Лабораторна робота № 8.

Тема: ***Символьні рядки. Регулярні вирази. Файли.***

Мета роботи: Ознайомлення з засобами обробки текстової інформації та роботою з файлами в C#.

Теоретичні відомості

Мова C# надає розв'язку широкий набір засобів обробки тестової інформації: окремі символи, масиви символів, змінювані й незмінні рядки та регулярні вирази.

Клас char

Символьний клас char, заснований на класі `System.Char`, що використовує двохбайтове кодування `Unicode` представлення символів. У цьому класі визначені статичні методи, що дозволяють задати вид і категорію символу, а також перетворити символ у верхній або нижній регістр і в число. Для цього типу в мові визначені символьні сталі - символьні літерали. Сталі можна задавати:

- символом, укладеним в одинарні лапки;
- escape-послідовністю, що задає код символу;
- Unicode-послідовністю, що задає Unicode-код символу.

```
char ch1='A', ch2 ='\x5A', ch3='\u0058';
char ch = new char();
```

Три символьні змінні ініціалізовані сталими, значення яких задано трьома різними способами. Змінна `ch` оголошується в об'єктному стилі, використовуючи `new` і виклик конструктора класу. Тип `char`, як і всі типи `C#`, є класом. Цей клас успадковує властивості та методи класу `object`.

Масив символів, як і масив будь-якого іншого типу, побудований на основі базового класу `Array`. Застосування цих методів дозволяє ефективно вирішувати деякі задачі. У мові `C#` визначений клас `char[]` і його можна використовувати для вистави рядків постійної довжини. Оскільки масиви в `C#` динамічні, то розширюється клас задач, у яких можна використовувати масиви символів.

Клас string

Основним типом при роботі з рядками є тип `string`, що задає рядка змінної довжини. Клас `string` у мові `C#` відноситься до типів посилання типів та призначений для робот з рядок символів у кодуванні `Unicode`. Йому відповідає базовий тип класу `System.String` бібліотеки `.Net`. Над рядками - об'єктами цього класу - визначений широкий набір операцій.

Над рядками визначені наступні операції:

- присвоєння (=);
- дві операції перевірки еквівалентності (= =) і (!=);
- конкатенація або зчеплення рядків (+);
- індексації ([]).

Кожний об'єкт `string` - це незмінна послідовність символів `Unicode`, тобто методи, призначені для зміни рядків, повертають змінені копії, вихідні ж рядки

залишаються незмінними. Невикористані "старі" копії автоматично віддаляються збирачем сміття.

Рядка типу StringBuilder

Можливості, надавані класом **`string`**, широкі, однак вимога незмінності його об'єктів може виявитися незручним. У цьому випадку для роботи з рядками застосовується клас **`StringBuilder`**, певний у просторі імен **`System.Text`**, що й дозволяє змінювати значення своїх екземплярів.

Регулярні вирази

Стандартний клас string дозволяє виконувати над рядками різні операції, у тому числі пошук, заміну, вставку і видалення підрядків. Тим не менш, є класи задач з обробки символьної інформації, де стандартних можливостей не вистачає. Щоб полегшити вирішення подібних завдань, у мові **C#** вбудований більш потужний апарат роботи з рядками, заснований на регулярних виразах.

Регулярні вирази призначені для обробки текстової інформації і забезпечують:

1. Ефективний пошук в тексті за заданим шаблоном;
2. Редагування тексту;
3. Формування підсумкових звітів за результатами роботи з текстом.

Регулярний вираз - це шаблон, по якому виконується пошук відповідного фрагмента тексту. Мова опису регулярних виразів складається з символів двох видів: звичайних символів і метасимволів. Звичайний символ представляє у виразі сам себе, а метасимвол - деякий клас символів.

Розглянемо найбільш вживані метасимволи:

Клас символів	Опис
`.`	Будь-який символ, крім \n.
`[]`	Будь-який одинарний символ з послідовності, записаної всередині дужок. Допускається використання діапазонів символів.
`[^]`	Будь-який одинарний символ, який не входить в послідовність, записану в середині дужок. Допускається використання діапазонів символів.
`\w`	Будь-який алфавітно-цифровий символ.
`\W`	Будь-який не алфавітно-цифровий символ.
`\s`	Будь-який пробільний символ.
`\S`	Будь-який не пробільний символ.
`\d`	Будь-яка десяткова цифра
`\D`	Будь-який символ, який не є десяткового цифрою

Крім метасимволів, що позначають класи символів, можуть застосовуватися уточнюючі метасимволи:

Уточнюючі символи	Опис
`^`	Фрагмент,що співпадає з регулярними виразами, слід шукати лише на початку рядка

$	Фрагмент, що співпадає з регулярними виразами, слід шукати тільки в кінці рядка
\A	Фрагмент, що співпадає з регулярними виразами, слід шукати лише на початку багаторядкового рядка
\Z	Фрагмент,що співпадає з регулярними виразами, слід шукати тільки в кінці багаторядкового рядка
\b	Фрагмент, що співпадає з регулярними виразами, починається або закінчується на межі слова, тобто між символами, які відповідають мета символам **\w** і **\W**
\B	Фрагмент, що співпадає з регулярними виразами, не повинен зустрічатися намежі слів

У регулярних виразах часто використовуються повторювачі - метасимволи, які розташовуються безпосередньо після звичайного символу або групи символів і задають кількість його повторень у виразі.

