Microsoft Partner
Silver Learning

C# Стартовый

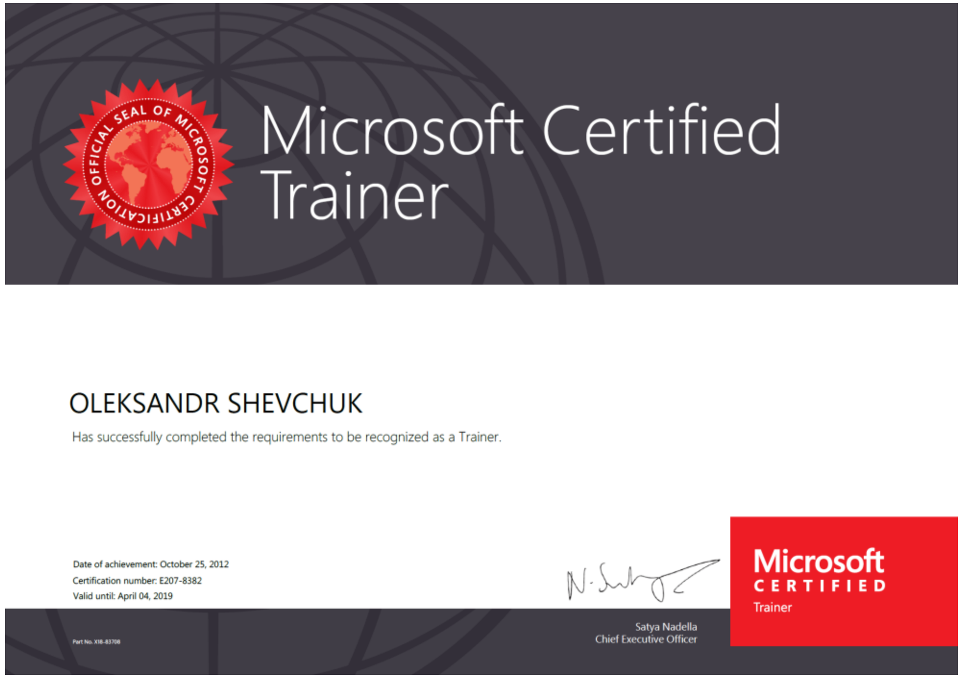
ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

Операции над числовыми переменными

ITVDN
IT VIDEO DEVELOPERS NETWORK

Introduction

Александр Шевчук



Microsoft Certified Trainer

OLEKSANDR SHEVCHUK

Has successfully completed the requirements to be recognized as a Trainer.

Date of achievement: October 25, 2012
Certification number: E207-8382
Valid until: April 04, 2019

