Școala informală de IT

Primitive Data Types and Variables in C#

Curriculum

- Primitive Data Types
  - Integer
  - Floating-Point
  - Boolean
  - Character
  - String
  - Object
- Declaring and Using Variables
  - Identifiers
  - Declaring Variables and Assigning Values
  - Literals
- Nullable Types
- Var Type
- Expressions

Primitive Data Types

How Computing Works?

- Computers are machines that process data
  - Data is stored in the computer memory in variables
  - Variables have name, data type and value
- Example of variable definition and assignment in C#

Variable name

Data type

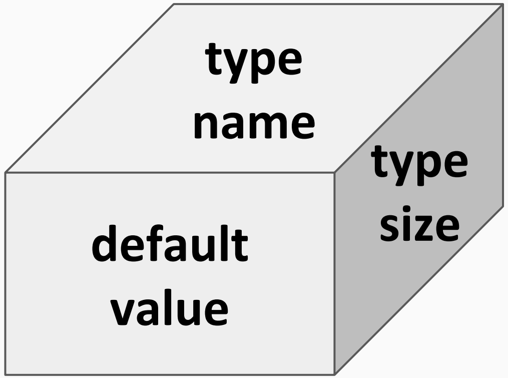
```
int count = 5;
```

Variable value

What is a Data Type?

- A data type:
  - Is a domain of values of similar characteristics
  - Defines the type of information stored in the computer memory (in a variable)

- Examples:
  - Positive integers: 1, 2, 3, ...
  - Alphabetical characters: a, b, c, ...
  - Days of week: `Monday`, `Tuesday`, ...



Data Type Characteristics

- A data type has:
  - Name (C# keyword or .NET type)
  - Size (how much memory is used)
  - Default value
- Example:
  - Integer numbers in C#
  - Name: `int`
  - Size: 32 bits (4 bytes)
  - Default value: 0

Integer Types

What are Integer Types?

- Integer types:
  - Represent whole numbers
  - May be signed or unsigned
  - Have range of values, depending on the size of memory used
- The default value of integer types is:
  - `0` – for integer types, except
  - `0L` – for the long type

