# Лабораторна робота №3

**Тема:** Рядки та робота з файлами у C#.

**Мета:** Вивчити особливості обробки символьних даних та файлів засобами ООП.

**Завдання:** Здобути навички створення, відлагодження та тестування проектів обробки символьних даних та файлів засобами ООП мовою ***C#***.

**(max: 5 балів)**

## Теоретичні відомості:

Для обробки текстової інформації в C# є широкий набір засобів: окремі символи, масиви символів, змінні і незмінні рядки і регулярні вирази.

### 1. Тип char. Масиви типу char

Символьний тип даних char представляє один символ. Він призначений для зберігання символів у кодуванні Unicode. Символьний тип відноситься до вбудованих типів даних C# і відповідає стандартному класу Char бібліотеки .NET з простору імен System. У цьому класі визначені статичні методи, що дозволяють задати вигляд і категорію символу, а також перетворити символ у верхній або нижній регістр і в число. Деякі корисні методи наведено в таблиці 1.

**Таблиця 1. Деякі методи класу System.Char**

| Метод | Опис |
|---|---|
| GetNumericValue | Повертає числове значення символу, якщо він є цифрою, і –1 інакше |
| IsControl | Повертає true, якщо символ є управляючим |
| IsDigit | Повертає true, якщо символ є десятковою цифрою |
| IsNumber | Повертає true, якщо символ є цифрою |
| IsLetter | Повертає true, якщо символ є буквою |
| IsLower | Повертає true, якщо символ заданий в нижньому регістрі |
| IsUpper | Повертає true, якщо символ записаний у верхньому регістрі |
| IsWhiteSpace | Повертає true, якщо символ є пробілом (пропуск, перехід до нового рядка і повернення каретки) |
| Parse | Перетворює рядок в символ (рядок повинен складатися з одного символу) |
| ToLower | Перетворює символ в нижній регістр |
| MaxValue, MinValue | Повертають символи з максимальним і мінімальним кодами (ці символи не мають видимого представлення) |

**Приклад 1. Використання методів класу System.Char**

```
using System;
namespace ConsoleApplication1
{  class Class1
   {  static void Main()
     {
        try
        {
           char b = 'B', c = '\x63', d = '\u0032';                    // 1
           Console.WriteLine( "{0}  {1}  {2}", b, c, d );
           Console.WriteLine( "{0}  {1}  {2}",
                char.ToLower(b), char.ToUpper(c), char.GetNumericValue(d) );
           char a;
           do                                  // 2
           {
               Console.Write( "Введіть символ: " );
               a = char.Parse( Console.ReadLine() );
               Console.WriteLine( "Введено символ {0}, його  код - {1}",
                                  a, (int)a );
```

```
                if (char.IsLetter(a))      Console.WriteLine("Літера");
                if (char.IsUpper(a))       Console.WriteLine("Верхній рег.");
                if (char.IsLower(a))       Console.WriteLine("Нижній рег.");
                if (char.IsControl(a))     Console.WriteLine("Управляючий");
                if (char.IsNumber(a))      Console.WriteLine("Число");
                if (char.IsPunctuation(a)) Console.WriteLine("Роздільник");
            } while (a != 'q');
        }
        catch
        {
            Console.WriteLine( "Виникло виключення" );
            return;
        }
    }
  }
}
```

**Приклад 2.** Вдосконалення консольного калькулятора (дивитись приклад в лаб 2). Метод `IsNumeric` перевіряє чи правильно введене число. У рядку `NumericText` знаходиться число, введене з консолі.

```
static bool IsNumeric(string NumericText)
        {
            //перевірка чи у рядку число
            bool isnumber = true;
            foreach (char c in NumericText)
            {
                isnumber = char.IsNumber(c);
                if (!isnumber)
                {
                    if (c != ',')
                        return isnumber;
                }
            }
            return isnumber;
        }
```

Масив символів, як і масив будь-якого іншого типу побудований на основі базового класу `System.Array`.

**Приклад 3**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
          char[] a = { 'm', 'a', 's', 's', 'i', 'v' };      // 1
          char[] b = "через тиждень у нас перший кредит".ToCharArray(); // 2
          PrintArray( "масив a:", a );
          int pos = Array.IndexOf( a, 'm' );
          a[pos] = 'M';
          PrintArray( "Змінений масив a:", a );
          PrintArray( "Масив b:", b );
          Array.Reverse( b );
          PrintArray( "Змінений масив b:", b );
```

```
            Console.ReadKey();
        }
        public static void PrintArray( string header, Array a )
        {
            Console.WriteLine( header );
            foreach ( object x in a ) Console.Write( x );
            Console.WriteLine( "\n" );
        }
    }
    }
```

В цьому прикладі створюється два масиви типу char: a,b. Зверніть увагу, що в ініціалізаторі масиву b використовуються явне перетворення рядка в масив символів викликом методу ToCharArray().

Далі викликається метод PrintArray( "масив a:", a ). В цьому методі елементи масиву виводяться на консоль. Використовується цикл foreach.

В наступних операторах спочатку знаходиться індекс символу 'm' в масиві a, який замінюється символом M, і знову викликається метод PrintArray( "Змінений масив a:", a );

Після цього викликається PrintArray( "Масив b:", b ); для друку масиву b. Потім викликається метод Array.Reverse( b ), який переставляє всі символу масиву у зворотньому порядку. Знову викликається метод PrintArray( "Змінений масив b:", b ); для виводу обернего масиву b.

**2. Робота з рядками. Клас String і його методи**

Тип string, призначений для роботи з рядками символів в кодуванні Unicode, є вбудованим посилковим типом C#. Йому відповідає базовий клас System.String бібліотеки .NET.

Створити рядок можна декількома способами:

