Data Type Conversions in C#

- The process of converting the value of one data type (int, float, double, etc.) to another data type is known as data type conversion.
- This is important for ensuring that data is correctly interpreted and manipulated according to the needs of your program.
- In C#, there are two types of casting:
    - Implicit conversion
    - Explicit conversion

Implicit Conversion

- When two data types are compatible implicit casting can automatically converts a smaller data type to larger data type .
- Generally, smaller types like int (having less memory size) are automatically converted to larger types like double (having larger memory size).
- When use implicit casting there is no data loss.
- char -> int -> long -> float -> double
- in This type of casting does not require explicit instructions from the programmer.
- **Note: In implicit type conversion, smaller types are converted to larger types. Hence, there is no loss of data during the conversion.**

Automatic conversion of a value from one data type to another data type by a programming language, without the programmer specifically doing so, is **called implicit type conversion**. It happens whenever a binary operator has two operands of different data types. Depending on the operator, one of the operands is going to be converted to the data type of the other data type.

Explicit Conversion

- In implicit conversion, we saw that we can directly convert a derived class into base class without losing any data but in case if there is a chance of data loss then the compiler will require performing an explicit conversion.
- Explicit conversion converts a larger data type to smaller data type.
- These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.
- Explicit conversions are performed using **Parse()** and **TryParse()** method.
- Explicit conversions can result in data loss.
- can be convert
- double -> float -> long -> int -> char
- A data type can be converted into another data type by using methods present in the **convert** class or by using a **Parse** & **TryParse** method that is available for the various numeral types. **TryParse** is more useful if we are converting a string into the numeral.

  Ex : **int number = Int32.Parse("123");**

  Using Convert class :

  ```
  int number = 75;
  string s = Convert.ToString(number);
  ```

Difference Between Parse() and TryParse()

Parse() : We use the **Parse()** method to convert a string representation of a number to its numerical value. It returns the converted numerical value if the conversion is successful. The Parse() method throws an exception if the conversion fails.

The **Parse** method is designed to convert a string representation of a value into an instance of a particular type. This is commonly used when dealing with user input or reading from sources where data is in string format.

Basic syntax of Parse:

```
public static Type Parse(string s);
```

TryParse() :

In C#, the **TryParse** method is a more robust alternative to **Parse** for converting a string representation of a value into a specific data type. Unlike Parse, which throws exceptions if the conversion fails, **TryParse** provides a safer way to handle parsing by returning a boolean indicating success or failure. This method is available for many built-in types, including numeric types

The **TryParse()** method helps us avoid exceptions when parsing. The last parameter contains the result of the conversion and is preceded by the **out** keyword.

Ex :

```
string intString = "123";

bool success = int.TryParse(intString, out int number);

if (success)

{

        Console.WriteLine("Conversion successful: " + number);

}

else
{

        Console.WriteLine("Conversion failed.");

}
```

Implicit conversion example

```
using System;

namespace ConsoleApp1

{

   class Program

   {

       static void Main(string[] args)

           {

                    int x = 10;

                     //given number is in integer data type

                   //integer can be converted in double without data loss

                     double y = x;

                     Console.WriteLine(y);

           }

    }

}
```

Explicit conversion example

```
using System;

namespace ConsoleApp1

{

   class Program

  {

          static void Main(string[] args)

        {

                  float x = 10.1F;

                   //given number is in float data type

                  //float  cannot  implicit convert in integer and data loss.

                  //  It gives exception but use parse() can be converted

                  int y = int.Parse(x);

                  Console.WriteLine(y);

         }

   }

}
```

Arrays

- Array is a collection of elements of the same data type.
- They provide a way to manage and access multiple values through indexed positions. Arrays are zero-based, meaning that the index of the first element is 0.
- The data types of the elements may be any valid data type like char, int, float, etc. and the elements are stored in a contiguous location.

Syntax of array declaration and initialization

<Data_type>[] <array_name> = new <Data_type>[<size_Of_array>];

Ex: `int[] numbers = new int[3];`

Assign values to array

`int [] numbers = new int[3] { 1,2,3 }`

Assign values in different line

```
array1[0] = 1;
array1[1] = 2;
array1[2] = 3;
array1[3] = 4;
```

Types of arrays in C#.

1. Single-dimensional arrays
2. Multi-dimensional arrays

Single-dimensional arrays:

- Single-dimensional arrays are the simplest form of arrays.
- All items in a single-dimension array are stored contiguously, starting from 0 to the size of the Array -1.

Array Declaration

```
int[] intArray;

intArray = new int[3];
```

declaration and **initialization** an array of three items of integer type.

```
int[] Elements = new int[3] {1, 3, 5};
```

Multi-dimensional arrays:

- A multi-dimensional array, also known as a rectangular array, has more than one dimension.
- The form of a multi-dimensional array is a matrix.

Declaring a multi-dimensional array:

```
string[,] mutliDimStringArray;
```

Initializing multi-dimensional arrays:

```
int[,] numbers = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

string[,] names = new string[2, 2] { { "Rosy", "Amy" }, { "Peter", "Albert" } };
```

Accessing multi-dimensional arrays:

```
Console.WriteLine(numbers[0, 0]);
Console.WriteLine(numbers[0, 1]);
Console.WriteLine(numbers[1, 0]);
Console.WriteLine(numbers[1, 1]);
Console.WriteLine(numbers[2, 0]);
Console.WriteLine(numbers[2, 2]);
```

Array example

```
using System;

public class Numbers

{

   public static void Main(string[] args)

   {

      int[] numbers = new int[10];

      for (int i = 0; i < numbers.Length; i++)

       {

            numbers[i] = i + 1;

       }

       Console.WriteLine("Array elements:");

       for (int i = 0; i < numbers.Length; i++)

       {

          Console.WriteLine($"Element at index {i}: {numbers[i]}");

       }

   }

}
```

Length of the array specifies the number of elements present in the array.

