Programiz

C# Variables and (Primitive) Data Types

In this tutorial, we will learn about variables, how to create variables in C# and different data types that C# programming language supports.

A variable is a symbolic name given to a memory location. Variables are used to store data in a computer program.

How to declare variables in C#?

Here's an example to declare a variable in C#.

```
int age;
```

In this example, a variable `age` of type `int` (integer) is declared and it can only store integer values.

We can assign a value to the variable later in our p Contents h:

```
int age;
... ... ...
age = 24;
```

However, the variable can also be initialized to some value during declaration. For example,

```
int age = 24;
```

Here, a variable `age` of type `int` is declared and initialized to `24` at the same time.

Since, it's a variable, we can change the value of variables as well. For example,

```
int age = 24;
age = 35;
```

Here, the value of `age` is changed to 35 from 24.

Variables in C# must be declared before they can be used. This means, the name and type of variable must be known before they can be assigned a value. This is why C# is called a statically-typed language.

Once declared, the datatype of a variable can not be changed within a scope. A scope can be thought as a block of code where the variable is visible or available to use. If you don't understand the previous statement, don't worry we'll learn about scopes in the later chapters.

For now remember,we can not do the following in C#:

Contents

```
int age;
age = 24;
... ... ...
float age;
```

Implicitly typed variables

Alternatively in C#, we can declare a variable without knowing its type using `var` keyword. Such variables are called **implicitly typed local variables**.

Variables declared using `var` keyword must be initialized at the time of declaration.

```
var value = 5;
```

The compiler determines the type of variable from the value that is assigned to the variable. In the above example, `value` is of type `int`. This is equivalent to:

```
int value;
value = 5;
```

You can learn more about implicitly typed local variables.

Rules for Naming Variables in C#

There are certain rules we need to follow while naming a variable. The rules for naming a variable in C# are:

Contents

1. The variable name can contain letters (uppercase and lowercase), underscore(_) and digits only.

2. The variable name must start with either letter, underscore or @ symbol. For example,

Rules for naming variables in C#	
Variable Names	Remarks
`name`	Valid
`subject101`	Valid
`_age`	Valid (Best practice for naming private member variables)
`@break`	Valid (Used if name is a reserved keyword)
`101subject`	Invalid (Starts with digit)
`your_name`	Valid
`your name`	Invalid (Contains whitespace)

3. C# is case sensitive. It means `age` and `Age` refers to 2 different variables.
4. A variable name must not be a C# keyword. For example, `if`, `for`, `using` can not be a variable name. We will be discussing more about C# keywords in the next tutorial.

Best Practices for Naming a Variable

1. Choose a variable name that make sense. For example, `name`, `age`, `subject` makes more sense than `n`, `a` and `s`.
2. Use **camelCase** notation (starts with lowercase le Contents r local variables. For example, `numberOfStudents`, `age`, etc.

3. Use **PascalCase** or **CamelCase** (starts with uppercase letter) for naming public member variables. For example, `FirstName` , `Price` , etc.
4. Use a leading underscore (_) followed by **camelCase** notation for naming private member variables. For example, `_bankBalance` , `_emailAddress` , etc.

You can learn more about naming conventions in C# here.

Don't worry about public and private member variables. We will learn about them in later chapters.

C# Primitive Data Types

Variables in C# are broadly classified into two types: **Value types** and **Reference types**. In this tutorial we will be discussing about primitive (simple) data types which is a subclass of Value types.

Reference types will be covered in later tutorials. However, if you want to know more about variable types, visit C# Types and variables (official C# docs).

Boolean (bool)

- Boolean data type has two possible values: `true` or `false`
- **Default value**: `false`
- Boolean variables are generally used to check conditions such as in *if statements*, *loops*, etc.

For Example:

```
using System;
namespace DataType
{
    class BooleanExample
    {
        public static void Main(string[] args)
```

```
        {
            bool isValid = true;
            Console.WriteLine(isValid);
        }
    }
}
```

When we run the program, the output will be:

```
True
```

Signed Integral

These data types hold integer values (both positive and negative). Out of the total available bits, one bit is used for sign.

