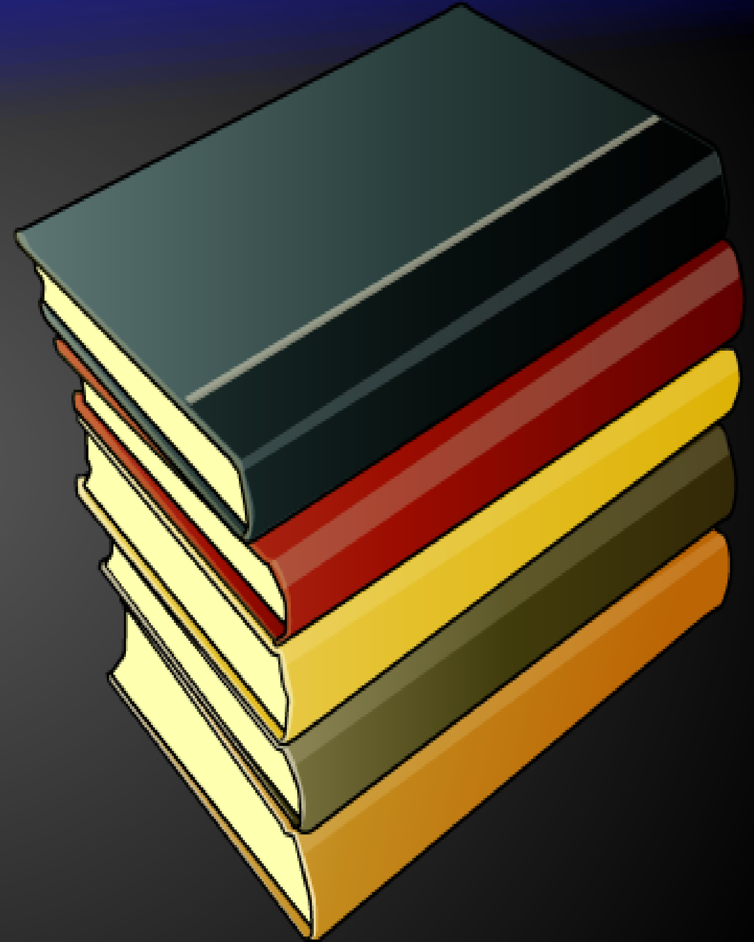
C# Language Overview (Part I)

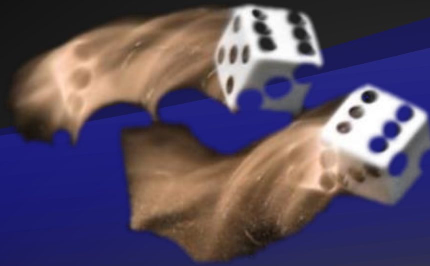
Data Types, Operators, Expressions, Statements, Console I/O, Loops, Arrays, Methods

Abdallah MOUJAHID – PMP®, COBIT® V5, ITIL® V3, ISO27002
abdallah.moujahid@uic.ac.ma

Table of Contents

1. Data Types
2. Operators
3. Expressions
4. Console I/O
5. Conditional Statements
6. Loops
7. Arrays
8. Methods





Primitive Data Types

Integer Types

- Integer types are:
  - **sbyte** (-128 to 127): signed 8-bit
  - **byte** (0 to 255): unsigned 8-bit
  - **short** (-32,768 to 32,767): signed 16-bit
  - **ushort** (0 to 65,535): unsigned 16-bit
  - **int** (-2,147,483,648 to 2,147,483,647): signed 32-bit
  - **uint** (0 to 4,294,967,295): unsigned 32-bit

Integer Types (2)

- More integer types:
  - `long` (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807): signed 64-bit
  - `ulong` (0 to 18,446,744,073,709,551,615): unsigned 64-bit

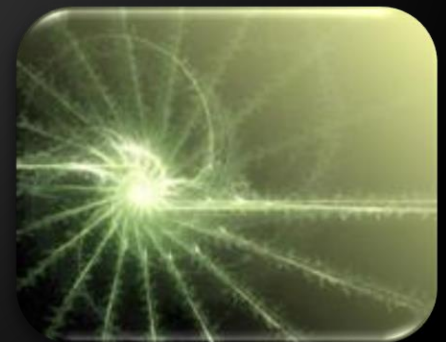
Integer Types – Example

- Measuring time
  - Depending on the unit of measure we may use different data types:

```
byte centuries = 20;    // Usually a small number
ushort years = 2000;
uint days = 730480;
ulong hours = 17531520; // May be a very big number
Console.WriteLine("{0} centuries is {1} years, or {2}
days, or {3} hours.", centuries, years, days, hours);
```