Повторювач	Опис
*	Нуль або більше повторень попереднього елемента
+	Одне або більше повторень попереднього елемента
?	Не більше одного повторення попереднього елемента
{n}	Рівно n повторень попереднього елемента
{n,}	Принаймні n повторень попереднього елемента
{n, m}	Від n до m повторень попереднього елемента

Регулярний вираз записується у вигляді рядкового літерала, причому перед рядком необхідно ставити символ **@**, який говорить про те, що рядок потрібно буде розглядати і в тому випадку, якщо він буде займати кілька рядків на екрані. Однак символ **@** можна не ставити, якщо як шаблон використовується шаблон без метасимволів.

Зауваження. Якщо потрібно знайти якийсь символ, який є метасимволом, наприклад, точка, можна це зробити поставивши перед ним зворотній слеш (**\.**).

Пошук в тексті за шаблоном

Простір імен бібліотеки базових класів **System.Text.RegularExpressions** містить всі об'єкти платформи **.NET Framework**, що мають відношення до регулярних виразів. Найважливішим класом, що підтримує регулярні вирази, є клас **Regex**. Для опису регулярного виразів у класі визначено кілька перевантажених конструкторів:

1. **Regex ()** - створює порожній вираз;
2. **Regex (String)** - створює заданий вираз;
3. **Regex (String, RegexOptions)** - створює заданий вираз і задає параметри для його обробки за допомогою елементів **RegexOptions** (наприклад, розрізняти чи ні великі та малі літери).

Пошук фрагментів рядків, які відповідають заданому виразу, виконується за допомогою методів **IsMatch**, **Match**, **Matches** класу **Regex**.

Метод **IsMatch** повертає **true**, якщо відповідний фрагмент в заданій рядку знайдено, і **false** у противному випадку.

Наприклад, необхідно визначити, чи зустрічається в заданому тексті слово собака:

```
static void Main ()
{
  Regex r = new Regex ("собака", RegexOptions.IgnoreCase);
  string text1 = "Кіт у будинку, собака в будці.";
  string text2 = "Котик в будинку, собачка в будці.";
  Console.WriteLine (r.IsMatch (text1));
  Console.WriteLine (r.IsMatch (text2));
}
```

Зауваження. **RegexOptions.IgnoreCase** - означає, що регулярний вираз застосовується без врахування регістра символів.

Визначення, чи є в заданих рядках номера телефону у форматі **xx-xx-xx** або **xxx-xx-xx** задається:

```
static void Main ()
{
  Regex r = new Regex (@ "\ d {2,3} (- \ d \ d) {2}");
  string text1 = "tel :123-45-67";
  string text2 = "tel: no";
  string text3 = "tel :12-34-56";
  Console.WriteLine (r.IsMatch (text1));
  Console.WriteLine (r.IsMatch (text2));
  Console.WriteLine (r.IsMatch (text3));
}
```

Метод **Match** класу **Regex** не просто визначає, чи міститься текст, відповідний шаблону, а повертає об'єкт класу **Match** - послідовність фрагментів тексту, які співпали з шаблоном. Наступний приклад дозволяє знайти всі номери телефонів в зазначеному фрагменті тексту:

```
static void Main ()
{
  Regex r = new Regex (@ "\ d {2,3} (- \ d \ d) {2}");
  string text = @ "Контакти в Києві tel :044-45-67-85, 044-45-34-56;
fax :123-56-45
                  Контакти в Чернівцях  tel :037-12-34-56; fax :
037-12-56-45 ";
  Match tel = r.Match (text);
  while (tel.Success)
  {
    Console.WriteLine (tel);
    tel = tel.NextMatch ();
  }
}
```

Наступний приклад дозволяє підрахувати суму цілих чисел, що зустрічаються в тексті:

```
static void Main ()
{
```

```
  Regex r = new Regex (@"[-+]? \ d + ");
  string text = @ "5 * 10 = 50 -80/40 =- 2";
  Match teg = r.Match (text);
  int sum = 0;
  while (teg.Success)
  {
    Console.WriteLine (teg);
    sum + = int.Parse (teg.ToString ());
    teg = teg.NextMatch ();
  }
  Console.WriteLine ("sum =" + sum);
}
```

Метод **Matches** класу **Regex** повертає об'єкт класу **MatchCollection** - колекцію всіх фрагментів заданого рядки, що збіглися з шаблоном. При цьому метод **Matches** багаторазово запускає метод **Match**, щоразу починаючи пошук із того місця, на якому закінчився попередній пошук.

```
  static void Main (string [] args)
  {
    string text = @ "5 * 10 = 50 -80/40 =- 2";
    Regex theReg = new Regex (@"[-+]? \ d + ");
    MatchCollection theMatches = theReg.Matches (text);
    foreach (Match theMatch in theMatches)
    {
      Console.Write ("{0}", theMatch.ToString ());
    }
    Console.WriteLine ();
  }
}
```

Редагування тексту

Регулярні вирази можуть ефективно використовуватися для редагування тексту. Наприклад, метод **Replace** класу **Regex** дозволяє виконувати заміну одного фрагмента тексту іншим або видалення фрагментів тексту.

