# C# Стартовый

## ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

Комментарии

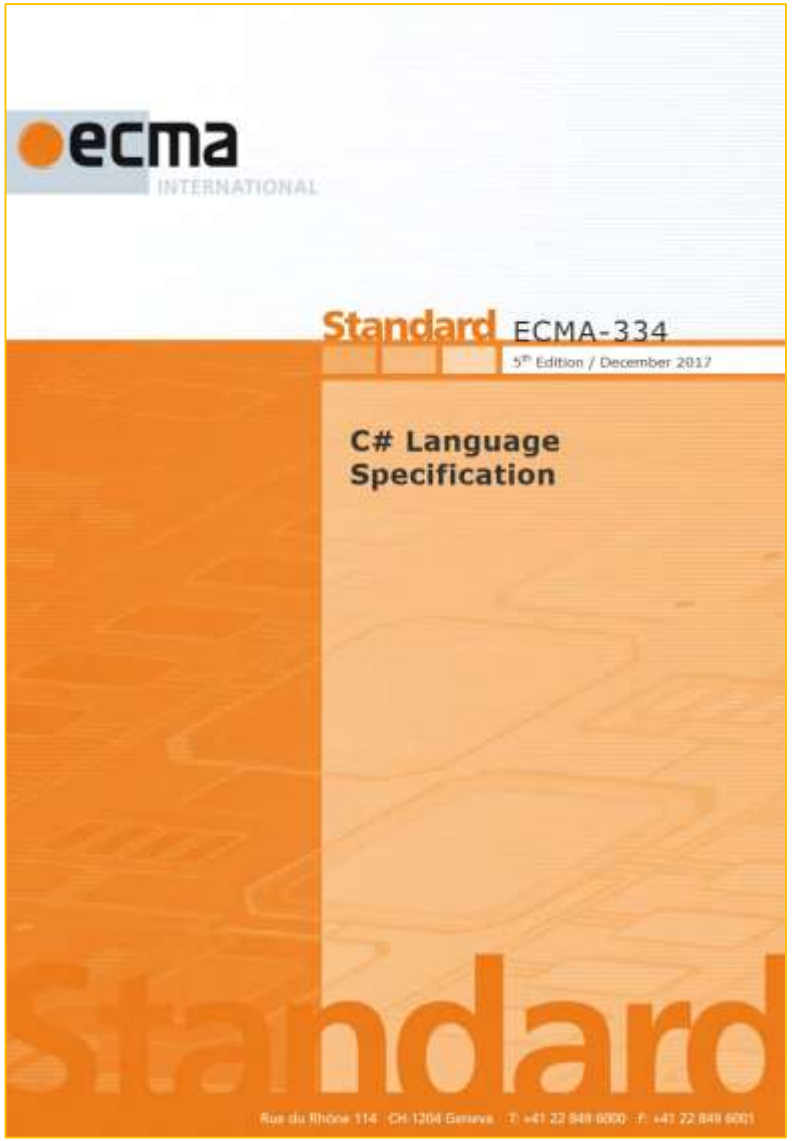
# Introduction


Александр Шевчук


MCID: 9230440

Тема урока

# Комментарии






```
/* Hello, world program
     This program writes "hello, world" to the console
*/
class Hello
{
   static void Main() {
      System.Console.WriteLine("hello, world");
   }
}
```

includes a delimited comment. *end example*]

A ***single-line comment*** begins with the characters // and extends to the end of the line. [*Example*: The example

```
// Hello, world program
//    This program writes "hello, world" to the console
//
class Hello // any name will do for this class
{
   static void Main() { // this method must be named "Main"
      System.Console.WriteLine("hello, world");
   }
}
```

shows several single-line comments. *end example*]

*comment::*
*single-line-comment*
*delimited-comment*

*single-line-comment::*
// *input-characters*$_{opt}$

*input-characters::*
*input-character*
*input-characters input-character*

*input-character::*
*Any Unicode character except a new-line character*

*new-line-character::*
*Carriage return character (U+000D)*
*Line feed character (U+000A)*
*Next line character (U+0085)*
*Line separator character (U+2028)*
*Paragraph separator character (U+2029)*

