University of Anbar
College of Computer Science
and Information Technology
Computer Network Systems Department

Visual Programming I

Lecture Two

Third Stage

First Course 2021 - 2022

Seddiq Qais Abd Al-Rahman

MSc Computer Science

co.sedeikaldossary@uoanbar.edu.iq

Visual Programming

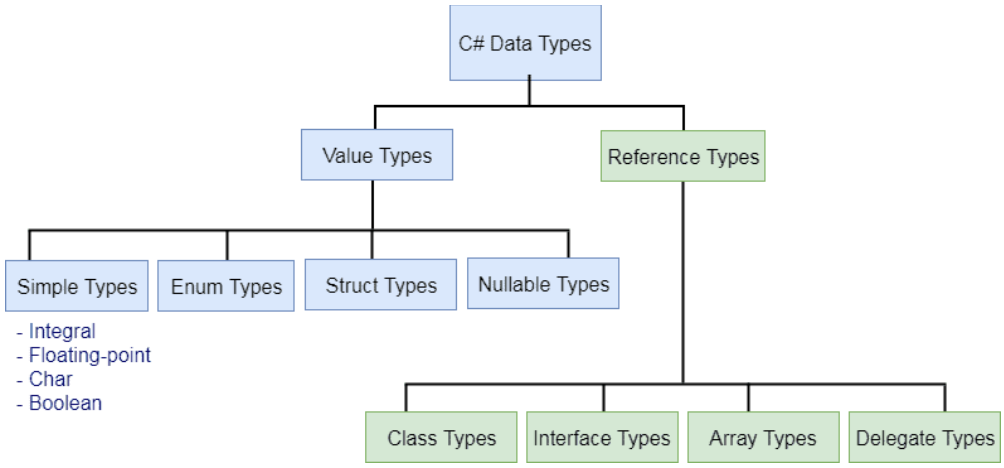
Data Types

C# is a strongly-typed language. It means we must declare the type of a variable that indicates the kind of values it is going to store, such as integer, float, decimal, text, etc.The following declares and initialized variables of different data types.

Example: Variables of Different Data Types

```
string stringVar = "Hello World!!";
int intVar = 100;
float floatVar = 10.2f;
char charVar = 'A';
bool boolVar = true;
```

C# mainly categorized data types in two types: Value types and Reference types. Value types include simple types (such as int, float, bool, and char), enum types, struct types, and Nullable value types. Reference types include class types, interface types, delegate types, and array types. Learn about value types and reference types in detail in the next chapter.



Predefined Data Types in C#

C# includes some predefined value types and reference types. The following table lists predefined data types:

Type	Description	Range
byte	8-bit unsigned integer	0 to 255
sbyte	8-bit signed integer	-128 to 127
short	16-bit signed integer	-32,768 to 32,767
ushort	16-bit unsigned integer	0 to 65,535
int	32-bit signed integer	-2,147,483,648 to 2,147,483,647
uint	32-bit unsigned integer	0 to 4,294,967,295

long	64-bit signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	64-bit unsigned integer	0 to 18,446,744,073,709,551,615
float	32-bit Single-precision floating point type	-3.402823e38 to 3.402823e38
double	64-bit double-precision floating point type	-1.79769313486232e308 to 1.79769313486232e308
decimal	128-bit decimal type for financial and monetary calculations	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
char	16-bit single Unicode character	Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode)
bool	8-bit logical true/false value	True or False
object	Base type of all other types.	
string	A sequence of Unicode characters	
DateTime	Represents date and time	0:00:00am 1/1/01 to 11:59:59pm 12/31/9999

Default Values

Every data type has a default value. Numeric type is 0, boolean has false, and char has '\0' as default value. Use the default(typename) to assign a default value of the data type or C# 7.1 onward, use default literal.

```
int i = default(int); // 0
float f = default(float);// 0
decimal d = default(decimal);// 0
bool b = default(bool);// false
char c = default(char);// '\0'
// C# 7.1 onwards
int i = default; // 0
float f = default;// 0
decimal d = default;// 0
bool b = default;// false
char c = default;// '\0'
```

Converting C++ Data Types to C#

If you've ever had to write any interop code to use an unmanaged library in your C# application, you know how tricky it can be to get the data types correct. I often find myself scouring the internet looking for the correct conversions, so I thought I would document everything I have learned so far. This is by no means a comprehensive list of all C++ data types, just the ones I find myself frequently writing interop code for.