Part No. X18-83708

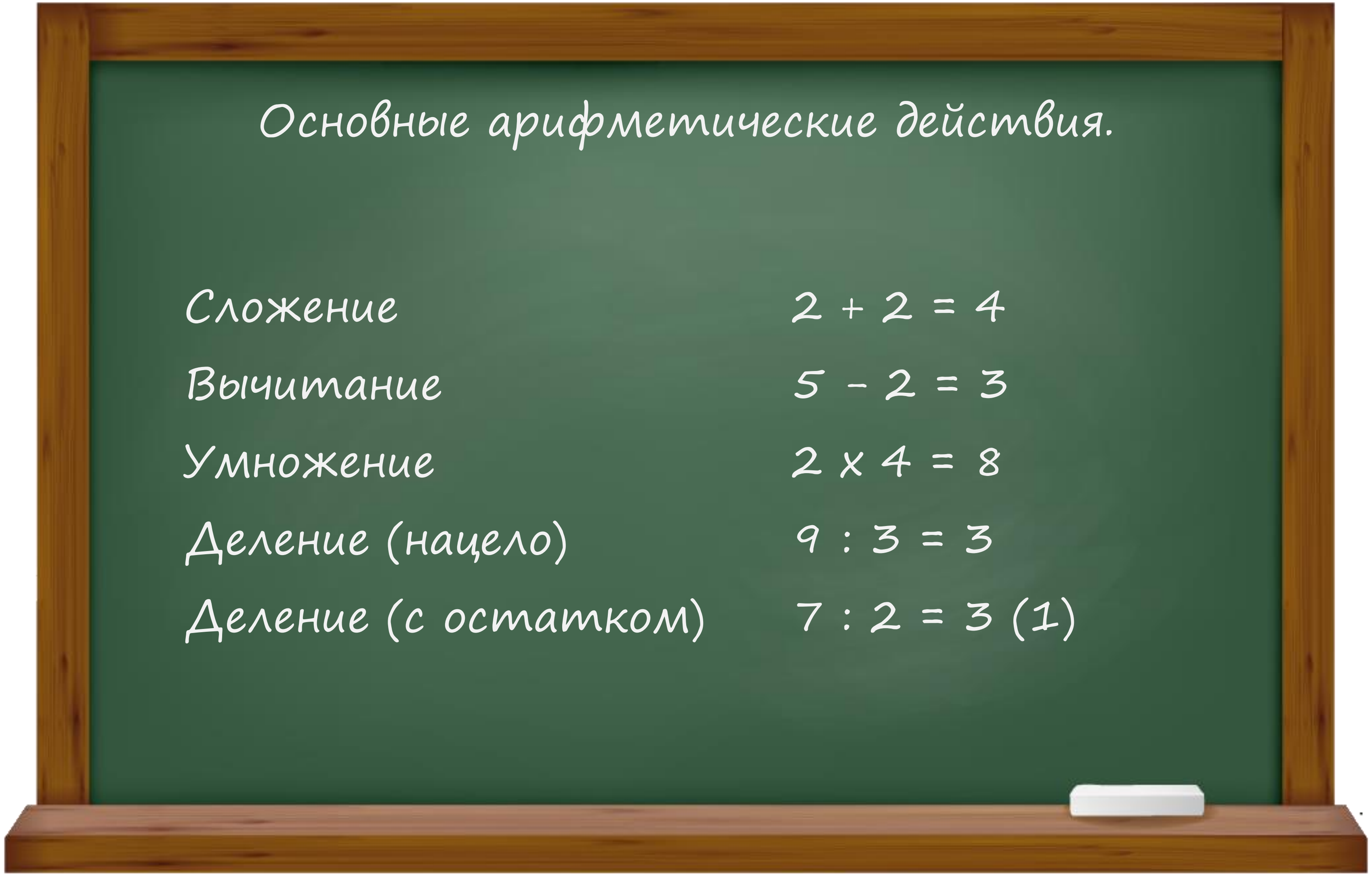
Satya Nadella
Chief Executive Officer

Microsoft CERTIFIED Trainer

MCID: 9230440

Тема урока

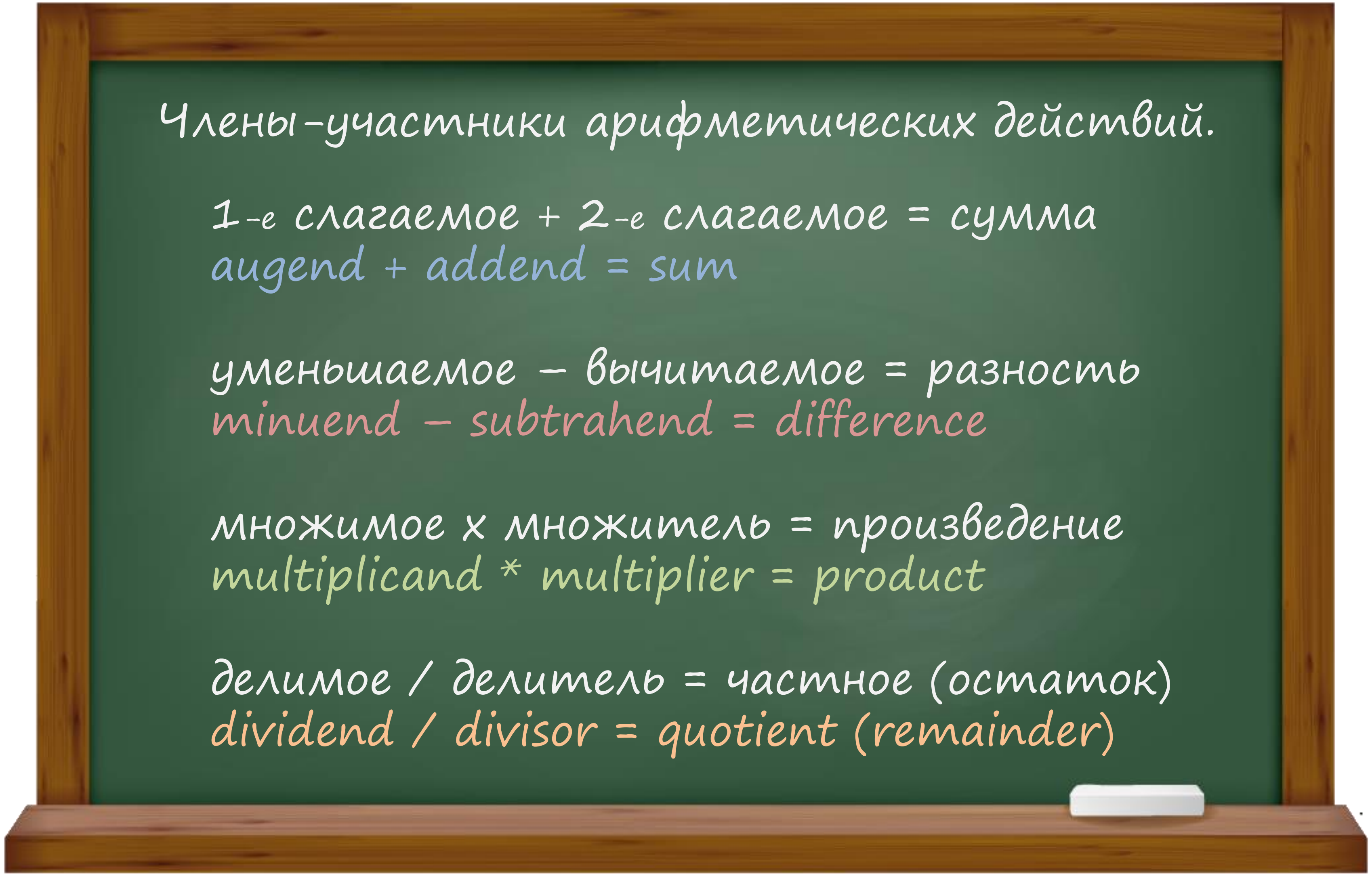
Операции над числовыми переменными



Основные арифметические действия.

Сложение	2 + 2 = 4
Вычитание	5 - 2 = 3
Умножение	2 x 4 = 8
Деление (нацело)	9 : 3 = 3
Деление (с остатком)	7 : 2 = 3 (1)

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#



Члены-участники арифметических действий.

1-е слагаемое + 2-е слагаемое = сумма
augend + addend = sum

уменьшаемое – вычитаемое = разность
minuend – subtrahend = difference

множимое х множитель = произведение
multiplicand * multiplier = product

делимое / делитель = частное (остаток)
dividend / divisor = quotient (remainder)

Математическое выражение

Математическое выражение – это одна или несколько величин (переменных или констант), соединенных между собой знаками арифметических действий (+, -, *, /, …) и знаками последовательности действий – () – круглыми скобками.

Для того чтобы преобразовать результат выражения к другому типу, следует всё выражение взять в круглые скобки и перед выражением поставить оператор преобразования типа.

Операторы арифметических действий над двумя операндами

```
// Сложение
int int.operator + (int left, int right)
uint uint.operator + (uint left, uint right)
long long.operator + (long left, long right)
ulong ulong.operator + (ulong left, ulong right)
float float.operator + (float left, float right)
double double.operator + (double left, double right)
decimal decimal.operator + (decimal left, decimal right)
// Вычитание
int int.operator - (int left, int right)
uint uint.operator - (uint left, uint right)
long long.operator - (long left, long right)
ulong ulong.operator - (ulong left, ulong right)
float float.operator - (float left, float right)
double double.operator - (double left, double right)
decimal decimal.operator - (decimal left, decimal right)
// Умножение
int int.operator * (int left, int right)
uint uint.operator * (uint left, uint right)
long long.operator * (long left, long right)
ulong ulong.operator * (ulong left, ulong right)
float float.operator * (float left, float right)
double double.operator * (double left, double right)
decimal decimal.operator * (decimal left, decimal right)
```

```
// Деление нацело
int int.operator / (int left, int right)
uint uint.operator / (uint left, uint right)
long long.operator / (long left, long right)
ulong ulong.operator / (ulong left, ulong right)
float float.operator / (float left, float right)
double double.operator / (double left, double right)
decimal decimal.operator / (decimal left, decimal right)
// Деление с получением остатка
int int.operator % (int left, int right)
uint uint.operator % (uint left, uint right)
long long.operator % (long left, long right)
ulong ulong.operator % (ulong left, ulong right)
float float.operator % (float left, float right)
double double.operator % (double left, double right)
decimal decimal.operator % (decimal left, decimal right)
```