Наприклад, зміна номерів телефонів:

```
static void Main (string [] args)
 {
 string text = @ "Контакти в Києві tel : 044-23-45-67, 044-23-34-56;
fax : 044-23-56-45.
Контакти в Чернівцях tel :037-12-34-56; fax : 037-11-56-45 ";
 Console.WriteLine ("Старі дані \ n" + text);
 string newText = Regex.Replace (text, "044 -", "037 -");
 Console.WriteLine ("Нові дані \ n" + newText);
 }
```

Видалення всіх номерів телефонів з тексту:

```
  static void Main (string [] args)
 {
 string text = @ "Контакти в Києві tel : 044-23-45-67, 044-23-34-56;
fax : 044-23-56-45.
Контакти в Чернівцях tel :037-12-34-56; fax : 037-11-56-45 ";
```

```
    Console.WriteLine ("Старі дані \ n" + text);
    string newText = Regex.Replace (text, @ "\ d {2,3} (- \ d \ d)
{2}", "");
    Console.WriteLine ("Нові дані \ n" + newText);
  }
}
```

Розбиття вихідного тексту на фрагменти:

```
static void Main (string [] args)
 {
 string text = @ "Контакти в Києві tel : 044-23-45-67, 044-23-34-56;
fax : 044-23-56-45.
Контакти в Чернівцях tel :037-12-34-56; fax : 037-11-56-45 ";
  string [] newText = Regex.Split (text, "[,.:;]+");
  foreach (string a in newText)
  Console.WriteLine (a);
}
```

Потоки. Файли.

Програми C # виконують операції введення-виведення за допомогою потоків, які побудовані на ієрархії класів. *Потік* (stream) - це абстракція, яка генерує та приймає дані. За допомогою потоку можна читати дані з різних джерел (клавіатура, файл, тощо) і записувати в різні джерела (принтер, екран, файл, тощо). Незважаючи на те, що потоки зв'язуються з різними фізичними обладнаннями, характер поведінки всіх потоків однаковий. Тому класи та методи введення-виведення можна застосувати до багатьом типам обладнань.

На найнижчому рівні ієрархії потоків уведення-виведення знаходяться, що оперують байтами. Це пояснюється тим, що багато обладнань при виконанні операцій введення-виведення орієнтовані на байти. Однак для людини привичко оперувати символами, тому розроблені символьні потоки, які фактично є оболонки, що виконують перетворення байтові потоки у символьні та навпаки. Крім цього, реалізовані потоки для роботи з `int`-, `double`-, `short`-значеннями, які також представляють оболонку для байтових потоків, але працюють не із самими значеннями, а з їхнім внутрішнім представленням у вигляді двійкових кодів.

Центральною частин потоків C# є класс **`Stream`** з простору імен **`System.IO`**. Клас **`Stream`** представляє собою потік байтів та є базовим для всіх інших потокових класів. Із класу **`Stream`** успадковуються такі байтові класи потоків як:

1. **`Filestream`** - байтовий потік, розроблений для файлового введення-виведення;
2. **`Bufferedstream`** - містить в оболонку байтовий потік і додає буферизацію, яка в багатьох випадках збільшує продуктивність програми;
3. **`Memorystream`** - байтовий потік, який використовує пам'ять для зберігання даних.

Програміст може будувати власні потокові класи. Однак для переважної більшості програм досить вбудованих потоків.

Байтовий потік

Для створення байтового потоку, пов'язаного із файлом, створюється об'єкт класу **`Filestream`**. При цьому в класі визначено декілька конструкторів. Найчастіше викори-стовується конструктор, який відкриває потік для читання й/або запису:

```
Filestream(string filename, Filemode mode)
```

де:

1. параметр `filename` визначає ім'я файлу, з яким буде зв'язаний потік введення-виведення даних; при цьому `filename` визначає або повний шлях до файлу, або ім'я файлу, який перебуває в папці bin/debug вашого проекту.

2. параметр mode визначає режим відкриття файлу, який може ухвалювати одне з можливих значень, певних перерахуванням `Filemode`:
   - `Filemode.Append` - призначений для додавання даних у кінець файлу;
   - `Filemode.Create` - призначений для створення нового файлу, при цьому якщо існує файл із таким же іменем, то він буде попередньо вилучений;
   - `Filemode.Createnew` - призначений для створення нового файлу, при цьому файл із таким же іменем не повинен існувати;
   - `FileMode.Open` - призначений для відкриття існуючого файлу;
   - `Filemode.OpenOrCreate` - якщо файл існує, то відкриває його, а якщо ні, то створює новий
   - `Filemode.Truncate` - відкриває існуючий файл, але усікає його довжину до нуля

   Якщо спроба відкрити файл виявилася неуспішної, то генерується одне з виключень:
   `Filenotfoundexception` - файл неможливо відкрити через його відсутність,
   `Ioexception` - файл неможливо відкрити через помилку введення-виведення,
   `Argumentnullexception` - ім'я файлу являє собою null-`значення`,
   `Argumentexception` - некоректний параметр mode,
   `Securityexception` - користувач не має права доступу,
   `Directorynotfoundexception` - некоректно заданий каталог.

Символьний потік

Щоб створити символьний потік потрібно помістити об'єкт класу `Stream` (наприклад, `Filestream`) "в середину" об'єкта класу `Streamwriter` або об'єкта класу `Streamreader`. У цьому випадку байтовый потік буде автоматично перетворюватися в символьний.

Клас Streamwriter призначений для організації вихідного символьного потоку. У ньому визначене трохи конструкторів. Один з них записується в такий спосіб:

```
Streamwriter(Stream stream);
```

де параметр stream визначає ім'я вже відкритого байтового потоку.

