



ПРОГРАМНІ ЗАСОБИ ДЛЯ РОБОТИ З ТЕКСТОМ

Питання 2.4.

Робота з текстом

Простір імен	Тип	Опис
System	Char	Зберігання одного текстового символу
System	String	Зберігання кількох текстових символів
System.Text	StringBuilder	Ефективне керування рядками
System.Text.RegularExpressions	Regex	Ефективне керування рядками, які відповідають шаблонам

Управляющая последовательность	Описание
\'	Вставляет в строковый литерал символ одинарной кавычки
\"	Вставляет в строковый литерал символ двойной кавычки
\\	Вставляет в строковый литерал символ обратной косой черты. Особенно полезна при определении путей к файлам и сетевым ресурсам
\a	Заставляет систему выдавать звуковой сигнал, который в консольных приложениях может служить аудио-подсказкой пользователю
\n	Вставляет символ новой строки (на платформах Windows)
\r	Вставляет символ возврата каретки
\t	Вставляет в строковый литерал символ горизонтальной табуляции

- Як і в інших С-подібних мовах, рядкові літерали C# можуть містити різні керуючі послідовності (escape sequences), які дозволяють уточнювати вигляд символьних даних у вихідному потоці.

Використання управляючих послідовностей та дослівних рядків

```
static void EscapeChars()
{
    Console.WriteLine("=> Escape characters:\a");
    string strWithTabs = "Model\tColor\tSpeed\tPet Name\a ";
    Console.WriteLine(strWithTabs);

    Console.WriteLine("Everyone loves \"Hello World\"\a ");
    Console.WriteLine("C:\\MyApp\\bin\\Debug\a ");

    // Додати 4 пустих строки і знову надати звуковий сигнал.
    Console.WriteLine("All finished.\n\n\n\a ");
    Console.WriteLine();
}
```

```
// Следующая строка воспроизводится дословно,
// с отображением всех управляющих последовательностей.
Console.WriteLine(@"C:\MyApp\bin\Debug");
```

```
// При использовании дословных строк пробельные символы предохраняются.
string myLongString = @"This is a very
    very
        long string";
Console.WriteLine(myLongString);
```

- Дослівні рядки дозволяють відключати обробку керуючих послідовностей у літералах та виводити значення string у том виді, як вони є.
 - Для цього до рядкового літерала додається префікс @
 - Найбільш корисна можливість при роботі з рядками, які представляють шляхи до каталогів і мережевих ресурсів.
 - Також можуть застосовуватись для зберігання пробільних символів у рядках, розділених на кілька рядків виводу.
 - Використовуючи дослівні рядки, можна прямо в текст вставляти подвійні лапки:

```
Console.WriteLine(@"Cerebus said ""Darrrr! Pret-ty sun-sets""");
```