```
string s;                                   // відкладена ініціалізація
string t = "qqq";                    // ініціалізація строковим літералом
string u = new string(' ', 20); // конструктор створює рядок з 20 пробілів
char[] a = { '0', '0', '0' };         // масив для ініціалізації рядка
string v = new string( a );     // створення з масиву символів
```

Для рядків визначені наступні операції:

- призначення (=);
- перевірка на рівність (==);
- перевірка на нерівність (!=);
- звернення по індексу ([]);
- конкатенація рядків (+).

Не дивлячись на те, що рядки є посилковим типом даних, на рівність і нерівність перевіряються не посилання, а значення рядків. Рядки рівні, якщо мають однакову кількість символів і збігаються посимвольно.

Звертатися до окремого елементу рядка за індексом можна лише для отримання значення, але не для його зміни. Це пов'язано з тим, що рядки типу string відносяться до так званих незмінних типів даних. Методи, що змінюють вміст рядка, насправді створюють нову копію рядка. Невживані "старі" копії автоматично видаляються при збірці сміття.

В базовому класі **System.String** реалізовано багато різних методів, що виконують операції над рядками. Основні властивості і методи класу String перелічені у табл. 2.

| **Таблиця 2. Деякі властивості і методи класу System.String** | |
|---|---|
| `Length` | Властивість. Дозволяє отримати кількість символів в рядку. |
| `Concat()` | Дозволяє з'єднати декілька рядків або змінних типу object. |
| `CompareTo()` | Дозволяє порівняти два рядки. В разі рівності рядків результат виконання функції дорівнює нулю. При позитивному значенні функції більшим є рядок, для якого викликався метод. |
| `Copy()` | Створює нову копію існуючого рядка. |
| `Format()` | Застосовується для форматування рядка з використанням різних примітивів (рядків і числових даних) і підстановлювальних виразів вигляду {0}. |
| `Insert()` | Дозволяє вставити один рядок всередину існуючого. |

| | |
|---|---|
| `Remove()`<br>`Replace()` | Видаляють або замінюють символи в рядку. |
| `ToUpper()`<br>`ToLower()` | Перетворюють всі символи рядка в строкові або прописні. |
| `Chars` | Дозволяє отримати символ, що знаходиться в певній позиції рядка. |
| `Join()` | Створює рядок, сполучаючи задані рядки і розділяючи їх рядком-роздільником. |
| `Replace()` | Замінює один символ рядка іншим. |
| `Split()` | Повертає масив рядків з елементами - підрядками основного рядка, між якими знаходяться символи-роздільники. |
| `Substring()` | Дозволяє отримати підрядок основного рядка, заданої довжини, що починається з певного символу. |
| `Trim()` | Видаляє пропуски або набір заданих символів на початку і кінці основного рядка. |
| `ToCharArray()` | Створює масив символів і вставляє в нього символи початкового рядка. |

При роботі з рядками в C# необхідно враховувати таке. Тип string є *посилковим* типом. Проте, не дивлячись на це, при використанні операцій порівняння відбувається порівняння *значень рядкових об'єктів*, а не адрес цих об'єктів, розміщених в оперативній пам'яті. Крім того, оператор "+" об'єкту string перевизначений тому при його використанні застосовується метод Concat().

**Приклад 4**

```
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         string s = "Чого бажає моя королева";
         Console.WriteLine(s);
         string sub = s.Substring(3).Remove(2, 6);            // 1
         Console.WriteLine(sub);

         string[] mas = s.Split(' ');                    // 2
         string joined = string.Join("! ", mas);
         Console.WriteLine(joined);

         Console.WriteLine("Задайте рядок");
         string x = Console.ReadLine();                  // 3
         Console.WriteLine("Ви надрукували рядок " + x);
         double a = 12.234;
         int b = 29;
         Console.WriteLine(" a = {0,6:C}  b = {1,2:X}", a, b);   // 4
         Console.ReadKey();
      }
   }
}
```

**Результат роботи програми:**

```
Чого бажає моя королева
о моя королева
Чого! Бажає! Моя! Королева!
Задайте рядок
не хочу!
Ви надрукували рядок не хочу!
a = 12,23р.  b = 1D
```

**Форматування рядків:** У операторі 4 цього прикладу неявно застосовується метод `Format`, який замінює всі входження параметрів у фігурних дужках значеннями відповідних змінних із списку виводу. Після номера параметра можна задати мінімальну ширину поля виводу, а також вказати специфікатор формату, який визначає форму представлення значення, що виводиться.

У загальному вигляді параметр задається таким чином:

```
{n [,m[:специфікатор_форматування]]}
```

Тут n — номер параметра. Параметри нумеруються з нуля, нульовий параметр замінюється значенням першої змінної із списку виводу, перший параметр — другою змінною, і так далі Параметр m визначає мінімальну ширину поля, яке відводиться під значення, що виводиться. Якщо числу, що виводиться, досить меншої кількості позицій, невживані позиції заповнюються пробілами. Якщо числу потрібно більше позицій, параметр ігнорується.

Специфікатор форматування визначає формат виведення значення. Наприклад, специфікатор C (Currency) означає, що параметр повинен форматуватися як валюта з урахуванням національних особливостей представлення, а специфікатор X (Hexadecimal) задає шістнадцяткову форму представлення значення, що виводиться.

Досить часто при роботі з рядками виникає необхідність розділити рядок на підрядки, відокремлені один від одного заданими символами-роздільниками. Виділення лексем використовується в задачах лексичного аналізу в усіх компіляторах. У наступному прикладі задається рядок символів, в якому є декілька символів-роздільників. За допомогою функції **Split** рядок розділяється на підрядки, які потім виводяться на екран кожний в окремому рядку. Для завдання символів-роздільників використовується масив символів. В цьому прикладі також застосовується функція **Trim**, необхідна для того, щоб переконатися в тому, що заданий рядок не складається з одних лише пробілів.

**Приклад 5**

```
static void Main(string[] args)
        {
          string words = "рядок, що містить декілька слів, а також знаки пунктуації: такі як двокрапка і крапка.";
          string [] split = words.Split(new Char [] {' ', ',', '.', ':'});
          foreach (string s in split)
          {
           if (s.Trim() != "")
           Console.WriteLine(s);
          }
          Console.ReadKey();
        }
```

В прикладі спочатку створюється рядок **words** і масив **split**, в який записуються виділені слова. Метод **Split** повертає масив рядків з елементами - підрядками основного рядка, між якими знаходяться символи-роздільники. Ці символи ми розмістили в масиві типу Char. Далі в циклі для кожного слова викликається метод Trim(), який видаляє пробільні символи з рядка. В нашому прикладі оператор

```
if (s.Trim() != "")
```

перевіряє чи виділене слово не є пробілом. Якщо слово не пробіл, воно виводиться на консоль.

**Приклад 6.** Вдосконалення консольного калькулятора. Програма вводить одразу арифметичний вираз, наприклад, 2+3. Необхідно розбити рядок на окремі частини (лексичний аналіз). Фрагмент коду може бути таким:

```
.....................................................................................
.........
      string str;      //Рядки для розбору виразу
      string[] arrStr = new string[2];
      string op = "";
      Console.WriteLine("Введіть арифметичний вираз");
      str = Console.ReadLine();
      //аналіз вхідного виразу
      string[] split = str.Split(new Char[] { ' ', '*', '+', '-', '/' });
      int i = 0;
      foreach (string s in split)
       {
         if (s.Trim() != "")
          {
```

```
            Console.WriteLine(s);
            arrStr[i] = s;
            i++;
           }
        }
       //пошук операції
      char[] array0p = new char[4] { '+', '-', '*', '/' };
      foreach (char c in array0p)
      {
        int indexop = str.IndexOf(c);
        if (indexop != -1)
         {
           op = str[indexop].ToString();
           break;
         }
}
```

### 3. Клас StringBuilder і його методи

Іноді слід уникати ситуацій, коли в результаті виконання операції створюється новий рядок, оскільки це пов'язано з додатковими витратами пам'яті та інших ресурсів комп'ютера при виконанні операції.

C# містить спеціальний клас StringBuilder, використовуючи який можна уникнути створення копій рядків при їх обробці. Всі зміни, що вносяться до об'єкту даного класу, негайно відображаються в ньому, що ефективніше, ніж робота з копіями рядка.

Основною операцією, яка найчастіше використовується класом **StringBuilder**, є операція додавання до рядка вмісту. Для цього існує метод **Append**. Наступний код додає один рядок до іншого і виводить результат на консоль. При цьому змінюється оригінал рядка, копія не створюється:

```
StringBuilder sb = new StringBuilder("Наступного тижня у нас кредит");
sb.Append(", потрібно захистити лаб.роботи");
Console.Write(sb);
```

Окрім додавання клас **StringBuilder** містить інші методи, найбільш значимі з яких наведені нижче. Після того, як усі необхідні дії, пов'язані з обробкою рядка, були виконані, необхідно викликати метод ToString() для перетворення вмісту об'єкту в звичайний тип даних string.

| **Таблиця 3. Деякі методи класу StringBuilder** | |
|---|---|
| Append | Додавання заданого рядка в кінець рядка об'єкту. |
| AppendFormat | Додавання заданого форматованого рядка (рядка, що містить управляючі символи) в кінець рядка об'єкту. |
| CopyTo | Копіювання символів заданого сегменту рядка в задані комірки масиву символів. |
| Insert | Додавання рядка в задану позицію рядка об'єкту. |
| Remove | Видалення заданої кількості символів з рядка об'єкту |
| Replace | Заміна заданого символу або рядка об'єкту на інший заданий символ або рядок. |

При інтенсивній роботі з рядками рекомендується використовувати клас **StringBuilder**, оскільки це дозволяє зменшити накладні витрати, пов'язані із створенням копії рядка при виконанні кожної операції.

### 4. Обробка помилок. Клас Exception.

Мова C#, як і багато інших об'єктно-орієнтованих мов, реагує на помилки і ненормальні ситуації за допомогою виклику виключень (exceptions) – програмних переривань.

**Виключення** - це об'єкт, що містить інформацію про незвичайний програмний випадок.

Коли у програмі виникає виняткова ситуація, вона викликає (throws) виключення. Після виклику виключення виконання поточної функції переривається і стек звільняється, поки не буде знайдений відповідний обробник виключення.

*Обробник виключення (exception handler) - це програмний блок, призначений для реагування на виклик виключення. Обробники виключень реалізуються в операторах try...catch.*

Якщо в тексті функції є фрагмент, який має бути виконаний незалежно від того, чи було викликано яке-небудь виключення (наприклад, фрагмент, що звільняє виділені програмі ресурси), то його можна помістити в блок *finally*, де цей фрагмент програми напевно буде виконаний навіть після виклику виключення.

У мові C# виключеннями можуть виступати лише об'єкти типу System. Exception або об'єкти похідного від нього типа.

Виключення генерує або середовище виконання, або програміст за допомогою оператора throw.

Простір імен System середовища CLR включає цілий ряд типів виключень, якими можна користуватися в програмі.

**Таблиці 4. Часто використовувані стандартні виключення**

| **Ім'я** | **Опис** |
|---|---|
| `DivideByZeroException` | Спроба ділення на нуль |
| `FormatException` | Спроба передати в метод аргумент невірного формату |
| `IndexOutOfRangeException` | Індекс масиву виходить за границі діапазону |
| `InvalidCastException` | Помилка перетворення типу |
| `OutOfMemoryException` | Недостатньо пам'яті для створення нового об'єкту |
| `OverFlowException` | Переповнення при виконанні арифметичних операцій |
| `StackOverFlowException` | Переповнення стеку |

Виключення виявляються і обробляються в операторі **try**.

Оператор try містить три частини:

- контрольований блок — складений оператор, перед яким стоїть ключове слово **try**. У контрольований блок включаються потенційно небезпечні оператори програми;
- один або декілька обробників виключень — блоків catch, в яких описується, як обробляються помилки різних типів;
- блок завершення finally виконується незалежно від того, виникла помилка в контрольованому блоці чи ні.

Синтаксис оператора try:

```
try блок [ блоки catch ] [ блок finally ]
```

Бути відсутніми можуть або блоки catch, або блок finally, але не обидва одночасно.

Розглянемо, яким чином реалізується обробка виключень.

1. Обробка виключення починається з появи помилки. Функція або операція, в якій виникла помилка, генерує виключення.
2. Виконання поточного блоку припиняється, відшукується відповідний обробник виключення, і йому передається управління.
3. Виконується блок finally, якщо він присутній.
4. Якщо обробник не знайдений, викликається стандартний обробник виключення. Зазвичай він виводить на екран вікно з інформацією про виключення і завершує поточний процес.

Обробники виключень повинні розташовуватися безпосередньо за блоком **try**. Вони починаються з ключового слова **catch**, за яким в дужках вказується тип оброблюваного виключення. Можна записати один або декілька обробників відповідно до типів оброблюваних виключень. Блоки **catch** є видимими в тому порядку, в якому вони записані, поки не буде знайдений відповідний тип викинутого виключення.

Існують три форми запису обробників:

```
catch( тип ім'я ) { ... /* тіло обробника */ }
```

```
catch( тип )      { ... /* тіло обробника */ }
catch             { ... /* тіло обробника */ }
```

Перша форма застосовується, коли ім'я параметра використовується в тілі обробника для виконання яких-небудь дій.

Друга форма не передбачає використання інформації про виключення, грає роль лише його тип.

Третя форма застосовується для перехоплення всіх виключень. Оскільки обробники є видимими в тому порядку, в якому вони записані, обробник третього типу (він може бути лише один) слід розміщати після всіх інших.

**Приклад 7:**

```
try {
    ...  // Контрольований блок
}
catch ( OverflowException e ) {
    ...    // Обробка виключень класу OverflowException (переповнення)
}
catch ( DivideByZeroException ) {
    ... // Обробка виключень класу DivideByZeroException (ділення на 0)
}
catch {
    ...    // Обробка всіх інших виключень
}
```

Якщо виключення в контрольованому блоці не виникло, всі обробники пропускаються.

В будь-якому разі, сталося виключення чи ні, управління передається в блок завершення **finally** (якщо він існує), а потім — першому оператору, який знаходиться безпосередньо за оператором try.

Оператори try можуть багато разів вкладатися один в одного. Виключення, яке виникло у внутрішньому блоці try і не було перехоплене відповідним блоком catch, передається на верхній рівень, де продовжується пошук відповідного обробника. Цей процес називається *поширенням виключення.*

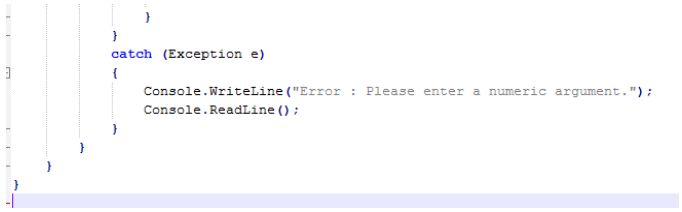
Клас System. Exception має ряд корисних методів і властивостей. Зокрема, властивість **Message** надає інформацію про виключення, наприклад про причини його виникнення. Властивість Message доступна лише для читання, а програма, що викликає виключення, може передати цю властивість конструктору виключення як аргумент.

В цьому прикладі в методі Power(int x, int n) обчислюється $x^n$. Значення x і n вводяться з консолі. Для контролю введених даних використовується блок try... catch.

**Приклад 8:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Lab5
{
    class Program
    {
        static public double Power(int x, int n)
        {
            double z = 1;
            for (int i = 1; n >= i; i++)
            {
                z = z * x;
            }
            return z;
        }
        static void Main(string[] args)
        {
            int x;
            int n;
            try
            {
                Console.WriteLine("Enter x:");
                x = Convert.ToInt32(Console.ReadLine());
                if ((x >= 0) & (x <= 999))
                {
                    Console.WriteLine("Enter n:");
                    n = Convert.ToInt32(Console.ReadLine());
                    if ((n >= 1) & (n <= 100))
                    {
                        Console.WriteLine("The power n of x is {0}",
                        Power(x, n));
                        Console.ReadLine();
                    }
                    else
                    {
                        Console.WriteLine("Error : n must be in [1..100]");
                        Console.ReadLine();
```



```
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("Error : Please enter a numeric argument.");
            Console.ReadLine();
        }
    }
}
}
```

**Оператор throw.** Щоб проінформувати про ненормальну ситуацію, що виникла в якому-небудь класі C#, програміст викликає виключення, користуючись ключовим словом throw, наприклад:

```
throw new System.Exceptlon():
```

Виклик виключення відразу ж припиняє виконання програми, а середовище CLR тим часом шукає процедуру обробки виключення. Якщо обробник виключення відсутній в поточному методі, стек викликів звільняється. Якщо CLR пройде стек аж до Main(), не знайшовши обробник викликаного виключення, виконання програми завершується.

### 5. Комплексний приклад. Вдосконалений консольний калькулятор

Створення консольного калькулятора на 4 дії. Обчислюваний вираз вводиться одним рядком. Виконується розбір рядка і перевірка правильності його введення. У разі виявлення помилки викликається обробник помилки.

```
namespace ConsoleCalculator
{///Реалізувати  метод обчислення виразу a X b, де a, b – числа,
    ///X={+,-,*,/}. Вираз ввести з консолі, результат вивести на консоль.
    ///При виборі операції використати оператор switch.
    class Program
    {
        static bool IsNumeric(string NumericText)
        {
            //перевірка чи у рядку число
            bool isnumber = true;
            foreach (char c in NumericText)
            {
                isnumber = char.IsNumber(c);
                if (!isnumber)
                {
                    if (c != ',')
                        return isnumber;
                }
            }
            return isnumber;
        }
        static double ConvertDouble(string NumericText)
        {
            double digit;
            //конвертація і округлення числа
            digit = double.Parse(NumericText);
            digit = Math.Round(digit, 2);
            return digit;
        }

        static void Main(string[] args)
        {
            string str;     //Рядки для розбору виразу
            string[] arrStr = new string[2];
            string op = "";
            double digit1, digit2, result;      //числа
            do
            {
                Console.WriteLine("Введіть арифметичний вираз");
                str = Console.ReadLine();
                //аналіз вхідного виразу
```

```
                try
                {
                    if (str.Length == 0)
                    {
                        throw new Exception("Нічого не введено! Введіть правильний
вираз.");
                    }
                    string[] split = str.Split(new Char[] { ' ', '*', '+', '-', '/'
});
                    int i = 0;
                    foreach (string s in split)
                    {
                        if (s.Trim() != "")
                        {
                            Console.WriteLine(s);
                            arrStr[i] = s;
                            i++;
                        }
                    }

                    //пошук операції
                    char[] array0p = new char[4] { '+', '-', '*', '/' };
                    foreach (char c in array0p)
                    {
                        int indexop = str.IndexOf(c);
                        if (indexop != -1)
                        {
                            op = str[indexop].ToString();
                            break;
                        }
                    }
                    if (op.Length == 0)
                    {
                        throw new Exception("Операція не знайдена у виразі");
                    }
                    //перевірка чисел

                    if (!IsNumeric(arrStr[0]))
                    {
                        throw new Exception("Перший член виразу - не число");

                    }
                    digit1 = ConvertDouble(arrStr[0]);

                    if (!IsNumeric(arrStr[1]))
                    {
                        throw new Exception("Перший член виразу - не число");
                    }
                    digit2 = ConvertDouble(arrStr[1]);

                    switch (op)
                    {
                        case ("+"):
                            result = digit1 + digit2;
                            break;
                        case ("-"):
                            result = digit1 - digit2;
                            break;
                        case ("/"):
                            result = digit1 / digit2;
                            break;
                        case ("*"):
                            result = digit1 * digit2;
                            break;
                        default:
                            throw new Exception("Невідома операція");
```

```
                    }//switch

                    // Console.WriteLine("s1={0},op={1},s2={2}", s1, op, s2);
                    Console.WriteLine("аргумент1={0},операція={1},аргумент2={2}",
digit1, op, digit2);
                    Console.WriteLine("Результат= " + result);
                    Console.ReadKey();
                }
                catch (Exception e)
                {
                    Console.WriteLine("Помилка у вхідних даних" + e.Message);
                    Console.ReadLine();
                }
                Console.WriteLine("Для завершення роботи введіть exit");
            } while (Console.ReadLine() != "exit");
        }
    }
}
```