Наприклад, створити екземпляр класу `Streamwriter` можна в такий спосіб:

```
Streamwriter fileout=new Streamwriter(new Filestream("text.txt",
                                      Filemode.Create,
                                      Fileaccess.Write));
```

Цей конструктор генерує виключення типу `Argumentexception`, якщо потік stream не відкритий для виведення, і виключення типу `Argumentnullexception`, якщо він (потік) має null-значення.

Інший вид конструктора дозволяє відкрити потік відразу через звертання до файлу:

```
Streamwriter(string name);
```

де параметр name визначає ім'я файлу, що відкривається.

Наприклад, звернутися до даного конструктора можна в такий спосіб:

```
Streamwriter fileout=new Streamwriter("c:\temp\t.txt");
```

І ще один варіант конструктора `Streamwriter`:

```
Streamwriter(string name, bool appendflag);
```

де параметр `name` визначає ім'я файлу, що відкривається;

параметр `appendflag` може ухвалювати значення `true` - якщо потрібно додавати дані в кінець файлу, або `false` - якщо файл необхідно перезаписати.

Наприклад:

```
Streamwriter fileout=new Streamwriter("t.txt", true);
```

Тепер для запису даних у потік `fileout` можна звернутися до методу `Writeline`. Це можна зробити в такий спосіб:

```
fileout.Writeline("test");
```

У цьому випадку в кінець файлу `t.txt` буде дописане слово `test`.

Клас Streamreader призначений для організації вхідного символьного потоку. Один з його конструкторів виглядає в такий спосіб:

```
Streamreader(Stream stream);
```

де параметр `stream` визначає ім'я вже відкритого байтового потоку.

Цей конструктор генерує виключення типу `Argumentexception`, якщо потік `stream` не відкритий для введення.
Наприклад, створити екземпляр класу `Streamwriter` можна в такий спосіб:

```
Streamreader filein = new Streamreader(new Filestream("text.txt",
                                    Filemode.Open,
                                    Fileaccess.Read));
```

Як і у випадку із класом `Streamwriter` у класу `Streamreader` є й інший вид конструктора, який дозволяє відкрити файл прямо:

```
Streamreader (string name);
```

де параметр `name` визначає ім'я файлу, що відкривається.
Звернутися до даного конструктора можна в такий спосіб:

```
Streamreader filein=new Streamreader ("c:\temp\t.txt");
```

В C# символи реалізуються кодуванням Unicode. Для того, щоб можна було обробляти текстові файли, що містять російські символи, створені, наприклад, у Блокноті, рекомендується викликати наступний вид конструктора Streamreader:

```
Streamreader filein=new Streamreader ("c:\temp\t.txt",
                        Encoding.Getencoding(1251));
```

Параметр `Encoding.Getencoding(1251)` говорить про те, що буде виконуватися перетворення з коду Windows-1251 (одна з модифікацій коду ASCII, що містить російські символи) в Unicode. `Encoding.Getencoding(1251)` реалізований у просторі імен `System.Text`.
Тепер для читання даних з потоку `filein` можна скористатися методом `Readline`. При цьому якщо буде досягнутий кінець файлу, то метод `Readline` поверне значення null.
Розглянемо приклад, у якім дані з одного файлу копіюються в інший, але вже з використанням класів `Streamwriter` і `Streamreader`.

Двійкові потоки

Двійкові файли зберігають дані в тому ж виді, у якім вони представлені в оперативній пам'яті, тобто у внутрішній представлені. Двійкові файли не застосовуються для перегляду людиною, вони використовуються тільки для програмної обробки.

Вихідний потік `Binarywriter` підтримує довільний(прямий) доступ, тобто є можливість виконувати запис у довільну позицію двійкового файлу. Найбільш важливі методи потоку `Binarywriter`:

Член класу	Опис
Basestream	Визначає базовий потік, з яким працює об'єкт Binarywriter
Close	Закриває потік
Flush	Очищає буфер
Seek	Установлює позицію в поточному потоці
Write	Записує значення в поточний потік

Найбільш важливі методи вихідного потоку `Binaryreader`:

Член класу	**Опис**
`Basestream`	Визначає базовий потік, з яким працює об'єкт Binaryreader
`Close`	Закриває потік
`Peekchar`	Повертає наступний символ потоку без переміщення внутрішнього вказівник в потоці
`Read`	Зчитує черговий потік байтів або символів і зберігає в масиві, переданому у вхідному параметрі
`Readboolean`, `Readbyte`, `Readint32` і т.д	Зчитує з потоку дані певного типу

Двійковий потік відкривається на основі базового потоку (наприклад, `Filestream`), при цьому двійковий потік буде перетворювати байтовый потік у значення `int`-, `double`-, `short`- і т.д.

У свою чергу параметр `pos` повинен бути задано одним зі значень перерахування `Seekorigin`:

Значення	Опис
`Seekorigin.Begin`	Пошук від початку файлу
`Seekorigin.Current`	Пошук від поточної позиції покажчика
`Seekorigin.End`	Пошук від кінця файлу

Після виклику методу `Seek` наступні операції читання або записи будуть виконуватися з нової позиції внутрішнього покажчика файлу.