Властивості та методи класу Char

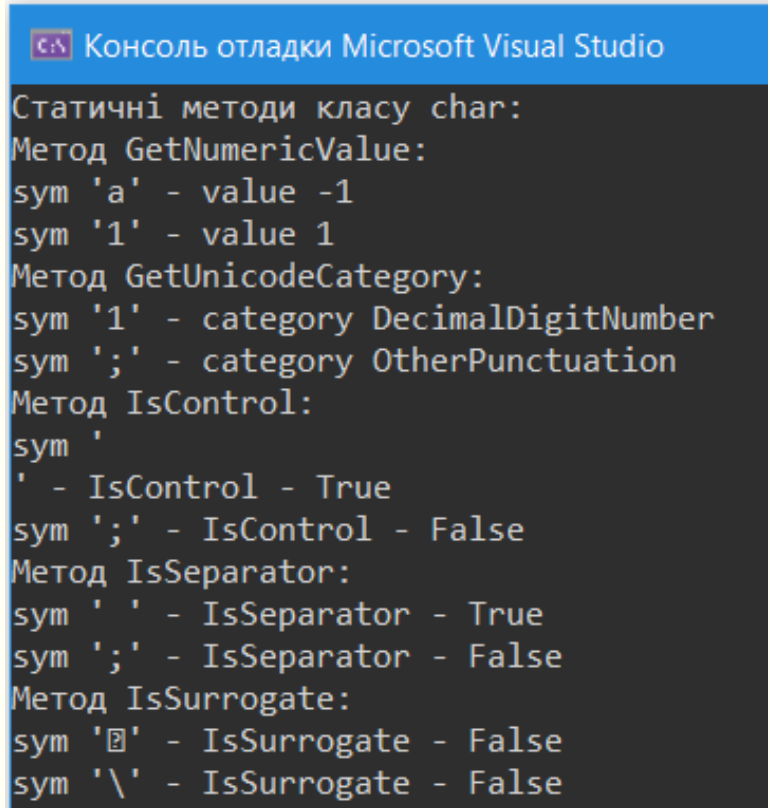
Метод	Опис
GetNumericValue	Повертає числове значення символу, якщо він є цифрою, інакше – (-1)
GetUnicodeCategory	Метод повертає Unicode-категорію символу.
IsControl	Повертає true, якщо символ є керуючим
IsDigit	Повертає true, якщо символ є десятковою цифрою
IsLetter	Повертає true, якщо символ є буквою
IsLetterOrDigit	Повертає true, якщо символ є буквою або цифрою
IsLower	Повертає true, якщо символ задано в нижньому регістрі
IsNumber	Повертає true, якщо символ є числом (десятковою або шістнадцятковою цифрою)
IsPunctuation	Повертає true, якщо символ є пунктуаційним знаком
IsSeparator	Повертає true, якщо символ є роздільником
IsSurrogate	Метод повертає true, якщо символ є сурогатним: деякі символи Unicode з кодом з інтервалу [0x1000, 0x10FFF] представляються двома 16-бітними "сурогатними" символами
IsUpper	Повертає true, якщо символ задано в верхньому регістрі
IsWhiteSpace	Повертає true, якщо символ є "білим пробілом". До білих пробілів, крім власне пробілу, відносяться символ кінця рядка, символ переводу каретки та ін.
Parse	Перетворює рядок з одного символу в символ
ToLower	Зводить символ до нижнього регістра
ToUpper	Зводить символ до верхнього регістра
MaxValue, MinValue	Властивості, які повертають символи з максимальним та мінімальним кодом. Ці символи не мають видимого образу

```
public static int SayCode(char sym) { return sym; }
```

```
static void Main(string[] args)
{
```

```
    Console.OutputEncoding = Encoding.UTF8;
    Console.WriteLine("Статичні методи класу char:");
    char ch = 'a', ch1 = '1', lim = ';', chc = '\xA';
    double d1, d2;
    d1 = char.GetNumericValue(ch);
    d2 = char.GetNumericValue(ch1);
    Console.WriteLine("Метод GetNumericValue:");
    Console.WriteLine("sym 'a' - value {0}", d1);
    Console.WriteLine("sym '1' - value {0}", d2);
    System.Globalization.UnicodeCategory cat1, cat2;
    cat1 = char.GetUnicodeCategory(ch1);
    cat2 = char.GetUnicodeCategory(lim);
    Console.WriteLine("Метод GetUnicodeCategory:");
    Console.WriteLine("sym '1' - category {0}", cat1);
    Console.WriteLine("sym ';' - category {0}", cat2);
    Console.WriteLine("Метод IsControl:");
    Console.WriteLine("sym '\xA' - IsControl - {0}", char.IsControl(chc));
    Console.WriteLine("sym ';' - IsControl - {0}", char.IsControl(lim));
    Console.WriteLine("Метод IsSeparator:");
    Console.WriteLine("sym ' ' - IsSeparator - {0}", char.IsSeparator(' '));
    Console.WriteLine("sym ';' - IsSeparator - {0}", char.IsSeparator(lim));
    Console.WriteLine("Метод IsSurrogate:");
    Console.WriteLine("sym '\u10FF' - IsSurrogate - {0}", char.IsSurrogate('\u10FF'));
    Console.WriteLine("sym '\\\n' - IsSurrogate - {0}", char.IsSurrogate('\n'));
    ...
```

Демонстрація властивостей та методів класу Char

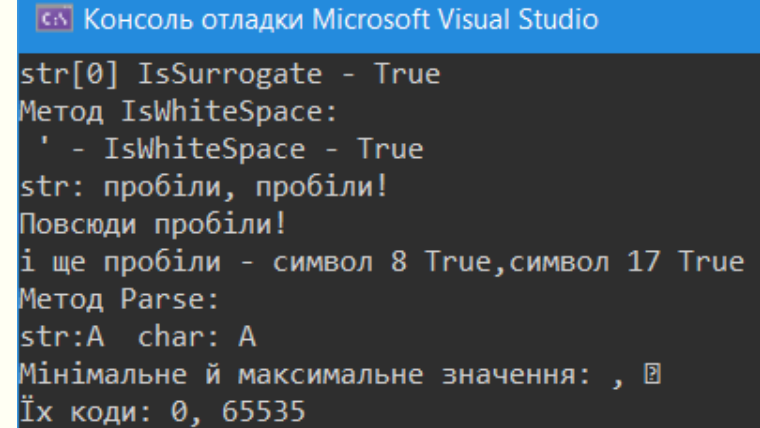


Консоль отладки Microsoft Visual Studio

```
Статичні методи класу char:
Метод GetNumericValue:
sym 'a' - value -1
sym '1' - value 1
Метод GetUnicodeCategory:
sym '1' - category DecimalDigitNumber
sym ';' - category OtherPunctuation
Метод IsControl:
sym '
' - IsControl - True
sym ';' - IsControl - False
Метод IsSeparator:
sym ' ' - IsSeparator - True
sym ';' - IsSeparator - False
Метод IsSurrogate:
sym '\u10FF' - IsSurrogate - False
sym '\\\n' - IsSurrogate - False
```