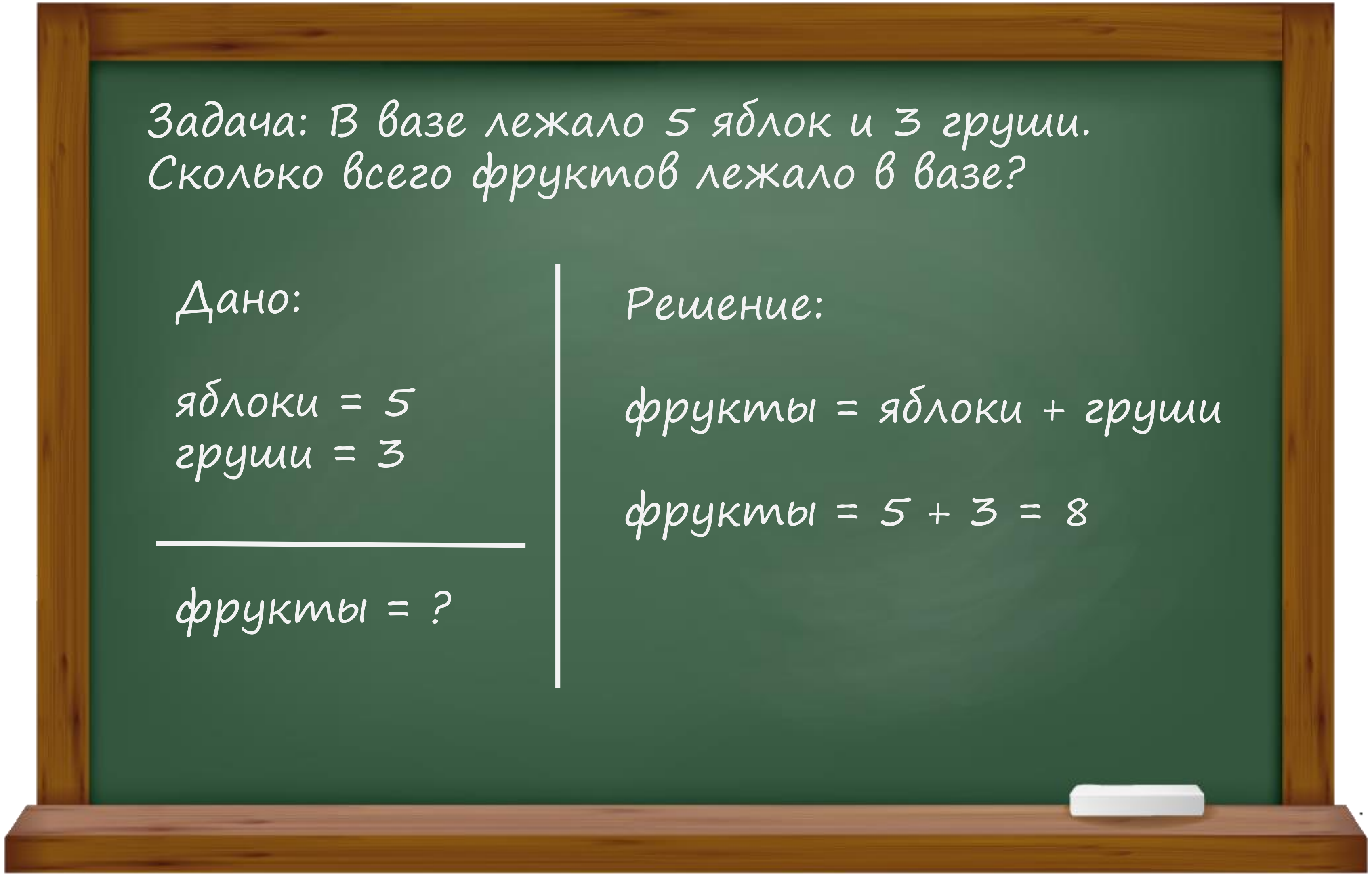
Операторы арифметических действий над двумя операндами

```
// Сложение
int int.operator + (int left, int right)
uint uint.operator + (uint left, uint right)
long long.operator + (long left, long right)
ulong ulong.operator + (ulong left, ulong right)
float float.operator + (float left, float right)
double double.operator + (double left, double right)
decimal decimal.operator + (decimal left, decimal right)
// Вычитание
int int.operator - (int left, int right)
uint uint.operator - (uint left, uint right)
long long.operator - (long left, long right)
ulong ulong.operator - (ulong left, ulong right)
float float.operator - (float left, float right)
double double.operator - (double left, double right)
decimal decimal.operator - (decimal left, decimal right)
// Умножение
int int.operator * (int left, int right)
uint uint.operator * (uint left, uint right)
long long.operator * (long left, long right)
ulong ulong.operator * (ulong left, ulong right)
float float.operator * (float left, float right)
double double.operator * (double left, double right)
decimal decimal.operator * (decimal left, decimal right)
```

```
// Деление нацело
int int.operator / (int left, int right)
uint uint.operator / (uint left, uint right)
long long.operator / (long left, long right)
ulong ulong.operator / (ulong left, ulong right)
float float.operator / (float left, float right)
double double.operator / (double left, double right)
decimal decimal.operator / (decimal left, decimal right)
// Деление с получением остатка
int int.operator % (int left, int right)
uint uint.operator % (uint left, uint right)
long long.operator % (long left, long right)
ulong ulong.operator % (ulong left, ulong right)
float float.operator % (float left, float right)
double double.operator % (double left, double right)
decimal decimal.operator % (decimal left, decimal right)
```

Типы: `byte`, `sbyte`, `short`, `ushort` – не имеют своих арифметических операторов.

По умолчанию используется `int.operator`



Задача: В вазе лежало 5 яблок и 3 груши.
Сколько всего фруктов лежало в вазе?

Дано:

яблоки = 5
груши = 3

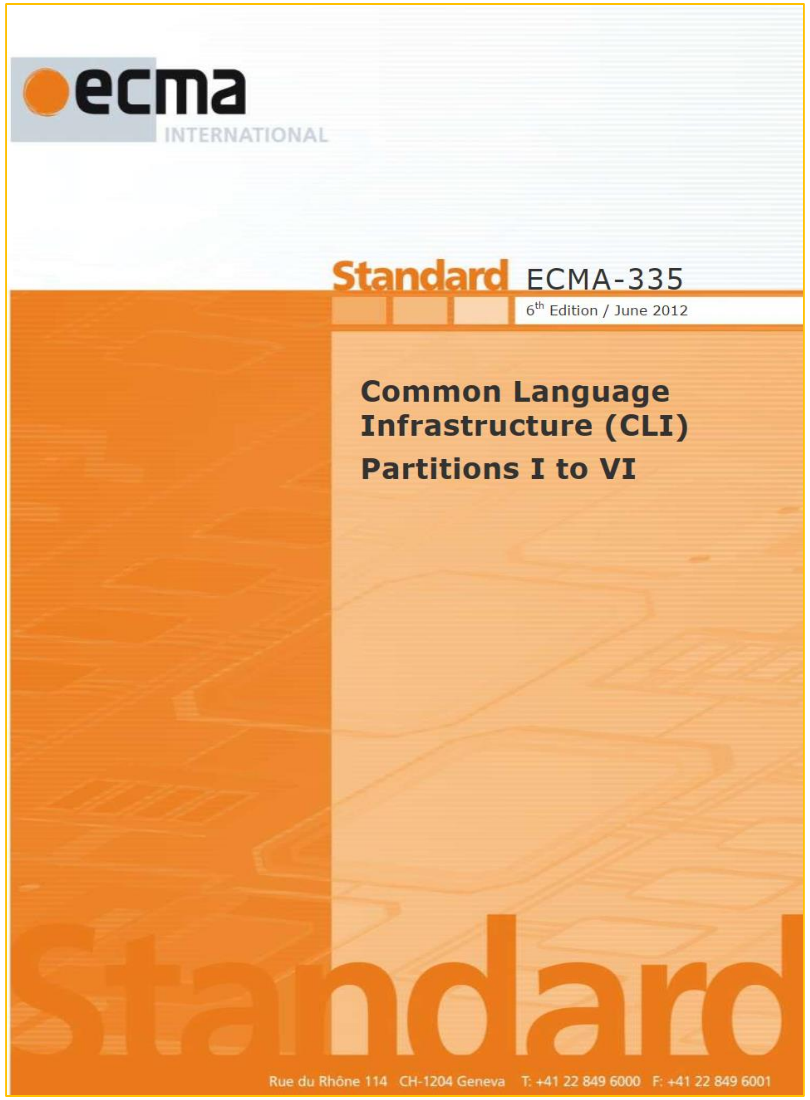
фрукты = ?