Приклад 1. Створити двійковий файл і записати в нього дійсні числа з діапазону від `a` до `b` із кроком `h`. Вивести на екран всі компоненти файлу з непарними порядковими номерами.

```
using System;
using System.Text;
using System.IO;

namespace Myprogram
{
 class Program
 {
  static void Main()
  {
    Console.Write("a= ");
    double a=double.Parse(Console.Readline());
    Console.Write("b= ");
    double b=double.Parse(Console.Readline());
    Console.Write("h= ");
    double h=double.Parse(Console.Readline());
    //Записуємо у файл t.dat дійсні числа із заданого діапазону
    Filestream f=new Filestream("t.dat",Filemode.Open);
    Binarywriter fout=new Binarywriter(f);
    for (double i=a; i<=b; i+=h)
    {
     fout.Write(i);;
    }
    fout.Close();
    //Об'єкти f і fin пов'язані з тим самим файлом
    f=new Filestream("t.dat",Filemode.Open);
    Binaryreader fin=new Binaryreader(f);
    long m=f.Length; //визначаємо кількість байт у потоці
    //Читаємо дані з файлу t.dat починаючи з елемента з номером 1, т.е з 8 байта,
    //переміщаючи внутрішній покажчик на 16 байт, тобто на два дійсні числа
    for (long i=8; i<m; i+=16)
    {
     f.Seek(i,Seekorigin.Begin);
     a=fin.Readdouble();
     Console.Write("{0:f2} ",a);
    }
    fin.Close();
    f.Close();
   }
 }
}
```

Приклад 2. Даний текстовий файл. Знайти кількість рядків, які починаються з даної букви. **Ф**айл повинен містити англійський текст.

```
using System;
using System.Text;
using System.IO;
using System.Text.Regularexpressions;
```

```
namespace Myprogram
{
 class Program
 {
  static void Main()
  {
    Console.Write("Уведіть задану букву: ");
    char a=char.Parse(Console.Readline());
    Streamreader filein = new Streamreader("text.txt");
    string text=filein.Readtoend(); //зчитуємо з файлу весь текст
    filein.Close();
    int k=0;
    //розбиваємо текст на слова використовуючи регулярні вираження
    string []newtext=Regex.Split(text,"[,.:;]+");
    //підраховуємо кількість слів, що починаються на задану букву
    foreach( string b in newtext)
     if (b[0]==a)++k;
    Console.Writeline("k= "+k);
   }
 }
}
```

Робота з файловою системою.

У просторі імен `System.IO` передбачено чотири класи, які призначені для роботи з файловою системою комп'ютера, т.е для створення, видалення переносу і т.д. файлів і каталогів. Перші два типи - `Directory` і `Fi1e` реалізують свої можливості за допомогою статичних методів, тому дані класи можна використовувати без створення відповідних об'єктів (екземплярів класів). Наступні типи - `Directoryinfo` і `Fileinfo` мають схожі функціональні можливості с `Directory` і `Fi1e`, але породжені від класу `FileSystemInfo` і тому реалізуються шляхом створення відповідних екземплярів класів.

Абстрактний клас FileSystemInfo

Значна частина членів `Filesysteminfo` призначена для роботи із загальними характеристиками файлу або каталогу (мітками часу, атрибутами й т.п.). Розглянемо деякі властивості `Filesysteminfo`:

Властивість	Опис
`Attributes`	Дозволяє одержати або встановити атрибути для даного об'єкта файлової системи. Для цієї властивості використовуються значення й перерахування Fileattributes
`Creationtime`	Дозволяє одержати або встановити час створення об'єкта файлової системи
`Exists`	Може бути використане для того, щоб визначити, чи існує даний об'єкт файлової системи
`Extension`	Дозволяє одержати розширення для файлу
`Fullname`	Повертає ім'я файлу або каталогу із вказівкою шляху до нього у файловій системі
`Lastaccesstime`	Дозволяє одержати або встановити час останнього звертання до об'єкта файлової системи
`Lastwritetime`	Дозволяє одержати або встановити час останнього внесення змін в об'єкт файлової системи
`Name`	Повертає ім'я зазначеного файлу. Ця властивість доступна тільки для читання. Для каталогів повертає ім'я останнього каталогу в ієрархії, якщо це можливо. Якщо ні, повертає повністю певне ім'я

В `Filesysteminfo` передбачене й кілька методів. Наприклад, метод Delete() - дозволяє вилучити об'єкт файлової системи з жорсткого диска, а `Refresh()` - обновити інформацію про об'єкт файлової системи.

Клас Directoryinfo

Даний клас успадковує члени класу `Filesysteminfo` і містить додатковий набір членів, які призначені для створення, переміщення, видалення, одержання інформації про каталоги й

підкаталоги у файловій системі. Найбільш важливі члени класу втримуються в наступній таблиці:

Член	Опис
`Create()` `Createsubdirectory()`	Створюють каталог (або підкаталог) по зазначеному шляхові у файловій системі
`Delete()`	Видаляє порожній каталог
`Getdirectories()`	Дозволяє одержати доступ до підкаталогів поточного каталогу (у вигляді масиву об'єктів `Directoryinfo`)
`Getfiles()`	Дозволяє одержати доступ до файлів поточного каталогу (у вигляді масиву об'єктів `Fileinfo`)
`Moveto()`	Переміщає каталог і весь його вміст на нову адресу у файловій системі
`Parent`	Повертає батьківський каталог в ієрархії файлової системи

Робота з типом `Directoryinfo` починається з того, що ми створюємо екземпляр класу (об'єкт), указуючи при виклику конструктора як параметра шлях до потрібного каталогу. Якщо ми прагнемо звернутися до поточного каталогу (тобто каталогу, у якім у цей час проводиться виконання додатка), замість параметра використовується позначення `"."`. Наприклад:

```
// Створюємо об'єкт Directoryinfo, якому буде звертатися до поточного каталогу
Directoryinfo dir1 = new Directoryinfo(".");
// Створюємо об'єкт Directoryinfo, якому буде звертатися до каталогу d:\prim
Directoryinfo dir2 = new Directoryinfo(@"d:\prim");
```