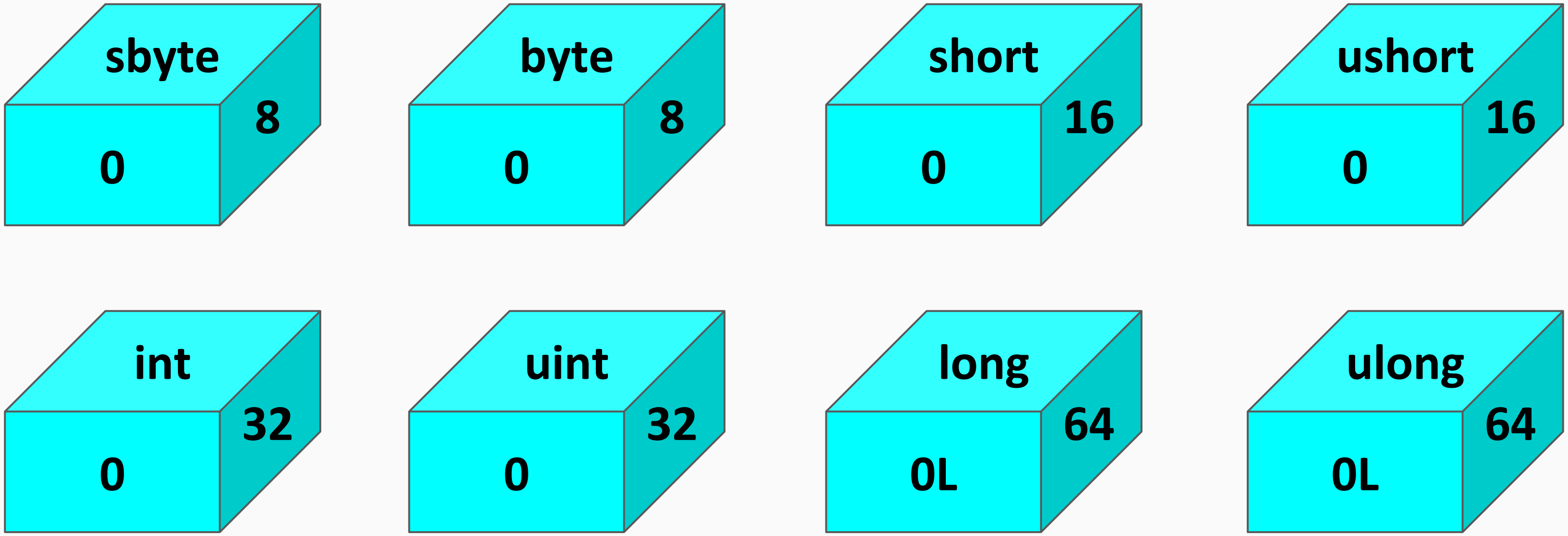
Integer Types

- 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

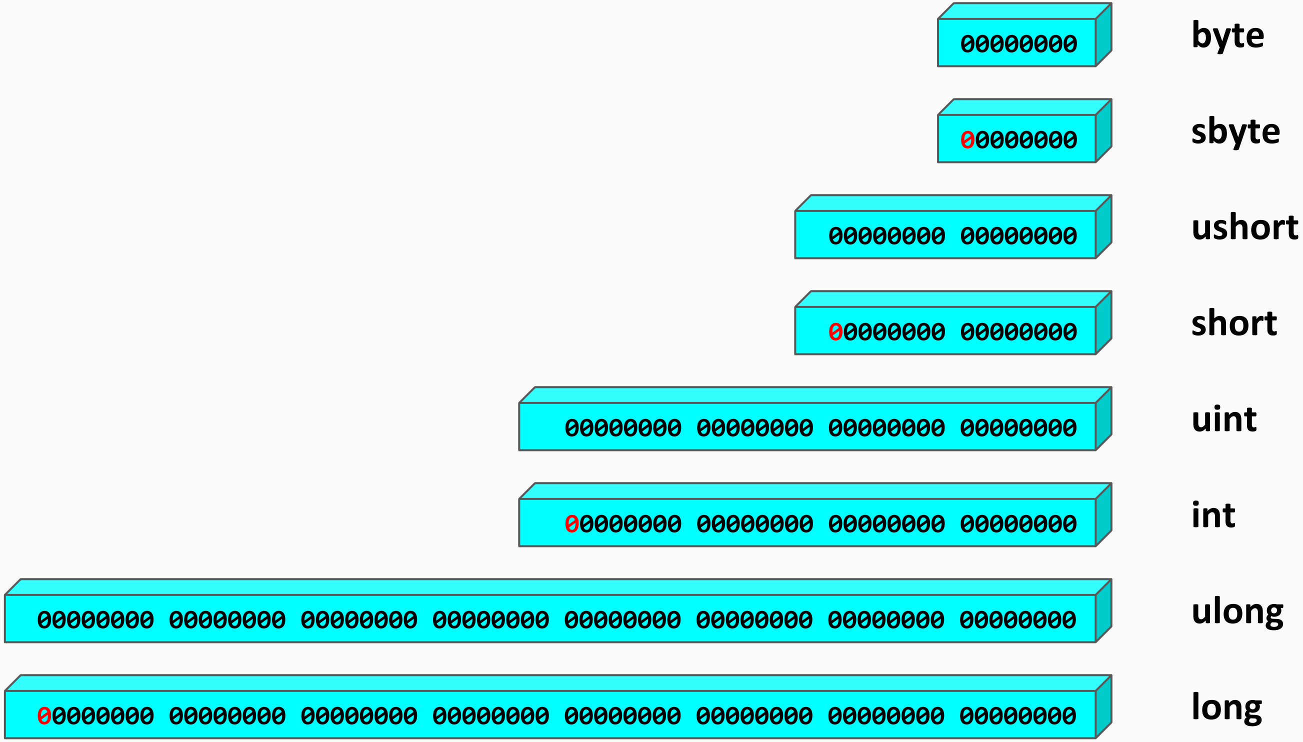
Integer Types (2)

