C# Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```
1  class Lecture2
2  {
3              "Data types, Variables, and Operators"
4  }
5
6  // Keywords:
7  byte, sbyte, short, ushort, int, uint, long, ulong, char, float,
8  double, decimal, bool, true, false
```

Example

Given the radius of a circle, say 10, determine the area.

- Input: how to store the data?
- Algorithm: how to compute the area?
- Output: how to show the result?

```
1  using System;
2
3  namespace ComputeCircleArea
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              // INPUT
10             int r = 10;
11
12             // ALGORITHM
13             double A = r * r * 3.14;
14
15             // OUTPUT
16             Console.WriteLine(A);
17         }
18     }
19 }
```

- Here we use two of primitive types: int, double.

Variable $\approx$ Box

Variable Declaration

- First, we name the variable, say x.
- We then need to determine a proper type for x.
- For example,

```
1 ...
2         int x; // x is a variable declared an integer type.
3 ...
```

Naming Rules

- The naming rule excludes the following cases:
  - cannot start with a digit;
  - cannot be any keyword[1];
  - cannot include any blank between letters;
  - cannot contain operators, like $+, -, *, /$.
- Note that C# is case sensitive.[2]
- These rules are also applicable to methods, classes (, and more).

---

[1] See `https://docs.microsoft.com/zh-tw/dotnet/csharp/language-reference/keywords/`. C# provides 78 reserved keywords and 30 contextual keywords.

[2] For example, the letter A is different from the letter a.

Things behind Variable Declaration

- Variable declaration is actually asking the compiler to allocate a memory space for the variable.
- In particular, its data type determines the size of memory allocation.
- The size is quantified in bits or bytes.
  - A bit presents a binary digit.
  - 1 byte is equal to 8 bits.
- For example, 32 bits (or 4 bytes) are allocated for an int value.

Digression: Binary System

- We are familiar with the decimal system. (Why?)
- For computers, the binary system is adopted because of its nature: the only two states, on and off.
- However, both are equivalent except that they differ in representations.
- For example,

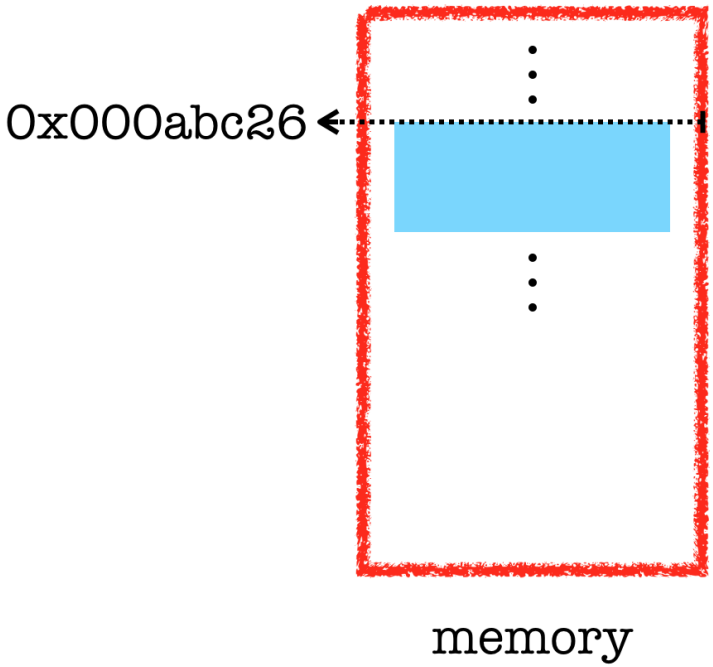
$$999_{10} = 9 \times 10^2 + 9 \times 10^1 + 9 \times 10^0.$$