Якщо ми спробуємо створити об'єкт `Directoryinfo`, зв'язавши його з неіснуючим каталогом, то буде сгенерировано виключення `System.IO.DirectoryNotFoundException`. Якщо ж усі нормально, то ми зможемо одержати доступ до даного каталогу. У прикладі, який наведений нижче, ми створюємо об'єкт `Dlrectoryinfo`, який пов'язаний з каталогом `d:\prim`, і виводимо інформацію про даний каталог:

Клас Directory

Працювати з каталогами файлової системи комп'ютера можна й за допомогою класу `Directory`, функціональні можливості якого багато в чому збігаються з можливостями `Directoryinfo`. Але члени даного класу реалізовані статично, тому для їхнього використання немає необхідності створювати об'єкт.

Розглянемо роботу з методами даного класу на прикладах.

Зауваження. Вилучите з диска `d` змінену папку `prim`. І ще раз скопіюйте її вихідну версію з розділу 12 даного електронного підручника.

```
Directory.Createdirectory(@"d:\prim\2020");//створили підкаталог 2020
Directory.Move(@"d:\prim\bmp",
        @"d:\prim\2020\bmp");//перенесли каталог bmp у каталог 2020
Directory.Move(@"d:\prim\letter",
        @"d:\prim\archives");//перейменували каталог letter в archives
```

Зауваження.

1. Видалення каталогу можливо тільки тоді, коли він порожній.
2. На практиці комбінують використання класів `Directory` і `Directoryinfo`.

Робота з файлами

Клас Filelnfo

Клас `Filelnfo` призначений для організації доступу до фізичного файлу, який утримується на жорсткому диску комп'ютера. Він дозволяє одержувати інформацію про цей файл (наприклад, про час його створення, розмірі, атрибутах і т.п.), а також робити різні операції, наприклад, по створенню файлу або його видаленню. Клас `Fileinfo` успадковує члени класу `Filesysteminfo` і містить додатковий набір членів, який наведений у наступній таблиці:

Член	Опис
`Appendtext()`	Створює об'єкт Streamwriter для додавання тексту до файлу
`Copyto()`	Копіює вже існуючий файл у новий файл

`Create()`	Створює новий файл і повертає об'єкт `Filestream` для взаємодії із цим файлом
`Createtext()`	Створює об'єкт `Streamwriter` для запису текстових даних у новий файл
`Delete()`	Видаляє файл, якому відповідає об'єкт `Fileinfo`
`Directory`	Повертає каталог, у якому розташований даний файл
`Directoryname`	Повертає повний шлях до даного файлу у файловій системі
`Length`	Повертає розмір файлу
`Moveto()`	Переміщає файл у зазначене користувачем місце (цей метод дозволяє одночасно перейменувати даний файл)
`Name`	Дозволяє одержати ім'я файлу
`Open()`	Відкриває файл із зазначеними користувачем правами доступу на читання, запис або спільне використання з іншими користувачами
`Openread()`	Створює об'єкт `Filestream`, доступний тільки для читання
`Opentext()`	Створює об'єкт `Streamreader` (про нього також буде розказано нижче), який дозволяє зчитувати інформацію з існуючого текстового файлу
`Openwrite()`	Створює об'єкт `Filestream`, доступний для читання й запису

Як ми бачимо, більшість методів `Fileinfo` повертає об'єкти (`Filestream`, `Streamwriter`, `Streamreader` і т.п.), які дозволяють різним образом взаємодіяти з файлом, наприклад, робити читання або запис у нього. Приймання роботи з даними потоками нам уже відомі. Тому розглянемо інші можливості класу `Fileinfo`.

Завдання до лабораторної роботи:

Порядок виконання роботи:

1) Клонування репозиторія C#: https://classroom.github.com/a/soLbqpEn
2) Написати C# програми згідно з варіантом завдання. При вирішенні завдання передбачити:
   - пошук ;
   - підрахунок кількості входжень;
   - пошук із заміною на нове значення.
3) Підготувати звіт в електронному виді надіслати мудл(https://moodle.chnu.edu.ua/course/view.php?id=3371) .

```
//  За бажанням студента для задач можна створювати консольний проект або WinForm
// Бажано для задач лаб. робіт створити окремі класи
// Виконання  виконати в стилі багатозаданості :
//   Lab8T2  lab8task2 = new Lab8T2; lab8task2.Run();
// При бажанні можна створити багатозадачний режим виконання задач.
```

Завдання 1. Варіанти задач. Задано текстовий файл. У файлі рядки в яких міститься осмислене текстове повідомлення. Слова повідомлення розділяються пробілами та розділовими знаками Записати у новий файл всі підтексти заданого формату, підрахувати їх кількість, вилучити та замінити деякі з них, за вказаними параметрами користувача.

1.1. У тексті може міститися дата в форматі **дд.мм.рррр**. Де **дд** - ціле число з діапазону від **1** до **31**, **мм** - ціле число з діапазону від **1** до **12**, а **рррр** - ціле число з діапазону від **1900** до **2099** (якщо якась

частина формату порушена, то даний підрядок в якості дати не розглядається).

1.2. У тексті можуть міститися IP-адреси комп'ютерів в форматі **d.d.d.d**, де **d** - ціле число з діапазону від **0** до **255**.