*delimited-comment::*
/* *delimited-comment-text*$_{opt}$ *asterisks* /

*delimited-comment-text::*
*delimited-comment-section*
*delimited-comment-text delimited-comment-section*

*delimited-comment-section::*
/
*asterisks*$_{opt}$ *not-slash-or-asterisk*

*asterisks::*
*
*asterisks* *

*not-slash-or-asterisk::*
*Any Unicode character except / or **

```
/* Hello, world program
      This program writes "hello, world" to the console
*/
class Hello
{
   static void Main() {
      System.Console.WriteLine("hello, world");
   }
}
```

includes a delimited comment. *end example*]

A ***single-line comment*** begins with the characters // and extends to the end of the line. [*Example*: The example

```
// Hello, world program
//    This program writes "hello, world" to the console
//
class Hello // any name will do for this class
{
   static void Main() { // this method must be named "Main"
      System.Console.WriteLine("hello, world");
   }
}
```





The

*delimited-comment-text::*
*delimited-comment-section*
*delimited-comment-text delimited-comment-section*

*delimited-comment-section::*
*/*
*asterisks*$_{opt}$ *not-slash-or-asterisk*

*asterisks::*
*
*asterisks* *

*not-slash-or-asterisk::*
*Any Unicode character except / or **

*input-section*::
    *input-section-part*
    *input-section input-section-part*

*input-section-part*::
    *input-elements*$_{opt}$ *new-line*
    *pp-directive*

*input-elements*::
    *input-element*
    *input-elements input-element*

*input-element*::
    *whitespace*
    *comment*
    *token*

Five basic elements make up the lexical structure of a C# source file: Line terminators (§7.3.2), white space (§7.3.4), comments (§7.3.3), tokens (§7.4), and pre-processing directives (§7.5). Of these basic elements, only tokens are significant in the syntactic grammar of a C# program (§7.2.4), except in the case of a > token being combined with another token to form a single operator (§7.4.6).

The lexical processing of a C# source file consists of reducing the file into a sequence of tokens that becomes the input to the syntactic analysis. Line terminators, white space, and comments can serve to separate tokens, and pre-processing directives can cause sections of the source file to be skipped, but otherwise these lexical elements have no impact on the syntactic structure of a C# program.

When several lexical grammar productions match a sequence of characters in a source file, the lexical processing always forms the longest possible lexical element. [*Example*: The character sequence // is processed as the beginning of a single-line comment because that lexical element is longer than a single / token. *end example*]

### 7.3.2 Line terminators

Line terminators divide the characters of a C# source file into lines.

*new-line*::
    *Carriage return character (U+000D)*
    *Line feed character (U+000A)*
    *Carriage return character (U+000D) followed by line feed character (U+000A)*
    *Next line character (U+0085)*
    *Line separator character (U+2028)*
    *Paragraph separator character (U+2029)*

For compatibility with source code editing tools that add end-of-file markers, and to enable a source file to be viewed as a sequence of properly terminated lines, the following transformations are applied, in order, to every source file in a C# program:

- If the last character of the source file is a Control-Z character (U+001A), this character is deleted.
- A carriage-return character (U+000D) is added to the end of the source file if that source file is non-empty and if the last character of the source file is not a carriage return (U+000D), a line feed (U+000A), a next line character (U+0085), a line separator (U+2028), or a paragraph separator (U+2029). [*Note*: The additional carriage-return allows a program to end in a *pp-directive* (§7.5) that does not have a terminating *new-line*. *end note*]

### 7.3.3 Comments

Комментарии

Two forms of comments are supported: delimited comments and single-line comments.