1. sbyte

- **Size**: 8 bits
- **Range**: -128 to 127.
- **Default value**: 0

For example:

```
using System;
namespace DataType
{
    class SByteExample
    {
        public static void Main(string[] args)
        {
```

Contents

```
            sbyte level = 23;
            Console.WriteLine(level);
        }
    }
}
```

When we run the program, the output will be:

```
23
```

Try assigning values out of range i.e. less than -128 or greater than 127 and see what happens.

---

2. short

- **Size**: 16 bits
- **Range**: -32,768 to 32,767
- **Default value**: 0

For example:

```
using System;
namespace DataType
{
    class ShortExample
    {
        public static void Main(string[] args)
        {
            short value = -1109;
            Console.WriteLine(value);
        }
    }
}
```

Contents

When we run the program, the output will be:

```
-1109
```

---

3. int

- **Size**: 32 bits
- **Range**: -231 to 231-1
- **Default value**: 0

For example:

```
using System;
namespace DataType
{
     class IntExample
     {
          public static void Main(string[] args)
          {
               int score = 51092;
               Console.WriteLine(score);
          }
     }
}
```

When we run the program, the output will be:

```
51092
```

Contents

4. long

- **Size**: 64 bits
- **Range**: -263 to 263-1
- **Default value**: `0L` [L at the end represent the value is of long type]

For example:

```
using System;
namespace DataType
{
     class LongExample
     {
          public static void Main(string[] args)
          {
               long range = -7091821871L;
               Console.WriteLine(range);
          }
     }
}
```

When we run the program, the output will be:

```
-7091821871
```

Unsigned Integral

These data types only hold values equal to or greater than 0. We generally use these data types to store values when we are sure, we won't have negative values.

Contents

1. byte

- **Size**: 8 bits
- **Range**: 0 to 255.
- **Default value**: 0

For example:

```
using System;
namespace DataType
{
      class ByteExample
      {
            public static void Main(string[] args)
            {
                  byte age = 62;
                  Console.WriteLine(level);
            }
      }
}
```

When we run the program, the output will be:

```
 62
```

---

2. ushort

- **Size**: 16 bits
- **Range**: 0 to 65,535
- **Default value**: 0

Contents

For example:

```
using System;
namespace DataType
{
      class UShortExample
      {
            public static void Main(string[] args)
            {
                  ushort value = 42019;
                  Console.WriteLine(value);
            }
      }
}
```

When we run the program, the output will be:

```
42019
```

---

3. uint

- **Size**: 32 bits
- **Range**: 0 to 232-1
- **Default value**: 0

For example:

```
using System;
namespace DataType
{
      class UIntExample
      {
```

Contents

```
            public static void Main(string[] args)
            {
                  uint totalScore = 1151092;
                  Console.WriteLine(totalScore);
            }
      }
}
```

When we run the program, the output will be:

```
1151092
```

---

4. ulong

- **Size**: 64 bits
- **Range**: 0 to 264-1
- **Default value**: 0

For example:

```
using System;
namespace DataType
{
      class ULongExample
      {
            public static void Main(string[] args)
            {
                  ulong range = 17091821871L;
                  Console.WriteLine(range);
            }
      }
}
```

Contents

When we run the program, the output will be:

```
17091821871
```

Floating Point

These data types hold floating point values i.e. numbers containing decimal values. For example, 12.36, -92.17, etc.

1. float

- Single-precision floating point type
- **Size**: 32 bits
- **Range**: 1.5 × 10−45 to 3.4 × 1038
- **Default value**: 0.0F [F at the end represent the value is of float type]

For example:

```
using System;
namespace DataType
{
    class FloatExample
    {
        public static void Main(string[] args)
        {
            float number = 43.27F;
            Console.WriteLine(number);
        }
    }
}
```

Contents

When we run the program, the output will be:

```
43.27
```

2. double

- Double-precision floating point type. What is the difference between single and double precision floating point?
- **Size**: 64 bits
- **Range**: 5.0 × 10−324 to 1.7 × 10308
- **Default value**: 0.0D [D at the end represent the value is of double type]

For example:

```
using System;
namespace DataType
{
    class DoubleExample
    {
        public static void Main(string[] args)
        {
            double value = -11092.53D;
            Console.WriteLine(value);
        }
    }
}
```