Floating-Point Types

- Floating-point types are:
  - `float` ($\pm1.5 \times 10^{-45}$ to $\pm3.4 \times 10^{38}$): 32-bits, precision of 7 digits
  - `double` ($\pm5.0 \times 10^{-324}$ to $\pm1.7 \times 10^{308}$): 64-bits, precision of 15-16 digits
- The default value of floating-point types:
  - Is `0.0F` for the `float` type
  - Is `0.0D` for the `double` type



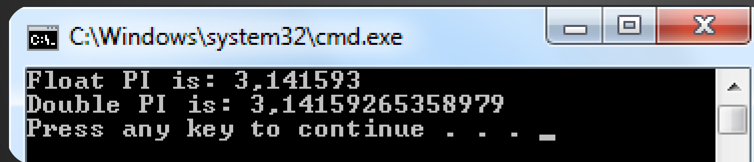
Fixed-Point Types

- There is a special fixed-point real number type:
  - `decimal` ($\pm 1,0 \times 10^{-28}$ to $\pm 7,9 \times 10^{28}$): 128-bits, precision of 28-29 digits
  - Used for financial calculations with low loss of precision
  - No round-off errors
- The default value of `decimal` type is:
  - `0.0M` (`M` is the suffix for decimal numbers)

PI Precision – Example

- See below the difference in precision when using `float` and `double`:

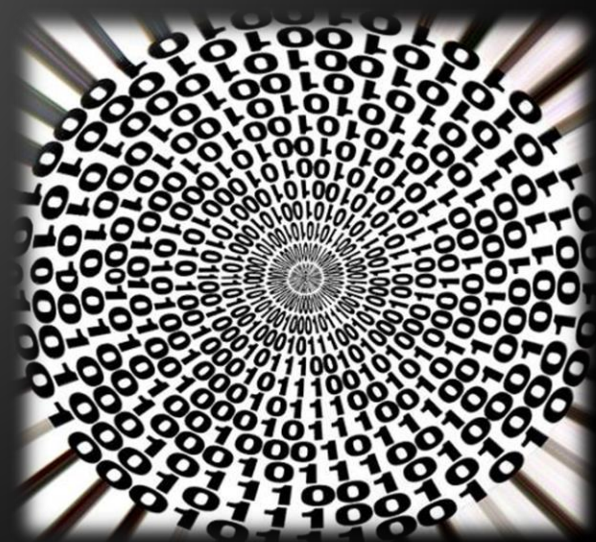
```
float floatPI = 3.141592653589793238f;
double doublePI = 3.141592653589793238;
Console.WriteLine("Float PI is: {0}", floatPI);
Console.WriteLine("Double PI is: {0}", doublePI);
```



- NOTE: The "f" suffix in the first statement!
  - Real numbers are by default interpreted as `double`!
  - One should explicitly convert them to `float`

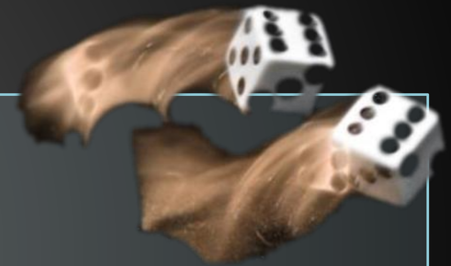
The Boolean Data Type

- The Boolean Data Type:
  - Is declared by the `bool` keyword
  - Has two possible values: `true` and `false`
  - Is useful in logical expressions
- The default value is `false`



Boolean Values – Example

- Here we can see how boolean variables take values of `true` or `false`:



```
int a = 1;
int b = 2;
bool greaterAB = (a > b);
Console.WriteLine(greaterAB);
// False
bool equalA1 = (a == 1);
Console.WriteLine(equalA1);
// True
```

The Character Data Type

- The Character Data Type:
  - Represents symbolic information
  - Is declared by the `char` keyword
  - Gives each symbol a corresponding integer code
  - Has a `'\0'` default value

