Event Driven Programming

Programming with Event Driven in C#

Chalew Tesfaye
Department of Computer Science
Debre Berhan University
2017

Objectives

- After the end this lesson the student will be able to
  - ✓ Understand C# language fundamentals
    - Data type, variables and constants, …
  - ✓ Write a C# program statement
  - ✓ Debug C# program in VS
  - ✓ Develop OO program with C#

Lesson Outline

- C# fundamentals
  - ✓ Data types, variables and constants
  - ✓ Comments in C#
  - ✓ Namespace
  - ✓ Type casting
  - ✓ Data type conversion
  - ✓ Operators and Expressions
- Console Input / Output
- C# Code control structure
  - ✓ Conditional statements
  - ✓ Looping statements
- Working with Date and string data types
- Arrays and Collections
- Methods and Event handlers
- Object oriented programming
  - ✓ Classes
  - ✓ Indexer, delegates, events and operators
  - ✓ Inheritance
  - ✓ Interface and generics

C# C-Sharp

High-level programming language

C#(C-Sharp)

- Microsoft C# (pronounced See Sharp) developed by Microsoft Corporation, USA
- New programming language that runs on the .NET Framework
- C# is simple, modern, type safe, and object oriented
- C# code is compiled as managed code
- Combines the best features of Visual Basic, C++ and Java

C# Features

- Simple
- Modern
- Object-Oriented
- Type-safe
- Versionable
- Compatible
- Secure

C# - Data Types

Domain of values

Data Types

- are sets (ranges) of values that have similar characteristics
- Data types are characterized by:
  - ✓ Name – for example, int;
  - ✓ Size (how much memory they use) – for example, 4 bytes;
  - ✓ Default value – for example 0.
- two major sets of data types in C#
  - ✓ value types
    - > store their own data
  - ✓ reference types
    - > Store a reference to the area of memory where the data is stored

Data Types …

- Basic data types in C# :
  - ✓ Integer types – sbyte, byte, short, ushort, int, uint, long, ulong;
  - ✓ Real floating-point types – float, double;
  - ✓ Real type with decimal precision – decimal;
  - ✓ Boolean type – bool;
  - ✓ Character type – char;
  - ✓ String – string;
  - ✓ Object type – object.
- These data types are called primitive (built-in types),
  - ✓ because they are embedded in C# language at the lowest level.

Data Types …

C# Keyword	Bytes	.Net Type	default	Min value	Max value
sbyte	1	SByte	0	-128	127
byte	1	Byte	0	0	255
short	2	Int16	0	-32768	32767
ushort	2	UInt16	0	0	65535
int	4	Int32	0	-2147483648	2147483647
uint	4	UInt32	0u	0	4294967295
long	8	Int64	0L	-9223372036854775808	9223372036854775807
ulong	8	UInt64	0u	0	18446744073709551615
float	4	Single	0.0f	$\pm 1.5 \times 10^{-45}$	$\pm 3.4 \times 10^{38}$
double	8	Double	0.0d	$\pm 5.0 \times 10^{-324}$	$\pm 1.7 \times 10^{308}$
decimal	16	Decimal	0.0m	$\pm 1.0 \times 10^{-28}$	$\pm 7.9 \times 10^{28}$
bool	1	Boolean	false	Two possible values: true and false	
char	2	Char	'\u0000'	'\u0000'	'\uffff‘
object	-	Object	null	- a reference to a String object	
string	-	String	null	- a reference to any type of object	

C# > Variables

A named area of memory

Variables

- a named area of memory
- stores a value from a particular data type, and
- is accessible in the program by its name.
- Stores a value that can change as the program executes
- Before use it must be declared
- Variable declaration contains
  - ✓ Data types
  - ✓ Name
  - ✓ Initial value
- Valid C# variable name
  - ✓ Start with A-Z or a-z or _
  - ✓ Can include number and _
  - ✓ cannot coincide with a keyword
    - > Use "@". For example, @char and @null
  - ✓ C# is case sensitive
  - ✓ E.g. salary, number1, total_mark
- Naming convention
  - ✓ camelNotation
    - > E,g. letterGrade
  - ✓ PascalNotation
    - > E.g. LetterGrade

Variables > keywords

- Reserved words by the language for some purpose
- Can't be used by programmer for other purpose except they are reserved for

abstract	as	base	bool	break	byte
case	catch	char	checked	class	const
continue	decimal	default	delegate	do	double
else	enum	event	explicit extern	false	
finally	fixed	float	for	foreach	goto
if	implicitin	int	interface	internal	
is	lock	long	namespace	new	null
object	operator	out	override	params	private
protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string
struct	switch	this	throw	true	try
typeof	uint	ulong	unchecked	unsafe	ushort
using	virtual	void	volatilewhile	…	

Variables > Declaration

- Variable declaration and initialization

  Syntax

  ```
  type variableName;
  variableName=value;
  ```

- Several ways of initializing:
  - ✓ By using the new keyword
  - ✓ By using a literal expression
  - ✓ By referring to an already initialized variable

Variables > Example

- Example

```
int num = new int(); // num = 0
int x;          //declaration statement
x=0;            //assignment statement
char grade='A'; //enclose a character value in single quotes
double price = 10.55; //declare and  initialize a value with 1 stmt
double scientificNumber= 3.45e+6; //scientific notation
decimal salary = 12345.67m; //m or M indicates a decimal value,
monetary(money)
float interestRate = 5.25f; //f or F indicates a float value
bool isValid = false; //declare and  initialize a value with 1 stmt
double scgpa = 3.2, cgpa=3.12; //initialize 2 variables with 1 stmt
string greeting = "Hello World!"; //enclose a string value in double quotes
string message = greeting;
```

Variables > Constant variables

- Constant
  - ✓ Stores a value that can't be changed as the program executes
  - ✓ Always declared and initialized in a single statement
  - ✓ Declaration statement begins with const keyword
  - ✓ Capitalize the first letter in each word of a constant also common practice – PascalNotation
- Example

```
const double Pension = 0.06;
const int DaysInWeek = 7;
const int Km = 100;
```