Продовження коду та виводу

```
// сурогатні символи Unicode з інтервалу [0x10000,0x10FFF]
// представляються двома 16-бітними символами
Console.WriteLine("str = {0}, str[0] = {1}", str, str[0]);
Console.WriteLine("str[0] IsSurrogate - {0}", char.IsSurrogate(str, 0));
Console.WriteLine("Метод IsWhiteSpace:");
str = "пробіли, пробіли!" + "\xD" + "\xA" + "Повсюди пробіли!";
Console.WriteLine("sym '\xD' - IsWhiteSpace - {0}", char.IsWhiteSpace('\xD'));
Console.WriteLine("str: {0}", str);
Console.WriteLine("і ще пробіли - символ 8 {0}," + "символ 17 {1}",
    char.IsWhiteSpace(str, 8), char.IsWhiteSpace(str, 17));
Console.WriteLine("Метод Parse:");
str = "A";
ch = char.Parse(str);
Console.WriteLine("str:{0} char: {1}", str, ch);
Console.WriteLine("Мінімальне й максимальне значення: {0}, {1}",
    char.MinValue.ToString(), char.MaxValue.ToString());
Console.WriteLine("Їх коди: {0}, {1}", SayCode(char.MinValue), SayCode(char.MaxValue));
}
```



Консоль отладки Microsoft Visual Studio

```
str[0] IsSurrogate - True
Метод IsWhiteSpace:
' - IsWhiteSpace - True
str: пробіли, пробіли!
Повсюди пробіли!
і ще пробіли - символ 8 True,символ 17 True
Метод Parse:
str:A char: A
Мінімальне й максимальне значення: , 
Їх коди: 0, 65535
```

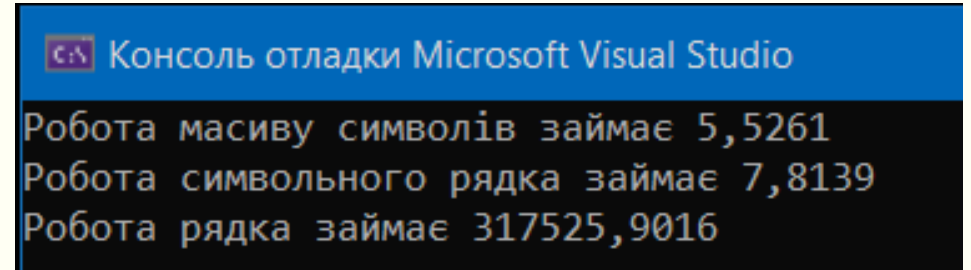
Масиви символів у C#

```
int n = 1000000;
TimeSpan time = Time(() => {
    // Use a new char array.
    char[] buffer = new char[n];
    for (int i = 0; i < n; i++)
    {
        buffer[i] = 'a';
    }
    string result = new string(buffer);
});
Console.WriteLine("Робота масиву символів займає {0}", time.TotalMilliseconds);

TimeSpan time2 = Time(() => {
    // Declare new StringBuilder and append to it 100 times.
    StringBuilder builder = new StringBuilder(n);
    for (int i = 0; i < n; i++)
    {
        builder.Append('a');
    }
    string result = builder.ToString();
});
Console.WriteLine("Робота символьного рядка займає {0}", time2.TotalMilliseconds);

TimeSpan time3 = Time(() => {
    // Declare new StringBuilder and append to it 100 times.
    string result = "";
    for (int i = 0; i < n; i++)
    {
        result += 'a';
    }
});
Console.WriteLine("Робота рядка займає {0}", time3.TotalMilliseconds);
```

- Виступають альтернативою до типу `StringBuilder` для швидкого зберігання інформації.



- Перевага `StringBuiler` – можливість додавати символи в буфер навіть після того, як `n` символів уже туди покладені.
- `char`-масиви працюють швидше, оскільки передбачають меншу функціональність та менше сценаріїв використання.
- Змінні `string` можуть бути зчепленими разом для створення рядків більшого розміру за допомогою операції `+` мови C# або виклику статичного методу `String.Concat()`.