- Similarly,

$$111_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7_{10}.$$

- No need to write code in binary because we are using high-level languages.

Variable as Alias of Memory Address



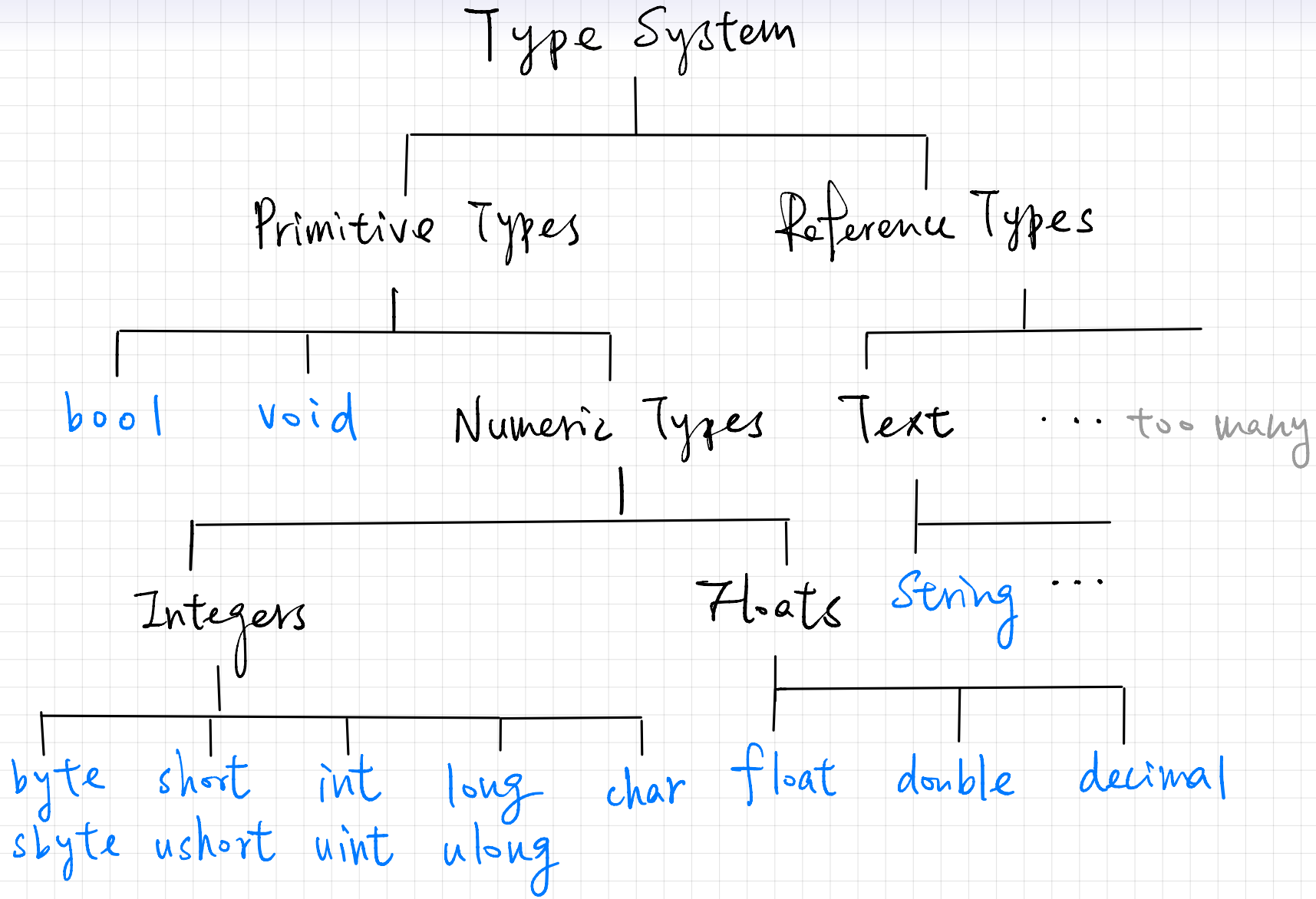
- Literals that start with 0x are hexadecimal (hex) integers.[3]
- Hexadecimal numbers are widely used to present, say a memory address and a color.[4]

[3] See `https://en.wikipedia.org/wiki/Hexadecimal`.
[4] Try `https://htmlcolorcodes.com/`.

Data Types

- Every variable needs a type.
- Also, every statement (or expression) has a type.
- The notion of data types is vital to programming languages.
  - I would say that, the idea of data types acts like the law of nature.
- C# is a static-typed language.
  - A variable is available after declaration.
- We now proceed to introduce the two categories of data types: primitive types, and reference types.[5]

[5] We will meet those reference types soon.


Type System
Primitive Types
Reference Types
bool
void
Numeric Types
Text
... too many
Integers
Floats
String
...
byte
short
int
long
char
sbyte
ushort
uint
ulong
float
double
decimal

Integers

Name	Bits	Range	Approx. range
byte	8	0 to 255	255
sbyte	8	-128 to 127	127
short	16	-32768 to 32767	$\pm 3 \times 10^4$
ushort	16	0 to 65535	$\sim 6 \times 10^4$
int	32	-2147483648 to 2147483647	$\pm 2 \times 10^9$
uint	32	0 to 4294967295	$\sim 4 \times 10^9$
long	64	-9223372036854775808 to 9223372036854775807	$\pm 9 \times 10^{18}$
ulong	64	0 to 18446744073709551615	$\sim 1.8 \times 10^{19}$

- The most commonly used integer type is int.
- As you can see, the range is limited due to finite size.
- If a value exceeds the feasible range of the type, then an overflow occurs.

Floats

Name	Bits	Approximate range
float	32	$\pm 1.4e-45$ to $\pm 3.4e+38$
double	64	$\pm 4.9e-324$ to $\pm 1.8e+308$
decimal	128	$\pm 1e-28$ to $\pm 7.9e+28$

- The notation e represents the scientific notation, based 10.
  - For example, $1e2 = 100$ and $-1.8e-3 = -0.0018$.
  - You may use the capital E for same purpose.
- Floats are used when evaluating expressions that require fractional precision.
  - For example, sin(), cos(), and sqrt().
- If so, the integers seem redundant!
- However, floating-point arithmetic can only approximate real arithmetic! (Why?)

Machine Error by Example[7]