Variables > String data type

- Strings are sequences of characters.
- declared by the keyword string
- enclosed in double quotation marks
- default value is null
  - ✓ Example

```
string firstName = “Abebe”;
string lastName = “Kebede”;
string fullName = firstName + “ “ + lastName;
```

- Various text-processing operations can be performed using strings:
  - ✓ concatenation (joining one string with another),
  - ✓ splitting by a given separator,
  - ✓ searching,
  - ✓ replacement of characters and others.
  - ✓ + concatenate, += append, More in working with Date and String class

Variables > Object Type

- Object type is a special type
  - ✓ which is the parent of all other types in the .NET Framework
  - ✓ Declared with the keyword object,
  - ✓ it can take values from any other type.
  - ✓ It is a reference type,
    - > i.e. an index (address) of a memory area which stores the actual value.
  - ✓ Example

```
object object1 = 1;
object object2 = “Two";
```

Variables > Nullable Types

- are specific wrappers around the value types (as int, double and bool) that allow storing data with a null value.
- provides opportunity for types that generally do not allow lack of value (i.e. value null)
  - ✓ to be used as reference types and to accept both normal values and the special one null.
- Thus nullable types hold an optional value.
- Wrapping a given type as nullable can be done in two ways:
  - ✓ Nullable<int> i1 = null;
  - ✓ int? i2 = i1;

Variables > Nullable Types – Example

```
int? someInteger = null;
Console.WriteLine("This is the integer with Null value -> " + someInteger);
someInteger = 5;
Console.WriteLine( "This is the integer with value 5 -> " +  someInteger);

double? someDouble = null;
Console.WriteLine("This is the real number with Null value -> " +
someDouble);
someDouble = 2.5;
Console.WriteLine("This is the real number with value 5 -> " +
someDouble);
```

Variables > Literals

- Primitive types, which we already met, are special data types built into the C# language.
- Their values specified in the source code of the program are called literals.
- types of literals:
  - ✓ Boolean
  - ✓ Integer
  - ✓ Real
  - ✓ Character
  - ✓ String
  - ✓ Object literal null

Variables > Literals > Escaping Sequences

- the most frequently used escaping sequences:
- \' – single quote
- \" – double quotes
- \\ – backslash
- \n – new line
- \t – offset (tab)
- \uXXXX – char specified by its Unicode number, for example \u1200.
- Example
  - ✓ char symbol = 'a'; // An ordinary symbol
  - ✓ symbol = '\u1200'; // Unicode symbol code in a hexadecimal format
  - ✓ symbol = '\u1201'; // ሁ (”Hu” Amharic Letter)
  - ✓ symbol = '\t'; // Assigning TAB symbol
  - ✓ symbol = '\n'; // Assigning new line symbol

Variables > String Literals

- Strings can be preceded by the @ character that specifies a quoted string (verbatim string)
- `string quotation = "\"Hello, C#\", he said.";`
- `string path = "C:\\Windows\\Notepad.exe";`
- `string verbatim = @"The \ is not escaped as \\.`

  `I am at a new line.";`

Variables > Scope

- scope
  - ✓ Visibility/accessibility of a variable
  - ✓ determines what codes has access to it
  - ✓ Code block scope
  - ✓ method scope
  - ✓ class scope
- Access Modifier / Accessibility/life time extender
  - ✓ Private – only in the same class
  - ✓ Internal – in the same assembly
  - ✓ Protected – same class or subclass (inherited class)
  - ✓ public – by any object in anywhere

Type Casting

- Casting – converting one data from one data type to another
- Two type casting
  - ✓ Implicit casting
    - > Performed automatically
    - > Widening conversion
      - Used to convert data with a less precise type to a more precise type
    - > byte-> short -> int -> long->decimal
    - > E.g.
      - double mark =85;
      - int grade ='A'
  - ✓ Explicit casting
    - > Narrowing conversion
      - Casts data from a more precise data type to a less precise data type
    - > (type) expression
    - > E.g
      - int mark = (int)85.25;

Convert data type

- .Net framework provides structures and classes that defines data type
- Structure defines a value type
- Class defines a reference type
- Structure
  - ✓ Byte – byte – an 8-bit unsigned integer
  - ✓ Int16 – short - a 16-bit signed integer
  - ✓ Int32 – int – A 32-bit signed integer
  - ✓ Int64 – long – A 64-bit signed integer
  - ✓ Single – float – a single-precision floating number
  - ✓ Double – double - a double precision floating number
  - ✓ Decimal – decimal – a 96-bit decimal value
  - ✓ Boolean – bool – a true or false value
  - ✓ Char – char - a single character
- Class
  - ✓ String – string – a reference to a String object
  - ✓ Object – object – a reference to any type of object

Convert data type …

- Methods used to convert data types
  - ✓ ToString
    - > Converts the value to its equivalent string representation using the specified format, if any
    - > ToString([format])
  - ✓ Parse
    - > A static method that converts the specified string to an equivalent data value
    - > If the string can't converted, an exception occurred
    - > Parse(string)
  - ✓ TryParse
    - > A static method that converts the specified string to an equivalent data value and
    - > Stores it in the result variable
    - > Returns a true value if the string is converted, otherwise, returns a false value
    - > If the string can't converted, an exception occurred
  - ✓ Static methods of Convert class
    - > ToDecima(value) – converts the value to the decimal data type
    - > ToDouble(value) - converts the value to the double data type
    - > ToInt32(value) – converts the value to the int data type
    - > …

Convert data type > Example

```
decimal salary = 2453.32m;
string salaryString = salary.ToString();
salary=Decimal.Parse(salaryString);
Decimal.TryParse(salaryString, out salary);
double cgpa = 3.85;
string studentCgpa = “CGPA: “+ cgpa; //automatic call to  ToString method
string salesString = “$45268.25”;
decimal sales = 0m;
Decimal.TryParse(salesString. out sales); //sales is 0
string gradePoint = “63.25”;
string totalCredit = “22”;
double gpa = Convert.ToDouble(pgradePoint)/Convert.ToInt32(totalCredit);
```

Convert data type > Formatted String

- using methods to convert formatted strings
  - ✓ C or c – Currency
  - ✓ P or p – Percent
  - ✓ N or n – Number
  - ✓ F or f – Float
  - ✓ D or d – Digits
  - ✓ E or e – Exponential
  - ✓ G or g – General
- Use
  - ✓ ToString method or
  - ✓ Format method