Решение:

фрукты = яблоки + груши

фрукты = 5 + 3 = 8

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#



ecma INTERNATIONAL

Standard ECMA-335

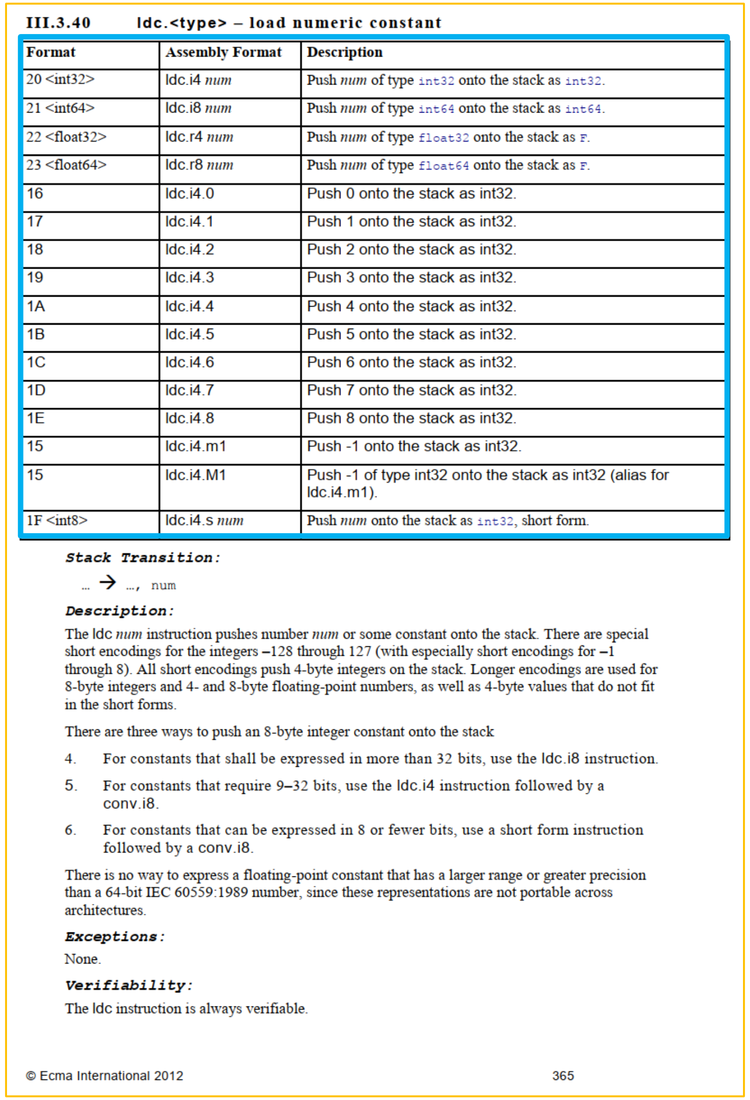
6th Edition / June 2012

Common Language Infrastructure (CLI)

Partitions I to VI

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001



III.3.40 ldc.<type> – load numeric constant

Format	Assembly Format	Description
20 <int32>	ldc.i4 *num*	Push *num* of type int32 onto the stack as int32.
21 <int64>	ldc.i8 *num*	Push *num* of type int64 onto the stack as int64.
22 <float32>	ldc.r4 *num*	Push *num* of type float32 onto the stack as F.
23 <float64>	ldc.r8 *num*	Push *num* of type float64 onto the stack as F.
16	ldc.i4.0	Push 0 onto the stack as int32.
17	ldc.i4.1	Push 1 onto the stack as int32.
18	ldc.i4.2	Push 2 onto the stack as int32.
19	ldc.i4.3	Push 3 onto the stack as int32.
1A	ldc.i4.4	Push 4 onto the stack as int32.
1B	ldc.i4.5	Push 5 onto the stack as int32.
1C	ldc.i4.6	Push 6 onto the stack as int32.
1D	ldc.i4.7	Push 7 onto the stack as int32.
1E	ldc.i4.8	Push 8 onto the stack as int32.
15	ldc.i4.m1	Push -1 onto the stack as int32.
15	ldc.i4.M1	Push -1 of type int32 onto the stack as int32 (alias for ldc.i4.m1).
1F <int8>	ldc.i4.s *num*	Push *num* onto the stack as int32, short form.

Stack Transition:

… → …, num

Description:

The ldc *num* instruction pushes number *num* or some constant onto the stack. There are special short encodings for the integers –128 through 127 (with especially short encodings for –1 through 8). All short encodings push 4-byte integers on the stack. Longer encodings are used for 8-byte integers and 4- and 8-byte floating-point numbers, as well as 4-byte values that do not fit in the short forms.

There are three ways to push an 8-byte integer constant onto the stack

4. For constants that shall be expressed in more than 32 bits, use the ldc.i8 instruction.

5. For constants that require 9–32 bits, use the ldc.i4 instruction followed by a conv.i8.

6. For constants that can be expressed in 8 or fewer bits, use a short form instruction followed by a conv.i8.

There is no way to express a floating-point constant that has a larger range or greater precision than a 64-bit IEC 60559:1989 number, since these representations are not portable across architectures.

Exceptions:

None.