```
1  using System;
2
3  class NumericalErrorDemo
4  {
5      static void Main(string[] args)
6      {
7          Console.WriteLine(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
8          // Output?
9      }
10 }
```

- This issue occurs for every number represented in floats.
- So floats are inaccurate due to finite precision unless the algorithm is designed elaborately for machine errors.
- In critical applications, we don't use floats but integers.[6]

[6] Also read https://news.cnyes.com/news/id/3680649.

[7] See https://en.wikipedia.org/wiki/Numerical_error and https://0.30000000000000004.com/.

Another Example

```
1 ...
2         Console.WriteLine(3.14 + 1e20 - 1e20);   // Output?
3         Console.WriteLine(3.14 + (1e20 - 1e20)); // Output?
4 ...
```

- Can you explain why?
- Read this article: What Every Computer Scientist Should Know About Floating-Point Arithmetic.

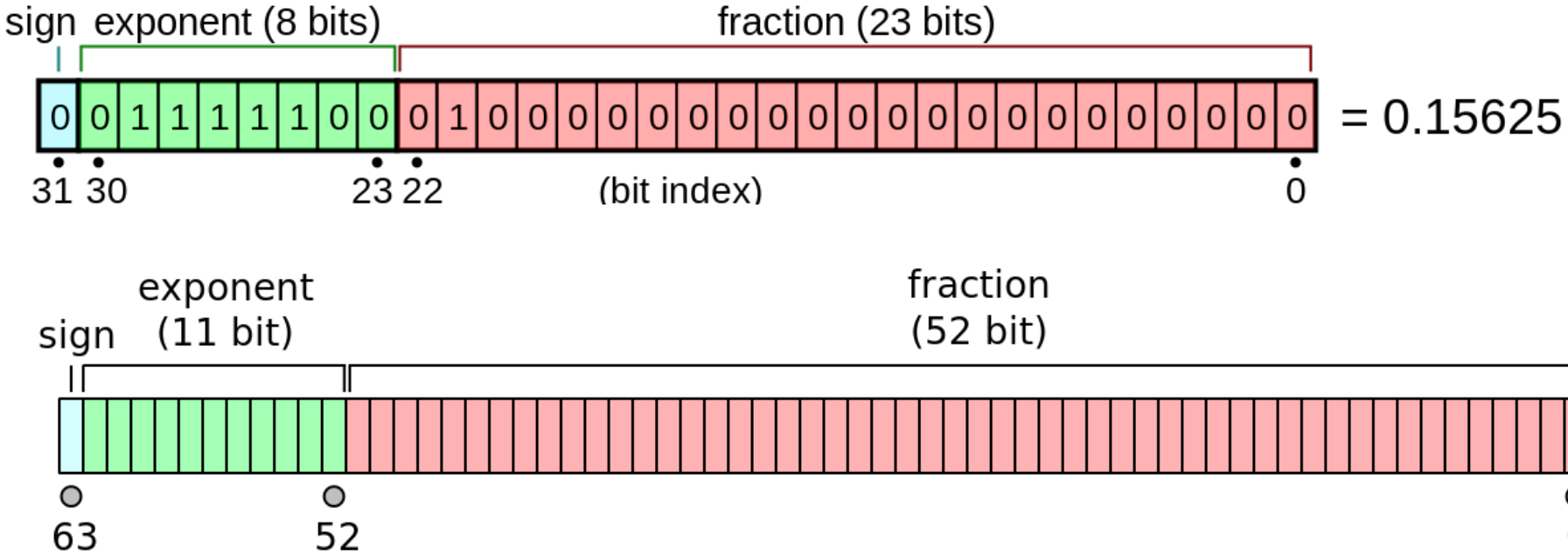
IEEE Floating-Point Representation[8]

$$x = (-1)^s \times M \times 2^E$$

- The sign bit s determines whether the number is negative ($s = 1$) or positive ($s = 0$).
- The mantissa M is a fractional binary number that ranges either between 1 and $2 - \varepsilon$, or between 0 and $1 - \varepsilon$.
- The exponent E weights the value by a (possibly negative) power of 2.

[8] IEEE754; William Kahan (1985). Also see `https://en.wikipedia.org/wiki/Double-precision_floating-point_format`.

Illustration



- That is why we call a double value.
- Double values have at least 16 significant digits; while decimal values have 29 significant digits!

Assignments

- An assignment statement designates a value to the variable.

```
1        int x;     // Variable declaration.
2        ...