Characters and Codes

- The example below shows that every symbol has an its unique code:

```
char symbol = 'a';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);

symbol = 'b';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);

symbol = 'A';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);
```

The String Data Type

- The String Data Type:
  - Represents a sequence of characters
  - Is declared by the `string` keyword
  - Has a default value `null` (no value)
- Strings are enclosed in quotes:

```
string s = "Microsoft .NET Framework";
```

- Strings can be concatenated

Saying Hello – Example

- Concatenating the two names of a person to obtain his full name:

```
string firstName = “Sami";
string lastName = “Mounir";
Console.WriteLine("Hello, {0}!", firstName);

string fullName = firstName + " " + lastName;
Console.WriteLine("Your full name is {0}.",
fullName);
```

- NOTE: a space is missing between the two names! We have to add it manually

The Object Type

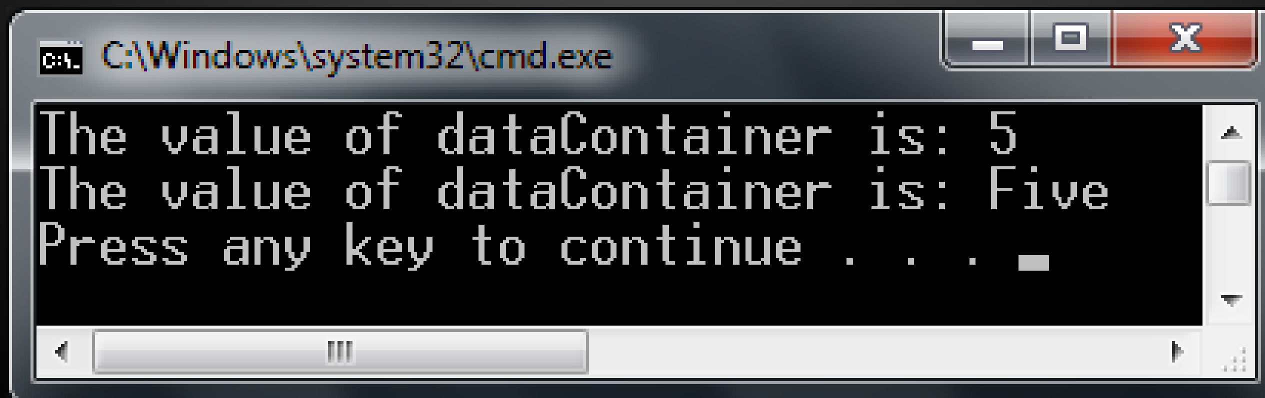
- The object type:
  - Is declared by the `object` keyword
  - Is the "parent" of all other types
  - Can take any types of values according to the needs

Using Objects

- Example of an object variable taking different types of data:

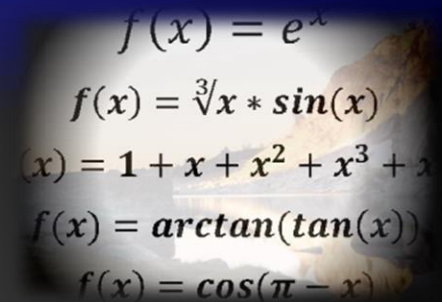
```
object dataContainer = 5;
Console.Write("The value of dataContainer is: ");
Console.WriteLine(dataContainer);

dataContainer = "Five";
Console.Write ("The value of dataContainer is: ");
Console.WriteLine(dataContainer);
```



Variables and Identifiers

Declaring Variables



- When declaring a variable we:
  - Specify its type
  - Specify its name (called identifier)
  - May give it an initial value
- The syntax is the following:

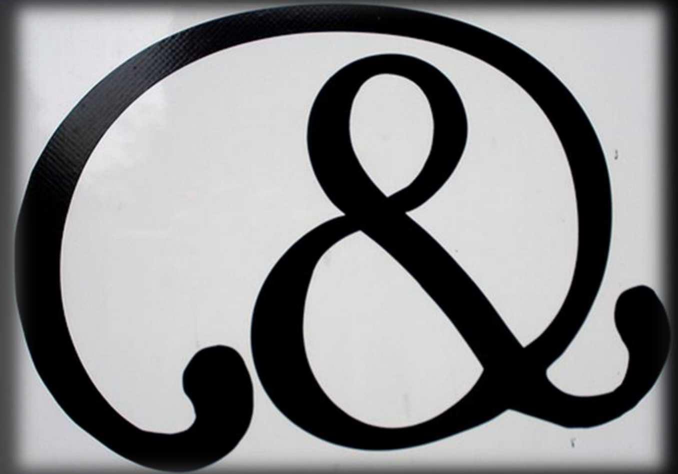
```
<data_type> <identifier> [= <initialization>];
```