**6. Введення-виведення даних на консоль**

Будь-яка програма при введенні даних і виведенні результатів взаємодіє із зовнішніми пристроями. Сукупність стандартних пристроїв введення і виведення, тобто клавіатури і екрану, називається **консоллю**. У мові C# немає операторів введення і виведення. Замість них для обміну із зовнішніми пристроями застосовуються стандартні об'єкти. Для роботи з консоллю в C# застосовується клас `Console`, визначений у просторі імен System.

Для організації консольного введення і виведення використовуються статичні методи цього класу: `Console.WriteLine` та `Console.Write`, наприклад:

```
Console.WriteLine(“Hello, World!”);
  Console.Write(“Hello, World!”);
```

Метод **WriteLine** відрізняється від **Write** тим, що завершує свою роботу обов'язковим виведенням Escape-послідовності line feed/carriage return, тобто, переходом на новий рядок.

**Приклад 9.**

```
using System;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            int     i = 3;
            double  y = 4.12;
            decimal d = 600m;
            string  s = "Вася";

            Console.WriteLine( "i = " + i );
// 1
            Console.WriteLine( "s = " + s );
// 2
            Console.WriteLine( "y = {0} \nd = {1}", y, d );    // 3
        }
    }
}
```

Результат роботи програми:

i = 3
s = Вася
y = 4,12
d = 600

В цьому прикладі використовується два варіанти виклику методу `WriteLine`. Якщо метод WriteLine викликаний з одним параметром, він може бути будь-якого вбудованого типу. У рядку,

поміченому коментарями "1" і "2", нам потрібно вивести в кожному рядку не одну, а дві величини, тому перш ніж передавати їх для виводу, їх потрібно "склеїти" в один рядок за допомогою операції +.

Перед об'єднанням рядка з числом треба перетворити число з його внутрішньої форми представлення в послідовність символів, тобто в рядок. Перетворення в рядок визначене у всіх стандартних класах C# — для цього є метод ToString(). В даному випадку він виконується неявно.

Оператор 3 ілюструє вивід формату. В цьому випадку використовується інший варіант методу WriteLine. Першим параметром методу передається рядковий літерал, що містить, окрім звичайних символів, призначених для виводу на консоль, параметри у фігурних дужках, а також управляючі послідовності. Параметри нумеруються з нуля, перед виводом вони замінюються значеннями відповідних змінних в списку виводу.

З управляючих послідовностей найчастіше використовуються символи переходу на новий рядок (\n) і горизонтальної табуляції (\t).

Розглянемо прості способи введення з клавіатури. У класі `Console` визначені методи введення рядка і окремого символу, але немає методів, які дозволяють безпосередньо читати з клавіатури числа. Введення числових даних виконується в два етапи:

1. Символи, що є числом, вводяться з клавіатури в змінну-рядок.
2. Виконується перетворення з рядка в змінну відповідного типу.

Перетворення можна виконати або за допомогою спеціального класу Convert, визначеного в просторі імен System, або за допомогою методу Parse, наявного в кожному стандартному арифметичному класі. У прикладі 10 використовуються обидва способи.

**Приклад 10.**

```
using System;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            Console.WriteLine( "Введіть рядок" );
            string s = Console.ReadLine();                          // 1
            Console.WriteLine( "s = " + s );

            Console.WriteLine( "Введіть символ" );
            char c = (char)Console.Read();                          // 2
            Console.ReadLine();                                     // 3
            Console.WriteLine( "c = " + c );

            string buf;            // рядок - буфер для введення чисел

            Console.WriteLine( "Введіть ціле число" );
            buf = Console.ReadLine();
            int i = Convert.ToInt32( buf );                         // 4
            Console.WriteLine( i );

           Console.WriteLine( "Введіть дійсне число" );
            buf = Console.ReadLine();
            double x = Convert.ToDouble( buf );                     // 5
            Console.WriteLine( x );

            Console.WriteLine( "Введіть дійсне число" );
            buf = Console.ReadLine();
            double y = double.Parse( buf );                         // 6
            Console.WriteLine( y );

            Console.WriteLine( "Введіть дійсне число" );
            buf = Console.ReadLine();
            decimal z = decimal.Parse( buf );                       // 7
            Console.WriteLine( z );
        }
    }
}
```

Введення рядка виконується в операторі 1. Довжина рядка не обмежена, введення виконується до символу кінця рядка.

Введення символу виконується за допомогою методу Read, який читає один символ з вхідного потоку (оператор 2). Метод повертає значення типу int, що є кодом символу, або –1, якщо символів у вхідному потоці немає (наприклад, користувач натиснув клавішу Enter). Оскільки нам потрібно не int, а char, а неявного перетворення від int до char не існує, доводиться застосувати операцію явного перетворення типу.

Оператор 3 читає залишок рядка і нікуди його не передає. Це необхідно тому, що введення даних виконується через буфер — спеціальну область оперативної пам'яті. Фактично дані спочатку заносяться в буфер, а потім читаються звідти процедурами введення. Занесення в буфер виконується по натисненні клавіші Enter разом з її кодом. Метод Read, на відміну від ReadLine, не очищає буфер, тому наступне після нього введення виконуватиметься з того місця, на якому закінчилося попереднє.

В операторах 4 і 5 використовуються методи класу Convert, в операторах 6 і 7 — методи Parse класів Double і Decimal бібліотеки .NET, які використовуються тут через імена типів C# double і decimal.

**Увага**

*При введенні дійсних чисел дробова частина відділяється від цілої за допомогою коми, а не крапки.*

Якщо символи, що вводяться з клавіатури, не можна інтерпретувати як дійсне число, генерується виключення.