3        x = 1;   // Assign 1 to x.
```

- The equal sign (=) is used as the assignment operator.
  - For example, is the expression $x = x + 1$ correct?
  - Direction: from the right-hand side to the left-hand side
- To assign a value to a variable, you must place the variable name to the left of the assignment operator.[9]
  - For example, $1 = x$ is wrong.
  - 1 cannot be resolved to a memory space.

[9] x can be a l-value and r-value, but 1 and other numbers can be only r-value but not l-value. See Value.

Two "Before" Rules

- A variable must be declared before it can be assigned a value.
  - In practice, do not declare the variable until you need it.
- A declared variable must be assigned a value before it can be used.

Arithmetic Operators

Operator	Operation	Example	Result
$+$	Addition	$12 + 34$	46
$-$	Subtraction	$56 - 78$	-22
$*$	Multiplication	90 * 12	1080
/	Division	3.0 / 2.0	1.5
%	Remainder	20 % 3	2

- What if 3 / 2?
- Note that the operator depends on the operands involved.

Check Point

```
1 ...
2         double x = 1 / 2;
3         Console.WriteLine(x); // Output?
4 ...
```

- What is the output?
- Can you explain this result?

Type Conversion and Compatibility

- If a type is compatible to another, then the compiler will perform the conversion implicitly.
  - For example, the integer 1 is compatible to a double value 1.0.
  - Therefore, C# is a weakly-typed language.[10]
- However, there is no automatic conversion from double to int. (Why?)
- To do so, you must use a cast, which performs an explicit conversion.
- Similarly, a long value is not compatible to int.

[10] See `https://android.jlelse.eu/magic-lies-here-statically-typed-vs-dynamically-typed-languages-d151c7`

Casting