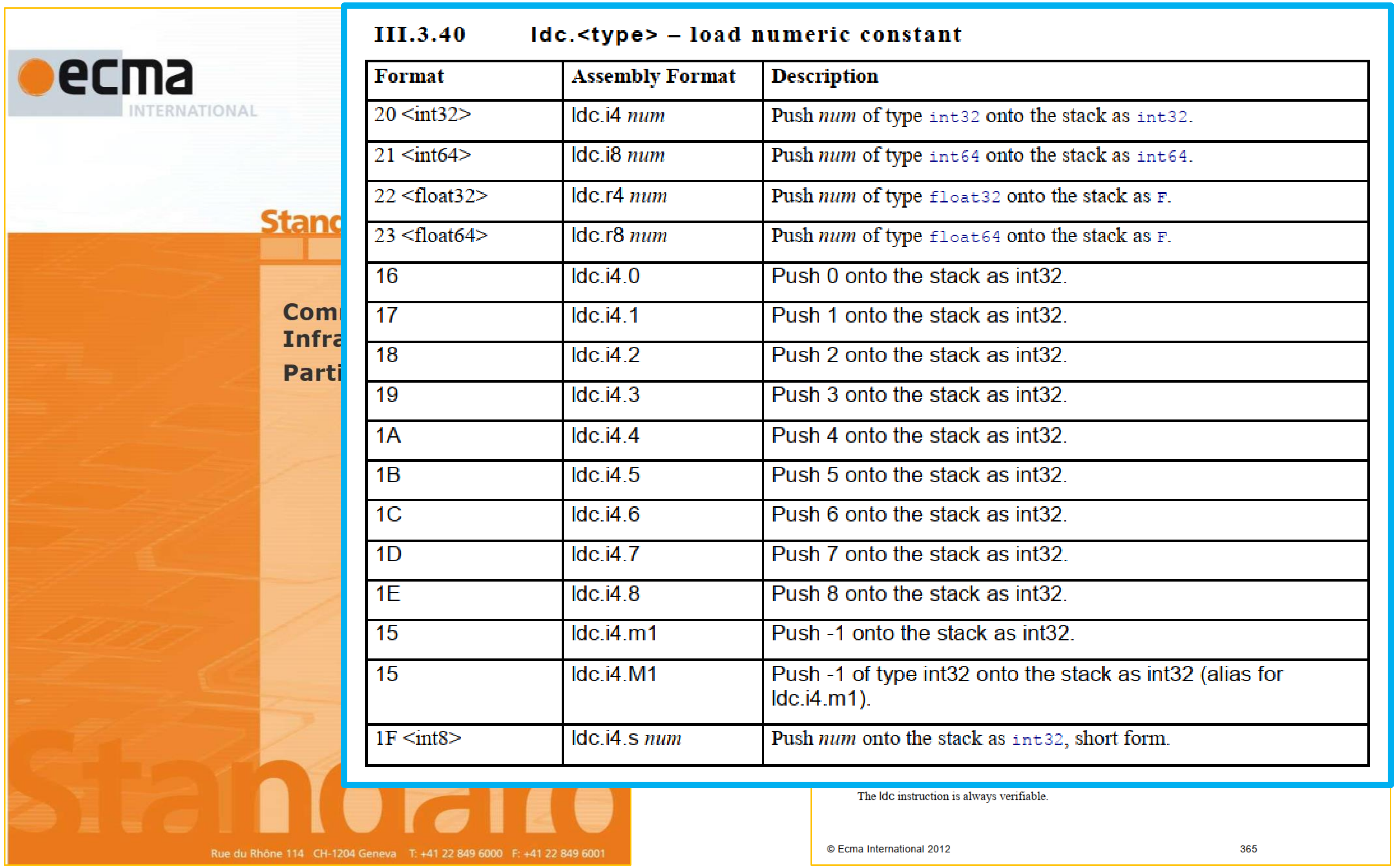
Verifiability:

The ldc instruction is always verifiable.

© Ecma International 2012 365

Стр. 365

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#


ecma
INTERNATIONAL
Stanc
Comi
Infra
Parti
Standard
Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

III.3.40 ldc.<type> – load numeric constant

Format	Assembly Format	Description
20 <int32>	ldc.i4 *num*	Push *num* of type int32 onto the stack as int32.
21 <int64>	ldc.i8 *num*	Push *num* of type int64 onto the stack as int64.
22 <float32>	ldc.r4 *num*	Push *num* of type float32 onto the stack as F.
23 <float64>	ldc.r8 *num*	Push *num* of type float64 onto the stack as F.
16	ldc.i4.0	Push 0 onto the stack as int32.
17	ldc.i4.1	Push 1 onto the stack as int32.
18	ldc.i4.2	Push 2 onto the stack as int32.
19	ldc.i4.3	Push 3 onto the stack as int32.
1A	ldc.i4.4	Push 4 onto the stack as int32.
1B	ldc.i4.5	Push 5 onto the stack as int32.
1C	ldc.i4.6	Push 6 onto the stack as int32.
1D	ldc.i4.7	Push 7 onto the stack as int32.
1E	ldc.i4.8	Push 8 onto the stack as int32.
15	ldc.i4.m1	Push -1 onto the stack as int32.
15	ldc.i4.M1	Push -1 of type int32 onto the stack as int32 (alias for ldc.i4.m1).
1F <int8>	ldc.i4.s *num*	Push *num* onto the stack as int32, short form.

The ldc instruction is always verifiable.

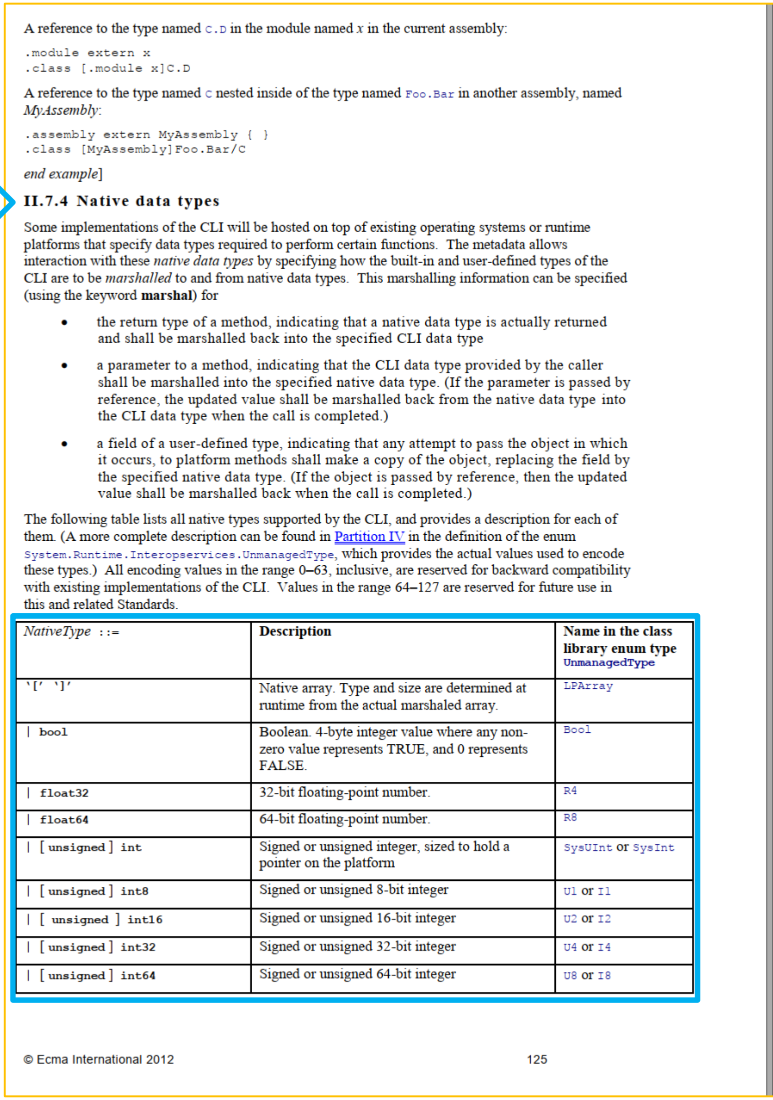
© Ecma International 2012 365

Стр. 365

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

II.7.4 Native data types

Стр. 125



A reference to the type named C.D in the module named *x* in the current assembly:

```
.module extern x
.class [.module x]C.D
```

