Hay dos clases de tipos en C#: *tipos de valor* y *tipos de referencia*. Las variables de tipos de valor contienen directamente los datos, mientras que las variables de los tipos de referencia almacenan referencias a los datos, lo que se conoce como objetos. Con los tipos de referencia, es posible que dos variables hagan referencia al mismo objeto y que, por tanto, las operaciones en una variable afecten al objeto al que hace referencia la otra variable. Con los tipos de valor, cada variable tiene su propia copia de los datos y no es posible que las operaciones en una variable afecten a la otra (excepto en el caso de las variables de parámetro `ref` y `out`).

Los tipos de valor de C# se dividen en *tipos simples*, *tipos de enumeración*, *tipos de estructura* y *tipos de valores NULL*. Los tipos de referencia de C# se dividen en *tipos de clase*, *tipos de interfaz*, *tipos de matriz* y *tipos delegados*.

A continuación se proporciona información general del sistema de tipos de C#.

- Tipos de valor
  - Tipos simples
    - Entero con signo: sbyte, short, int,long
    - Entero sin signo: byte, ushort, uint,ulong
    - Caracteres Unicode: char
    - Punto flotante binario IEEE: float, double
    - Punto flotante decimal de alta precisión: decimal
    - Booleano: bool
  - Tipos de enumeración
    - Tipos definidos por el usuario con el formato enum E {...}
  - Tipos de estructura
    - Tipos definidos por el usuario con el formato struct S {...}
  - Tipos de valor que aceptan valores NULL
    - Extensiones de todos los demás tipos de valor con un valor null
- Tipos de referencia
  - Tipos de clase
    - Clase base definitiva de todos los demás tipos: object
    - Cadenas Unicode: string
    - Tipos definidos por el usuario con el formato class C {...}
  - Tipos de interfaz
    - Tipos definidos por el usuario con el formato interface I {...}
  - Tipos de matriz
    - Unidimensional y multidimensional; por ejemplo, int[] y int[,]
  - Tipos delegados
    - Tipos definidos por el usuario con el formato delegate int D(...)

Para obtener más información sobre los tipos numéricos, vea Tabla de tipos enteros y Tabla de tipos de punto flotante.

El tipo bool de C# se utiliza para representar valores booleanos; valores que son true o false.

El procesamiento de caracteres y cadenas en C# utiliza la codificación Unicode. El tipo charrepresenta una unidad de código UTF-16 y el tipo string representa una secuencia de unidades de código UTF-16.

Los programas de C# utilizan declaraciones de tipos para crear nuevos tipos. Una declaración de tipos especifica el nombre y los miembros del nuevo tipo. Cinco de las categorías de tipos de C# las define el usuario: tipos de clase, tipos de estructura, tipos de interfaz, tipos de enumeración y tipos delegados.

A tipo class define una estructura de datos que contiene miembros de datos (campos) y miembros de función (métodos, propiedades y otros). Los tipos de clase admiten herencia única y polimorfismo, mecanismos por los que las clases derivadas pueden extender y especializar clases base.

Un tipo struct es similar a un tipo de clase, por el hecho de que representa una estructura con miembros de datos y miembros de función. Sin embargo, a diferencia de las clases, las estructuras son tipos de valor y no suelen requerir la asignación del montón. Los tipos struct no admiten la herencia especificada por el usuario y todos los tipos de struct se heredan implícitamente del tipo object.

Un tipo interface define un contrato como un conjunto con nombre de miembros de función públicos. Un class o struct que implementa un interface debe proporcionar implementaciones de miembros de función de la interfaz. Un interface puede heredar de varias interfaces base, y un class o struct pueden implementar varias interfaces.

Un tipo delegate representa las referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto. Los delegados permiten tratar métodos como entidades que se puedan asignar a variables y se puedan pasar como parámetros. Los delegados son análogos a los tipos de función proporcionados por los lenguajes funcionales. Son similares al concepto de punteros de función en otros lenguajes, pero a diferencia de los punteros de función, los delegados están orientados a objetos y presentan seguridad de tipos.

Los tipos class, struct, interface y delegate admiten parámetros genéricos, mediante los cuales se pueden parametrizar con otros tipos.

Un tipo enum es un tipo distinto con constantes con nombre. Cada tipo enum tiene un tipo subyacente, que debe ser uno de los ocho tipos

enteros. El conjunto de valores de un tipo enum es igual que el conjunto de valores del tipo subyacente.

C# admite matrices unidimensionales y multidimensionales de cualquier tipo. A diferencia de los tipos enumerados anteriormente, los tipos de matriz no tienen que ser declarados antes de usarlos.En su lugar, los tipos de matriz se crean mediante un nombre de tipo entre corchetes. Por ejemplo, int[] es una matriz unidimensional de int, int[,] es una matriz bidimensional de int y int[][]es una matriz unidimensional de la matriz unidimensional de int.

Los tipos de valor NULL tampoco tienen que ser declarados antes de usarlos. Para cada tipo de valor distinto de NULL T, existe un tipo de valor NULL correspondiente T?, que puede tener un valor adicional, null. Por ejemplo, int? es un tipo que puede contener cualquier número entero de 32 bits o el valor null.

Hay varios tipos de variables en C#, entre otras, campos, elementos de matriz, variables locales y parámetros. Las variables representan ubicaciones de almacenamiento, y cada variable tiene un tipo que determina qué valores pueden almacenarse en la variable, como se muestra a continuación.

- Tipo de valor distinto a NULL
  - Un valor de ese tipo exacto
- Tipos de valor NULL
  - Un valor null o un valor de ese tipo exacto
- objeto
  - Una referencia null, una referencia a un objeto de cualquier tipo de referencia o una referencia a un valor de conversión boxing de cualquier tipo de valor
- Tipo de clase
  - Una referencia null, una referencia a una instancia de ese tipo de clase o una referencia a una instancia de una clase derivada de ese tipo de clase
- Tipo de interfaz
  - Un referencia null, una referencia a una instancia de un tipo de clase que implementa dicho tipo de interfaz o una referencia a un valor de conversión boxing de un tipo de valor que implementa dicho tipo de interfaz
- Tipo de matriz
  - Una referencia null, una referencia a una instancia de ese tipo de matriz o una referencia a una instancia de un tipo de matriz compatible
- Tipo delegado
  - Una referencia null o una referencia a una instancia de un tipo delegado compatible