Convert data type > Formatted String > Example

- decimal amount = 15632.20m;
- decimal interest = 0.023m;
- int quantity = 15000;
- float payment = 42355.5;
- ToString method
  - ✓ string monthlyAmount = amount.ToString("c"); // $15,632.20
  - ✓ string interestRate = interest.ToString("p1"); //2.3%
  - ✓ string quantityString = quantity.ToString("n0"); // 15,000
  - ✓ string paymentString = payment.ToString("f3"); //42,355.500
- Format method of String Class
  - ✓ string monthlyAmount = String.Format("0:c", amount); // $15,632.20
  - ✓ string interestRate = String.Format("0:p1", interest); //2.3%
  - ✓ string quantityString = String.Format("0:n0", quantity); // 15,000
  - ✓ string paymentString = String.Format("0:f3", payment); //42,355.500

Comments in C#

- To include information about your code
- Not readable by the compiler
- Single line comment

```
//comment
```

- Multiple line comment

```
/* -------------
    ---- multiline comment -------
    -------------*/
```

Exercise

1. Which of the ff a valid C# variable name?
   A. new
   B. 1stYear
   C. grade
   D. father-name
2. Declare a variable that stores
   a. letter grade of a subject
   b. CGPA of a student
   c. Balance of a customer
3. Assign an initial value for the above variables
   a. To their domain default
   b. Using new key word
4. What is Nullable data type and its advantage? Give example.
5. Write the output of the ff code snippet

```
string exercise = @“Please Students \ answer this exercise \\.
                  Please participate actively.";
Console.WriteLine(exercise );
```

6. Give an example for implicit and explicit type casting.

C# > Operators and Expressions

Performing Simple Calculations with C#

Operators and Expressions

- Operator
  - ✓ is an operation performed over data at runtime
  - ✓ Takes one or more arguments (operands)
  - ✓ Produces a new value
  - ✓ Operators have precedence
  - ✓ Precedence defines which will be evaluated first
- Expressions
  - ✓ are sequences of operators and operands that are evaluated to a single value

Operators in C#

- Operators in C# :
  - ✓ Unary – take one operand
  - ✓ Binary – take two operands
  - ✓ Ternary – takes three operands
- Except for the assignment operators, all binary operators are left-associative
- The assignment operators and the conditional operator (?:) and ?? are right-associative

Operators in C# > Categories of Operators

Category	Operators
Arithmetic	+, -, *, /, %, ++, --
Logical	&&, ||, ^, !,
Binary	&, |, ^, ~, <<, >>
Comparison	==, !=, < >, <=, >=
Assignment	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
String concatenation	+
Type conversion	is, as, typeof
Other	., [], (), ?:, ??, new

Operators in C# > Operators Precedence

Precedence	Operators
Highest	++ -- (postfix) new typeof
	++ -- (prefix) + - (unary) ! ~
	* / %
	+ -
	<< >>
	< > <= >= is as
	== !=
	&
Lower	^

Operators in C# > Operators Precedence

Precedence	Operators
Higher	|
	&&
	||
	?:, ??
Lowest	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

- Parenthesis is operator always has highest precedence
- Note: prefer using parentheses, even when it seems difficult to do so

Arithmetic Operators

- Arithmetic operators +, -, * are the same as in math
- Division operator / if used on integers returns integer (without rounding) or exception
- Division operator / if used on real numbers returns real number or Infinity or NaN
- Remainder operator % returns the remainder from division of integers
- The special addition operator ++ increments a variable
- The special subtraction operator -- decrements a variable

Arithmetic Operators > Example

- ✓ int squarePerimeter = 17;
- ✓ double squareSide = squarePerimeter/4.0;
- ✓ double squareArea = squareSide*squareSide;
- ✓ Console.WriteLine(squareSide); // 4.25
- ✓ Console.WriteLine(squareArea); // 18.0625

- ✓ int a = 5;
- ✓ int b = 4;
- ✓ Console.WriteLine(a + b); // 9
- ✓ Console.WriteLine(a + b++); // 9
- ✓ Console.WriteLine(a + b); // 10
- ✓ Console.WriteLine(a + (++b)); // 11
- ✓ Console.WriteLine(a + b); // 11

- ✓ Console.WriteLine(11 / 3); // 3
- ✓ Console.WriteLine(11 % 3); // 2
- ✓ Console.WriteLine(12 / 3); // 4

Logical Operator

- Logical operators take boolean operands and return boolean result
- Operator ! (logical Negation) turns true to false and false to true
- Behavior of the operators &&(AND), ||(OR) and ^ (exclusive OR)

Operation	||	||	||	||	&&	&&	&&	&&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

Logical Operator > Example

- Example
  - ✓ bool a = true;
  - ✓ bool b = false;
  - ✓ Console.WriteLine(a && b); // False
  - ✓ Console.WriteLine(a || b); // True
  - ✓ Console.WriteLine(a ^ b); // True
  - ✓ Console.WriteLine(!b); // True
  - ✓ Console.WriteLine(b || true); // True
  - ✓ Console.WriteLine(b && true); // False
  - ✓ Console.WriteLine(a || true); // True
  - ✓ Console.WriteLine(a && true); // True
  - ✓ Console.WriteLine(!a); // False
  - ✓ Console.WriteLine((5>7) ^ (a==b)); // False

Bitwise Operators

- Bitwise operator `~` turns all `0` to `1` and all `1` to `0`
  - ✓ Like `!` for boolean expressions but bit by bit
- The operators `|`, `&` and `^` behave like `||`, `&&` and `^` for boolean expressions but bit by bit
- The `<<` and `>>` move the bits (left or right)
- Bitwise operators are used on integer numbers (`byte`, `sbyte`, `int`, `uint`, `long`, `ulong`)
- Bitwise operators are applied bit by bit
- Behavior of the operators `|`, `&` and `^`:

Operation	|	|	|	|	&	&	&	&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

Bitwise Operators > Example

- Examples:

```
ushort a = 3;              // 00000000 00000011
ushort b = 5;              // 00000000 00000101
Console.WriteLine( a | b); // 00000000 00000111
Console.WriteLine( a & b); // 00000000 00000001
Console.WriteLine( a ^ b); // 00000000 00000110
Console.WriteLine(~a & b); // 00000000 00000100
Console.WriteLine( a<<1 ); // 00000000 00000110
Console.WriteLine( a>>1 ); // 00000000 00000001
```

Comparison Operators

- Comparison operators are used to compare variables
  - ✓ ==, <, >, >=, <=, !=
- Comparison operators example:

```
int a = 5;
int b = 4;
Console.WriteLine(a >= b); // True
Console.WriteLine(a != b); // True
Console.WriteLine(a == b); // False
Console.WriteLine(a == a); // True
Console.WriteLine(a != ++b); // False
Console.WriteLine(a > b); // False
```

Assignment Operators

- Assignment operators are used to assign a value to a variable ,
  - ✓ =, +=, -=, |=, …
- Assignment operators example:

```
int x = 6;
int y = 4;
Console.WriteLine(y *= 2); // 8
int z = y = 3; // y=3 and z=3
Console.WriteLine(z); // 3
Console.WriteLine(x |= 1); // 7
Console.WriteLine(x += 3); // 10
Console.WriteLine(x /= 2); // 5
```

Other Operators

- String concatenation operator + is used to concatenate strings
- If the second operand is not a string, it is converted to string automatically
- example

```
string name= “Name";
string fatherName= “Father Name";
Console.WriteLine(name + fatherName);
// NameFather Name
string output = "The number is : ";
int number = 5;
Console.WriteLine(output + number);
// The number is : 5
```

Other Operators …

- Member access operator `.` is used to access object members
- Square brackets `[]` are used with arrays, indexers and attributes
- Parentheses `( )` are used to override the default operator precedence
- Class cast operator `(type)` is used to cast one compatible type to another
- Conditional operator `?:` has the form
  - ✓ `b?x:y`
    - > if `b` is true then the result is `x` else the result is `y`
- Null coalescing/joining operator `??`
  - ✓ `x=y??0;`
    - > If `y` is `null` set `x=0` else set `x=y`
- The `new` operator is used to create new objects
- The `typeof` operator returns `System.Type` object (the reflection of a type)
- The `is` operator checks if an object is compatible with given type

Other Operators > Example

- Examples

```
int a = 6;
int b = 4;
Console.WriteLine(a > b ? "a>b" : "b>=a"); // a>b
Console.WriteLine((long) a); // 6

int c = b = 3; // b=3; followed by c=3;
Console.WriteLine(c); // 3
Console.WriteLine(a is int); // True
Console.WriteLine((a+b)/2); // 4
Console.WriteLine(typeof(int)); // System.Int32

int d = new int();
Console.WriteLine(d); // 0
```

Expressions

- Expressions are sequences of operators, literals and variables that are evaluated to some value
- Expressions has:
  - ✓ Type (integer, real, boolean, ...)
  - ✓ Value
- Examples:

```
int r = (150-20) / 2 + 5; // r=70
// Expression for calculation of circle area
double surface = Math.PI * r * r;
// Expression for calculation of circle perimeter
double perimeter = 2 * Math.PI * r;
int a = 2 + 3; // a = 5
int b = (a+3) * (a-4) + (2*a + 7) / 4;  // b = 12
bool greater = (a > b) || ((a == 0) && (b == 0));
```

Using the Math Class

- Math class provides methods to work with numeral data
  - ✓ Math.Round(decimalnumber[, precision, mode]);
  - ✓ Math.Pow(number, power);
  - ✓ Math.Sqrt(number);
  - ✓ Math.Min(number1, number2);
  - ✓ Math.Max(number1, number2);
- Example

```
double gpa = Math.Round(gpa,2);
double area = Math.Pow(radius, 2) * Math.PI; //area of a circle
double maxGpa = Math.Max(lastYearGpa, thisYearGpa);
double sqrtX = Math.Sqrt(x);
```

Exercise

1. Write conditional expression (Ternary expression) that checks if given integer is odd or even.
2. Write the output of the ff code fragment.

```
sbyte x = 9;
Console.WriteLine( "Shift of {0} is {1}.", x, ~x<<3);
```

3. Write the output of the ff code fragment.

```
ushort a = 3;
ushort b = 5;
Console.WriteLine(a ^ ~b);
```

4. Write the output of the ff code fragment.

```
int a = 3;
int b = 5;
Console.WriteLine(“a + b = ” + a + b);
```

5. Write a program that calculates the area of a circle for a given radius r (eqn: a= πr^2).

Console Input / Output

Reading and Writing to the Console

Console Input / Output

- Printing to the Console
  - ✓ Printing Strings and Numbers
- Reading from the Console
  - ✓ Reading Characters
  - ✓ Reading Strings
  - ✓ Parsing Strings to Numeral Types
  - ✓ Reading Numeral Types

Printing to the Console

- Console is used to display information in a text window
- Can display different values:
  - ✓ Strings
  - ✓ Numeral types
  - ✓ All primitive data types
- To print to the console use the class `Console` (`System.Console`)

The Console Class

- Provides methods for console input and output
  - ✓ Input
    - `Read(…)` – reads a single character
    - `ReadKey(…)` – reads a combination of keys
    - `ReadLine(…)` – reads a single line of characters
  - ✓ Output
    - `Write(…)` – prints the specified argument on the console
    - `WriteLine(…)` – prints specified data to the console and moves to the next line

Console.Write(…)

- Printing an integer variable

```
int a = 15;
...
Console.Write(a); // 15
```

- Printing more than one variable using a formatting string

```
double a = 15.5;
int b = 14;
...
Console.Write("{0} + {1} = {2}", a, b, a + b);
// 15.5 + 14 = 29.5
```

- Next print operation will start from the same line

Console.WriteLine(…)

- Printing a string variable

```
string str = "Hello C#!";
...
Console.WriteLine(str);
```

- Printing more than one variable using a formatting string

```
string name = “Fatuma";
int year = 1990;
...
Console.WriteLine("{0} was born in {1}.", name, year);
// Fatuma was born in 1990.
```