When we run the program, the output will be:

```
-11092.53
```

Contents

Character (char)

- It represents a 16 bit unicode character.
- **Size**: 16 bits
- **Default value**: '\0'
- **Range**: U+0000 ('\u0000') to U+FFFF ('\uffff')

For example:

```
using System;
namespace DataType
{
      class CharExample
      {
            public static void Main(string[] args)
            {
                  char ch1 ='\u0042';
                  char ch2 = 'x';
                  Console.WriteLine(ch1);
                  Console.WriteLine(ch2);
            }
      }
}
```

When we run the program, the output will be:

```
B
x
```

The unicode value of `'B'` is `'\u0042'`, hence printing `ch1` will print `'B'`.

Decimal

Contents

- Decimal type has more precision and a smaller range as compared to floating point types (double and float). So it is appropriate for monetary calculations.
- **Size**: 128 bits
- **Default value**: 0.0M [M at the end represent the value is of decimal type]
- **Range**: (-7.9 x 1028 to 7.9 x 1028) / (100 to 28)

For example:

```
using System;
namespace DataType
{
     class DecimalExample
     {
          public static void Main(string[] args)
          {
            decimal bankBalance = 53005.25M;
                Console.WriteLine(bankBalance);
          }
     }
}
```

When we run the program, the output will be:

```
53005.25
```

The suffix `M` or `m` must be added at the end otherwise the value will be treated as a double and an error will be generated.

C# Literals

Contents

Let's look at the following statement:

```
int number = 41;
```

Here,

- `int` is a data type
- `number` is a variable and
- `41` is a literal

Literals are fixed values that appear in the program. They do not require any computation. For example, `5`, `false`, `'w'` are literals that appear in a program directly without any computation.

Boolean Literals

- true and false are the available boolean literals.
- They are used to initialize boolean variables.

For example:

```
bool isValid = true;
bool isPresent = false;
```

Integer Literals

- Integer literals are used to initialize variables of integer data types i.e. `sbyte`, `short`, `int`, `long`, `byte`, `ushort`, `uint` and `ulong`.
- If an integer literal ends with `L` or `l`, it is of ty Contents t practice use `L` (not `l`).

```
long value1 = 4200910L;
long value2 = -10928190L;
```

- If an integer literal starts with a `0x`, it represents hexadecimal value. Number with no prefixes are treated as decimal value. Octal and binary representation are not allowed in C#.

```
int decimalValue = 25;
int hexValue = 0x11c;// decimal value 284
```

Floating Point Literals

- Floating point literals are used to initialize variables of float and double data types.
- If a floating point literal ends with a suffix `f` or `F`, it is of type float. Similarly, if it ends with `d` or `D`, it is of type double. If neither of the suffix is present, it is of type double by **default**.
- These literals contains e or E when expressed in scientific notation.

```
double number = 24.67;// double by default
float value = -12.29F;
double scientificNotation = 6.21e2;// equivalent to 6.21 x 102 i.e. 621
```

Character and String Literals

- Character literals are used to initialize variables of char data types.
- Character literals are enclosed in single quotes. For example, `'x'`, `'p'`, etc.
- They can be represented as character, hexadecimal escape sequence, unicode representation or integral values casted to char.


```
char ch1 = 'R';// character
char ch2 = '\x0072';// hexadecimal
```

```
char ch3 = '\u0059';// unicode
char ch4 = (char)107;// casted from integer
```

- String literals are the collection of character literals.
- They are enclosed in double quotes. For example, `"Hello"`, `"Easy Programming"`, etc.

```
string firstName = "Richard";
string lastName = " Feynman";
```

- C# also supports escape sequence characters such as:

Character	Meaning
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash
`\n`	Newline
`\r`	Carriage return
`\t`	Horizontal Tab
`\a`	Alert
`\b`	Backspace

Contents

C# Tutorials

Contents

Contents

Contents

TUTORIALS

Python Tutorials

C Tutorials

Java Tutorials

Kotlin Tutorials

C++ Tutorials

Swift Tutorials

R Tutorials

Algorithms Tutorials

EXAMPLES

Python Examples

C Examples

Java Examples

Kotlin Examples

C++ Examples

R Examples

COMPANY

About

Contents

Contents