```
1 ...
2         int x = 1;
3         double y = x; // Compatible; implicitly converted.
4         x = y;        // Not allowed unless casting.
5         x = (int) y;  // Succeeded!!
6 ...
```

- Note that the C# compiler does only type checking but no real execution before compilation.
- In other words, the values of x and y are unknown until they are really executed.

Type Conversion and Compatibility (concluded)

- Small-size types $\rightarrow$ large-size types.
- Small-size types $\nleftarrow$ large-size types (need a cast).
- Simple types $\rightarrow$ complicated types.
- Simple types $\nleftarrow$ complicated types (need a cast).

Text: Characters & String

- Each character is encoded in a sequence of 0's and 1's.
  - For example, ASCII. (See the next page.)
- C# uses Unicode to represent characters denoted by char, which is a 16-bit unsigned value.[11]
- However, we often use string to present text, as shown before.
- As an analogy, a molecule (string) consists of atoms (characters).[12]

[11] Unicode defines a fully international character set that can represent all of the characters found in all human languages.

[12] A string object comprises characters equipped with plentiful tools.

ASCII (7-bit version)

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Example

```
1 ...
2         char c = 'a';  // A char value should be single-quoted.
3         Console.WriteLine(c + 1); // Output 98!! (Why?)
4         Console.WriteLine((char)(c + 1)); // Output b.
5
6         string s = "C#"; // A string should be double-quoted.
7         Console.WriteLine(s + 328); // Output C#328.
8 ...
```

- Why applying arithmetic operators on characters, for example, Line 4?[13]
- In Line 7, the result of applying the + operator to string is totally different from Line 3 & 4. (Why?)

[13] See https://en.wikipedia.org/wiki/Cryptography.

Boolean Values[15]

- The program is supposed to do decision making by itself, for example, self-driving cars.[14]
- To do this, C# provides the bool-type flow controls (branching and iteration).
- This type has only two possible values, true and false.
- Note that true and false cannot be converted to 1 and 0, respectively.

[14] See `https://www.google.com/selfdrivingcar/`.

[15] George Boole (1815–1864) is the namesake of the branch of algebra known as Boolean algebra. See `https://en.wikipedia.org/wiki/George_Boole`.

Relational Operators

Operator	Name
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

- Relational operators take two operands and return a bool value.
- Note that the mathematical equality operator is ==.

Example

```
1 ...
2         int x = 2;
3         Console.WriteLine(x > 1);      // Output true.
4         Console.WriteLine(x < 1);      // Output false.
5         Console.WriteLine(x == 1);     // Output false.
6         Console.WriteLine(x != 1);     // Output true.
7         Console.WriteLine(1 < x < 3); // Sorry?
8 ...
```

- In Line 7, $1 < x < 3$ is syntactically wrong.
- You need to split a complex statement into several basic statements and joint them by logical operators.
- For example, $1 < x < 3$ should be $1 < x \,\&\&\, x < 3$, where && presents the AND operator.

Conditional Logical Operators[16]

Operator	Name
!	NOT
&&	AND
||	OR
$\wedge$	EXCLUSIVE-OR

- We use XOR to denote the EXCLUSIVE-OR operator.

[16] I skip over the bit-wise operators because most of you do not use these directly. See `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators` if necessary.

Truth Table

- Let X and Y be two bool variables.
- Then the truth table for logical operators is as follows:

X	Y	!X	X&&Y	X $\|\|$ Y	X $\wedge$ Y
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

- It is worth to know that basic instructions of computers, such as arithmetic operators, are implemented by Boolean logics.[17]
  - For example, the single-bit adder can be implemented by using the XOR operator. (Try!)

[17] The combination of these very basic elements (0, 1, AND, OR, NOT) with jumps produces AlphaGo beat human beings in 2016.

"Logic is the anatomy of thought."

– John Locke (1632–1704)

"This sentence is false."

– anonymous

"I know that I know nothing."

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)

Arithmetic Compound Assignment Operators[18]

Operator	Operator name
++	Increment (by one)
+=	Addition assignment
−=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
−−	Decrement (by one)

[18] Note that these shorthand operators are not available in languages such as Matlab and R.

Example