### 7. Файлове введення-виведення даних

Основні особливості і правила роботи з пристроями введення/виведення в сучасних мовах високого рівня описуються в рамках **класів потоків**. Для мов платформи .NET місцем опису загальних властивостей потоків є клас System.IO.Stream.

Простір імен System.IO не підключається до збірки автоматично, тому його потрібно імпортувати.

```
using System.IO;
```

Класи цього простору імен дозволяють виконувати введення і виведення даних в **потоки (Stream) і файли (File).**

У бібліотеці FCL окрім класів **File** і **Stream**, що описують файли і потоки, є багато інших класів, пов'язаних з цими поняттями - **FileStream, StreamWriter**, **StreamReader, FileInfo** і інші класи.

Статичний клас **File** надає різні методи для роботи з файлами, основні з яких наведено в таблиці 5 (всі методи класу є статичними).

**Таблиця 5.** Методи файлового введення-виведення

| Метод | Призначення |
|---|---|
| FileStream **Create**(string path) | Повертає файл - об'єкт класу FileStream. Вхідний аргумент path задає шлях до файлу. Можна вказати лише ім'я файлу, тоді за замовчанням передбачатиметься поточна директорія. Якщо вказаний файл не існує, то файл буде створений, інакше існуючий файл буде відкритий для запису за умови, що для файлу не заданий атрибут доступу «лише для читання». Новий файл має атрибут доступу «ReadWrite» (читання і запис) і не дозволяє діставати доступ до нього іншим користувачам, поки файл не буде закритий. |
| StreamWriter **CreateText**(string path) | Створює новий або відкриває для запису існуючий текстовий файл з кодуванням UTF-8. |
| FileStream **Open**(string path, FileMode mode) | Повертає файл - об'єкт класу FileStream. Метод перегружений, його додаткові аргументи такі самі, як і в методі Create. Існуючий файл буде відкритий відповідно до режиму, заданого параметром FileMode. Можливі значення FileMode:<br>**Create** - якщо файл вже існує, то буде відкритий, інакше буде створений новий файл;<br>**CreateNew**- якщо файл не існує, то буде створений, інакше виникне виключення;<br>**Open** - якщо файл існує, то він буде відкритий, інакше виникне виключення;<br>**OpenOrCreate** - якщо файл існує, він буде відкритий, інакше буде |

| | |
|---|---|
| | створений;<br>**Truncate** - якщо файл існує, він буде відкритий, а його вміст буде видалений, інакше виникне виключення. |
| **Append** | якщо файл існує, то він відкритий для запису в кінець файлу, якщо файлу не існує, то буде створений новий файл, відкритий для запису. |
| StreamReader<br>**OpenText**(string path) | відкриває для читання існуючий текстовий файл. Якщо вказаного файлу немає, то виникне виключення |
| FileStream<br>**OpenRead**(string path) | відкриває існуючий файл для читання, повертаючи об'єкт класу FileSream. Виняткова ситуація виникає, якщо вказаного файлу немає. Метод **OpenWrite** виконує ті самі дії, але відкриває файл для запису. |
| StreamWriter<br>**AppendText**(string path) | аналогічний за своєю дією методу CreateText за тим виключенням, що при відкритті існуючого файлу відбувається не переписування його, а додавання нових записів в кінець файлу. |
| Методи Copy, Replace, Move, Delete, Exist | дозволяють копіювати файл, заміняти його вміст, переміщувати, видаляти і визначити, чи існує файл з вказаним іменем. |
| Read | Група методів Read дозволяє відкрити існуючий файл, прочитати його вміст і закрити файл. |
| Write | Група методів Write дозволяє писати у файл, виконуючи операції обернені читанню. |

### 8. Потоки. Абстрактний клас Stream і його нащадки

Класи потоків введення/виведення в Framework .NET ґрунтуються на абстрактному класі **Stream**. Призначення цього класу полягає в оголошенні загального стандартного набору операцій (стандартного інтерфейсу), що забезпечують роботу з пристроями введення/виведення, незалежно від їх конкретної реалізації, від джерел і одержувачів інформації.

При цьому класи конкретних потоків забезпечують власну реалізацію інтерфейсів цього абстрактного класу.

Нащадками класу Stream є, зокрема, три класи байтових потоків:

- **BufferedStream** – забезпечує буферизацію байтового потоку. Як правило, буферизовані потоки є продуктивнішими ніж не буферизовані;
- **FileStream** – байтовий потік, що забезпечує файлові операції введення/виведення;
- **MemoryStream** – байтовий потік, що використовує як джерело і сховище інформації оперативну пам'ять.

Робота з потоками передбачає СПРЯМОВАНІСТЬ виконуваних дій. Інформацію з потоку можна ПРОЧИТАТИ, а можна її в потік ЗАПИСАТИ. Як читання, так і запис передбачають реалізацію певних механізмів байтового обміну з пристроями.

В рамках Framework .NET, незалежно від характеристик того або іншого пристрою введення/виведення, програміст ЗАВЖДИ може визначити:

- чи можна читати з потоку – bool CanRead (якщо можна – значення має бути встановлене в true);
- чи можна писати в потік – bool CanWrite (якщо можна – значення має бути встановлене в true);
- чи можна задати в потоці поточну позицію – bool CanSeek (якщо послідовність, в якій виконується читання/запис, не є жорстко детермінованою і можливе позиціювання в потоці – значення має бути встановлене в true);
- позицію поточного елементу потоку – long Position (можливість позиціювання в потоці передбачає можливість програмної зміни значення цієї властивості);
- загальна кількість символів потоку (довжину потоку) – long Length.