- Example:

```
int height = 200;
```

Identifiers

- Identifiers may consist of:
  - Letters (Unicode)
  - Digits [0-9]
  - Underscore "_"
- Identifiers
  - Can begin only with a letter or an underscore
  - Cannot be a C# keyword



Identifiers (2)

- Identifiers
  - Should have a descriptive name
  - It is recommended to use only Latin letters
  - Should be neither too long nor too short
- Note:
  - In C# small letters are considered different than the capital letters (case sensitivity)

Operators in C#

Categories of Operators in C#

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

Arithmetic Operators

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

Logical Operators

- Logical operators take boolean operands and return boolean result
- Operator `!` turns `true` to `false` and `false` to `true`
- Behavior of the operators `&&`, `||` (`1 == true`, `0 == false`) :

<table>
<tr><th>Operation</th><th>||</th><th>||</th><th>||</th><th>||</th><th>&&</th><th>&&</th><th>&&</th><th>&&</th></tr>
<tr><td>Operand1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>Operand2</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>Result</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>

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 == b); // False
Console.WriteLine(a == a); // True
Console.WriteLine(a != ++b); // False
```

Assignment Operators

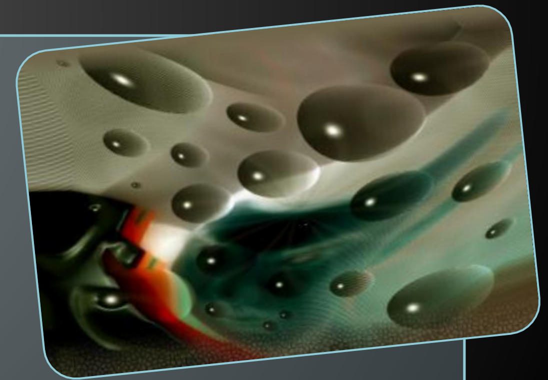
- Assignment operators are used to assign a value to a variable ,
  - =, +=, -=, |=, ...
- Assignment operators example:

```
int x = 8;
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); // 8
Console.WriteLine(x += 3); // 11
Console.WriteLine(x /= 2); // 4
```

Other Operators

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

```
string first = "First";
string second = "Second";
Console.WriteLine(first + second);
// FirstSecond
string output = "The number is : ";
int number = 5;
Console.WriteLine(output + number);
// The number is : 5
```



Other Operators (3)

- Conditional operator `?:` has the form

