# C# Type Conversion

The process of converting the value of one type (int, float, double, etc.) to another type is known as type conversion.

In C#, there are two basic types of type conversion:

1. **Implicit Type Conversions**
2. **Explicit Type Conversions**

## 1. Implicit Type Conversion in C#

In implicit type conversion, the C# compiler automatically converts one type to another.

Generally, smaller types like int (having less memory size) are automatically converted to larger types like double (having larger memory size).

### Example: Implicit Type Conversion

using System;

namespace MyApplication {

class Program {

static void Main(string[] args) {

int numInt = 500;

// get type of numInt

Type n = numInt.GetType();

// Implicit Conversion

double numDouble = numInt;

// get type of numDouble

Type n1 = numDouble.GetType();

// Value before conversion

Console.WriteLine("numInt value: "+numInt);

Console.WriteLine("numInt Type: " + n);

// Value after conversion

Console.WriteLine("numDouble value: "+numDouble);

Console.WriteLine("numDouble Type: " + n1);

Console.ReadLine();

}

}

}

**Output**

numInt value: 500

numInt Type: System.Int32

numDouble value: 500

numDouble Type: System.Double

In the above example, we have created an int type variable named numInt.

Notice the line,

// Implicit Conversion

double numDouble = numInt;

Here, we are assigning the int type variable to a double type variable. In this case, the C# compiler automatically converts the int type value to double.

Notice that we have used the GetType() method to check the type of numInt and numDouble variables.

**Note**: In implicit type conversion, smaller types are converted to larger types. Hence, there is no loss of data during the conversion.

## 2. C# Explicit Type Conversion

In explicit type conversion, we explicitly convert one type to another.

Generally, larger types like double (having large memory size) are converted to smaller types like int (having small memory size).

### Example: Explicit Type Conversion

using System;

namespace MyApplication {

class Program {

static void Main(string[] args) {

double numDouble = 1.23;

// Explicit casting

int numInt = (int) numDouble;

// Value before conversion

Console.WriteLine("Original double Value: "+numDouble);

// Value before conversion

Console.WriteLine("Converted int Value: "+numInt);

Console.ReadLine();

}

}

}

**Output**

Original double value: 1.23

Converted int value: 1

In the above example, we have created a double variable named numDouble. Notice the line,

// Explicit casting

int numInt = (int) numDouble;

Here, (int) is a **cast expression** that explicitly converts the double type to int type.

We can see the original value is **1.23** whereas the converted value is **1**. Here, some data is lost during the type conversion. This is because we are explicitly converting the larger data type double to a smaller type int.

**Note**: The explicit type conversion is also called type casting.

## C# Type Conversion using Parse()

In C#, we can also use the Parse() method to perform type conversion.

Generally, while performing type conversion between non-compatible types like int and string, we use Parse().

### Example: Type Conversion using Parse()

using System;

namespace Conversion {

class Program {

static void Main(string[] args) {

string n = "100";

// converting string to int type

int a = int.Parse(n);

Console.WriteLine("Original string value: "+n);

Console.WriteLine("Converted int value: "+a);

Console.ReadLine();

}

}

}

**Output**

Original string value: 100

Converted int value: 100

In the above example, we have converted a string type to an int type.

// converting string to int type

int a = int.Parse(n);

Here, the Parse() method converts the numeric string **100** to an integer value.