Рядки є незмінюваними (immutable)

- Після присвоєння об'єкту `string` початкового значення символьні дані не можуть бути зміненими.
 - Якщо переглянути роботу методів типу `string` «за кулісами», можна помітити, що насправді вони повертають абсолютно новий об'єкт `string` в модифікованому вигляді.
 - У прикладі початковий об'єкт `string` (`s1`) не було перетворено в верхній регістр, коли викликався метод `ToUpper()`.
 - Насправді, була повернена копія змінної типу `string` у зміненому форматі.
 - Той же закон діє і при здійсненні присвоєння для рядків.

```
static void StringsAreImmutable()
{
    // Установить первоначальное значение для строки.
    string s1 = "This is my string.";
    Console.WriteLine("s1 = {0}", s1);

    // Преобразование s1 в верхний регистр?
    string upperString = s1.ToUpper();
    Console.WriteLine("upperString = {0}", upperString);

    // Нет! s1 по-прежнему остается в том же формате!
    Console.WriteLine("s1 = {0}", s1);
}
```

```
s1 = This is my string.
upperString = THIS IS MY STRING.
s1 = This is my string.
```


Причини та наслідки незмінюваності класу

- Причини незмінюваності типу string
- Клас string може стати неефективним та призводити до «розбухання» коду при неправильному використанні, особливо при виконанні конкатенації рядків.
 - Проте коли необхідно представляти базові символні дані, такі як ідентифікаційний номер, ім'я та прізвище, прості фрагменти тексту, що використовуються всередині додатку, string є ідеальним варіантом.
 - Якщо будується додаток, в якому будуть часто змінюватись текстові дані (наприклад, текстовий процесор), то представлення текстових даних, які обробляються, застосовуючи об'єкти string, буде дуже невдалим рішенням.

Работа з рядковими даними

Член System.String	Описание
Length	Свойство, которое возвращает длину текущей строки
Compare()	Статический метод, который позволяет сравнить две строки
Contains()	Метод, который позволяет определить, содержится ли в строке определенная подстрока
Equals()	Метод, который позволяет проверить, содержатся ли в двух строковых объектах идентичные символьные данные
Format()	Статический метод, позволяющий сформатировать строку с использованием других элементарных типов данных (например, числовых данных или других строк) и нотации {0}, которая рассматривалась ранее в этой главе
Insert()	Метод, который позволяет вставить строку внутрь другой определенной строки
PadLeft() PadRight()	Методы, которые позволяют дополнить строку некоторыми символами
Remove() Replace()	Методы, которые позволяют получить копию строки с соответствующими изменениями (удалением или заменой символов)
Split()	Метод, который возвращает массив string, содержащий подстроки в этом экземпляре, которые разделяются элементами из указанного массива char или string
Trim()	Метод, который удаляет все вхождения набора указанных символов с начала и конца текущей строки
ToUpper() ToLower()	Методы, которые создают копию текущей строки в верхнем или нижнем регистре

Базові маніпуляції з рядками

```
static void BasicStringFunctionality()
{
    Console.WriteLine("=> Basic String functionality:");
    string firstName = "Freddy";
    // Значение firstName.
    Console.WriteLine("Value of firstName: {0}", firstName);
    // Длина firstName.
    Console.WriteLine("firstName has {0} characters.", firstName.Length);
    // firstName в верхнем регистре.
    Console.WriteLine("firstName in uppercase: {0}", firstName.ToUpper());
    // firstName в нижнем регистре.
    Console.WriteLine("firstName in lowercase: {0}", firstName.ToLower());
    // Содержит ли firstName букву y?
    Console.WriteLine("firstName contains the letter y?: {0}",
        firstName.Contains("y"));
    // firstName после замены.
    Console.WriteLine("firstName after replace: {0}", firstName.Replace("dy", ""));
    Console.WriteLine();
}
```

```
***** Fun with Strings *****
=> Basic String functionality:
Value of firstName: Freddy
firstName has 6 characters.
firstName in uppercase: FREDDY
firstName in lowercase: freddy
firstName contains the letter y?: True
firstName after replace: Fred
```