- 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



Integer Types



Measuring Time - Example

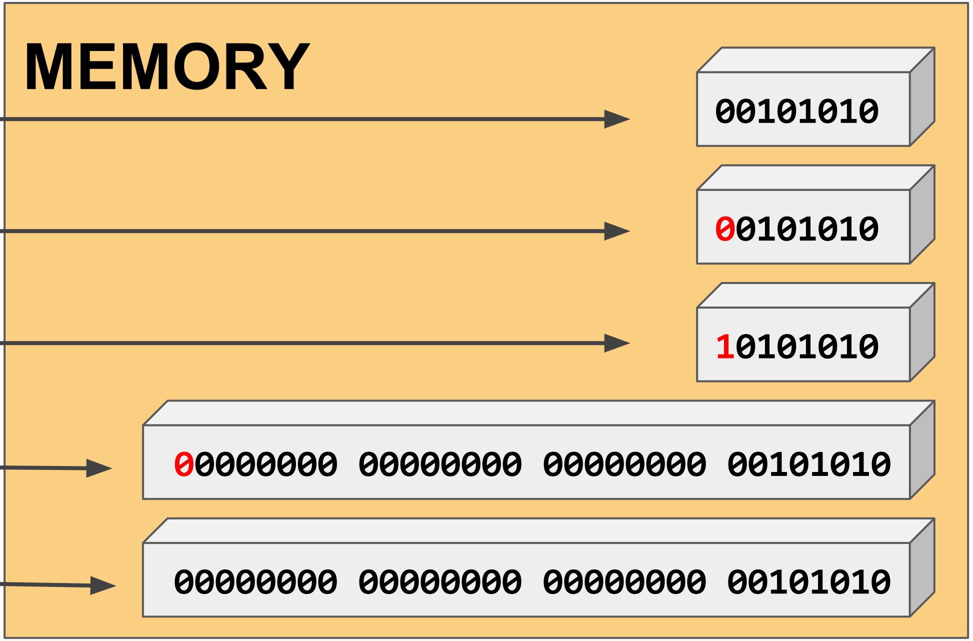
- Depending on the unit of measure we may use different data types:

```
byte centuries = 12;    // Usually a small number
ushort years = 12345;
uint days = 123456789;
ulong hours = 123456789023456789; // May be a very big number
Console.WriteLine("{0} centuries is {1} years, or {2} days,
or {3} hours.", centuries, years, days, hours);
```

Measuring Time - Example

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

```
byte myByte = 42;
sbyte myPosSByte = +42;
sbyte myNegSByte = -42;
int myInt = 42;
uint myUInt = 42U;
```



Floating-Point and Decimal Floating-Point Types

What are Floating-Point Types?

- Floating-point types:
  - Represent real numbers
  - May be signed or unsigned
  - Have range of values and different precision depending on the used memory
  - Can behave abnormally in the calculations

Floating-Point Types

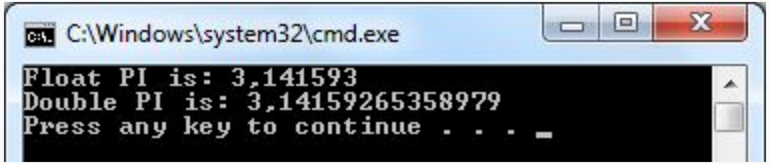
- Floating-point types are:
  - `float` (as small as $\pm 1.5 \times 10^{-45}$ to as big as $\pm 3.4 \times 10^{38}$)
    - 32-bits
    - precision of 7 digits
  - `double` (as small as $\pm 5.0 \times 10^{-324}$ to as big as $\pm 1.7 \times 10^{308}$)
    - 64-bits
    - precision of 15-16 digits