C++ Type	C# Type	Size
BOOL	bool	1 byte
BYTE	byte	1 byte
CHAR	byte	1 byte
DECIMAL	Decimal	16 bytes
DOUBLE	double	8 bytes
DWORD	uint, UInt32	4 bytes
FLOAT	float, single	4 bytes
INT, signed int	int, Int32	4 bytes
INT16, signed short int	short, Int16	2 bytes
INT32, signed int	int, Int32	4 bytes
INT64	long, Int64	8 bytes
LONG	int, Int32	4 bytes
LONG32, signed int	int, Int32	4 bytes
LONG64	long, Int64	8 bytes
LONGLONG	long, Int64	8 bytes
SHORT, signed short int	short, Int16	2 bytes
UCHAR, unsigned char	byte	1 byte
UINT, unsigned int	uint, UInt32	4 bytes
UINT16, WORD	ushort, UInt16	2 bytes
UINT32, unsigned int	uint, UInt32	4 bytes
UINT64	ulong, UInt64	8 bytes
ULONG, unsigned long	uint, UInt32	4 bytes
ULONG32	uint, UInt32	4 bytes
ULONG64	ulong, UInt64	8 bytes
ULONGLONG	ulong, UInt64	8 bytes
WORD	ushort	2 bytes
void*, pointers	IntPtr	x86=4 bytes, x64=8 bytes

Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable. The basic value types provided in C# can be categorized as:

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

C# also allows defining other value types of variable such as **enum** and reference types of variables such as **class**, which we will cover in subsequent chapters.

Defining Variables

Syntax for variable definition in C# is −

```
<data_type> <variable_list>;
```

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas. Some valid variable definitions are shown here:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

You can initialize a variable at the time of definition as −

```
int i = 100;
```

Initializing Variables

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is −

```
variable_name = value;
```

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as −

```
<data_type> <variable_name> = value;
```

Some examples are:

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

It is a good programming practice to initialize variables properly, otherwise sometimes program may produce unexpected result.

The following example uses various types of variables −

```
using System;
namespace VariableDefinition {
  class Program {
    static void Main(string[] args) {
      short a;
      int b ;
      double c;
      /* actual initialization */
      a = 10;
```

```
            b = 20;
            c = a + b;
            Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
            Console.ReadLine();
        }
    }
}
```

When the above code is compiled and executed, it produces the following result −

```
a = 10, b = 20, c = 30
```

Input and Output

C# Output

In order to output something in C#, we can use

```
System.Console.WriteLine() OR
System.Console.Write()
```

Here, System is a namespace, Console is a class within namespace System and WriteLine and Write are methods of class Console. Let's look at a simple example that prints a string to output screen.

Example: Printing String using WriteLine()

```
using System;
namespace Sample
{
        class Test
        {
                public static void Main(string[] args)
                {
                        Console.WriteLine("C# is cool");
                }
        }
}
```

When we run the program, the output will be

```
C# is cool
```

Difference between WriteLine() and Write() method

The main difference between WriteLine() and Write() is that the Write() method only prints the string provided to it, while the WriteLine() method prints the string and moves to the start of next line as well. Let's take at a look at the example below to understand the difference between these methods. Example: How to use WriteLine() and Write() method?

```
using System;
namespace Sample
```

```
{
        class Test
        {
                public static void Main(string[] args)
                {
                        Console.WriteLine("Prints on ");
                        Console.WriteLine("New line");
                        Console.Write("Prints on ");
                        Console.Write("Same line");
                }
        }
}
```

When we run the program, the output will be

```
Prints on
New line
Prints on Same line
```

Printing Variables and Literals using WriteLine() and Write()

The WriteLine() and Write() method can be used to print variables and literals. Here's an example.

Example: Printing Variables and Literals

```
using System;
namespace Sample
{
        class Test
        {
                public static void Main(string[] args)
                {
                        int value = 10;
                        // Variable
                        Console.WriteLine(value);
                        // Literal
                        Console.WriteLine(50.05);
                }
        }
}
```

When we run the program, the output will be

```
10
50.05
```

Combining (Concatenating) two strings using + operator and printing them

Strings can be combined/concatenated using the + operator while printing.

Example: Printing Concatenated String using + operator

```
using System;
namespace Sample
{
        class Test
        {
                public static void Main(string[] args)
                {
                        int val = 55;
                        Console.WriteLine("Hello " + "World");
                        Console.WriteLine("Value = " + val);
                }
        }
}
```

When we run the program, the output will be

```
Hello World
Value = 55
```

Printing concatenated string using Formatted String [Better Alternative]

A better alternative for printing concatenated string is using formatted string. Formatted string allows programmer to use placeholders for variables. For example, The following line,

```
Console.WriteLine("Value = " + val);
```

can be replaced by,

```
Console.WriteLine("Value = {0}", val);
```

{0} is the placeholder for variable val which will be replaced by value of val. Since only one variable is used so there is only one placeholder. Multiple variables can be used in the formatted string. We will see that in the example below.