Інші члени класу string

Член	Описание
Trim, TrimStart и TrimEnd	Удаляют пробельные символы в начале и/или конце строки
ToUpper и ToLower	Преобразуют символы строки в прописные или строчные
Insert и Remove	Добавляют или удаляют указанный текст в переменной типа string
Replace	Замещает указанный текст
String.Concat	Конкатенирует две переменные типа string. Оператор + вызывает этот метод, если используется между переменными типа string
String.Join	Конкатенирует одну или несколько переменных типа string с указанным символом между ними
string.IsNullOrEmpty	Проверяет, хранит ли переменная типа string значение null или она пустая ("")
string.IsNullOrWhiteSpace	Проверяет, является ли переменная типа string значением null, пустой строкой или строкой, состоящей только из пробельных символов (например, табуляции, пробела, возврата каретки, перевода строки и т. д.)
String.Empty	Можно задействовать вместо выделения памяти каждый раз, когда вы применяете литеральное значение string, используя пару двойных кавычек без содержимого ("")
string.Format	Устаревший альтернативный метод вывода форматированных строк, применяющий позиционированные параметры вместо именованных

```

class Program
{
    static void Main(string[] args)
    {
        string city = "London";
        WriteLine($"{city} is {city.Length} characters long.");
        WriteLine($"First char is {city[0]} and third is {city[2]}.");

        string cities = "Paris,Berlin,Madrid,New York";
        string[] citiesArray = cities.Split(',');
        foreach (string item in citiesArray)
        {
            WriteLine(item);
        }

        string fullname = "Alan Jones";
        int indexOfTheSpace = fullname.IndexOf(' ');
        string firstname = fullname.Substring(0, indexOfTheSpace);
        string lastname = fullname.Substring(indexOfTheSpace + 1);
        WriteLine($"{lastname}, {firstname}");

        string company = "Microsoft";
        bool startsWithM = company.StartsWith("M");
        bool containsN = company.Contains("N");
        WriteLine($"Starts with M: {startsWithM}, contains an N:{containsN}");
    }
}

```

Консоль отладки Microsoft Visual Studio

```

London is 6 characters long.
First char is L and third is n.
Paris
Berlin
Madrid
New York
Jones, Alan
Starts with M: True, contains an N:False

```



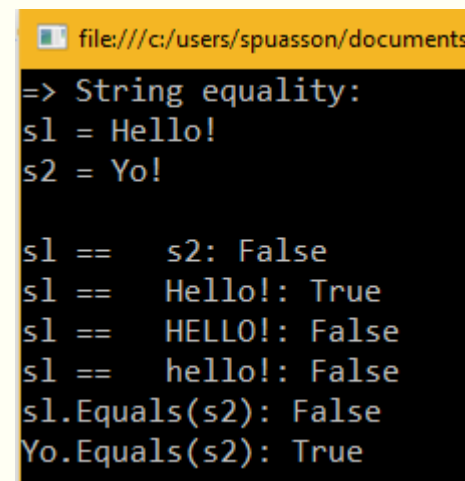
stringUsage.cs

Отримання довжини рядка
 Отримання символів рядка
 Розділення рядків
 Отримання фрагменту рядка
 Перевірка вмісту рядків

Рядки та рівність

- За умовчанням при виконанні перевірки на рівність посилальних типів (за допомогою операцій `==` та `!=`) значення `true` буде повертатись тоді, коли обидва посилання вказують на один і той же об'єкт у пам'яті.
 - Незважаючи на те, що `string` є посилальним типом, операції порівняння для нього були переозначені так, щоб можна було порівнювати значення об'єктів `string`, а не посилання на об'єкти в пам'яті.

```
static void StringEquality()
{
    Console.WriteLine("=> String equality:");
    string s1 = "Hello!";
    string s2 = "Yo!";
    Console.WriteLine("s1 = {0}", s1);
    Console.WriteLine("s2 = {0}", s2);
    Console.WriteLine();
    // Проверить строки на предмет равенства.
    Console.WriteLine("s1 == s2: {0}", s1 == s2);
    Console.WriteLine("s1 == Hello!: {0}", s1 == "Hello!");
    Console.WriteLine("s1 == HELLO!: {0}", s1 == "HELLO!");
    Console.WriteLine("s1 == hello!: {0}", s1 == "hello!");
    Console.WriteLine("s1.Equals(s2): {0}", s1.Equals(s2));
    Console.WriteLine("Yo.Equals(s2): {0}", "Yo!".Equals(s2));
    Console.WriteLine();
}
```



```
file:///c:/users/spuasson/documents
=> String equality:
s1 = Hello!
s2 = Yo!

s1 == s2: False
s1 == Hello!: True
s1 == HELLO!: False
s1 == hello!: False
s1.Equals(s2): False
Yo.Equals(s2): True
```

Тип System.Text.StringBuilder

```
static void FunWithStringBuilder()
{
    Console.WriteLine("=> Using the StringBuilder:");
    StringBuilder sb = new StringBuilder("**** Fantastic Games ****");
    sb.Append("\n");
    sb.AppendLine("Half Life");
    sb.AppendLine("Morrowind");
    sb.AppendLine("Deus Ex" + "2");
    sb.AppendLine("System Shock");
    Console.WriteLine(sb.ToString());
    sb.Replace("2", " Invisible War");
    Console.WriteLine(sb.ToString());
    Console.WriteLine("sb has {0} chars.", sb.Length);
    Console.WriteLine();
}
```

