**Лабораторна робота № 9.**

*Тема: Файли* ***.***

**Мета роботи:** Розробка програм із використанням файлів**.**

**Теоретичні відомості**

**Виняткові ситуації**

Виняток це помилка, що відбувається під час виконання програми. За допомогою підсистем обробки винятків для С# можна обробляти такі помилки, не викликаючи краху програми. Обробка винятків у С# виконується застосуванням ключових слів: **try**, **catch**, **throw** і **finally**. Ці ключові слова утворюють взаємозалежну підсистему, у якій використання одного із ключових слів спричиняє використання інших. Основа обробки винятку заснована на використанні блоків **try,** **catch** і **finally**.

**try** {

*Блок \_ коду \_ для \_ якого \_ виконується \_ моніторинг \_ помилок*

**catch** (Exceptypel ех) {

*Оброблювач \_ винятків \_ Exceptypel* }

**catch** (*ЕхсерТуре2 ех*) {

*Оброблювач \_ винятків \_ ЕхсерТуре2* }

**finally** {

Код блоку finally. }

Основні системні винятків наведені в наступній таблиці:

|  |  |
| --- | --- |
| Винятки | Значення |
| **Arraytypemismatchexception** | Тип збереженого значення несумісний з типом масиву. |
| **Dividebyzeroexception** | Почата спроба розподілу на нуль. |
| **Indexoutofrangeexception** | Індекс масиву виходить за межі діапазону. |
| **Invalidcastexception** | Некоректне перетворення в процесі виконання. |
| **Outofmemoryexception** | Виклик new був невдалим через недостатності пам'яті. |
| **Overflowexception** | Переповнення при виконанні арифметичної операції. |
| **Stackoverflowexception** | Переповнення стека. |

Тип винятків в операторові **catch** повинен відповідати типу винятків, що перехоплюється. Не перехоплене винятків неодмінне приводить до дострокового припинення виконання програми. Для виконання перехоплення винятку незалежно від їхнього типу (перехоплення всіх винятку) можливе використання **catch** без параметрів. Тому що оператор **catch** не викликається із програми, то після виконання блоку **catch** керування **не передається** назад операторові програми, при виконанні якого виник виняток. Виконання програми продовжується з операторів, що перебувають після блоку **catch**. З метою запобігання цієї ситуації можлива вказівка блоку коду, який викликається після виходу із блоку **try/catch**, за допомогою блоку f**inally** наприкінці послідовності **try/catch**. Блок **finally** буде викликатися незалежно від того, з'явиться винятків чи ні, і незалежно від причин виникнення такого.

Винятків автоматично генеруються системою. Однак винятків може бути згенеровано за допомогою оператора **throw**.

**throw** exceptob;

Винятків, перехоплене одним оператором **catch**, може генеруватися повторно, завдяки чому воно може перехоплюватися зовнішнім оператором **catch**. Для цього вказується ключове слово **throw** без імені винятків.

Можна створювати замовлені винятків, що виконують обробку помилок у користувацькому коді. Генерування винятку не представляє особливих складностей. Просто визначите клас, успадкований з класу **Exception**. У якості загального правила необхідно керуватись тим, що задані користувачем винятки успадковуються із класу **АрplicationException**, тому що вони є ієрархію зарезервованих винятку, пов'язаних з додатками. Успадковані класи не мають потреби у фактичній реалізації в якому-небудь виді, оскільки саме їхнє існування в системі типів даних дозволяє скористатися ними в якості винятку. Створювані користувачем класи винятку автоматично одержують доступні для них властивості й методи, заданих в класі **Exception**.

**Потоки. Файли.**

Програми **С#** 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 С# 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;
   * **FileMоde.Open** - призначений для відкриття існуючого файлу;
   * **Filemode.ОpenOrCreate** - 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 **Streamwrite**r 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 байтовий 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 -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 вираз**

**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 **Fi1е** 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 c **Directory** і **Fi1е**, 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 **Fileattribute**s |
| **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, a 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 шляху в 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 в 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.

**Directory.Createdirectory(@"d:\prim\2018");**//створили підкаталог 2018

**Directory.Move(@"d:\prim\bmp", @"d:\prim\2018\bmp");**

//перенесли каталог **bmp** у каталог 2018

**Directory.Move(@"d:\prim\letter", @"d:\prim\archives");**

//перейменували каталог **letter** в **archives**

#### 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 |
| **Ореn()** | 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 |

Зауважимо, що 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. Реалізувати програму мовою **C#** з обробкою виняткових ситуацій відповідно до варіанта. При роботі з файлами, зі вмістом файлів та обчисленнях передбачити обробку виняткових ситуацій. Для обробки винятків побудувати класи, які будуть спадкоємцями(нащадками) класу **Exception**.
2. Підготувати звіт у твердій копії та в електронному виді.