A ***delimited comment*** begins with the characters /* and ends with the characters */. Delimited comments can occupy a portion of a line, a single line, or multiple lines. [*Example*: The example



Стр. 16



```
/* Hello, world program
    This program writes "hello, world" to the console
*/
class Hello
{
   static void Main() {
      System.Console.WriteLine("hello, world");
   }
}
```

includes a delimited comment. *end example*]

A ***single-line comment*** begins with the characters // and extends to the end of the line. [*Example*: The example

```
// Hello, world program
//    This program writes "hello, world" to the console
//
class Hello // any name will do for this class
{
   static void Main() { // this method must be named "Main"
      System.Console.WriteLine("hello, world");
   }
}
```

shows several single-line comments. *end example*]

*comment*::
    *single-line-comment*
    *delimited-comment*

*single-line-comment*::
    *// input-characters*$_{opt}$

*input-characters*::
    *input-character*
    *input-characters input-character*

*input-character*::
    *Any Unicode character except a new-line character*

*new-line-character*::
    *Carriage return character (U+000D)*
    *Line feed character (U+000A)*
    *Next line character (U+0085)*
    *Line separator character (U+2028)*
    *Paragraph separator character (U+2029)*

*delimited-comment*::
    */* delimited-comment-text*$_{opt}$ *asterisks /*

*delimited-comment-text*::
    *delimited-comment-section*
    *delimited-comment-text delimited-comment-section*

*delimited-comment-section*::
    */*
    *asterisks*$_{opt}$ *not-slash-or-asterisk*

*asterisks*::
    **
    *asterisks* **

*not-slash-or-asterisk*::
    *Any Unicode character except / or **



Стр. 17

Стр. 16

Многострочные *(разделённые)*

### 7.3.3 Comments

Two forms of comments are supported: delimited comments and single-line comments.

A ***delimited comment*** begins with the characters /* and ends with the characters */. Delimited comments can occupy a portion of a line, a single line, or multiple lines. [*Example*: The example

**16**

**7 Lexical structure**

```
/* Hello, world program
      This program writes "hello, world" to the console
*/
class Hello
{
   static void Main() {
      System.Console.WriteLine("hello, world");
   }
}
```

includes a delimited comment. *end example*]

Однострочные

A ***single-line comment*** begins with the characters // and extends to the end of the line. [*Example*: The example

```
// Hello, world program
//    This program writes "hello, world" to the console
//
class Hello // any name will do for this class
{
   static void Main() { // this method must be named "Main"
      System.Console.WriteLine("hello, world");
   }
}
```

Стр. 17

*input-section::*
*input-section-part*
*input-section input-section-part*

*input-section-part::*
*input-elements*$_{opt}$ *new-line*
*pp-directive*

*input-elements::*
*input-element*
*input-elements input-element*

*input-element::*
*whitespace*
*comment*
*token*

Five basic elements make up the lexical structure of a C# source file: Line terminators (§7.3.2), white space (§7.3.4), comments (§7.3.3), tokens (§7.4), and pre-processing directives (§7.5). Of these basic elements, only tokens are significant in the syntactic grammar of a C# program (§7.2.4), except in the case of a > token being combined with another token to form a single operator (§7.4.6).

The lexical processing of a C# source file consists of reducing the file into a sequence of tokens that becomes the input to the syntactic analysis. Line terminators, white space, and comments can serve to separate tokens, and pre-processing directives can cause sections of the source file to be skipped, but otherwise these lexical elements have no impact on the syntactic structure of a C# program.

When several lexical grammar productions match a sequence of characters in a source file, the lexical processing always forms the longest possible lexical element. [*Example*: The character sequence // is processed as the beginning of a single-line comment because that lexical element is longer than a single / token. *end example*]

### 7.3.2 Line terminators

Line terminators divide the characters of a C# source file into lines.

*new-line::*
*Carriage return character (U+000D)*
*Line feed character (U+000A)*
*Carriage return character (U+000D) followed by line feed character (U+000A)*
*Next line character (U+0085)*
*Line separator character (U+2028)*
*Paragraph separator character (U+2029)*

For compatibility with source code editing tools that add end-of-file markers, and to enable a source file to be viewed as a sequence of properly terminated lines, the following transformations are applied, in order, to every source file in a C# program:

- If the last character of the source file is a Control-Z character (U+001A), this character is deleted.
- A carriage-return character (U+000D) is added to the end of the source file if that source file is non-empty and if the last character of the source file is not a carriage return (U+000D), a line feed (U+000A), a next line character (U+0085), a line separator (U+2028), or a paragraph separator (U+2029). [*Note*: The additional carriage-return allows a program to end in a *pp-directive* (§7.5) that does not have a terminating *new-line*. *end note*]

### 7.3.3 Comments

Two forms of comments are supported: delimited comments and single-line comments.