- Як і System.String, клас StringBuilder визначає методи, які дозволяють, наприклад, замінювати або форматовувати сегменти.
 - Щоб використовувати цей клас в файлах коду C #, насамперед знадобиться імпортувати простір імен System.Text;
- Унікальним у StringBuilder є те, що при виклику його членів проводиться безпосередня зміна внутрішніх символічних даних об'єкта (роблячи його більш ефективним), без отримання копії даних в модифікованому форматі.
 - При створенні екземпляра StringBuilder початкові значення об'єкта можуть бути задані через один з конструкторів.
- За замовчуванням StringBuilder здатний зберігати рядок довжиною не більше 16 символів (але при необхідності буде автоматично розширюватися), однак значення початкової довжини можна змінити через додатковий аргумент конструктора:
 - // Створити StringBuilder з початковим розміром на 256 символів.
StringBuilder sb = new StringBuilder("*** Fantastic Games ***", 256);

Регулярні вирази в .NET

- Потужна нотація регулярних виразів як шаблонів для зіставлення дозволяє швидко розбирати (парсити) великі обсяги тексту для:
 - знаходження конкретних символічних шаблонів;
 - валідації тексту з метою перевірки відповідності щодо заданого шаблону;
 - витягування, редагування, заміни чи видалення підрядків тексту;
 - додавання витягнутих рядків до колекції з метою генерування звіту.
- У .NET центром обробки тексту за допомогою регулярних виразів є двигун (engine) регулярних виразів, представлений об'єктом System.Text.RegularExpressions.Regex.
 - Для роботи двигуна мінімально потрібні **шаблон** у вигляді регулярного виразу та **текст** для зіставлення.
 - Шаблони регулярних виразів мають спеціальний синтаксис, сумісний з регулярними виразами Perl 5, з деякими включеннями, на зразок зіставлення справа наліво (right-to-left matching).

Синтаксис регулярних виразів

Символ	Значение	Символ	Значение
^	Начало ввода	\$	Конец ввода
\d	Одна цифра	\D	Любой нецифровой символ
\w	Пробельный символ	\W	Любой символ, кроме пробельного
[A-Za-z0-9]	Диапазон символов	\^	Символ ^ (каретки)
[aeiou]	Набор символов	[^aeiou]	Любой символ, кроме входящего в набор
.	Один символ	\.	Символ . (точка)

- Символы, які впливають на попередній символ у регулярному виразі

Символ	Значение	Символ	Значение
+	Один или больше	?	Один или ни одного
{3}	Точно три	{3,5}	От трех до пяти
{3,}	Три или больше	{,3}	До трех

.NET FRAMEWORK REGULAR EXPRESSIONS



SINGLE CHARACTERS

Use	To match any character
[set]	In that set
[^set]	Not in that set
[a-z]	In the a-z range
[^a-z]	Not in the a-z range
.	Any except \n (new line)
\char	Escaped special character

CONTROL CHARACTERS

Use	To match	Unicode
\t	Horizontal tab	\u0009
\v	Vertical tab	\u000B
\b	Backspace	\u0008
\e	Escape	\u001B
\r	Carriage return	\u000D
\f	Form feed	\u000C
\n	New line	\u000A
\a	Bell (alarm)	\u0007
\c char	ASCII control character	—

NON-ASCII CODES

Use	To match character with
\octal	2-3 digit octal character code
\x hex	2-digit hex character code
\u hex	4-digit hex character code

CHARACTER CLASSES

Use	To match character
\p{ctgry}	In that Unicode category or block
\P{ctgry}	Not in that Unicode category or block
\w	Word character
\W	Non-word character
\d	Decimal digit
\D	Not a decimal digit
\s	White-space character
\S	Non-white-space char

QUANTIFIERS

Greedy	Lazy	Matches
*	*?	0 or more times
+	+	1 or more times
?	??	0 or 1 time
{n}	{n}?	Exactly n times
{n,}	{n,}?	At least n times
{n,m}	{n,m}?	From n to m times

ANCHORS

Use	To specify position
^	At start of string or line
\A	At start of string
\z	At end of string
\Z	At end (or before \n at end) of string
\$	At end (or before \n at end) of string or line
\G	Where previous match ended
\b	On word boundary
\B	Not on word boundary

GROUPS

Use	To define
(exp)	Indexed group
(?<name>exp)	Named group
(?<name1-name2>exp)	Balancing group
(?:exp)	Noncapturing group
(?=exp)	Zero-width positive lookahead
(?!exp)	Zero-width negative lookahead
(?<=exp)	Zero-width positive lookbehind
(?<!exp)	Zero-width negative lookbehind
(?>exp)	Non-backtracking (greedy)

INLINE OPTIONS

Option	Effect on match
i	Case-insensitive
m	Multiline mode
n	Explicit (named)
s	Single-line mode
x	Ignore white space

Use	To
(?imnsx-imnsx)	Set or disable the specified options
(?imnsx-imnsx:exp)	Set or disable the specified options within the expression

June 2014

© 2014 Microsoft. All rights reserved.