A reference to the type named C nested inside of the type named Foo.Bar in another assembly, named *MyAssembly*:

```
.assembly extern MyAssembly { }
.class [MyAssembly]Foo.Bar/C
```

end example]

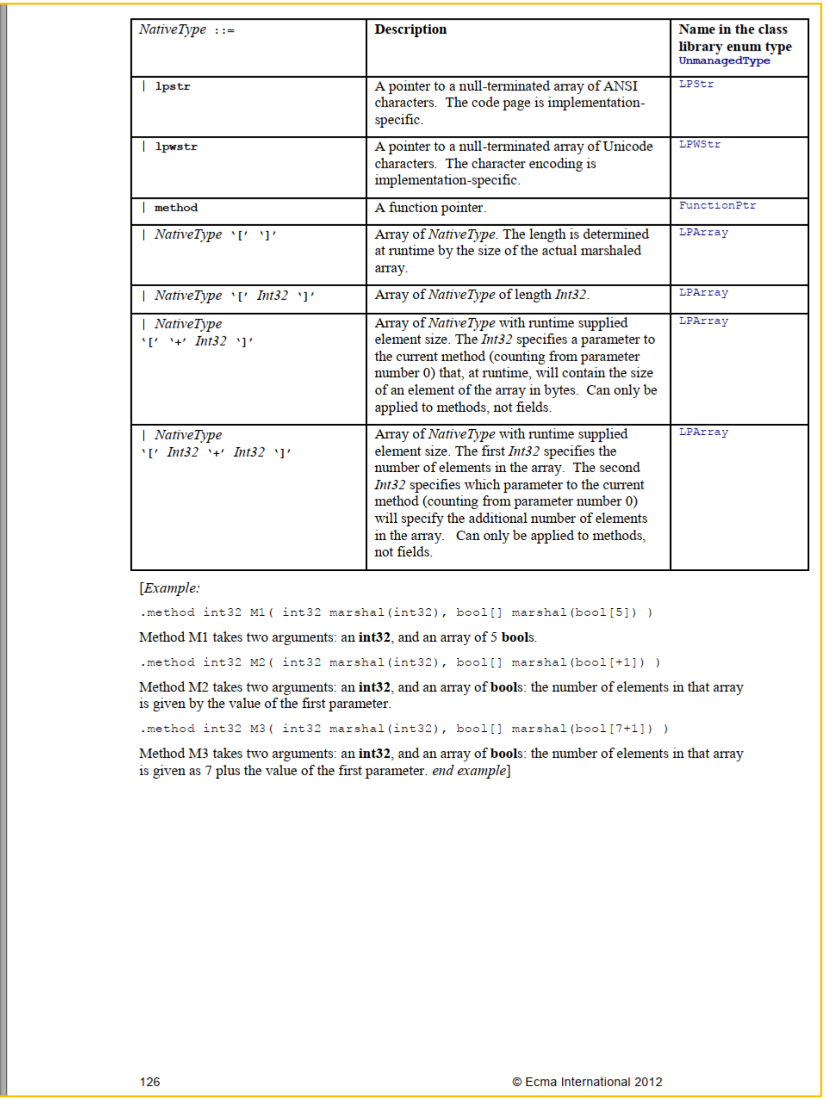
II.7.4 Native data types

Some implementations of the CLI will be hosted on top of existing operating systems or runtime platforms that specify data types required to perform certain functions. The metadata allows interaction with these *native data types* by specifying how the built-in and user-defined types of the CLI are to be *marshalled* to and from native data types. This marshalling information can be specified (using the keyword **marshal**) for

- the return type of a method, indicating that a native data type is actually returned and shall be marshalled back into the specified CLI data type
- a parameter to a method, indicating that the CLI data type provided by the caller shall be marshalled into the specified native data type. (If the parameter is passed by reference, the updated value shall be marshalled back from the native data type into the CLI data type when the call is completed.)
- a field of a user-defined type, indicating that any attempt to pass the object in which it occurs, to platform methods shall make a copy of the object, replacing the field by the specified native data type. (If the object is passed by reference, then the updated value shall be marshalled back when the call is completed.)

The following table lists all native types supported by the CLI, and provides a description for each of them. (A more complete description can be found in Partition IV in the definition of the enum System.Runtime.Interopservices.UnmanagedType, which provides the actual values used to encode these types.) All encoding values in the range 0–63, inclusive, are reserved for backward compatibility with existing implementations of the CLI. Values in the range 64–127 are reserved for future use in this and related Standards.

NativeType ::=	**Description**	**Name in the class library enum type** **UnmanagedType**
'[' ']'	Native array. Type and size are determined at runtime from the actual marshaled array.	LPArray
| **bool**	Boolean. 4-byte integer value where any non-zero value represents TRUE, and 0 represents FALSE.	Bool
| **float32**	32-bit floating-point number.	R4
| **float64**	64-bit floating-point number.	R8
| [**unsigned**] **int**	Signed or unsigned integer, sized to hold a pointer on the platform	SysUInt or SysInt
| [**unsigned**] **int8**	Signed or unsigned 8-bit integer	U1 or I1
| [**unsigned**] **int16**	Signed or unsigned 16-bit integer	U2 or I2
| [**unsigned**] **int32**	Signed or unsigned 32-bit integer	U4 or I4
| [**unsigned**] **int64**	Signed or unsigned 64-bit integer	U8 or I8



NativeType ::=	**Description**	**Name in the class library enum type** **UnmanagedType**
| **lpstr**	A pointer to a null-terminated array of ANSI characters. The code page is implementation-specific.	LPStr
| **lpwstr**	A pointer to a null-terminated array of Unicode characters. The character encoding is implementation-specific.	LPWStr
| **method**	A function pointer.	FunctionPtr
| *NativeType* '[' ']'	Array of *NativeType*. The length is determined at runtime by the size of the actual marshaled array.	LPArray
| *NativeType* '[' *Int32* ']'	Array of *NativeType* of length *Int32*.	LPArray
| *NativeType* '[' '+' *Int32* ']'	Array of *NativeType* with runtime supplied element size. The *Int32* specifies a parameter to the current method (counting from parameter number 0) that, at runtime, will contain the size of an element of the array in bytes. Can only be applied to methods, not fields.	LPArray
| *NativeType* '[' *Int32* '+' *Int32* ']'	Array of *NativeType* with runtime supplied element size. The first *Int32* specifies the number of elements in the array. The second *Int32* specifies which parameter to the current method (counting from parameter number 0) will specify the additional number of elements in the array. Can only be applied to methods, not fields.	LPArray

[*Example:*

```
.method int32 M1( int32 marshal(int32), bool[] marshal(bool[5]) )
```

Method M1 takes two arguments: an **int32**, and an array of 5 **bool**s.

```
.method int32 M2( int32 marshal(int32), bool[] marshal(bool[+1]) )
```

Method M2 takes two arguments: an **int32**, and an array of **bool**s: the number of elements in that array is given by the value of the first parameter.