**Note**: We cannot use Parse() to convert a textual string like "test" to an int. For example,

String str = "test";

int a = int.Parse(str); // Error Code

## C# Type Conversion using Convert Class

In C#, we can use the Convert class to perform type conversion. The Convert class provides various methods to convert one type to another.

|  |  |
| --- | --- |
| **Method** | **Description** |
| ToBoolean() | converts a type to a Boolean value |
| ToChar() | converts a type to a char type |
| ToDouble() | converts a type to a double type |
| ToInt16() | converts a type to a 16-bit int type |
| ToString() | converts a type to a string |

Let us look at some examples:

### Example: Convert int to String and Double

using System;

using System;

namespace Conversion {

class Program {

static void Main(string[] args) {

// create int variable

int num = 100;

Console.WriteLine("int value: " + num);

// convert int to string

string str = Convert.ToString(num);

Console.WriteLine("string value: " + str);

// convert int to Double

Double doubleNum = Convert.ToDouble(num);

Console.WriteLine("Double value: " + doubleNum);

Console.ReadLine();

}

}

}

**Output**

int value: 100

string value: 100

Double value: 100

In the above example,

* **Convert.ToString(a)** - converts an int type num to string
* **Convert.ToDouble(a)** - converts num to the Double type

### Example: Convert string to Double and vice-versa

using System;

namespace Conversion {

class Program {

static void Main(string[] args) {

// create string variable

string str = "99.99";

Console.WriteLine("Original string value: " + str);

// convert string to double

Double newDouble = Convert.ToDouble(str);

Console.WriteLine("Converted Double value: " + newDouble);

// create double variable

double num = 88.9;

Console.WriteLine("Original double value: " + num);

// converting double to string

string newString = Convert.ToString(num);

Console.WriteLine("Converted string value: " + newString);

Console.ReadLine();

}

}

}

**Output**

Original string value: 99.99

Converted Double value: 99.99

Original double value: 88.9

Converted string value: 88.9

In the above example,

* **Convert.ToDouble(str)**- converts a string type str to Double
* **Convert.ToString(num)** - converts a double type num to the string

### Example 3: Convert int to Boolean

using System;

namespace Conversion {

class Program {

static void Main(string[] args) {

// create int variables

int num1 = 0;

int num2 = 1;

// convert int to Boolean

Boolean bool1 = Convert.ToBoolean(num1);

Boolean bool2 = Convert.ToBoolean(num2);

Console.WriteLine("Boolean value of 0 is: " + bool1);

Console.WriteLine("Boolean value of 1 is: " + bool2);

Console.ReadLine();

}

}

}

**Output**

Boolean value of 0 is: False

Boolean value of 1 is: True

In the above example, we have created two integer variables: num1 and num2 with values **0** and **1** respectively. Here,

* **Convert.ToBoolean(num1)** - converts **0** to a Boolean value False
* **Convert.ToBoolean(num2)** - converts **1** to a Boolean value True

**Note**: In C#, the integer value **0** is considered False and all other values are considered True.

Type conversion or Type Casting in C#

When the variable of one data type is changed to another data type is known as the **Type Casting**. According to our needs, we can change the type of data. At the time of the compilation, C# is a statically-typed i.e., after the declaration of the variable, we cannot declare it again. The value of the variable cannot be assigned to another type of variable unless we implicitly change the type of the variable.

Here we will take an example of the string data type. We cannot convert the string implicitly to the int. Therefore, if we declare the variable 'i' as an int, we cannot assign the string value "Hello" into it.

1. **int** z;
2. z = "hii"; // error CS0029:  we cannot implicitly convert string type' to 'integer' type

However, we can face a situation when there is a need to copy the value of one variable into another variable or method parameter of another type. For example, we have a variable integer, and we need to pass it to a method parameter whose type is **double**. Or the situation can be to assign the class variable to the variable of the type of interface. These types of operations are called **Type Conversion**.

3.6M

375

Method Overloading vs Overriding in Java

## In C#, we can perform a different kinds of conversions.

* **Implicit Conversion:** For the implicit conversion, there is not any need for the special syntax. This type of conversion is safe; in this conversion, there is not any loss of the data. Implicit conversions include the conversion of the small type to large integral types, and from the derived class to the base class conversion.
* **Explicit Conversion (Type Caste):** Explicit conversion will be done with the cast operator (). We will do the casting when there is the situation of the data loss, or when the conversion is not succeeded. There can be any other reason for the explicit conversion. The example of the casting is the conversion of the numeric type to the less precision or smaller range. Explicit Conversion also includes the conversion of the base-class instance to the derived class.
* **User-Defined Conversion:** We can do this conversion by defining the method. We can use the technique to enable the explicit-implicit conversion between the custom type, which does not have any relationship with the base-class or derived-class.