- The default value of floating-point types:
  - Is `0.0F` for the `float` type
  - Is `0.0D` for the double type

PI Precision - Example

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

```
float floatPI =   3.14159265358979323 8F;
```



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

Abnormalities in the Floating-Point Calculations

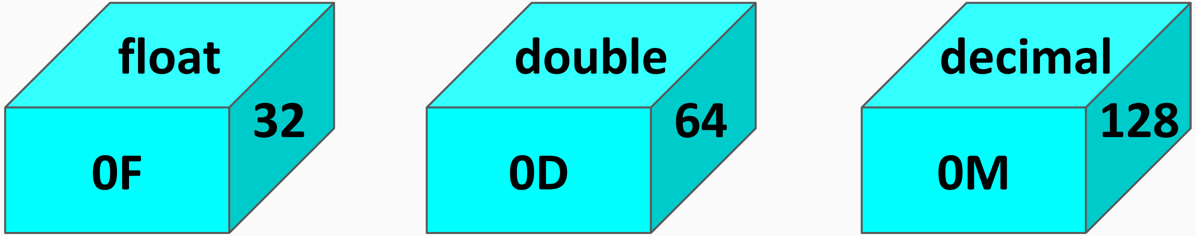
- **Sometimes abnormalities can be observed when using floating-point numbers**
  - **Comparing floating-point numbers can not be performed directly with the `==` operator**
- **Example:**

```
double a = 0.66F;
double b = 0.34F;
bool equal = (a + b == 1); // False!!!
Console.WriteLine($"a + b = {a + b} == 1, this is {equal}");
```

Decimal Floating-Point Types

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

Decimal Floating-Point Types



Boolean Type

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
  - 8-bits
- The default value is `false`

Boolean Values - Example

- Example of boolean variables taking 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
```

Character Type

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 (unicode null `\u0000`)
  - Takes 16 bits (2 bytes) of memory (from `U+0000` to `U+FFFF`)

Characters and Codes

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

```
char symbol = 'A'; // character literal
Console.WriteLine("Character {0} as literal", symbol);

symbol = '\x0041'; // hexadecimal
Console.WriteLine("Character {0} as decimal", symbol);

symbol = (char)65; // cast from integer type
Console.WriteLine("Character {0} as integer type", symbol);

symbol = '\u0041'; // unicode
Console.WriteLine("Character {0} as unicode", symbol);
```

String Type

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, not the same null as char `\0`)
- Strings are enclosed in quotes:

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

- Strings can be concatenated
  - Using the + operator

Saying Hello - Example

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

```
string firstName = "John";
string lastName = "Smith";
Console.WriteLine("Hello, {0}!\n", 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**

Object Type

The Object Type

- **The object type:**
  - **Is declared by the `object` keyword**
  - **Is the base type of all other types**
  - **Can hold values of any type**

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);
```

Introducing Variables

What is a variable?

- A variable is a:
  - Placeholder of information that can usually be changed at run-time
- Variables allow you to:
  - Store information
  - Retrieve the stored information
  - Manipulate the stored information

Variable Characteristics

- A variable has:
  - Name
  - Type (of stored data)
  - Value
- Example:

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

  - Name: `counter`
  - Type: `int`
  - Value: 5

Declaring and Using Variables

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)

Identifiers - Example

- **Examples of correct identifiers:**

```
int New = 2; // Here N is capital
int _2Pac; // This identifiers begins with _

string 你好 = "Hello"; // Unicode symbols used
// The following is more appropriate:
string greeting = "Hello";

int n = 100; // Undescriptive
int numberOfClients = 100; // Descriptive

// Overdescriptive identifier:
int numberOfPrivateClientOfTheFirm = 100;
```

- **Examples of incorrect identifiers:**