- Next printing will start from the next line

Reading from the Console

- Reading Strings and Numeral Types
- We use the console to read information from the command line
- We can read:
  - ✓ Characters
  - ✓ Strings
  - ✓ Numeral types (after conversion)
- To read from the console we use the methods
  - ✓ `Console.Read()` and
  - ✓ `Console.ReadLine()`

Console.Read()

- Gets a single character from the console (after [Enter] is pressed)
  - ✓ Returns a result of type `int`
  - ✓ Returns -1 if there aren't more symbols
- To get the actually read character we need to cast it to `char`

```
int i = Console.Read();
char ch = (char) i;  // Cast the int to char
// Gets the code of the entered symbol
Console.WriteLine("The code of '{0}' is {1}.", ch, i);
```

Console.ReadKey()

- Waits until a combination of keys is pressed
  - ✓ Reads a single character from console or a combination of keys
- Returns a result of type `ConsoleKeyInfo`
  - ✓ `KeyChar` – holds the entered character
  - ✓ `Modifiers` – holds the state of [Ctrl], [Alt], …

```
ConsoleKeyInfo key = Console.ReadKey();
Console.WriteLine();
Console.WriteLine("Character entered: " + key.KeyChar);
Console.WriteLine("Special keys: " +  key.Modifiers);
```

Console.ReadLine()

- Gets a line of characters
- Returns a `string` value
- Returns `null` if the end of the input is reached

```
Console.Write("Please enter your first name: ");
string firstName = Console.ReadLine();

Console.Write("Please enter your last name: ");
string lastName = Console.ReadLine();

Console.WriteLine("Hello, {0} {1}!", firstName, lastName);
```

Reading Numeral Types

- Numeral types can not be read directly from the console
- To read a numeral type do the following:
  1. Read a string value
  2. Convert (parse) it to the required numeral type
- `int.Parse(string)` – parses a `string` to `int`

```
string str = Console.ReadLine()
int number = int.Parse(str);
Console.WriteLine("You entered: {0}", number);
string s = "123";
int i = int.Parse(s); // i = 123
long l = long.Parse(s); // l = 123L
string invalid = "xxx1845";
int value = int.Parse(invalid); // FormatException
```

Reading Numbers from the Console > Example

```
static void Main()
{
    int a = int.Parse(Console.ReadLine());
    int b = int.Parse(Console.ReadLine());

    Console.WriteLine("{0} + {1} = {2}", a, b, a+b);
    Console.WriteLine ("{0} * {1} = {2}", a, b, a*b);
    float f = float.Parse(Console.ReadLine());
    Console.WriteLine("{0} * {1} / {2} = {3}", a, b, f, a*b/f);
}
```

Error Handling when Parsing

- Sometimes we want to handle the errors when parsing a number
  - ✓ Two options: use `TryParse()` or `try-catch` block (later in Error Handling Section)
- Parsing with `TryParse()`:

```
string str = Console.ReadLine();
int number;
if (int.TryParse(str, out number))
{
   Console.WriteLine("Valid number: {0}", number);
}
else
{
   Console.WriteLine("Invalid number: {0}", str);
}
```

Console IO > Example

- Calculating an Area

```
Console.WriteLine("This program calculates the area of a rectangle or a triangle");
Console.Write("Enter a and b (for rectangle) or a and h (for triangle): ");
//read inputs
int a = int.Parse(Console.ReadLine());
int b = int.Parse(Console.ReadLine());
Console.Write("Enter 1 for a rectangle or 2 for a triangle: ");
int choice = int.Parse(Console.ReadLine());
//calculate the area
double area = (double) (a*b) / choice;
//display the result
Console.WriteLine("The area of your figure is {0}", area);
```

Conditional Statements

Implementing Control Logic in C#

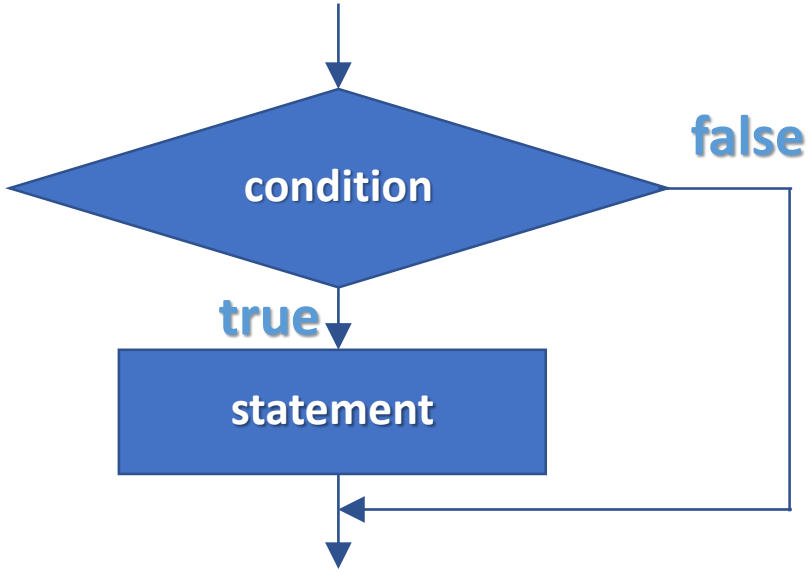
Conditional Statements

- Implementing Control Logic in C#
- if Statement
- if-else Statement
- nested if Statements
- multiple if-else-if-else-…
- switch-case Statement

The `if` Statement

- The most simple conditional statement
- Enables you to test for a condition
- Branch to different parts of the code depending on the result
- The simplest form of an `if` statement:

```
if (condition)
{
    statements;
}
```