**Conversion with the helper class:** For the conversion of the non-compatible type like **integer** and **System.DateTime** objects or hexadecimal strings and byte arrays, we can use System.BitConversion class, System. Convert class and the Parse methods of the built-in numeric type like as int32 Parse.

**Implicit Conversion:** We can easily understand and use the implicit conversion. Here we are going to assign the integer to the double is known as the implicit conversion because we are haven't lost any data in this conversion.

To understand this conversion here, we are going to take an example.

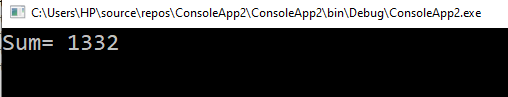
1. Int value1=567;
2. Int value2=765;
3. Long sum;
4. sum=value1+value2

Here we have two variable integer type value1 and value2. We are going to do the sum of two integer type value and store result in the long variable. Here this will not show any error, and we do not lose any data. This type of conversion is known as implicit conversion.

### Example:

1. using System;
2. namespace ConsoleApp2
3. {
4. **class** SumProgramme
5. {
6. **static** **void** Main(string[] args)
7. {
8. **int** value1 = 567;
9. **int** value2 = 765;
10. **long** summation;
12. summation = value1 + value2;
14. Console.WriteLine("summation = " + summation);
16. Console.ReadLine();
17. }
18. }
19. }

**Output:**



### Explicit Conversion

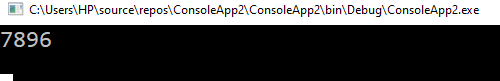
We can do the explicit conversion by defining the method. Users will do the explicit conversion. Users will do the conversion as per their requirements. The compiler will do the execution as per our command.

Now we will do this conversion by applying the following code:

### Example:

1. using System;
2. using System.Collections.Generic;
3. using System. Linq;
4. using System. Text;
5. using System.Threading.Tasks;
7. namespace ConsoleApp2
8. {
9. **class** ProgramExplicit
10. {
11. **static** **void** Main(string[] args)
12. {
13. **double** db = 7896.45;
14. **int** xy;
16. // here we do the cast double to int.
17. xy = (**int**)db;
18. Console.WriteLine(xy);
19. Console.ReadKey();
20. }
21. }
22. }

**Output:**



### Conversion Operators

Conversion Operators have the following properties:

* Conversions declared as "implicit" will occur automatically when it is required.
* Conversions declared as "explicit" will require the cast to be called.
* All the conversions must be declared as positive.

Now we will take an example that follows the following code.

1. using System;
2. using System.Collections.Generic;
3. using System. Linq;
4. using System. Text;
5. using System.Threading.Tasks;
7. namespace UserDefinedConversion
8. {
9. **class** Program
10. {
12. **public** struct ImperialMeasurement
13. {
14. **public** **float** feet;
15. **public** ImperialMeasurement(**float** r)
16. {
17. **this**.feet = r;
18. }
19. **public** **static** explicit operator ImperialMeasurement(**int** m)
20. {
21. **float** ConversionResult = 3.28f \* m;
22. ImperialMeasurement temp = **new** ImperialMeasurement(ConversionResult);
23. **return** temp;
24. }
25. }
26. **static** **void** Main(string[] args)
27. {
28. Console.WriteLine("Please enter a whole number measurement in meters");
29. **int** nm = Convert.ToInt32(Console.ReadLine());
30. ImperialMeasurement im = (ImperialMeasurement)nm;
31. Console.WriteLine($"The measument of {nm} in meters is {im.feet} in feet ");
32. Console.ReadKey();
33. }
34. }

37. }

**Output:**