Example: Printing Concatenated string using String formatting

```
using System;
namespace Sample
{
        class Test
        {
                public static void Main(string[] args)
                {
                        int firstNumber = 5, secondNumber = 10, result;
                        result = firstNumber + secondNumber;
                        Console.WriteLine("{0} + {1} = {2}", firstNumber,
secondNumber, result);
                }
        }
```

```
}
```

When we run the program, the output will be

```
5 + 10 = 15
```

Here, {0} is replaced by firstNumber, {1} is replaced by secondNumber and {2} is replaced by result. This approach of printing output is more readable and less error prone than using + operator. To know more about string formatting, visit *C# string formatting*.

C# Input

In C#, the simplest method to get input from the user is by using the ReadLine() method of the Console class. However, Read() and ReadKey() are also available for getting input from the user. They are also included in Console class.

Example: Get String Input From User

```
using System;
namespace Sample
{
    class Test
    {
        public static void Main(string[] args)
        {
            string testString;
            Console.Write("Enter a string - ");
            testString = Console.ReadLine();
            Console.WriteLine("You entered '{0}'", testString);
        }
    }
}
```

When we run the program, the output will be:

```
Enter a string - Hello World
You entered 'Hello World'
```

Difference between ReadLine(), Read() and ReadKey() method:

The difference between ReadLine(), Read() and ReadKey() method is:

- ReadLine(): The ReadLine() method reads the next line of input from the standard input stream. It returns the same string.
- Read(): The Read() method reads the next character from the standard input stream. It returns the ascii value of the character.
- ReadKey(): The ReadKey() method obtains the next key pressed by user. This method is usually used to hold the screen until user press a key.

If you want to know more about these methods, here is an interesting discussion on StackOverflow on: Difference between Console.Read() and Console.ReadLine()?.

Example: Difference between Read() and ReadKey() method

```
using System;
namespace Sample
{
    class Test
    {
        public static void Main(string[] args)
        {
            int userInput;
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
            Console.WriteLine();
            Console.Write("Input using Read() - ");
            userInput = Console.Read();
            Console.WriteLine("Ascii Value = {0}",userInput);
        }
    }
}
```

When we run the program, the output will be

```
Press any key to continue...
x
Input using Read() - Learning C#
Ascii Value = 76
```

From this example, it must be clear how `ReadKey()` and `Read()` method works. While using `ReadKey()`, as soon as the key is pressed, it is displayed on the screen. When `Read()` is used, it takes a whole line but only returns the ASCII value of first character. Hence, `76` (ASCII value of `L`) is printed. Reading numeric values (integer and floating point types). Reading a character or string is very simple in C#. All you need to do is call the corresponding methods as required. But, reading numeric values can be slightly tricky in C#. We'll still use the same `ReadLine()` method we used for getting string values. But since the `ReadLine()` method receives the input as string, it needs to be converted into integer or floating point type. One simple approach for converting our input is using the methods of `Convert` class.

Example: Reading Numeric Values from User using Convert class

```
using System;
namespace UserInput
{
    class MyClass
    {
        public static void Main(string[] args)
        {
            string userInput;
            int intVal;
```

```
                    double doubleVal;
                    Console.Write("Enter integer value: ");
                    userInput = Console.ReadLine();
                    /* Converts to integer type */
                    intVal = Convert.ToInt32(userInput);
                    Console.WriteLine("You entered {0}",intVal);
                    Console.Write("Enter double value: ");
                    userInput = Console.ReadLine();
                    /* Converts to double type */
                    doubleVal = Convert.ToDouble(userInput);
                    Console.WriteLine("You entered {0}",doubleVal);
                }
            }
        }
```

When we run the program, the output will be

```
Enter integer value: 101
You entered 101
Enter double value: 59.412
You entered 59.412
```

The ToInt32() and ToDouble() method of Convert class converts the string input to integer and double type respectively. Similarly we can convert the input to other types. Here is a complete list of available methods for Convert class.

Program Structure

Before we study basic building blocks of the C# programming language, let us look at a bare minimum C# program structure so that we can take it as a reference in upcoming chapters.