BACKREFERENCES

Use	To match
<code>\n</code>	Indexed group
<code>\k<name></code>	Named group

ALTERNATION

Use	To match
<code>a b</code>	Either <i>a</i> or <i>b</i>
<code>(?(exp)</code>	<i>yes</i> if <i>exp</i> is matched
<code>yes no)</code>	<i>no</i> if <i>exp</i> isn't matched
<code>(?(name)</code>	<i>yes</i> if <i>name</i> is matched
<code>yes no)</code>	<i>no</i> if <i>name</i> isn't matched

SUBSTITUTION

Use	To substitute
<code>\$n</code>	Substring matched by group number <i>n</i>
<code>\${name}</code>	Substring matched by group <i>name</i>
<code>\$\$</code>	Literal \$ character
<code>\$&</code>	Copy of whole match
<code>\$`</code>	Text before the match
<code>\$'</code>	Text after the match
<code>\$+</code>	Last captured group
<code>\$_</code>	Entire input string

COMMENTS

Use	To
<code>(?# comment)</code>	Add inline comment
<code>#</code>	Add x-mode comment

For detailed information and examples, see <http://aka.ms/regex>

To test your regular expressions, see <http://regexlib.com/RETester.aspx>

SUPPORTED UNICODE CATEGORIES

Category	Description
Lu	Letter, uppercase
Li	Letter, lowercase
Lt	Letter, title case
Lm	Letter, modifier
Lo	Letter, other
L	Letter, all
Mn	Mark, nonspacing combining
Mc	Mark, spacing combining
Me	Mark, enclosing combining
M	Mark, all diacritic
Nd	Number, decimal digit
Nl	Number, letterlike
No	Number, other
N	Number, all
Pc	Punctuation, connector
Pd	Punctuation, dash
Ps	Punctuation, opening mark
Pe	Punctuation, closing mark
Pi	Punctuation, initial quote mark
Pf	Punctuation, final quote mark
Po	Punctuation, other
P	Punctuation, all
Sm	Symbol, math
Sc	Symbol, currency
Sk	Symbol, modifier
So	Symbol, other
S	Symbol, all
Zs	Separator, space
Zl	Separator, line
Zp	Separator, paragraph
Z	Separator, all
Cc	Control code
Cf	Format control character
Cs	Surrogate code point
Co	Private-use character
Cn	Unassigned
C	Control characters, all

For named character set blocks (e.g., Cyrillic), search for "supported named blocks" in the MSDN Library.

REGULAR EXPRESSION OPERATIONS

Class: `System.Text.RegularExpressions.Regex`

Pattern matching with Regex objects

To initialize with	Use constructor
Regular exp	<code>Regex(String)</code>
+ options	<code>Regex(String, RegexOptions)</code>
+ time-out	<code>Regex(String, RegexOptions, TimeSpan)</code>

Pattern matching with static methods

Use an overload of a method below to supply the regular expression and the text you want to search.

Finding and replacing matched patterns

To	Use method
Validate match	<code>Regex.IsMatch</code>
Retrieve single match	<code>Regex.Match (first)</code> <code>Match.NextMatch (next)</code>
Retrieve all matches	<code>Regex.Matches</code>
Replace match	<code>Regex.Replace</code>
Divide text	<code>Regex.Split</code>
Handle char escapes	<code>Regex.Escape</code> <code>Regex.Unescape</code>

Getting info about regular expression patterns

To get	Use Regex API
Group names	<code>GetGroupNames</code> <code>GetGroupNameFromNumber</code>
Group numbers	<code>GetGroupNumbers</code> <code>GetGroupNumberFromName</code>
Expression	<code>ToString</code>
Options	<code>Options</code>
Time-out	<code>MatchTimeout</code>

Приклади регулярних виразів

Выражение	Значение
\d	Одна цифра где-либо в вводе
a	Символ где-либо в вводе
Bob	Слово Bob где-либо в вводе
^Bob	Слово Bob в начале ввода
Bob\$	Слово Bob в конце ввода
^\d{2}\$	Точно две цифры
^[0-9]{2}\$	Точно две цифры
^[A-Z]{4,}\$	Не менее четырех прописных букв
^[A-Za-z]{4,}\$	Не менее четырех прописных или строчных букв
^[A-Z]{2}\d{3}\$	Точно две прописные буквы и три цифры
^d.g\$	Буква d, далее любой символ, а затем буква g, так что это выражение совпадет со словами типа dig, dog и др. с любым символом между буквами d и g
^d\.g\$	Буква d, далее точка (.), а затем буква g, поэтому данное выражение совпадает только с последовательностью d.g