**Задача 1. Робота із двійковими файлами.**

1. Створити файл і записати в нього степені числа 3. Вивести на екран всі компоненти файлу з парним порядковим номером.
2. Створити файл та записати в нього зворотні натуральні числа 1,1/2,...,1/n. Вивести на екран всі компоненти файлу з порядковим номером, кратним 3.
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. Дана послідовність із n цілих чисел. Створити файл і записати в нього всі парні числа послідовності. Вивести вміст файлу на екран.
5. Дана послідовність із n цілих чисел. Створити файл і записати в нього всі додатні числа послідовності. Вивести вміст файлу на екран.
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.
7. Дана послідовність із **n** цілих чисел. Створити файл і записати в нього числа послідовності, не кратні заданому числу. Вивести вміст файлу на екран.
8. Дана послідовність із **n** дійсних чисел. Записати всі ці числа у файл. Вивести на екран всі компоненти, що не попадають у даний діапазон.
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.
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.
11. Дана послідовність із **n** дійсних чисел. Записати всі ці числа у файл. Вивести на екран усі позитивні компоненти файлу.
12. Дана послідовність із **n** дійсних чисел. Записати всі ці числа у файл. Підрахувати середнє арифметичне компонентів файлу, що розташовуються на парних позиціях.
13. Дана послідовність із **n** дійсних чисел. Записати всі ці числа у файл. Знайти максимальне значення серед компонентів файлу, що розташовуються на непарних позиціях.
14. Дана пропозиція. Створити файл і записати в нього всі символи даного пропозиції, відмінні від розділових знаків. Вивести вміст файлу на екран.
15. Дана пропозиція. Створити файл і записати в нього всі символи даного пропозиції, відмінні від цифр. Вивести вміст файлу на екран.
16. Створити файл, що складається зі слів. Вивести на екран усі слова, які починаються на задану букву.
17. Створити файл, що складається зі слів. Вивести на екран усі слова, довжина яких дорівнює заданому числу.
18. Створити файл, що складається зі слів. Вивести на екран усі слова, які починаються й закінчуються однієї буквою.
19. Створити файл, що складається зі слів. Вивести на екран всі слова, які починаються на ту ж букву, що й останнє слово.

**Задача 2. Робота з текстовим (символьним) файлом.**

1. Даний текстовий файл. Знайти кількість рядків, які починаються й закінчуються однієї буквою.
2. Даний текстовий файл. Знайти самий довгий рядок і її довжину.
3. Даний текстовий файл. Знайти самий короткий рядок і її довжину.
4. Даний текстовий файл. Знайти номер самого довгого рядка.
5. Даний текстовий файл. Знайти номер самого короткого рядка.
6. Даний текстовий файл. З'ясувати, є чи в ньому рядок, який починається з даної букви. Якщо так, то надрукувати його.
7. Даний текстовий файл. Надрукувати перший символ кожного рядка.
8. Даний текстовий файл. Надрукувати символи з **k1** по **k2** у кожному рядкові.
9. Даний текстовий файл. Надрукувати всі непарні рядки.
10. Даний текстовий файл. Надрукувати всі рядки, у яких є хоча б один пробіл.
11. Даний текстовий файл. Надрукувати всі рядки, довжина яких дорівнює даному числу.
12. Даний текстовий файл. Надрукувати всі рядки, довжина яких менше заданого числа.
13. Даний текстовий файл. Надрукувати всі рядки з номерами від **k1** до **k2**.
14. Даний текстовий файл. Одержати слово, утворене **k**-тими символами кожного рядка.
15. Даний текстовий файл. Переписати в новий файл всі його рядка, вставивши в кінець кожного рядка її номер.
16. Даний текстовий файл. Переписати в новий файл всі його рядка, вставивши в кінець кожного рядка кількість символів у ній.
17. Даний текстовий файл. Переписати в новий файл всі його рядка, довжина яких більше заданого числа.
18. Даний текстовий файл. Переписати в новий файл всі його рядка парної довжини.
19. Даний текстовий файл. Переписати в новий файл всі його рядка, вилучивши з них символи, що коштують на парних місцях.

**Задача 3.** Для всіх.Текст у кутових дужка замінити відповідним чином**.**

Звадання. 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 Андрій Іванович, 1990*> року народження, місце проживання <*м. Київ*>
   3. створіть файл t2.txt, у який запишіть наступний текст:
   4. *<Петренко Сергій Федорович, 1991* > року народження, місце проживання м. Чернівці
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 < прізвище\_студента>2 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.