Creating Hello World Program

A C# program consists of the following parts −

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

Let us look at a simple code that prints the words "Hello World" −

```
using System;
namespace HelloWorldApplication {
  class HelloWorld {
```

```
        static void Main(string[] args) {
          /* my first program in C# */
          Console.WriteLine("Hello World");
          Console.ReadKey();
        }
      }
    }
```

When this code is compiled and executed, it produces the following result −

```
Hello World
```

Let us look at the various parts of the given program −

- The first line of the program **using System;** - the **using** keyword is used to include the **System** namespace in the program. A program generally has multiple **using** statements.
- The next line has the **namespace** declaration. A **namespace** is a collection of classes. The *HelloWorldApplication* namespace contains the class *HelloWorld*.
- The next line has a **class** declaration, the class *HelloWorld* contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the *HelloWorld* class has only one method **Main**.
- The next line defines the **Main** method, which is the **entry point** for all C# programs. The **Main** method states what the class does when executed.
- The next line /*...*/ is ignored by the compiler and it is put to add **comments** in the program.
- The Main method specifies its behavior with the statement **Console.WriteLine("Hello World");**

  WriteLine is a method of the *Console* class defined in the *System* namespace. This statement causes the message "Hello, World!" to be displayed on the screen.
- The last line **Console.ReadKey();** is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

It is worth to note the following points −

- C# is case sensitive.
- All statements and expression must end with a semicolon (;).
- The program execution starts at the Main method.
- Unlike Java, program file name could be different from the class name.

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# has rich set of built-in operators and provides the following type of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

This tutorial explains the arithmetic, relational, logical, bitwise, assignment, and other operators one by one.

Arithmetic Operators

Following table shows all the arithmetic operators supported by C#. Assume variable A holds 10 and variable B holds 20 then −

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Relational Operators

Following table shows all the relational operators supported by C#. Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C#. Assume variable A holds Boolean value true and variable B holds Boolean value false, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Assignment Operators

There are following assignment operators supported by C#:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
***=**	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

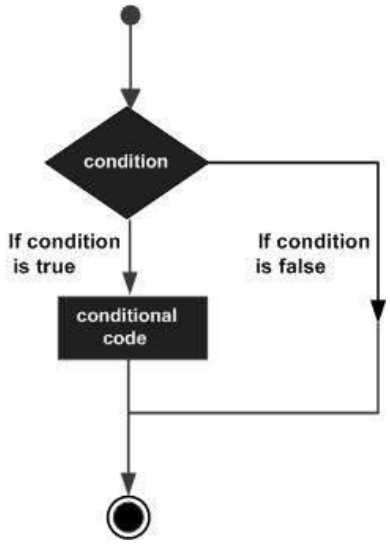
Miscellaneous Operators

There are few other important operators including sizeof, typeof and ? : supported by C#.

Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*****	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

Decision Making

Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false. Following is the general form of a typical decision making structure found in most of the programming languages:



C# provides following types of decision making statements. Click the following links to check their detail.

No.	Statement & Description
1	if statement An **if statement** consists of a boolean expression followed by one or more statements.

2	if...else statement An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
3	nested if statements You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
4	switch statement A **switch** statement allows a variable to be tested for equality against a list of values.
5	nested switch statements You can use one **switch** statement inside another **switch** statement(s).

The ? : Operator

C# includes a decision-making operator ?: which is called the conditional operator or ternary operator. It is the short form of the if else conditions.

```
condition ? statement 1 : statement 2
```

The ternary operator starts with a boolean condition. If this condition evaluates to true then it will execute the first statement after ?, otherwise the second statement after : will be executed. The following example demonstrates the ternary operator.

Example: Ternary operator

```
int x = 20, y = 10;
var result = x > y ? "x is greater than y" : "x is less than y";
Console.WriteLine(result);
```

output:

```
x is greater than y
```

Above, a conditional expression x > y returns true, so the first statement after ? will be execute. The following executes the second statement.

Example: Ternary operator

```
int x = 10, y = 100;
var result = x > y ? "x is greater than y" : "x is less than y";
Console.WriteLine(result);
```

output:

```
x is less than y
```

Thus, a ternary operator is short form of if else statement. The above example can be re-write using if else condition, as shown below.