```
int new;   // new is a keyword
int 2Pac;  // Cannot begin with a digit
```

Assigning values to variables

Assigning values

- Assigning of values to variables
  - Is achieved by the = operator
- The = operator has
  - Variable identifier on the left
  - Value of the corresponding data type on the right
  - Could be used in a cascade calling, where assigning is done from right to left

Assigning values - Examples

- **Assigning values example:**

```
int firstValue = 5;
int secondValue;
int thirdValue;

// Using an already declared variable:
secondValue = firstValue;

// The following cascade calling assigns 3 to firstValue and then
// firstValue to thirdValue, so both variables have the value 3
// as a result:

thirdValue = firstValue = 3; // Avoid this!
```

Initializing variables

- Initializing
  - Is assigning of initial value
  - Must be done before the variable is used!
- Several ways of initializing:
  - By using the `new` keyword
  - By using a literal expression
  - By referring to an already initialized variable

Initialization - Examples

- **Example of some initializations:**

```
// The following would assign the default
// value of the int type to num:
int num = new int(); // num = 0

// This is how we use a literal expression:
float heightInMeters = 1.74F;

// Here we use an already initialized variable:
string greeting = "Hello World!";
string message = greeting;
```

Literals

What are Literals?

- Literals are:
  - Representations of values in the source code
- There are six types of literals
  - Boolean
  - Integer
  - Real
  - Character
  - String
  - The `null` literal

Boolean and Integer Literals

- The boolean literals are:
  - `true`
  - `false`
- The integer literals:
  - Are used for variables of type `int`, `uint`, `long`, and `ulong`
  - Consist of digits
  - May have a sign (`+`, `-`)
  - May be in a hexadecimal format

Integer Literals

- Examples of integer literals
  - The '`0x`' and '`0X`' prefixes mean a hexadecimal value, e.g. `0xA8F1`
  - The '`u`' and '`U`' suffixes mean a `ulong` or `uint` type, e.g. `1234567U`
  - The '`l`' and '`L`' suffixes mean a `long` or `ulong` type, e.g. `9876543L`

Integer Literals - Example

- The letter ‘l’ is easily confused with the digit ‘1’ so it’s better to use ‘L’!!!

```
// The following variables are initialized with the same value:
int numberInHex = -0x10;
int numberInDec = -16;

// The following causes an error, because 234u is of type uint
int unsignedInt = 234U;

// The following causes an error, because 234L is of type long
int longInt = 234L;

object myObject = null;
```

Real Literals

- The real literals:
  - Are used for values of type `float`, `double` and `decimal`
  - May consist of digits, a sign and "`.`"
  - May be in exponential notation: `6.02E+23`
- The "`f`" and "`F`" suffixes mean `float`
- The "`d`" and "`D`" suffixes mean `double`
- The "`m`" and "`M`" suffixes mean `decimal`
- The default interpretation is `double`

Real Literals - Example

- **Example of incorrect `float` literal:**

```
// The following causes an error because 12.5 is double by default
float realNumber = 12.5;
```

- **A correct way to assign floating-point value (using also the exponential format):**

```
// The following is the correct way of assigning the value:
float realNumber = 12.5F;

// This is the same value in exponential format:
realNumber = 1.25E+7F;
```

Character Literals

- The character literals:
  - Are used for values of the `char` type
  - Consist of two single quotes surrounding the character value: `'<value>'`
- The value may be:
  - Symbol
  - The code of the symbol
  - Escaping sequence

Escaping Sequences

- Escaping sequences are:
  - Means of presenting a symbol that is usually interpreted otherwise (like `'` or `\`)
  - Means of presenting system symbols (like the new line symbol)
- Common escaping sequences are:
  - `\'` for single quote `\"` for double quote
  - `\\` for backslash `\n` for new line
  - `\uXXXX` for denoting any other Unicode symbol

Character Literals - Example

- **Examples of different character literals:**