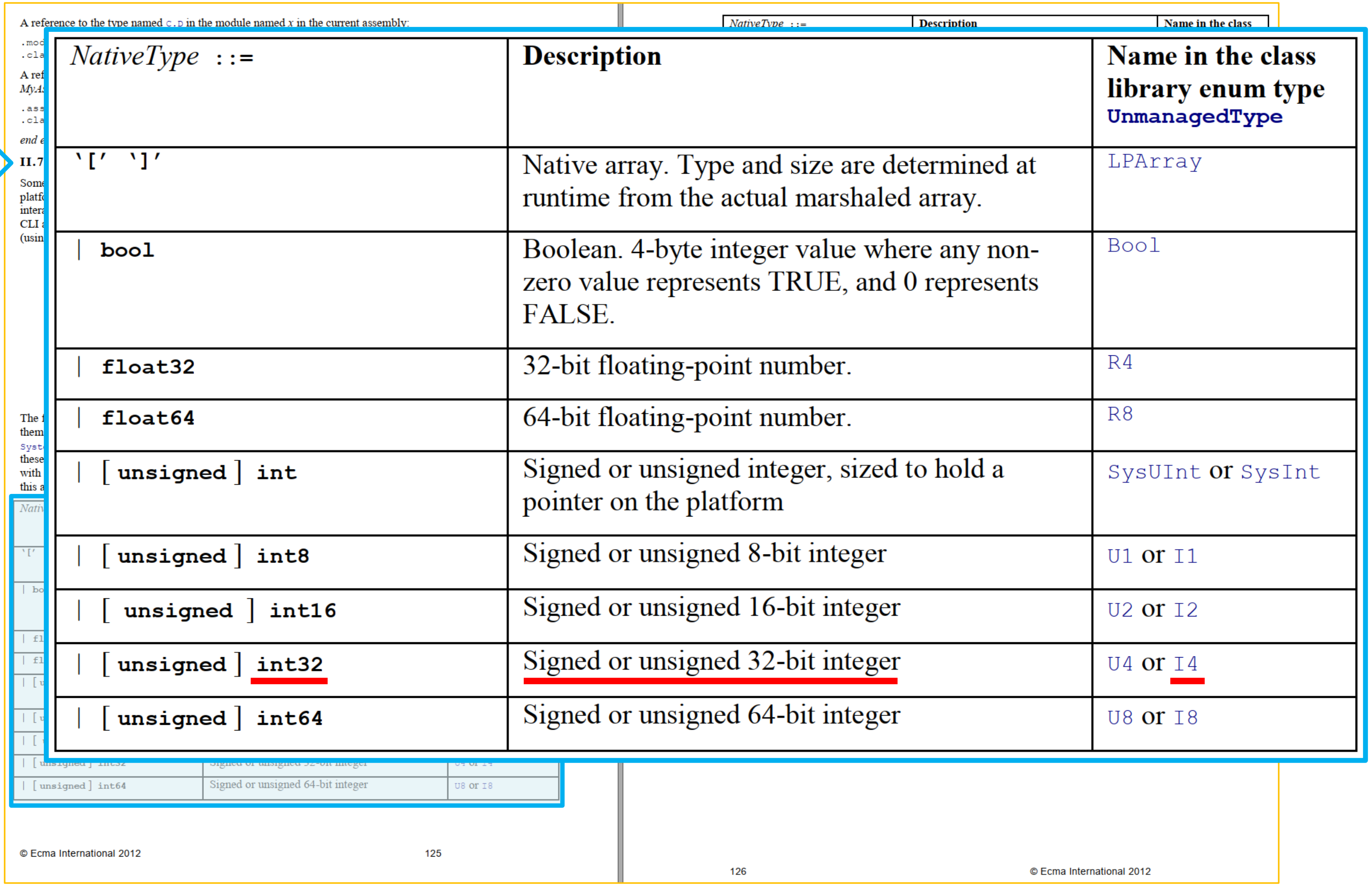
```
.method int32 M3( int32 marshal(int32), bool[] marshal(bool[7+1]) )
```

Method M3 takes two arguments: an **int32**, and an array of **bool**s: the number of elements in that array is given as 7 plus the value of the first parameter. *end example*]

Стр. 126

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

II.7.4 Native data types



A reference to the type named c.p in the module named x in the current assembly:

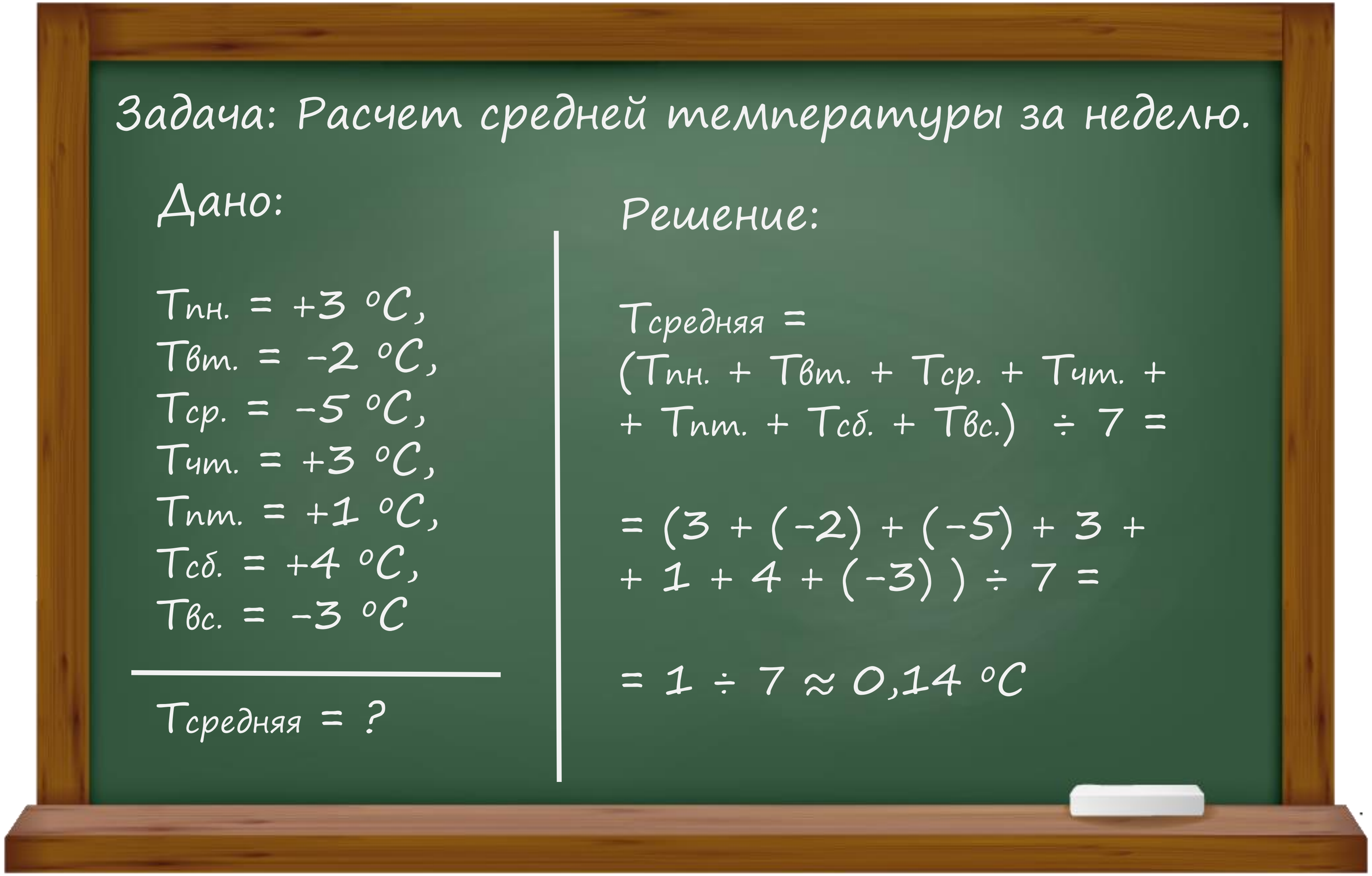
NativeType ::=	**Description**	**Name in the class library enum type `UnmanagedType`**
`'['  ']'`	Native array. Type and size are determined at runtime from the actual marshaled array.	`LPArray`
| **`bool`**	Boolean. 4-byte integer value where any non-zero value represents TRUE, and 0 represents FALSE.	`Bool`
| **`float32`**	32-bit floating-point number.	`R4`
| **`float64`**	64-bit floating-point number.	`R8`
| [**`unsigned`**] **`int`**	Signed or unsigned integer, sized to hold a pointer on the platform	`SysUInt` or `SysInt`
| [**`unsigned`**] **`int8`**	Signed or unsigned 8-bit integer	`U1` or `I1`
| [**`unsigned`**] **`int16`**	Signed or unsigned 16-bit integer	`U2` or `I2`
| [**`unsigned`**] **`int32`**	Signed or unsigned 32-bit integer	`U4` or `I4`
| [**`unsigned`**] **`int64`**	Signed or unsigned 64-bit integer	`U8` or `I8`

© Ecma International 2012 125

126 © Ecma International 2012

Стр. 125

Стр. 126



Задача: Расчет средней температуры за неделю.

Дано:

$T_{\text{пн.}} = +3\ {}^{o}C,$
$T_{\text{вт.}} = -2\ {}^{o}C,$
$T_{\text{ср.}} = -5\ {}^{o}C,$
$T_{\text{чт.}} = +3\ {}^{o}C,$
$T_{\text{пт.}} = +1\ {}^{o}C,$
$T_{\text{сб.}} = +4\ {}^{o}C,$
$T_{\text{вс.}} = -3\ {}^{o}C$

$T_{\text{средняя}} = ?$

Решение:

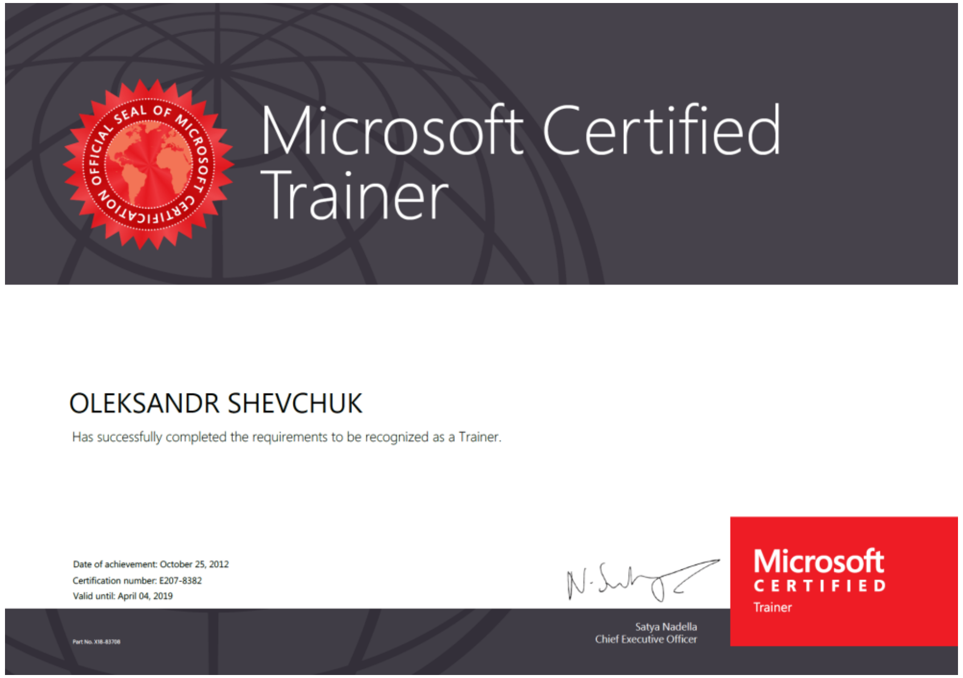
$T_{\text{средняя}} =$
$(T_{\text{пн.}} + T_{\text{вт.}} + T_{\text{ср.}} + T_{\text{чт.}} +$
$+ T_{\text{пт.}} + T_{\text{сб.}} + T_{\text{вс.}}) \div 7 =$

$= (3 + (-2) + (-5) + 3 +$
$+ 1 + 4 + (-3)) \div 7 =$

$= 1 \div 7 \approx 0{,}14\ {}^{o}C$

Спасибо за внимание! До новых встреч!

Александр Шевчук


OFFICIAL SEAL OF MICROSOFT CERTIFICATION

Microsoft Certified Trainer

OLEKSANDR SHEVCHUK

Has successfully completed the requirements to be recognized as a Trainer.

Date of achievement: October 25, 2012
Certification number: E207-8382
Valid until: April 04, 2019

Part No. X18-83708

Satya Nadella
Chief Executive Officer

Microsoft CERTIFIED Trainer

MCID: 9230440

ITVDN

IT VIDEO DEVELOPERS NETWORK