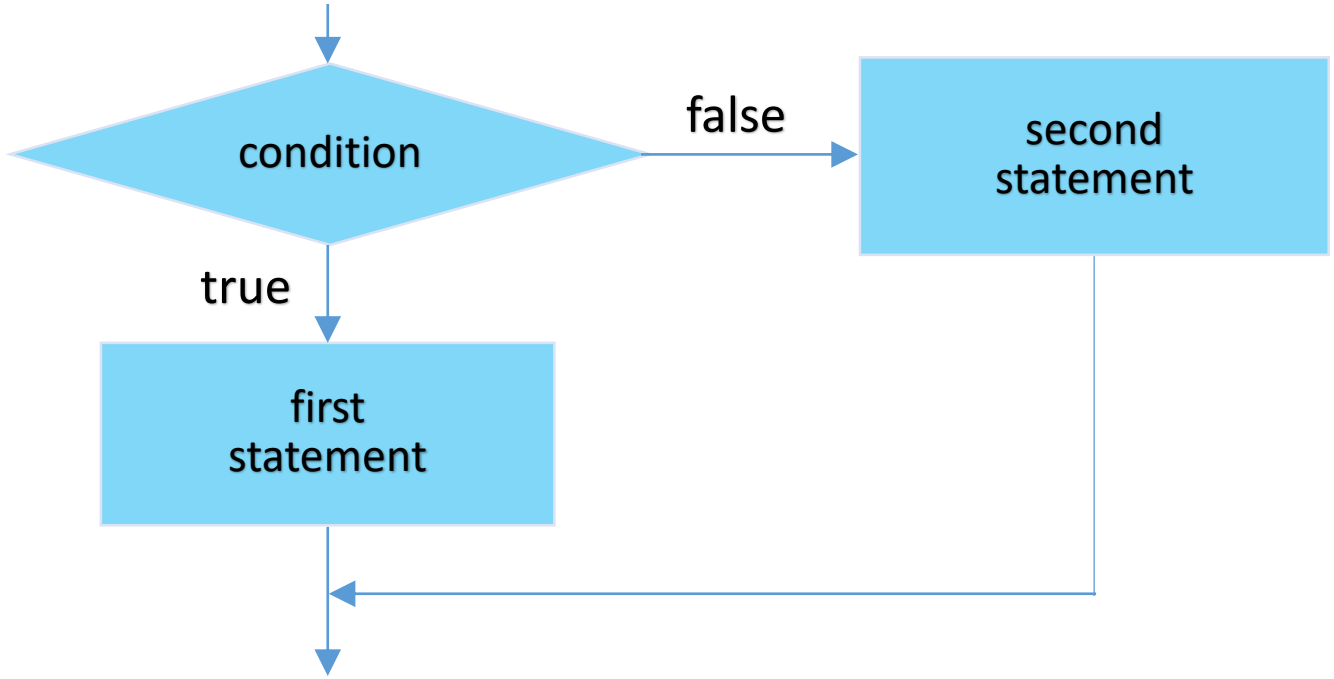
The `if` Statement > Example

```
static void Main()
{
    Console.WriteLine("Enter two numbers.");

    int biggerNumber = int.Parse(Console.ReadLine());
    int smallerNumber = int.Parse(Console.ReadLine());

    if (smallerNumber > biggerNumber)
    {
        biggerNumber = smallerNumber;
    }

    Console.WriteLine("The greater number is: {0}", biggerNumber);
}
```

The `if-else` Statement

- More complex and useful conditional statement
- Executes one branch if the condition is true, and another if it is false
- The simplest form of an `if-else` statement:

```
if (condition)
{
   statement1;
}
else
{
   statement2;
}
```



if-else Statement > Example

- Checking a number if it is odd or even

```
string s = Console.ReadLine();
int number = int.Parse(s);

if (number % 2 == 0)
{
    Console.WriteLine("This number is even.");
}
else
{
    Console.WriteLine("This number is odd.");
}
```

Nested `if` Statements

- `if` and `if-else` statements can be nested, i.e. used inside another `if` or `else` statement
- Every `else` corresponds to its closest preceding `if`

```
if (expression)
{
   if (expression)
   {
      statement;
   }
   else
   {
      statement;
   }
}
else
   statement;
```

Nested if Statements > Example

```
if (first == second)
{
    Console.WriteLine("These two numbers are equal.");
}
else
{
    if (first > second)
    {
        Console.WriteLine("The first number is bigger.");
    }
    else
    {
        Console.WriteLine("The second is bigger.");
    }
}
```

Multiple if-else-if-else-…

- Sometimes we need to use another `if`-construction in the `else` block
  - ✓ Thus `else if` can be used:

```
int ch = 'X';
if (ch == 'A' || ch == 'a')
{
   Console.WriteLine("Vowel [ei]");
}
else if (ch == 'E' || ch == 'e')
{
   Console.WriteLine("Vowel [i:]");
}
else if  …
else …
```

The switch-case Statement

- Selects for execution a statement from a list depending on the value of the `switch` expression

```
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("Error!"); break;
}
```

How switch-case Works?

1. The expression is evaluated
2. When one of the constants specified in a case label is equal to the expression
   - ✓ The statement that corresponds to that case is executed
3. If no case is equal to the expression
   - ✓ If there is default case, it is executed
   - ✓ Otherwise the control is transferred to the end point of the switch statement

switch-case > Usage good practice

- Variables types like `string`, `enum` and integral types can be used for `switch` expression
- The value `null` is permitted as a case label constant
- The keyword `break` exits the switch statement
- "No fall through" rule – you are obligated to use `break` after each case
- Multiple labels that correspond to the same statement are permitted
- There must be a separate `case` for every normal situation
- Put the normal case first
  - ✓ Put the most frequently executed cases first and the least frequently executed last
- Order cases alphabetically or numerically
- In `default` use case that cannot be reached under normal circumstances

Multiple Labels – Example

- You can use multiple labels to execute the same statement in more than one case

```
switch (animal)
{
    case "dog" :
        Console.WriteLine("MAMMAL");
        break;
    case "crocodile" :
    case "tortoise" :
    case "snake" :
        Console.WriteLine("REPTILE");
        break;
    default :
        Console.WriteLine("There is no such animal!");
        break;
}
```