```
char symbol = 'a'; // An ordinary symbol
symbol = '\x006F'; // Unicode symbol code in a hexadecimal format
symbol = '\u8449'; // 葉 (Leaf in Traditional Chinese)
symbol = '\''; // Assigning the single quote symbol
symbol = '\\'; // Assigning the backslash symbol
symbol = '\n'; // Assigning new line symbol
symbol = '\0'; // Assigning null symbol
symbol = '\t'; // Assigning tab symbol
symbol = "a"; // Incorrect: use single quotes
```

String Literals

- String literals:
  - Are used for values of the string type
  - Consist of two double quotes surrounding the value: `"<value>"`
  - May have a `@` prefix which ignores the used escaping sequences: `@"<value>"`
- The value is a sequence of character literals

```
string s = "I am a string literal";
```

String Literals - Example

- **Benefits of quoted strings (the @ prefix):**

```
// Here is a string literal using escape sequences
string quotation = "\"Hello, Jude\", he said.";
string path = "C:\\WINNT\\Darts\\Darts.exe";

// Here is an example of the usage of @
quotation = @"""Hello, Jimmy!"", she answered.";
path = @"C:\WINNT\Darts\Darts.exe";

string str = @"some
        Text on the other line";
```

- **In quoted strings `\"` is used instead of `""`!**

Nullable Types

Nullable Types

- `Nullable` types are instances of the `System.Nullable` struct
  - Wrapper over the primitive types
  - E.g. `int?`, `double?`, etc.
- `Nullable` type can represent the normal range of values for its underlying value type, plus an additional `null` value
- Useful when dealing with databases or other structures that have default value `null`

Nullable Types - Example

Example with `integer`:

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


age = 5;
Console.WriteLine("This is the integer with value 5 -> " +  age);
```

Nullable Types - Example (2)

Example with double:

```
double? weight = null;
Console.WriteLine("This is the double with Null value -> " + weight);


weight = 2.5;
Console.WriteLine("This is the double with value 5 -> " + weight);
```

Var type

Var type

- var type - implicit type determined by compiler
- no performance penalty
- syntactic sugar

```
var x; // it will not compile! value needed for inference
var y = 10; // type of y inferred to int
var z = 10M; // type of z inferred to decimal
var obj = new Object(); // type of obj inferred to Object
var myDog = new Dog(); // type of myDog inferred to Dog
```

Implicit and Explicit Type Conversions

Implicit Type Casting

- Implicit type casting
  - Automatic conversion of value of one data type to value of another data type
  - Allowed when no loss of data is possible

    "Larger" types can implicitly take values of smaller "types"
  - Example:

```
int myInt = 5;
long myLong = myInt;    // implicit conversion
```

Explicit Type Casting

- Explicit type casting
  - Manual conversion of a value of one data type to a value of another data type
  - Allowed only explicitly by (type) operator
  - Required when there is a possibility of loss of data or precision
  - Example:

```
long myLong = 5;            // implicit conversion
int myInt = (int)myLong;    // explicit conversion
```

Type Casting - Examples

- **Example of implicit and explicit casting:**

```
float heightInMeters = 1.74F;

double maxHeight = heightInMeters;          // Implicit

double minHeight = (double)heightInMeters; // Explicit

float actualHeight = (float)maxHeight;     // Explicit
```

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

Expressions

Expressions

- Expressions are sequences of operators, literals and variables that are evaluated to some value
- Parentheses are used to force evaluation order and for readability
- Examples:

```
float r = (72 - 1) / 2 + 5;    // r = 40

// Expression for calculation of circle area
double surface = Math.PI * r * r;

// Expression for calculation of circle perimeter
double perimeter = 2 * Math.PI * r;
```

Expressions (2)

- Expressions has:
  - Type (integer, real, boolean, ...)
  - Value
- Examples:

Expression of type *int*. Calculated at compile time.

Expression of type *int*. Calculated at runtime.

Expression of type *bool*. Calculated at runtime.

```
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));
```