1.3. У тексті можуть міститися IP-адреси комп'ютерів в форматі **d.d.d.d**, де **d** - ціле шіснацяткове число з діапазону від 0 до FF.

1.4. У тексті можуть міститися IP-адреси комп'ютерів в форматі **d.d.d.d**, де **d** - ціле двійкове число з діапазону від 0 до 11111111.

1.5. У тексті можуть міститися адреси web-сайтів домена com.

1.6. У тексті можуть міститися адреси web-сайтів домена edu.ua.

1.7. У тексті можуть міститися адреси web-сайтів домена cv.ua.

1.8. У тексті можуть міститися електронні адреси.

1.9. У тексті можуть міститися електронні адреси gmail.

1.10. У тексті можуть міститися електронні адреси ukrnet.

1.11. У тексті може міститися час у форматі **гг: хх: сс**. У заданому форматі **гг** - ціле число з діапазону від **00** до **24**, **хх** і **сс** - цілі числа з діапазону від **00** до **60** (якщо якась частина формату порушена, то дана підрядок в якості дати не розглядається).

1.12. У тексті може міститися дата- час в форматі **рррр.мм.дд:гг:хх**. Де **дд** - ціле число з діапазону від **1** до **31**, **мм** - ціле число з діапазону від **1** до **12**, а **рррр** - ціле число з діапазону від **1900** до **2099**, **гг** - ціле число з діапазону від **00** до **24**, а **хх** - ціле число з діапазону від **00** до **60** (якщо якась частина формату порушена, то даний підрядок в якості дати не розглядається).

1.13. У тексті може міститися географічні координати у форматі

1.14. У тексті може міститися координати вектора у форматі

1.15. У тексті може міститися координати вектора у форматі

Завдання 2. Варіанти задач. Задано текстовий файл. Записати у новий файл результат задачі.

2.1. Визначте, чи міститься у тексті задане слово.

2.2. Виведіть всі слова заданої довжини.

2.3. Видаліть всі однобуквені слова та слова які починаються буквами: 'a', 'b', 'c', 'd' і 'e' .

2.4. Видаліть всі знаки пунктуації та цифри.

2.5. Видаліть із повідомлення тільки ті українські слова, які починаються на голосну літеру.

2.6. Видаліть із повідомлення тільки ті українські слова, які починаються на голосну літеру

2.7. Замінити всі англійські слова на три крапки.

2.8. Знайти максимальне ціле число, що зустрічається в тексті.

2.9. Замінити всі шістнадцяткові цифри ('0', '1', ..., '9', 'a'-'f') на знак '+';

2.10. Замінити послідовність букв в алфавітному порядку на скорочений запис (наприклад: abcdf -> a-f);
2.11. Замінити скорочений запис на послідовність букв (наприклад: c-e -> cdfe, k-t -> klmnoprst);
2.12. Видаліть слова, які мають префікс "re", "not" та "be" та замінє слова, які мають префікс "не" на "not".
2.13. Видаліть слова, які мають закінчення "re", "nd" та "less" та замінє слова, які мають префікс "to" на "at".
2.14. Видаліть всі ідентифікатори, де ідентифікатор – будемо називати послідовність букв латиського алфавіту, символу підкреслення та цифр, яка починається з букв латиського алфавіту або з символу підкреслення.
2.15. Видаліть всі не ідентифікатори де ідентифікатор – будемо називати послідовність букв латиського алфавіту, символу підкреслення та цифр, яка починається з букв латиського алфавіту або з символу підкреслення.

Завдання 3. Варіанти задач. Задано текстовий файл. Записати у новий файл результат задачі.

3.1. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає з цього тексту всі слова з подвоєнням літер і записує їх в окремий рядок, розділяючи пробілами. Друкує окремо вилучені слова і текст, що залишився після вилучення слів.
3.2. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає з цього тексту всі слова найбільшої довжини. (Слів найбільшої довжини може бути декілька). Друкує текст, що залишився після вилучення слів.
3.3. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі симетричні слова (наприклад, слово абввба є симетричним).
3.4. Задано два тексти, слова в яких розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із першого тексту всі слова, що містяться у другому тексті.
3.5. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає в кожному слові цього тексту всі наступні входження першої літери.
3.6. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає в кожному слові цього тексту всі попередні входження останньої літери.

3.7. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає з цього тексту всі повторні входження слів.

3.8. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і вилучає всі слова, що входять в цей текст по одному разу.

3.9. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка в словах непарної довжини цього тексту вилучає середню літеру.

3.10. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі слова, що входять у заданий текст по одному разу.

3.11. Задано текст, слова в якому розділені пробілами і розділовими знаками, та два окремих слова. Розробити програму, яка замінює всі входження в заданий текст першого слова другим словом.

3.12. Задано два тексти, слова в яких розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із другого тексту всі входження слів першого тексту.

3.13. Задано два тексти, слова в яких розділені пробілами і розділовими знаками. Розробити програму, яка створює третій текст із слів першого тексту, які не входять у другий текст, розділяючи їх пробілами.

3.14. Задано два тексти, слова в яких розділені пробілами і розділовими знаками, та окреме слово. Розробити програму, яка після кожного входження заданого слова в перший текст вставляє в нього другий текст.

3.15. Задано символ і текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі слова, що містять заданий символ найбільшу кількість разів.

3.16. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує найдовший ланцюжок із слів однакової довжини.

3.17. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із заданого тексту всі слова непарної довжини.