Looping Statements

Execute Blocks of Code Multiple Times

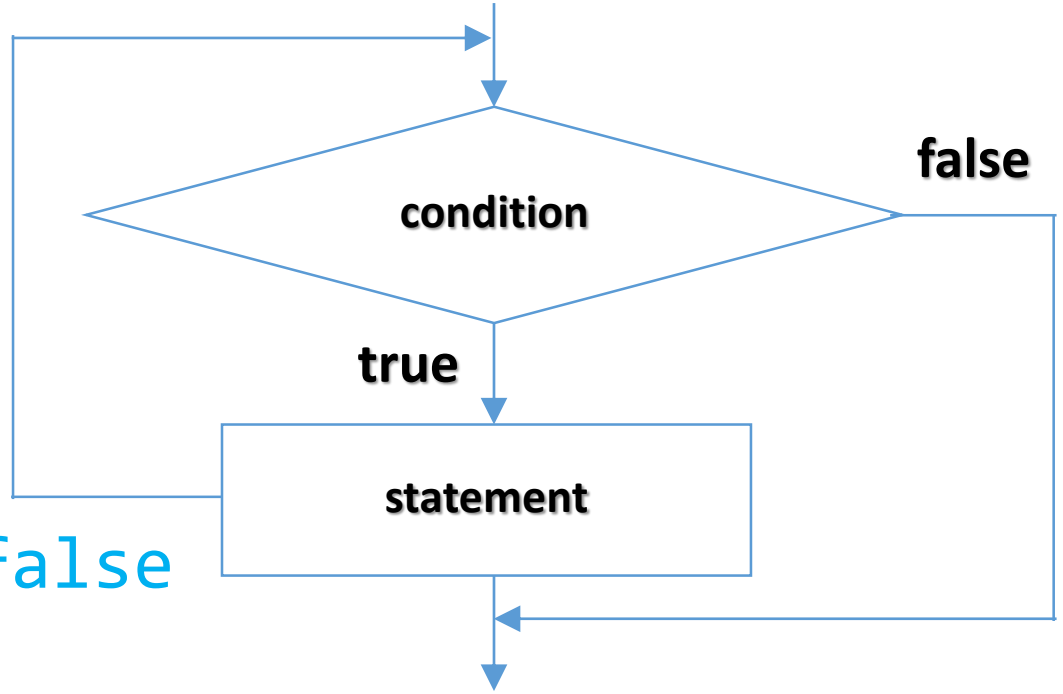
Looping Statements

- A loop is a control statement that allows repeating execution of a block of statements
  - ✓ May execute a code block fixed number of times
  - ✓ May execute a code block while given condition holds
  - ✓ May execute a code block for each member of a collection
- Loops that never end are called an infinite loops
- Loops in C#
  - ✓ `while` loops
  - ✓ `do` … `while` loops
  - ✓ `for` loops
  - ✓ `foreach` loops
- Nested loops

Using `while(…)` Loop

- Repeating a Statement While Given Condition Holds
- The simplest and most frequently used loop

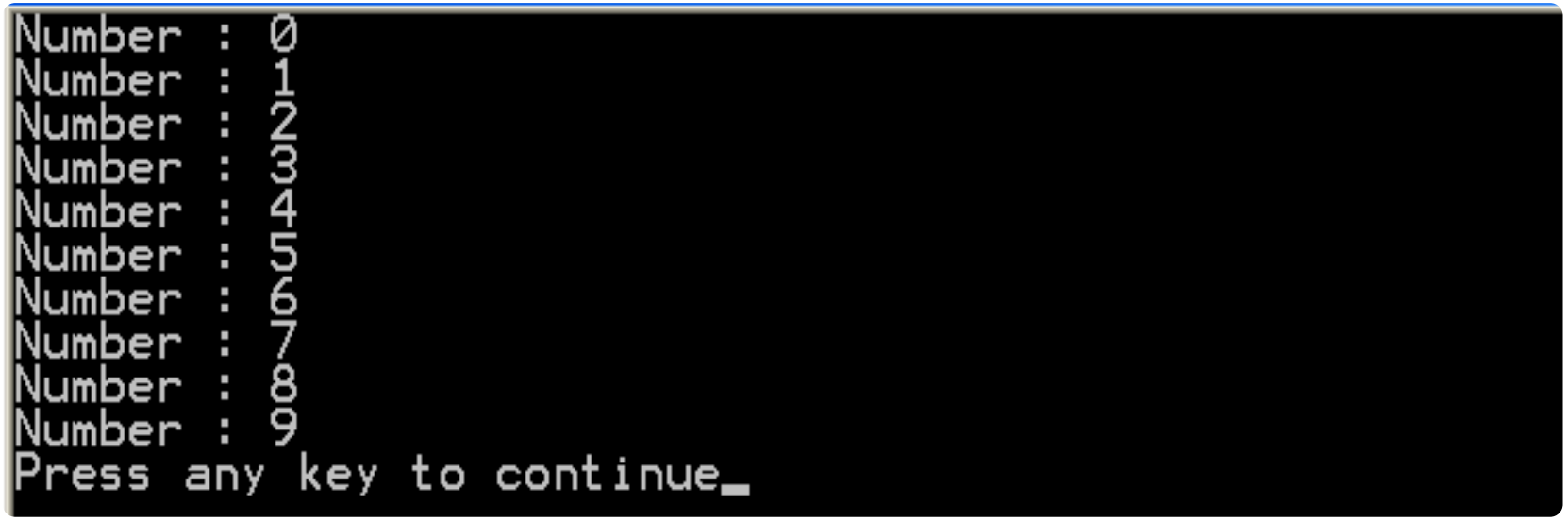
```
while (condition)
{
    statements;
}
```



- The repeat condition
  - ✓ Returns a boolean result of `true` or `false`
  - ✓ Also called loop condition

While Loop > Example

```
int counter = 0;
while (counter < 10)
{
    Console.WriteLine("Number : {0}", counter);
    counter++;
}
```