```
b ? x : y
```

(if `b` is true then the result is `x` else the result is `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

Type Conversions

- Example of implicit and explicit conversions:

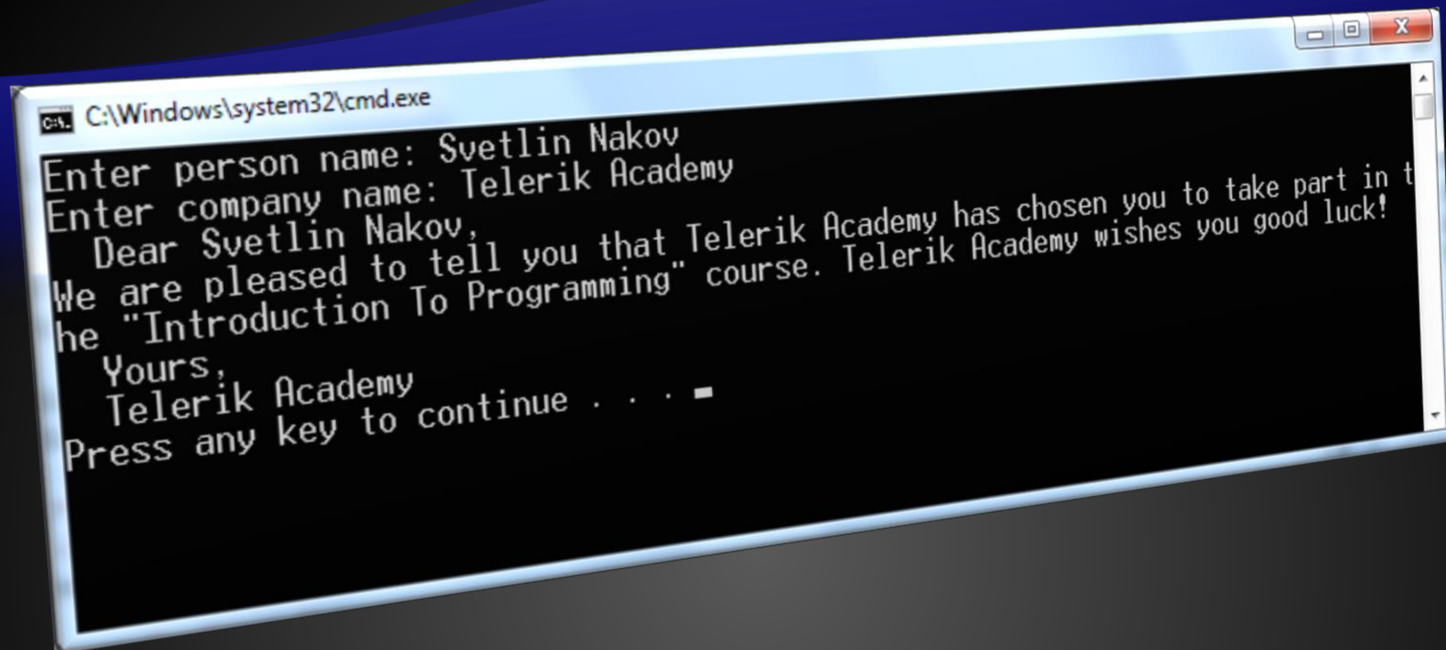
```
float heightInMeters = 1.74f; // Explicit conversion
double maxHeight = heightInMeters; // Implicit

double minHeight = (double) heightInMeters; // Explicit

float actualHeight = (float) maxHeight; // Explicit

float maxHeightFloat = maxHeight;
// Compilation error!
```

- Note: explicit conversion may be used even if not required by the compiler



Using to the Console

Printing / Reading Strings and Numbers

The Console Class

- Provides methods for input and output
- Input
  - `Read(…)` – reads a single character
  - `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 = "Marry";
int year = 1987;
...
Console.Write("{0} was born in {1}.", name, year);
// Marry was born in 1987.
```

- Next printing will start from the next line

Printing to the Console – Example

```
static void Main()
{
    string name = “Ahmed";
    int age = 18;
    string town = “Casablanca";

    Console.Write("{0} is {1} years old from {2}.",
        name, age, town);
    // Result: Ahmed is 18 years old from Casablanca.
    Console.Write("This is on the same line!");
    Console.WriteLine("Next sentence will be" +
        " on a new line.");

    Console.WriteLine("Bye, bye, {0} from {1}.",
        name, town);
}
```

Reading from the Console

- 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.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 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);
```

Converting Strings to Numbers

- Numeral types have a method `Parse(…)` for extracting the numeral value from a string
  - `int.Parse(string)` – `string` → `int`
  - `long.Parse(string)` – `string` → `long`
  - `float.Parse(string)` – `string` → `float`
  - Causes `FormatException` in case of error

```
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
```

Conditional Statements

Implementing Conditional Logic

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 (expression)
{
    statement1;
}
else
{
    statement2;
}
```

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.");
    }
}
```

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;
}
```

Loops

Repeating Statements Multiple Times

How To Use While Loop?