3.18. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує слово з

найбільшою кількістю однакових символів (якщо таких слів декілька, то взяти перше з них).

3.19. Задано текст із малих латинських літер, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі слова з літерами, розміщеними в лексикографічному порядку.

3.20. Задано два тексти, слова в яких розділені пробілами і розділовими знаками. Розробити програму, яка створює третій текст із слів першого тексту, які входять у другий текст, і розділяє їх пробілами.

3.21. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із заданого тексту всі попередні входження останнього слова.

3.22. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із кожного слова заданого тексту всі повторні входження кожної літери.

3.23. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає з цього тексту всі слова з повторенням літер.

3.24. Задано текст із малих латинських букв, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі слова, букви в яких розміщені в алфавітному порядку.

3.25. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і вилучає всі слова, літери в яких розміщені в лексикографічному порядку.

3.26. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із кожного слова заданого тексту всі попередні входження останньої літери.

3.27. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі слова, буква ’A‘ або a’ в яких зустрічається найбільшу кількість разів.

3.28. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка знаходить і друкує всі слова, букви в яких не повторюються.

3.29. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає у кожному слові цього тексту всі повторні входження кожної букви.

3.30. Задано текст, слова в якому розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із цього тексту всі слова, які починаються з голосної букви.

Завдання 4. Варіанти задач. **Робота із двійковими файлами.**

4.1. Створити файл і записати в нього степені числа 3. Вивести на екран всі компоненти файлу з парним порядковим номером.

4.2. Створити файл і записати в нього зворотні натуральні числа `1,1/2,...,1/n`. Вивести на екран всі компоненти файлу з порядковим номером, кратним 3.

4.3. Створити файл і записати в нього `n` перших членів послідовності Фібоначчі. Вивести на екран всі компоненти файлу з порядковим номером, не кратним 3.

4.4. Дана послідовність із `n` цілих чисел. Створити файл і записати в нього всі парні числа послідовності. Вивести вміст файлу на екран.

4.5. Дана послідовність із `n` цілих чисел. Створити файл і записати в нього всі додатні числа послідовності. Вивести вміст файлу на екран.

4.6. Дана послідовність із `n` цілих чисел. Створити файл і записати в нього числа послідовності, що попадають у заданий інтервал. Вивести вміст файлу на екран.

4.7. Дана послідовність із `n` цілих чисел. Створити файл і записати в нього числа послідовності, не кратні заданому числу. Вивести вміст файлу на екран.

4.8. Дана послідовність із `n` дійсних чисел. Записати всі ці числа у файл. Вивести на екран всі компоненти, що не попадають у даний діапазон.

4.9. Дана послідовність із `n` дійсних чисел. Записати всі ці числа у файл. Вивести на екран всі компоненти файлу з непарними номерами, більші заданого числа.

4.10. Дана послідовність із `n` дійсних чисел. Записати всі ці числа у файл. Вивести на екран всі компоненти файлу з парними номерами, менші заданого числа.

4.11. Дана послідовність із `n` дійсних чисел. Записати всі ці числа у файл. Вивести на екран усі позитивні компоненти файлу.

4.12. Дана послідовність із `n` дійсних чисел. Записати всі ці числа у файл. Підрахувати середнє арифметичне компонентів файлу, що розташовуються на парних позиціях.

4.13. Дана послідовність із `n` дійсних чисел. Записати всі ці числа у файл. Знайти максимальне значення серед компонентів файлу, що розташовуються на непарних позиціях.

4.14. Дана пропозиція. Створити файл і записати в нього всі символи даного пропозиції, відмінні від розділових знаків. Вивести вміст файлу на екран.
4.15. Дана пропозиція. Створити файл і записати в нього всі символи даного пропозиції, відмінні від цифр. Вивести вміст файлу на екран.
4.16. Створити файл, що складається зі слів. Вивести на екран усі слова, які починаються на задану букву.
4.17. Створити файл, що складається зі слів. Вивести на екран усі слова, довжина яких дорівнює заданому числу.
4.18. Створити файл, що складається зі слів. Вивести на екран усі слова, які починаються й закінчуються однієї буквою.
4.19. Створити файл, що складається зі слів. Вивести на екран всі слова, які починаються на ту ж букву, що й останнє слово.

Задача 5. Для всіх. Текст у кутових дужка замінити відповідним чином**.**

Завдання. Програмним шляхом:

1. У папці d:\temp створіть папки <прізвище_студента>1 і <прізвище_студента>2.
2. У папці <прізвище_студента>1:
    1. створіть файл t1.txt, у який запишіть наступний текст :
    2. *<Шевченко Степан Іванович, 2001>* року народження, місце проживання *<м. Суми>*
    3. створіть файл t2.txt, у який запишіть наступний текст:
    4. *<Комар Сергій Федорович, 2000 >* року народження, місце проживання <м. Київ>
3. У папці <прізвище_студента>2 створіть файл t3.txt, у який перепишіть спочатку текст із файлу t1.txt, а потім з t2.txt
4. Виведіть розгорнуту інформацію про створені файли.
5. Файл t2.txt перенесіть у папку < прізвище_студента>2.
6. Файл t1.txt скопіюйте в папку < прізвище_студента>2.
7. Папку K2 перейменуйте в ALL, а папку < прізвище_студента>1 вилучите.
8. Вивести повну інформацію про файли папки All.

Контрольні питання

1) Що розуміється під терміном файл?
2) Тип файлів.

3) Класи роботи із файловою системою.