A ***delimited comment*** begins with the characters /* and ends with the characters */. Delimited comments can occupy a portion of a line, a single line, or multiple lines. [*Example*: The example





```
/* Hello, world program
     This program writes "hello, world" to the console
*/
class Hello
{
   static void Main() {
      System.Console.WriteLine("hello, world");
   }
}
```

includes a delimited comment. *end example*]

A ***single-line comment*** begins with the characters // and extends to the end of the line. [*Example*: The example

```
// Hello, world program
//    This program writes "hello, world" to the console
//
class Hello // any name will do for this class
{
   static void Main() { // this method must be named "Main"
      System.Console.WriteLine("hello, world");
   }
}
```

shows several single-line comments. *end example*]

*comment::*
*single-line-comment*
*delimited-comment*

*single-line-comment::*
*// input-characters*$_{opt}$

*input-characters::*
*input-character*
*input-characters input-character*

*input-character::*
*Any Unicode character except a new-line character*

*new-line-character::*
*Carriage return character (U+000D)*
*Line feed character (U+000A)*
*Next line character (U+0085)*
*Line separator character (U+2028)*
*Paragraph separator character (U+2029)*

*delimited-comment::*
*/* *delimited-comment-text*$_{opt}$ *asterisks* */*

*delimited-comment-text::*
*delimited-comment-section*
*delimited-comment-text delimited-comment-section*

*delimited-comment-section::*
*/*
*asterisks*$_{opt}$ *not-slash-or-asterisk*

*asterisks::*
**
*asterisks* **

*not-slash-or-asterisk::*
*Any Unicode character except / or **





Comments do not nest. The character sequences /* and */ have no special meaning within a single-line comment, and the character sequences // and /* have no special meaning within a delimited comment.

Comments are not processed within character and string literals.

[*Note*: These rules must be interpreted carefully. For instance, in the example below, the delimited comment that begins before A ends between B and C(). The reason is that

```
// B */ C();
```

is not actually a single-line comment, since // has no special meaning within a delimited comment, and so */ does have its usual special meaning in that line.

Likewise, the delimited comment starting before D ends before E. The reason is that "D */ " is not actually a string literal, since it appears inside a delimited comment.

A useful consequence of /* and */ having no special meaning within a single-line comment is that a block of source code lines can be commented out by putting // at the beginning of each line. In general it does not work to put /* before those lines and */ after them, as this does not properly encapsulate delimited comments in the block, and in general may completely change the structure of such delimited comments.

Example code:

```
static void Main() {
  /* A
  // B */ C();
  Console.WriteLine(/* "D */ "E");
}
```

*end note*]

### 7.3.4 White space

White space is defined as any character with Unicode class Zs (which includes the space character) as well as the horizontal tab character, the vertical tab character, and the form feed character.

*whitespace::*
*whitespace-character*
*whitespace whitespace-character*

*whitespace-character::*
*Any character with Unicode class Zs*
*Horizontal tab character (U+0009)*
*Vertical tab character (U+000B)*
*Form feed character (U+000C)*

## 7.4 Tokens

### 7.4.1 General

There are several kinds of **tokens**: identifiers, keywords, literals, operators, and punctuators. White space and comments are not tokens, though they act as separators for tokens.

*token::*
*identifier*
*keyword*
*integer-literal*
*real-literal*
*character-literal*
*string-literal*
*operator-or-punctuator*



Стр. 16 - 18

# Виды эффективных и неэффективных комментариев

1. Повторение кода
2. Объяснение кода
3. Маркер в коде
4. Резюме кода
5. Описание цели кода
6. Информация, которую невозможно выразить в форме кода

# Спасибо за внимание! До новых встреч!


Александр Шевчук


MCID: 9230440


ITVDN
IT VIDEO DEVELOPERS NETWORK