Example: Ternary operator replaces if statement

```
int x = 10, y = 100;
if (x > y)
   Console.WriteLine("x is greater than y");
else
   Console.WriteLine("x is less than y");
```

output:

```
x is greater than y
```

Switch

The switch statement can be used instead of if else statement when you want to test a variable against three or more conditions. Here, you will learn about the switch statement and how to use it efficiently in the C# program. The following is the general syntax of the switch statement.

```
switch(match expression/variable)
{
   case constant-value:
      statement(s) to be executed;
      break;
   default:
      statement(s) to be executed;
      break;
}
```

The switch statement starts with the switch keyword that contains a match expression or a variable in the bracket switch(match expression). The result of this match expression or a variable will be tested against conditions specified as cases, inside the curly braces { }. A case must be specified with the unique constant value and ends with the colon :. Each case includes one or more statements to be executed. The case will be executed if a constant value and the value of a match expression/variable are equal. The switch statement can also contain an optional default label. The default label will be executed if no cases executed. The break, return, or goto keyword is used to exit the program control from a switch case. The following example demonstrates a simple switch statement.

Example: C# Switch Statement

```
int x = 10;
switch (x)
{
   case 5:
      Console.WriteLine("Value of x is 5");
      break;
   case 10:
      Console.WriteLine("Value of x is 10");
      break;
   case 15:
      Console.WriteLine("Value of x is 15");
      break;
   default:
      Console.WriteLine("Unknown value");
      break;
}
```

Output:

Value of x is 10

Above, the switch(x) statement includes a variable x whose value will be matched with the value of each case value. The above switch statement contains three cases with constant values 5, 10, and 15. It also contains the default label, which will be executed if none of the case value match with the switch variable/expression. Each case starts after : and includes one statement to be executed. The value of x matches with the second case case 10:, so the output would be Value of x is 10.

Note:

The switch statement can include any non-null expression that returns a value of type: char, string, bool, int, or enum.

The switch statement can also include an expression whose result will be tested against each case at runtime.

Example: C# Switch Statement

```
int x = 125;
switch (x % 2)
{
   case 0:
      Console.WriteLine($"{x} is an even value");
      break;
   case 1:
      Console.WriteLine($"{x} is an odd Value");
      break;
}
```

Output:

125 is an odd value

Multiple cases can be combined to execute the same statements.

Example: C# Combined Switch Cases

```
int x = 5;
switch (x)
{
   case 1:
      Console.WriteLine("x = 1");
      break;
   case 2:
      Console.WriteLine("x = 2");
      break;
   case 4:
   case 5:
      Console.WriteLine("x = 4 or x = 5");
      break;
```

```
        default:
            Console.WriteLine("x > 5");
            break;
    }
```

Exercises

1. Write a C# Sharp program to accept two integers and check whether they are equal or not.
2. Write a C# Sharp program to check whether a given number is even or odd.
3. Write a C# Sharp program to check whether a given number is positive or negative
4. Write a C# Sharp program to find whether a given year is a leap year or not.
5. Write a C# Sharp program to read the age of a candidate and determine whether it is eligible for casting his/her own vote
6. Write a C# Sharp program to read the value of an integer m and display the value of n is 1 when m is larger than 0, 0 when m is 0 and -1 when m is less than 0.
7. Write a C# Sharp program to accept the height of a person in centimeter and categorize the person according to their height.
8. Write a C# Sharp program to find the largest of three numbers.
9. Write a C# Sharp program to accept a coordinate point in an XY coordinate system and determine in which quadrant the coordinate point lies.
10. Write a C# Sharp program to find the eligibility of admission for a professional course based on the following criteria:
    Marks in Maths >=65
    Marks in Phy >=55
    Marks in Chem>=50
    Total in all three subject >=180
    or
    Total in Math and Subjects >=140
11. Write a C# Sharp program to calculate root of Quadratic Equation.
12. Write a C# Sharp program to read roll no, name and marks of three subjects and calculate the total, percentage and division.
13. Write a C# Sharp program to read temperature in centigrade and display a suitable message according to temperature state below :
    Temp < 0 then Freezing weather

Temp 0-10 then Very Cold weather
Temp 10-20 then Cold weather
Temp 20-30 then Normal in Temp
Temp 30-40 then Its Hot
Temp >=40 then Its Very Hot

14. Write a C# Sharp program to calculate root of Quadratic Equation.
15. Write a C# Sharp program to read roll no, name and marks of three subjects and calculate the total, percentage and division.

References:

- Tony Gaddis, “Starting out with Visual C#.”, Fourth edition, Boston, Pearson Inc., 2017.
- Salvatore A. Buono, “C# and Game Programming: A Beginner's Guide.” Second Edition, Boca Raton, CRC Press Inc., 2019.
- Eric Butow and Tommy Ryan, "C#: Your Visual Blueprint for Building .NET Applications.", Hungry Minds Inc., New York, 2002.
- Faraz Rasheed, “Programmer's Heaven: C# School.”, First Edition, Fuengirola, Synchron Data, 2006.