Це основні властивості класу **Stream.**

Відповідно до загальних принципів реалізації операцій введення/виведення, для роботи з потоком є набір методів (табл.6):

**Таблиця 6.** Методи роботи з потоками

| Метод | Призначення |
|---|---|
| int ReadByte(); | читання байта з потоку з поверненням цілочисельного представлення наступного доступного байта в потоці введення |
| int Read(byte[] buff, int index, int count); | читання певної (count) кількості байтів з потоку і розміщення їх в масиві buff, починаючи з елементу buff[index], з поверненням кількості успішно прочитаних байтів |
| void WriteByte(byte b); | запис в потік одного байта |
| int Write(byte[] buff, int index, int count); | запис в потік певної (count) кількості байтів з масиву buff, починаючи з елементу buff[index], з поверненням кількості успішно записаних байтів |
| long Seek (long index, SeekOrigin origin) | позиціювання в потоці (позиція поточного байта в потоці задається значенням зміщення index відносно позиції origin); |
| void Close() | закриття потоку |

### 9. Класи StreamReader, StreamWriter, StringReader, StringWriter

Ці класи лежать в основі роботи з текстовими файлами і дозволяють працювати з потоком як з текстовим файлом, читаючи і записуючи в потік дані різних скалярних типів. Класи Reader є нащадками абстрактного класу TextReader, класи Writer – нащадками класу TextWriter. Класи StringReader і StringWriter дозволяють розглядати рядок як потік.

- **StreamReader** – містить властивості і методи, що забезпечують читання даних з текстового файлу як пототу байтів.
- **StreamWriter** – містить властивості і методи, що забезпечують запис символів в байтовий потік.
-

**Таблиця 7.** Основні методи класу StreamReader

| Метод | Призначення |
|---|---|
| `int Read()`<br>`int Read (char[] buff, int index, int count)` | має дві реалізації. Перша повертає черговий символ, прочитаний з потоку, пересуваючи покажчик потоку вправо. Друга реалізація дозволяє прочитати задане число символів з потоку в масив, переданий як аргумент методу Read |
| `int ReadBlock(char[] array, int offset, int count)` | дозволяє прочитати задане число символів з потоку в масив |
| `int Peek()` | читає наступний символ, не змінюючи стан покажчика потоку. Повертає наступний доступний символ, не прочитуючи його насправді з потоку вхідних даних |
| `string ReadLine()` | читає і повертає черговий рядок тексту з потоку. Це означає, що читаються символи потоку, поки не зустрінуться символи кінця рядка або кінця файлу. Якщо в потоці символів немає і досягнутий кінець файлу, то в результаті повертається null – посилання на відсутнє значення. |
| `string ReadToEnd()` | читає всі символи з потоку, що залишилися, аж до кінця файлу |
| `void Close()` | закриває вхідний потік |

**Таблиця 8.** Основні методи класу StreamWriter

| Метод | Призначення |
|---|---|
| `Write` | має різні реалізації для виведення різних типів даних. |
| `WriteLine (...)` | виводить у потік дані, розділяючи їх символами символи кінця рядка |
| `void Close()` | закриває потік |

**Приклад 11.** В цьому прикладі спочатку створюється об'єкт f класу StreamWriter, який містить файл для виводу output.txt. Далі визначаються і ініціалізуються змінна int i та string s, які записуються у вихідний файл. Метод Close() закриває файл.

```
using System.IO;                                        // 1
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            StreamWriter f = new StreamWriter( "output.txt" );      // 2
            int     i = 3;
            string  s = "Мова програмування C#";
            f.WriteLine( "i = " + i );                              // 3
            f.WriteLine( "s = " + s );                              // 4
            f.Close();                                              // 5
        }
    }
}
```

Таким чином, щоб використовувати в програмі файли, необхідно:

1. Підключити простір імен, в якому описуються стандартні класи для роботи з файлами (оператор 1).
2. Оголосити файлову змінну і зв'язати її з файлом (оператор 2).
3. Виконати операції введення-виводу (оператори 3–4).
4. Закрити файл (оператор 5).

**Приклад 12.**

Введення даних з файлу виконується аналогічно. У наступному прикладі наведено програму, в якій введення виконується з файлу з іменем input.txt, розташованому в каталозі D:\C#.

```
using System;
using System.IO;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            StreamReader f = new StreamReader( "d:\\C#\\input.txt" );
            string s = f.ReadLine();
            Console.WriteLine( "s = " + s );
            char  c = (char)f.Read();
            f.ReadLine();
            Console.WriteLine( "c = " + c );
            string buf;
            buf = f.ReadLine();
            int i = Convert.ToInt32( buf );
            Console.WriteLine( i );
            buf = f.ReadLine();
            double x = Convert.ToDouble( buf );
            Console.WriteLine( x );
            buf = f.ReadLine();
            double y = double.Parse( buf );
            Console.WriteLine( y );
            buf = f.ReadLine();
            decimal z = decimal.Parse( buf );
            Console.WriteLine( z );
            f.Close();
        }
    }
}
```

**Приклад 13.** Збереження даних в тестовому файлі.

```
//створюємо об'єкт myfile класу FileInfo - це файл, в який будемо виводи
//властивість Application.StartupPath - шлях до програми в період виконання
System.IO.FileInfo myfile = new FileInfo(Application.StartupPath +
"\\tests.txt");
```

```
//потік для запису StreamWriter. Створюємо об'єкт sw
System.IO.StreamWriter sw;
//якщо файл з таким іменем існує, видаляємо його
if (myfile.Exists)
  {
    myfile.Delete();
  }
//якщо файлу немає - створюємо його і відкриваємо потік для запису
sw = myfile.CreateText(); //створюємо потік, який записує текст в новий файл
sw.Write(txtOut.Text);   //метод Write записує рядок у файл
sw.Close();              //метод Close закриває файл
```

**Приклад 14.** Читання даних з текстового файлу

```
using System;
using System.IO;
public class TextFromFile
{
    private const string FILE_NAME = "MyFile.txt";
    public static void Main(String[] args)
    {
        if (!File.Exists(FILE_NAME))
        {
            Console.WriteLine("{0} does not exist.", FILE_NAME);
            return;
        }
        using (StreamReader sr = File.OpenText(FILE_NAME))
        {
            String input;
            while ((input=sr.ReadLine())!=null)
            {
                Console.WriteLine(input);
            }
            Console.WriteLine ("The end of the stream has been
reached.");
            sr.Close();
        }
    }
```

**Індивідуальні завдання:**

(Для виконання індивідуальних завдань № варінта є порядковим номером прізвища студента в списку групи. Усі проекти завантажити у власні репозиторії на Git Hub та виконати їх збірку/build у середовищі Travis CI. Посилання на репозиторії проектів вказати у звіті та обов'язково долучати скріни успішної збірки в Travis CI.)

