C# Data Types

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

- **Value Data Types**
- **Reference Data Types**
- **Pointer Data Type**

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 :
    - **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
byte	System.byte	unsigned integer	8	0 to 255	0
uint	System.UInt32	unsigned integer	32	0 to 2^{32}	0
long	System.Int64	signed integer	64	-2^{63} to $2^{63}-1$	0L
ushort	System.UInt16	unsigned integer	16	0 to 65535	0
ulong	System.UInt64	unsigned integer	64	0 to 2^{63}	0

- **Floating Point Types :**There are 2 floating point data types which contain the decimal point.
  - **Float**: It is **32-bit single-precision** floating point type. It has 7 digit Precision. To initialize a float

variable, use the suffix f or F. Like, float x = 3.5F;. If the suffix F or f will not use then it is treated as double.

- **Double**:It is **64-bit double-precision** floating point type. It has 14 – 15 digit Precision. To initialize a double variable, use the suffix d or D. But it is not mandatory to use suffix because by default floating data types are the double type.

Alias	Type name	Size(bits)	Range (aprox)	Default Value
float	System.Single	32	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	0.0F
double	System.Double	64	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	0.0D

- **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	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9228 \times 10^{28}$	0.0M

- **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'

- **Example :**

```
// C# program to demonstrate
// the above data types
using System;
namespace ValueTypeTest {

class GeeksforGeeks {
```

```
// Main function
static void Main()
{

    // declaring character
    char a = 'G';

    // Integer data type is generally
    // used for numeric values
    int i = 89;

    short s = 56;

    // this will give error as number
    // is larger than short range
    // short s1 = 87878787878;

    // long uses Integer values which
```

```
// may signed or unsigned
long l = 4564;

// UInt data type is generally
// used for unsigned integer values
uint ui = 95;

ushort us = 76;
// this will give error as number is
// larger than short range

// ulong data type is generally
// used for unsigned integer values
ulong ul = 3624573;

// by default fraction value
// is double in C#
double d = 8.358674532;
```

```
            // for float use 'f' as suffix
            float f = 3.7330645f;

            // for float use 'm' as suffix
            decimal dec = 389.5m;

            Console.WriteLine("char: " + a);
            Console.WriteLine("integer: " + i);
            Console.WriteLine("short: " + s);
            Console.WriteLine("long: " + l);
            Console.WriteLine("float: " + f);
            Console.WriteLine("double: " + d);
            Console.WriteLine("decimal: " + dec);
            Console.WriteLine("Unsinged integer: " + ui);
            Console.WriteLine("Unsinged short: " + us);
            Console.WriteLine("Unsinged long: " + ul);
        }
    }
}
```

- **Output :**
- char: G
- integer: 89
- short: 56
- long: 4564
- float: 3.733064
- double: 8.358674532
- decimal: 389.5
- Unsinged integer: 95
- Unsinged short: 76
- Unsinged long: 3624573
- **Example :**

```
// C# program to demonstrate the Sbyte
// signed integral data type
using System;
namespace ValueTypeTest {

class GeeksforGeeks {
```

```
// Main function
static void Main()
{
    sbyte a = 126;

    // sbyte is 8 bit
    // singned value
    Console.WriteLine(a);

    a++;
    Console.WriteLine(a);

    // It overflows here because
    // byte can hold values
    // from -128 to 127
    a++;
    Console.WriteLine(a);

    // Looping back within
```

```
            // the range
            a++;
            Console.WriteLine(a);
        }
    }
    }
```

- **Output :**
- 126
- 127
- -128
- -127
- **Example :**

```
    // C# program to demonstrate
    // the byte data type
    using System;
    namespace ValueTypeTest {

    class GeeksforGeeks {
```

```
// Main function
static void Main()
{
    byte a = 0;

    // byte is 8 bit
    // unsigned value
    Console.WriteLine(a);

    a++;
    Console.WriteLine(a);

    a = 254;

    // It overflows here because
    // byte can hold values from
    // 0 to 255
    a++;
```

```
            Console.WriteLine(a);


            // Looping back within the range
            a++;
            Console.WriteLine(a);
        }
    }
    }
```

- **Output :**
- 0
- 1
- 255
- 0
- **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

- **Example :**

```
// C# program to demonstrate the
// boolean data type
using System;
namespace ValueTypeTest {

    class GeeksforGeeks {

    // Main function
    static void Main()
    {

        // boolean data type
        bool b = true;
        if (b == true)
            Console.WriteLine("Hi Geek");
```

```
    }
}
}
```

- **Output :**
- Hi Geek

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.**
    - **String :** It represents a sequence of Unicode characters and its type name is **System.String**. So, string and String are equivalent.
      Example :
    - string s1 = "hello"; // creating through string keyword
    - String s2 = "welcome"; // creating through String class
    - **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 :

```
// C# program to demonstrate
// the Reference data types
using System;
namespace ValueTypeTest {

class GeeksforGeeks {

    // Main Function
    static void Main()
    {

        // declaring string
        string a = "Geeks";

        //append in a
        a+="for";
```

```
            a = a+"Geeks";
            Console.WriteLine(a);

            // declare object obj
            object obj;
            obj = 20;
            Console.WriteLine(obj);

            // to show type of object
            // using GetType()
            Console.WriteLine(obj.GetType());
        }
    }
    }
```

Output :

```
GeeksforGeeks
20
System.Int32
```

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 :
4. type* identifier;

Example :

int* p1, p; // Valid syntax

int *p1, *p; // Invalid

Example :

```
// Note: This program will not work on
// online compiler
// Error: Unsafe code requires the `unsafe'
// command line option to be specified
// For its solution:
// Go to your project properties page and
// check under Build the checkbox Allow
```

```
// unsafe code.
using System;
namespace Pointerprogram {

class GFG {

    // Main function
    static void Main()
    {
        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);
        }
    }
}
}
```