- 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++;
}
```

```
Number : 0
Number : 1
Number : 2
Number : 3
Number : 4
Number : 5
Number : 6
Number : 7
Number : 8
Number : 9
Press any key to continue_
```

Using 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

Factorial – Example

- Calculating N factorial

```
static void Main()
{
    int n = int.parse(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
  - Boolean test expression
  - Update statement
  - Loop body block

For-Each Loops

- 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;
    }
    …
}
```

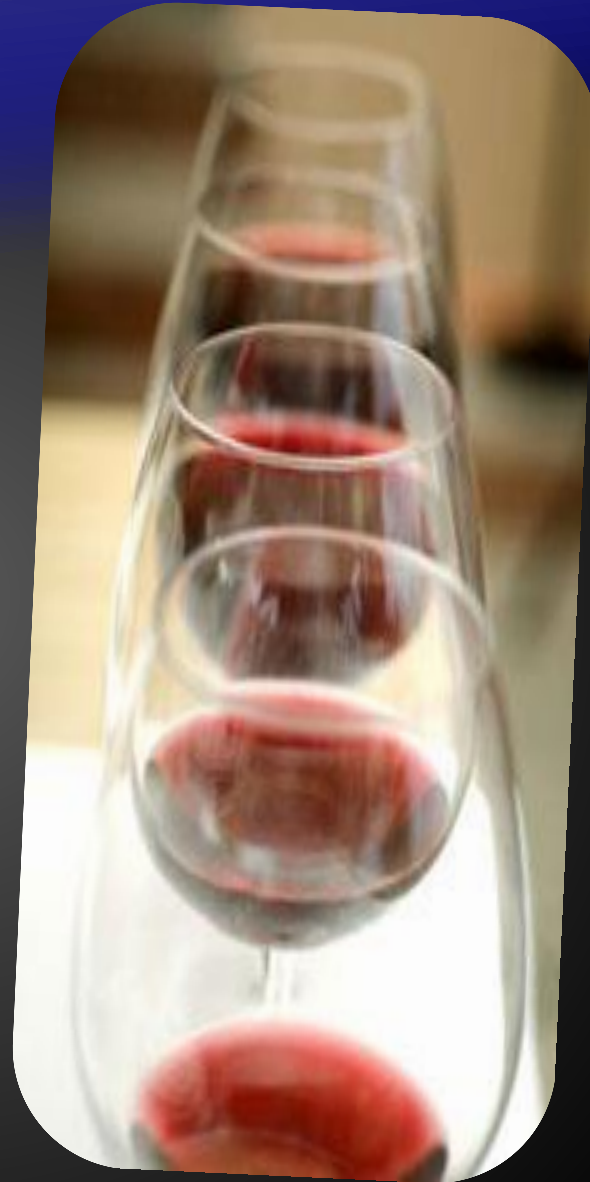
Nested Loops – Examples

- Print all combinations from TOTO 6/49

```
static void Main()
{
  int i1, i2, i3, i4, i5, i6;
  for (i1 = 1; i1 <= 44; i1++)
    for (i2 = i1 + 1; i2 <= 45; i2++)
      for (i3 = i2 + 1; i3 <= 46; i3++)
        for (i4 = i3 + 1; i4 <= 47; i4++)
          for (i5 = i4 + 1; i5 <= 48; i5++)
            for (i6 = i5 + 1; i6 <= 49; i6++)
              Console.WriteLine("{0} {1} {2} {3} {4} {5}",
                i1, i2, i3, i4, i5, i6);
}
```

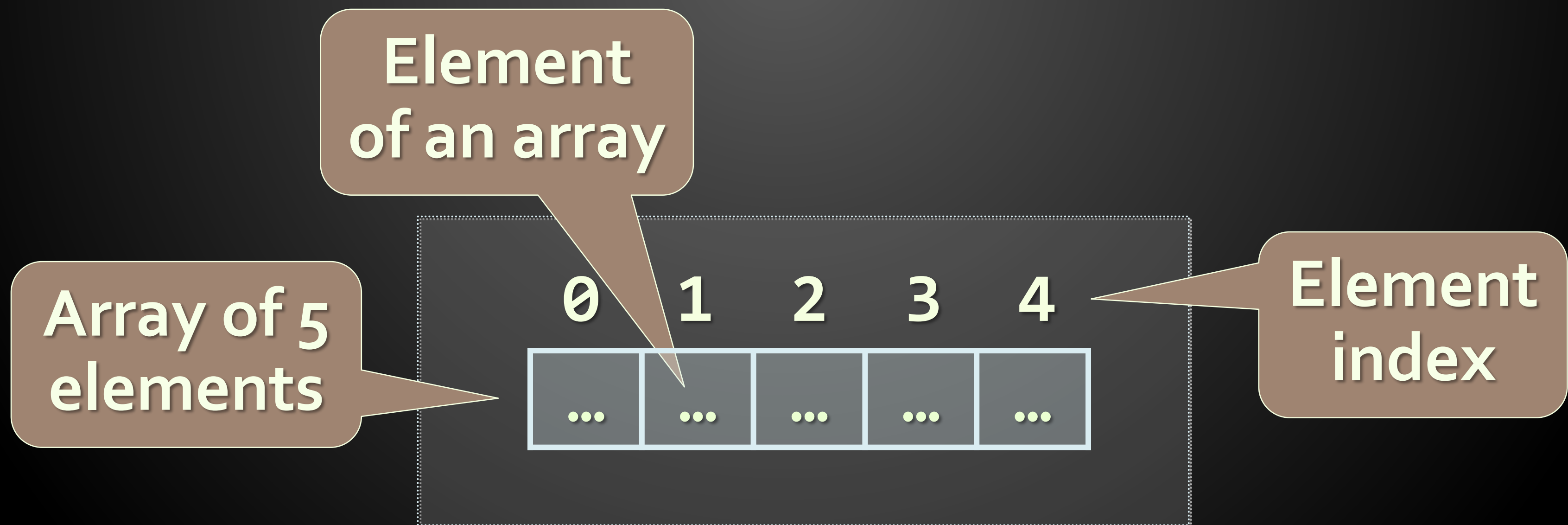
Warning: execution of this code could take too long time.

Arrays



What are Arrays?

- An array is a sequence of elements
  - All elements are of the same type
  - The order of the elements is fixed
  - Has fixed size (`Array.Length`)



Declaring Arrays

- Declaration defines the type of the elements
- Square brackets [ ] mean "array"
- Examples:
  - Declaring array of integers:

```
int[] myIntArray;
```

  - Declaring array of strings:

```
string[] myStringArray;
```

Creating Arrays

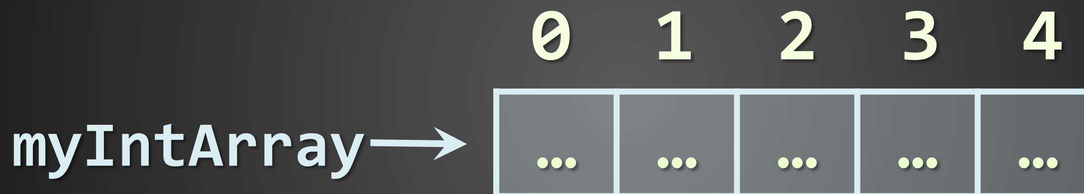
- Use the operator **new**
  - Specify array length
- Example creating (allocating) array of 5 integers:

```
myIntArray = new int[5];
```

Creating and Initializing Arrays

- Creating and initializing can be done together:

```
myIntArray = {1, 2, 3, 4, 5};
```



- The `new` operator is not required when using curly brackets initialization

Creating Array – Example

- Creating an array that contains the names of the days of the week

```
string[] daysOfWeek =
{
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
};
```

How to Access Array Element?

- Array elements are accessed using the square brackets operator `[]` (indexer)
  - Array indexer takes element's index as parameter
  - The first element has index `0`
  - The last element has index `Length-1`
- Array elements can be retrieved and changed by the `[]` operator

Processing Arrays: `foreach`

- How `foreach` loop works?

```
foreach (type value in array)
```

  - `type` – the type of the element
  - `value` – local name of variable
  - `array` – processing array

- Used when no indexing is needed
  - All elements are accessed one by one
  - Elements can not be modified (read only)

Processing Arrays Using foreach – Example

- Print all elements of a `string[]` array:

```
string[] capitals =
{
    "Sofia",
    "Washington",
    "London",
    "Paris"
};
foreach (string capital in capitals)
{
    Console.WriteLine(capital);
}
```

Multidimensional Arrays

- Multidimensional arrays have more than one dimension (2, 3, …)
  - The most important multidimensional arrays are the 2-dimensional
    - Known as matrices or tables
- Example of matrix of integers with 2 rows and 4 columns:

	0	1	2	3
0	5	0	-2	4
1	5	6	7	8

Declaring and Creating Multidimensional Arrays

- Declaring multidimensional arrays:

```
int[,] intMatrix;
float[,] floatMatrix;
string[,,] strCube;
```

- Creating a multidimensional array
  - Use new keyword
  - Must specify the size of each dimension