Операції в класі Regex

- Determine whether the regular expression pattern occurs in the input text by calling the Regex.IsMatch method. For an example that uses the IsMatch method for validating text, see How to: Verify that Strings Are in Valid Email Format.
- Retrieve one or all occurrences of text that matches the regular expression pattern by calling the Regex.Match or Regex.Matches method. The former method returns a System.Text.RegularExpressions.Match object that provides information about the matching text. The latter returns a MatchCollection object that contains one System.Text.RegularExpressions.Match object for each match found in the parsed text.
- Replace text that matches the regular expression pattern by calling the Regex.Replace method. For examples that use the Replace method to change date formats and remove invalid characters from a string, see How to: Strip Invalid Characters from a String and Example: Changing Date Formats.

Зіставлення шаблонів з регулярними виразами

```
1 using static System.Console;
2 using System.Text.RegularExpressions;
3
4 namespace ConsoleApp
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             Write("Enter your age: ");
11             string input = ReadLine();
12             var ageChecker = new Regex(@"^\d+$");
13             if (ageChecker.IsMatch(input))
14             {
15                 WriteLine("Thank you!");
16             }
17             else
18             {
19                 WriteLine($"This is not a valid age: {input}");
20             }
21         }
22     }
23 }
```

Консоль отладки Microsoft Visual Studio

```
Enter your age: 70
Thank you!
```

Консоль отладки Microsoft Visual Studio

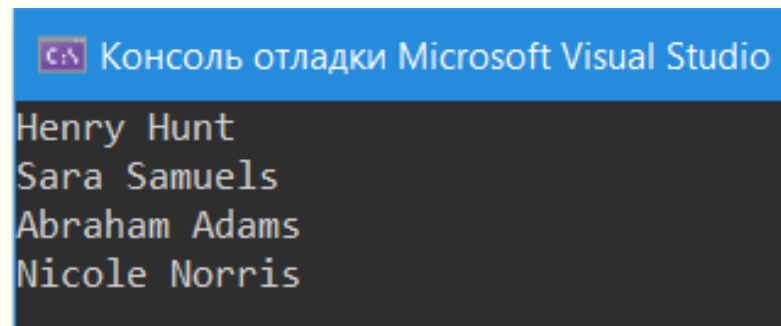
```
Enter your age: B52
This is not a valid age: B52
```


Приклад 1: заміна підрядків

```
using System;
using System.Text.RegularExpressions;

namespace UseRegex
{
    class Program
    {
        static void Main(string[] args)
        {
            string pattern = "(Mr\\.?.? |Mrs\\.?.? |Miss |Ms\\.?.? )";
            string[] names = { "Mr. Henry Hunt", "Ms. Sara Samuels",
                               "Abraham Adams", "Ms. Nicole Norris" };
            foreach (string name in names)
                Console.WriteLine(Regex.Replace(
                    name, pattern, String.Empty));
        }
    }
}
```

- Нехай mailing list містить звернення, які інколи включають title (Mr., Mrs., Miss або Ms.) біля ім'я та прізвища.
 - If you do not want to include the titles when you generate envelope labels from the list, you can use a regular expression to remove the titles
 - The regular expression pattern (Mr\\.?.? |Mrs\\.?.? |Miss |Ms\\.?.?) matches any occurrence of "Mr ", "Mr. ", "Mrs ", "Mrs. ", "Miss ", "Ms or "Ms. ".



```
Консоль отладки Microsoft Visual Studio
Henry Hunt
Sara Samuels
Abraham Adams
Nicole Norris
```

Example 2: Identify duplicated words

- Accidentally duplicating words is a common error that writers make. A regular expression can be used to identify duplicated words.
 - The regular expression pattern `\b(\w+?)\s\1\b` can be interpreted as follows:

Pattern	Interpretation
<code>\b</code>	Start at a word boundary.
<code>(\w+?)</code>	Match one or more word characters, but as few characters as possible. Together, they form a group that can be referred to as <code>\1</code> .
<code>\s</code>	Match a white-space character.
<code>\1</code>	Match the substring that is equal to the group named <code>\1</code> .
<code>\b</code>	Match a word boundary.

Example 3: Dynamically build a culture-sensitive regular expression

- power of regular expressions combined with the flexibility offered by .NET's globalization features.
 - It uses the NumberFormatInfo object to determine the format of currency values in the system's current culture.
 - It then uses that information to dynamically construct a regular expression that extracts currency values from the text.
 - For each match, it extracts the subgroup that contains the numeric string only, converts it to a Decimal value, and calculates a running total.
- Застосування регулярних виразів у Visual Studio

Теми для доповідей

- Об'єктна модель регулярних виразів у .NET