface, etc.).

**Modifier Keywords**

Modifier keywords are specific keywords that indicate who can modify types and type members. Modifiers allow or prevent certain parts of programs from being modified by other parts.

1. abstract
2. async
3. const
4. event
5. extern
6. new
7. override
8. partial
9. readonly
10. sealed
11. static
12. unsafe
13. virtual
14. volatile

**Virtual**

C# virtual method is a method that can be redefined in derived classes. In C#, a virtual method has an implementation in a base class as well as derived the class. It is used when a method's basic functionality is the same but sometimes more functionality is needed in the derived class. A virtual method is created in the base class that can be overridden in the derived class.

Create a virtual method in the base class using the virtual keyword and that method is overriden in the derived class using the override keyword.

# C# Namespace

Namespaces play an important role in managing related classes in C#. The .NET Framework uses namespaces to organize its built-in classes. A namespace is a container for classes and namespaces. The namespace also gives unique names to its classes thereby you can have the same class name in different namespaces.

# C# Variables

int num = 100;

float rate = 10.2f;

decimal amount = 100.50M;

char code = 'C';

bool isValid = true;

string name = "Steve";

The followings are naming conventions for declaring variables in C#:

* Variable names must be unique.
* Variable names can contain letters, digits, and the underscore \_ only.
* Variable names must start with a letter.
* Variable names are case-sensitive, num and Num are considered different names.
* Variable names cannot contain reserved keywords. Must prefix @ before keyword if want reserve keywords as identifiers.

# C# - var

In C#, variables must be declared with the data type. These are called explicitly typed variables.

C# 3.0 introduced var keyword to declare method level variables without specifying a data type explicitly.

# C# - Data Types

C# is a strongly-typed language.

Variables of Different Data Types

string stringVar = "Hello World!!";

int intVar = 100;

float floatVar = 10.2f;

char charVar = 'A';

bool boolVar = true;

C# mainly categorized data types in two types:

Diagram

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

# C# - Struct

struct is the [value type](https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type) data type that represents data structures. It can contain a parameterized constructor, static constructor, constants, fields, methods, properties, indexers, operators, events, and nested types.

struct can be used to hold small data values that do not require inheritance, e.g. coordinate points, key-value pairs, and complex data structure.

## **Structure Declaration**

**A structure is declared using struct keyword.** The default modifier is internal for the struct and its members.

The following example declares a structure Coordinate for the graph.

Example: Structure

struct Coordinate

{

public int x;

public int y;

}

# C# Enumerations Type - Enum

In C#, an enum (or enumeration type) is used to assign constant names to a group of numeric integer values. It makes constant values more readable, for example, WeekDays.Monday is more readable then number 0 when referring to the day in a week.

An enum is defined using the enum keyword, directly inside a namespace, class, or structure. All the constant names can be declared inside the curly brackets and separated by a comma.

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

## **Enum Values**

If values are not assigned to enum members, then the compiler will assign integer values to each member starting with zero by default. The first member of an enum will be 0, and the value of each successive enum member is increased by 1.

You can assign different values to enum member. A change in the default value of an enum member will automatically assign incremental values to the other members sequentially.

Example: Assign Values to Enum Members

enum Categories

{

Electronics, // 0

Food, // 1

Automotive = 6, // 6

Arts, // 7

BeautyCare, // 8

Fashion // 9

}

The enum can be of any numeric data type such as byte, sbyte, short, ushort, int, uint, long, or ulong.

However, an enum cannot be a string type.

Specify the type after enum name as : type

The following defines the byte enum.

Example: byte Enum

enum Categories: byte

{

Electronics = 1,

Food = 5,

Automotive = 6,

Arts = 10,

BeautyCare = 11,

Fashion = 15

}

## **Conversion**

Explicit casting is required to convert from an enum type to its underlying integral type.

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine(WeekDays.Friday); //output: Friday

int day = (int) WeekDays.Friday; // enum to int conversion

Console.WriteLine(day); //output: 4

var wd = (WeekDays) 5; // int to enum conversion

Console.WriteLine(wd);//output: Saturday

# C# - Interface

An interface includes the declarations of related functionalities. The entities that implement the interface must provide the implementation of declared functionalities.

In C#, an interface can be defined using the interface keyword.

An interface can contain declarations of methods, properties, indexers, and events. However, it cannot contain fields, auto-implemented properties.

interface IFile

{

void ReadFile();

void WriteFile(string text);

}

1. You cannot apply access modifiers to interface members.
2. All the members are public by default.
3. If you use an access modifier in an interface, then the C# compiler will give a compile-time error.

An interface can only contain declarations but not implementations.

## **Implementing an Interface**

A class or a Struct can implement one or more interfaces using colon (:).

Example: Interface Implementation

interface IFile

{

void ReadFile();

void WriteFile(string text);

}

class FileInfo : IFile

{

public void ReadFile()

{

Console.WriteLine("Reading File");

}

public void WriteFile(string text)

{

Console.WriteLine("Writing to file");

}

}

 Note:

Interface members must be implemented with the public modifier; otherwise, the compiler will give compile-time errors.

## **Implementing Multiple Interfaces**

A class or struct can implement multiple interfaces. It must provide the implementation of all the members of all interfaces.

Example: Implement Multiple Interfaces

interface IFile

{

void ReadFile();

}

interface IBinaryFile

{

void OpenBinaryFile();

void ReadFile();

}

class FileInfo : IFile, IBinaryFile

{

void IFile.ReadFile()

{

Console.WriteLine("Reading Text File");

}

void IBinaryFile.OpenBinaryFile()

{

Console.WriteLine("Opening Binary File");

}

void IBinaryFile.ReadFile()

{

Console.WriteLine("Reading Binary File");

}

public void Search(string text)

{

Console.WriteLine("Searching in File");

}

}

# C# - Dictionary<TKey, TValue>

The Dictionary<TKey, TValue> is a generic collection that stores key-value pairs in no particular order.

## **Dictionary Characteristics**

1. Dictionary<TKey, TValue> stores key-value pairs.
2. Comes under System.Collections.Generic namespace.
3. Implements [IDictionary<TKey, TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.idictionary-2?view=netframework-4.8" \t "_blank) interface.
4. Keys must be unique and cannot be null.
5. Values can be null or duplicate.
6. Values can be accessed by passing associated key in the indexer e.g. myDictionary[key]
7. Elements are stored as [KeyValuePair<TKey, TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.keyvaluepair-2?view=netframework-4.8" \t "_blank) objects.