```
int[,] intMatrix = new int[3, 4];
float[,] floatMatrix = new float[8, 2];
string[,,] stringCube = new string[5, 5, 5];
```

Creating and Initializing Multidimensional Arrays

- Creating and initializing with values multidimensional array:

```
int[,] matrix =
{
    {1, 2, 3, 4}, // row 0 values
    {5, 6, 7, 8}, // row 1 values
}; // The matrix size is 2 x 4 (2 rows, 4 cols)
```

- Matrices are represented by a list of rows
  - Rows consist of list of values
- The first dimension comes first, the second comes next (inside the first)

Reading Matrix – Example

- Reading a matrix from the console

```
int rows = int.Parse(Console.ReadLine());
int cols = int.Parse(Console.ReadLine());
int[,] matrix = new int[rows, cols];
for (int row=0; row<rows; row++)
{
  for (int col=0; col<cols; col++)
  {
    Console.Write("matrix[{0},{1}] = ", row, col);
    matrix[row, col] =
      int.Parse(Console.ReadLine());
  }
}
```

Printing Matrix – Example

- Printing a matrix on the console:

```
for (int row=0; row<matrix.GetLength(0); row++)
{
  for (int col=0; col<matrix.GetLength(1); col++)
  {
    Console.Write("{0} ", matrix[row, col]);
  }
  Console.WriteLine();
}
```

Methods

Declaring and Using Methods

What is a Method?

- A method is a kind of building block that solves a small problem
  - A piece of code that has a name and can be called from the other code
  - Can take parameters and return a value
- Methods allow programmers to construct large programs from simple pieces
- Methods are also known as functions, procedures, and subroutines

Declaring and Creating Methods

```
using System;

class MethodExample
{
    static void PrintLogo()
    {
        Console.WriteLine(“UIC is private");
        Console.WriteLine("www.uic.ac.ma");
    }

    static void Main()
    {
        // ...
    }
}
```

- Methods are always declared inside a `class`
- `Main()` is also a method like all others

Calling Methods

- To call a method, simply use:
  - The method's name
  - Parentheses (don't forget them!)
  - A semicolon (;)

```
PrintLogo();
```

- This will execute the code in the method's body and will result in printing the following:

```
UIC is private
www.uic.ac.ma
```

Defining and Using Method Parameters

```
static void PrintSign(int number)
{
    if (number > 0)
        Console.WriteLine("Positive");
    else if (number < 0)
        Console.WriteLine("Negative");
    else
        Console.WriteLine("Zero");
}
```

- Method's behavior depends on its parameters
- Parameters can be of any type
  - `int`, `double`, `string`, etc.
  - arrays (`int[]`, `double[]`, etc.)

Defining and Using Method Parameters (2)

- Methods can have as many parameters as needed:

```
static void PrintMax(float number1, float number2)
{
    float max = number1;
    if (number2 > number1)
        max = number2;
    Console.WriteLine("Maximal number: {0}", max);
}
```

- The following syntax is not valid:

```
static void PrintMax(float number1, number2)
```

Calling Methods with Parameters

- To call a method and pass values to its parameters:
  - Use the method's name, followed by a list of expressions for each parameter
- Examples:

```
PrintSign(-5);
PrintSign(balance);
PrintSign(2+3);

PrintMax(100, 200);
PrintMax(oldQuantity * 1.5, quantity * 2);
```

Returning Values From Methods

- A method can return a value to its caller
- Returned value:
  - Can be assigned to a variable:

```
string message = Console.ReadLine();
// Console.ReadLine() returns a string
```

  - Can be used in expressions:

```
float price = GetPrice() * quantity * 1.20;
```

  - Can be passed to another method:

```
int age = int.Parse(Console.ReadLine());
```

Defining Methods That Return a Value

- Instead of **void**, specify the type of data to return

```
static int Multiply(int firstNum, int secondNum)
{
    return firstNum * secondNum;
}
```

- Methods can return any type of data (**int**, **string**, array, etc.)
- **void** methods do not return anything
- The combination of method's name, parameters and return value is called **method signature**
- Use **return** keyword to return a result

The `return` Statement

- The `return` statement:
  - Immediately terminates method's execution
  - Returns specified expression to the caller
  - Example:

```
return -1;
```

- To terminate `void` method, use just:

```
return;
```

- Return can be used several times in a method body