While Loop > Example

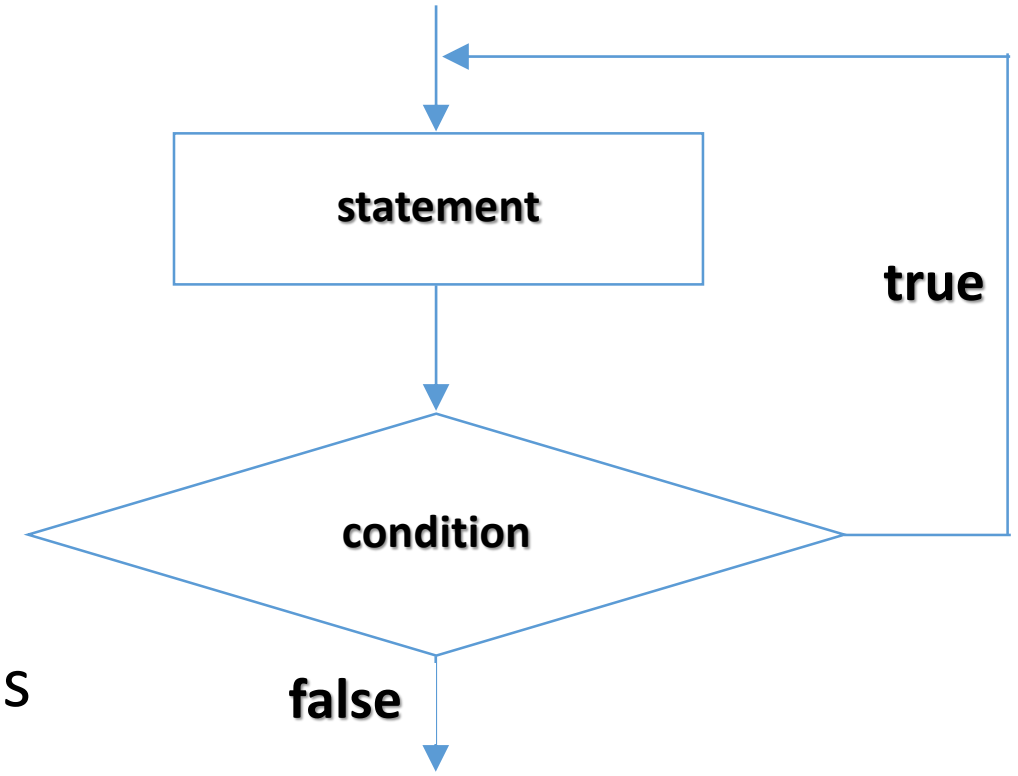
- Checking whether a number is prime or not

```
Console.Write("Enter a positive integer number: ");
uint number = uint.Parse(Console.ReadLine());
uint divider = 2;
uint maxDivider = (uint) Math.Sqrt(number);
bool prime = true;
while (prime && (divider <= maxDivider))
{
    if (number % divider == 0)
    {
        prime = false;
    }
    divider++;
}
Console.WriteLine("Prime? {0}", prime);
```

Do-While Loop

- Another loop structure is:

```
do
{
   statements;
}
while (condition);
```



- The block of statements is repeated
  - ✓ While the boolean loop condition holds
- The loop is executed at least once

Do … while > Example

- Calculating N factorial

```
static void Main()
{
    int n = Convert.ToInt32(Console.ReadLine());
    int factorial = 1;

    do
    {
        factorial *= n;
        n--;
    }
    while (n > 0);

    Console.WriteLine("n! = " + factorial);
}
```

for loops

- The typical `for` loop syntax is:

```
for (initialization; test; update)
{
    statements;
}
```

- Consists of
  - ✓ Initialization statement
    - > Executed once, just before the loop is entered
  - ✓ Boolean test expression
    - > Evaluated before each iteration of the loop
      - If `true`, the loop body is executed
      - If `false`, the loop body is skipped
  - ✓ Update statement
    - > Executed at each iteration after the body of the loop is finished
  - ✓ Loop body block

for Loop > Example

- A simple for-loop to print the numbers 0…9:

```
for (int number = 0; number < 10; number++)
{
    Console.Write(number + " ");
}
```

- A simple for-loop to calculate n!:

```
decimal factorial = 1;
for (int i = 1; i <= n; i++)
{
    factorial *= i;
}
```

foreach Loop

- Iteration over a Collection
- The typical `foreach` loop syntax is:

```
foreach (Type element in collection)
{
    statements;
}
```

- Iterates over all elements of a collection
  - ✓ The element is the loop variable that takes sequentially all collection values
  - ✓ The collection can be list, array or other group of elements of the same type

foreach Loop > Example

- Example of foreach loop:

```
string[] days = new string[] {
  "Monday", "Tuesday", "Wednesday", "Thursday",
  "Friday", "Saturday", "Sunday" };
foreach (string day in days)
{
  Console.WriteLine(day);
}
```

- The above loop iterates of the array of days
  - ✓ The variable day takes all its values

Nested Loops

- A composition of loops is called a nested loop
  - ✓ A loop inside another loop
- Example:

```
for (initialization; test; update)
{
   for (initialization; test; update)
   {
      statements;
   }
   …
}
```

Nested loop > Example

- Print the following triangle:

```
1
1 2
…
1 2 3 ... n
```

```
int n = int.Parse(Console.ReadLine());
for(int row = 1; row <= n; row++)
{
  for(int column = 1; column <= row; column++)
  {
    Console.Write("{0} ", column);
  }
  Console.WriteLine();
}
```

C# Jump Statements

- Jump statements are:
  - ✓`break`, `continue`, `goto`, `return`
- How `continue` woks?
  - ✓In `while` and `do-while` loops jumps to the test expression
  - ✓In `for` loops jumps to the update expression
- To exit an inner loop use `break`
- To exit outer loops use `goto` with a label
  - ✓ Avoid using `goto`! (it is considered harmful)
- `return` – to terminate method execution and go back to the caller method

Jump Statements > Example

```
int outerCounter = 0;
for (int outer = 0; outer < 10; outer++)
{
    for (int inner = 0; inner < 10; inner++)
    {
        if (inner % 3 == 0)
            continue;
        if (outer == 7)
            break;
        if (inner + outer > 9)
            goto breakOut;
    }
    outerCounter++;
}
breakOut:
```

Jump Statements > continue example

- Example: sum all odd numbers in [1, n] that are not divisors of 7:

```
int n = int.Parse(Console.ReadLine());
int sum = 0;
for (int i = 1; i <= n; i += 2)
{
   if (i % 7 == 0)
   {
      continue;
   }
   sum += i;
}
Console.WriteLine("sum = {0}", sum);
```

For more information

- Brian Bagnall, et al. C# for Java Programmers. USA. Syngress Publishing, Inc.
- http://www.tutorialspoint.com/csharp/
- Svetlin Nakov *et al*. Fundamentals of Computer Programming With C#. Sofia, 2013
- Joel Murach, Anne Boehm. Murach C# 2012, Mike Murach & Associates Inc USA, 2013

QUESTION!