Comments

- Comments can be used to explain code, and to make it more readable.
- C# compiler ignores comments

There are three types to add comment in code

- Single line comment - //
- Multi line comment - /* */
- XML Documentation comment - ///

If – Else statement in C# (Conditional Statements)

- C# provide a variety of **decision-making statements** that control the execution flow of a program based on specific logical conditions.
- **if statement:** The If statement is used to indicate which statement will execute according to the value of the given **boolean expression**. When the value of the boolean expression is **true**, then the if statement will execute the given then statement, **otherwise** it will return the control to the next statement after the if statement.
- **Else if statement :** Use **else if** to specify a new condition to test, if the (if condition) i.e first condition is false.
- **Else statement** : **else** statement is used with **if** statement to execute some block of code the given **if** condition is **false**.

Syntax of if statement : if (condition)

```
{
   // code if condition is true
}
```

Syntax of if-else / else if statement : if(condition1)

```
    {
       // code to be executed if condition1 is true
    }
    else if(condition2)
    {
       // code to be executed if condition2 is true
    }
    ….
    else
    {
       // code will be execute
    }
```

Example of if else statement:

```
using System;


public class IfStatement
{
   public static void Main(string[] args)
   {
          int age = 18;
          if(age >= 18 )
          {
                 Console.WriteLine("applicable for driving licence");
          }
          else
          {
                 Console.WriteLine("not applicable ");
          }
   }
}
```

Example of if else / else-if

```
using System;

class Program
{
   Public static void Main(String[] args)
   {
      Console.Write("Enter the marks obtained (0-100): ");
      int marks = Convert.ToInt32(Console.ReadLine()); // Input marks of the student
      // Check if the marks are in the valid range
      if (marks < 0 || marks > 100)
      {
          Console.WriteLine("Invalid marks. Please enter a value between 0 and 100.");
      }
     else
     {
          // Determine the grade based on the marks
          if (marks >= 90)
          {
                    Console.WriteLine("Grade: A");
          }
          else if (marks >= 80)
           {
                    Console.WriteLine("Grade: B");
           }
          else if (marks >= 70)
          {
                    Console.WriteLine("Grade: C");
           }
```

```
        else if (marks >= 60)
        {
                Console.WriteLine("Grade: D");
        }
        else
         {
                Console.WriteLine("Grade: F");
          }
      }
   }
}
```

Difference between && and & (Conditional Operators)

Both operators are used like:

Condition1 && Condition2

Condition1 & Condition2

&& - logical operator -

This operator is based on **Boolean** values to logically check the condition, and if the conditions are true, it returns 1. Otherwise, it returns 0 (False).

Ex : When && is used, then first condition is evaluated and if it comes out to be false then the second condition is not checked because according to the definition of && operator (AND) if both conditions are true only then && returns true else false.

& - bitwise operator –

Performs a bitwise **AND** operation. Each bit in the result is **1** if both corresponding bits in the operands are 1. Otherwise, it is **0**.

Ex : When & is used then both conditions are checked in every case. It first evaluates **condition1**, irrespective of its result it will always checks the **condition2**. Then it compares both outputs and if both are true only then it returns **true**.

Switch Statement in C# - Conditional statement

- In C#, the **switch** statement is a multi-branch control structure that allows a program to execute different code blocks based on the value of an expression.
- **C# Switch Statement** allows a variable to be tested for equality for a list of values (cases). Each case represents a possible value, and the corresponding code block is executed if a match is found.

- **Break and Default:** The break keyword is used to terminate the switch case once a match is found. The **default** keyword provides a pathway for the program to follow if none of the cases match the switch expression.
- **Return and Goto:** The **return** keyword can be used in a C# Switch case to return a value from a method once a match is found. The **goto** keyword allows jumping from one case to another, providing greater control over the flow of execution.

Syntax of switch statement :

```
switch(expression)
{
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
        break;
}
```

Example of Switch statement

```
using System;


public class Switch
{
   public static void Main(string[] args)
   {
      int day = 4;
         switch (day)
         {
                  case 1:
                        Console.WriteLine("Monday");
                         break;
                  case 2:
                        Console.WriteLine("Tuesday");
                        break;
                 case 3:
                        Console.WriteLine("Wednesday");
                        break;
                case 4:
                        Console.WriteLine("Thursday");
                        break;
                 case 5:
                        Console.WriteLine("Friday");
                        break;
                case 6:
                        Console.WriteLine("Saturday");
```

```
                    break;
                case 7:
                    Console.WriteLine("Sunday");
                    break;
                default :
                    Console.WriteLine("Invalid ");

            }
        }
}
```

using goto

In C#, the goto statement transfers control to some other part of the program.

```
using System;


namespace Switch
{
 class Program
{
   Public static void Main(String[] args)
   {
      int option = 2;
     switch (option)
      {
         case 1:
            Console.WriteLine("Option 1 selected.");
            goto case 3; // Jump to the case 3 block


         case 2:
            Console.WriteLine("Option 2 selected.");
            goto case 3; // Jump to the case 3 block


         case 3:
            Console.WriteLine("Option 3 selected.");
            break;


         default:
```

```
            Console.WriteLine("Invalid option.");
            break;
        }
        // Additional code after the switch statement
        Console.WriteLine("End of the switch statement.");
    }
}}
```