```
1  ...
2          int x = 1;
3          Console.WriteLine(x); // Output 1.
4
5          x = x + 1;
6          Console.WriteLine(x); // Output 2.
7
8          x += 2;
9          Console.WriteLine(x); // Output 4.
10
11         x++; // Equivalent to x += 1 and x = x + 1.
12         Console.WriteLine(x); // Output 5.
13 ...
```

++x vs. x++

```
1  ...
2          int x = 0;
3          int y = ++x;
4          Console.WriteLine(y); // Output 1.
5          Console.WriteLine(x); // Output 1.
6
7          int w = 0;
8          int z = w++;
9          Console.WriteLine(z); // Output 0.
10         Console.WriteLine(w); // Output 1.
11 ...
```

- ++x first increments the value of x and then returns x.
- Instead, x++ first returns the value of x and then increments itself.

Operator Precedence

Precedence	*Operator*
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Comparison)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.

Example: Reading Input From Console

```
1  ...
2          // INPUT
3          Console.WriteLine("Enter r?");
4          int r = int.Parse(Console.ReadLine());
5
6          // ALGORITHM
7          double A = r * r * 3.14;
8
9          // OUTPUT
10         Console.WriteLine(A);
11 ...
```

- In Line 4, we use **Console**.ReadLine() to receive any typing as a string input.
- We then convert the string to an integer by invoking int.Parse(). (Why?)

Exercise: Body Mass Index (BMI)

Write a program which takes user name, height (in cm), weight (in kgw) as input, and then outputs the user name attached with his/her BMI, which is

$$\text{BMI} = \frac{\text{weight (kgw)}}{\text{height}^2(\text{m}^2)}.$$

- Be careful about unit conversion!

```
1  ...
2          // INPUT
3          Console.WriteLine("Enter your name?");
4          string name = Console.ReadLine();
5
6          Console.WriteLine("Enter your height (cm)?");
7          double height = double.Parse(Console.ReadLine()) / 100;
8
9          Console.WriteLine("Enter your weight (kgw)?");
10         double weight = double.Parse(Console.ReadLine());
11
12         // ALGORITHM
13         double bmi = weight / height / height;
14
15         // OUTPUT: name (bmi)
16         Console.WriteLine(name + " (" + bmi + ")");
17 ...
```

- In Line 16, *name* is followed by " (", and then *bmi* converted to a string, and finally ")".

Special Issue: String Format[19]

- To insert values into a string, use { } to indicate the location of insertion with zero-based indexing.
- For example,

```
...
        Console.WriteLine("{0} ({1})", name, bmi);
...
```

- We can also specify the field width for the inserted value, even with the precision, for example,

```
...
        Console.WriteLine("{0} ({1, 5:F2})", name, bmi);
        //                              5 -> field width
        //                                F2 -> precision
        // So it outputs, say Arthur (23.51).
...
```

[19] See https://docs.microsoft.com/en-us/dotnet/api/system.string.format?view=netcore-3.1.

More Examples

```
1  ...
2          string company = "Microsoft";
3          double version = 3.1415926;
4          int fuji = -43;
5
6          Console.WriteLine(company);
7          // Output Microsoft
8          Console.WriteLine("{0}", company);
9          // Output Microsoft (Same as above.)
10         Console.WriteLine("{0, 12}", company);
11         // Output ___Microsoft
12         Console.WriteLine("{0, 6:F2}", version);
13         // Output __3.14.
14         Console.WriteLine("{0} {1} {2}", company, version, fuji);
15         // Output Microsoft 3.1415926 -43.
16         Console.WriteLine("{0} {0} {0}", company);
17         // Output Microsoft Microsoft Microsoft.
18 ...
```

Exercise: Two Simple Descriptive Statistics

Write a program which takes 3 numbers as user input, and calculates the average with the standard deviation.

- The standard deviation is

$$\sqrt{\frac{\sum_{i=1}^{3}(x_i - \overline{x})^2}{3}},$$

where $\overline{x} = \left(\sum_{i=1}^{3} x_i\right) / 3$.

- You may use the following **Math**[20] methods:
  - **Math**.Pow(double x , double y) for x^y.
  - **Math**.Sqrt(double x) for $\sqrt{x}$.

[20] See `https://docs.microsoft.com/en-us/dotnet/api/system.math?view=netframework-4.8`.