***1. З клавіатури вводиться текстовий рядок. Розробити консольний застосунок, який реалізує вказані дії.***

***(2 бала)***

1. а) видаляє кожне друге слово; б) знищує всі слова, які починаються і закінчуються за одну й ту ж літеру.
2. а) підраховує кількість цифр у тексті; б) виводить на екран слова, що починаються з приголосних літер.
3. а) підраховує кількість слів у тексті, які закінчуються на голосну літеру; б) виводить на екран всі слова, довжина яких менша п'яти символів.
4. а) видаляє всі слова, які містять хоча б одну латинську літеру; б) видаляє всі числа з тексту.
5. а) підраховує кількість слів у тексті, які починаються з голосної літери; б) виводить на екран всі слова, що мають непарну кількість приголосних літер.

6. а) замінює всі великі літери, що входять до тексту на відповідні малі; б) виводить на екран найдовше слово.
7. а) видаляє всі слова, що містять непарну кількість приголосних літер; б) видаляє з тексту всі слова-паліндроми.
8. а) кількість слів, які містять однакову кількість голосних і приголосних літер; б) виводить на екран найдовше слово.
9. а) виводить всі символи, які розташовані після першого символу ":"; б) підраховує кількість речень, що містять непарну кількість слів.
10. а) видаляє з тексту всі слова, які розташовані після ком. б) видаляє всі слова, передостання літера яких голосна.
11. а) підраховує кількість слів у кожному реченні; б) виводить на екран найдовше речення.
12. а) інвертує рядок, подаючи його у зворотному вигляді; б) підраховує кількість чисел у тексті.
13. а) видаляє всі слова, що починаються з голосних літер б) підраховує кількість слів у тексті.
14. а) знищує всі слова, які починаються і закінчуються за одну й ту ж літеру; б) підраховує кількість різних слів, що входять до заданого тексту.
15. а) замінює всі великі літери, що входять до тексту на відповідні малі; б) виводить всі слова, що мають парну кількість літер.
16. а) підраховує кількість слів, які мають непарну довжину; б) виводить на екран частоту входження кожної літери.
17. а) перевіряє, чи співпадає кількість відкритих і закритих дужок у введеному рядку (перевірити для круглих та квадратних дужок); б) видаляє текст, що розміщено в круглих дужках.
18. а) виводить на екран найдовше слово; б) видаляє всі слова, що складаються тільки з латинських літер.
19. а) підраховує кількість різних слів, що входять до заданого тексту; б) виводить на екран кількість використаних символів.
20. а) видаляє всі слова, що мають подвоєні літери; б) підраховує кількість слів у тексті.
21. а) виводить на екран слово, що містить найбільшу кількість голосних літер; б) видаляє з тексту всі непотрібні пробіли.
22. а) підраховує кількість розділових знаків у тексті; б) виводить всі слова, що мають парну кількість літер.
23. а) міняє місцями першу і останню літери кожного слова; б) видаляє всі слова, що починаються з малої літери.
24. а) підраховує кількість великих літер у тексті; б) виводить на екран слова, що мають найменшу кількість літер.
25. а) підраховує кількість чисел у тексті (не цифр, а саме чисел); б) виводить на екран всі слова, що складаються тільки з латинських літер.

***2. Створити програми для роз'язування задач згідно свого варіанта.***

***(2 бала)***

1. Задано 2 файли дійсних чисел (кількість елементів файлів та самі елементи вводяться з клавіатури). Записати у третій файл по черзі значення цих файлів.
2. Задано текстовий файл. Підрахувати кількість входжень в нього кожної з голосних букв.
3. Дано текстовий файл, в який записано код програми на JavaScript (Знайти приклад коду в Інтернеті). Перевірити відповідність розміщення круглих та фігурних дужок в ньому. (Кількість відкритих та закритих дужок кожного виду має співпадати)
4. Дано файл цілих чисел. Продублювати в ньому всі парні числа, які знаходяться на непарних місцях.
5. Дано текстовий файл, елементами якого є слова. Вивести на екран всі слова, що починаються на літеру "s". Розглянути наступні варіанти:
   а) відомо, що в існуючому файлі записано 10 слів;
   б) розмір існуючого файлу невідомий.
6. Дано файл, елементами якого є окремі символи. Визначити чи перші два символи є цифрами. Якщо так, то визначити чи знайдене число є парним.

7. Дано файл, елементами якого є окремі символи, що складають слово "*олгаритм*". Отримати новий файл, в якому літери слова "*алгоритм*" будуть розміщені правильно.
8. Дано текстовий файл. Видалити з перший нього рядок в кінці якого знаходиться символ "!". Результат записати в новий файл.
9. Дано текстовий файл. Створити на його основі цілочисловий масив *a*[*i*], який складається з чисел, що визначають кількість символів в кожному з рядків.
10. В консольному арифметичному калькуляторі замість консольного введення-виведення реалізувати роботу з файлом. Вхідний файл містить обчислюваний вираз. Вихідний – обчислюваний вираз, текст "Результат=" і сам результат обчислення.
11. Розробити консольний застосунок, що дозволяє писати і читати дані в текстовий файл. Дані є записами результатів іспиту – прізвище студента і його оцінка. Ім'я текстового файлу включає назву предмету.
12. Дано текстовий файл, що містить речення та змінна, що містить задане користувачем слово. Побудувати програму, що читає речення з файла і виводить на екран список речень, що містять задане слово.
13. Задано файл, що містить масив чисел. Створити інший файл і записати в нього числа з першого файла за таким правилом: парні числа помножені на 2, непарні числа збільшені на 1, усі від'ємні числа перетворити за модулем.
14. Записати у файл sport.txt, що знаходитиметься на диску D прізвища та результати 3 переможців олімпіади з інформатики. Вивести вміст файлу на екран для візуального контролю. Внести в кінець файлу прізвище та ім'я голови журі.
15. Відкрити диск D та створити в ньому текстовий файл Zavd2.txt, куди вписати своє прізвище та вік (через пробіл). Створити програму, яка зчитує з файлу інформацію, визначає кількість літер у введеному слові, вивести своє прізвище у зворотному порядку в стовпець та до віку додати число 10.
16. На диску D міститься файл 111.txt з числами (записаними через пробіл). Знайти суму елементів масиву.
17. Дано файл f.txt, який містить цілі числа. Переписати у файл g.txt ті з них, які є парними числами.
18. Дано файл, який містить цілі числа. Визначити кількість непарних чисел у ньому. Вивести на екран ці числа.
19. Дано файл, який містить текст, що складається з окремих речень і в ньому використовуються розділові знаки „.”, „?”, „!”. Порахувати кількість речень в даному тексті.
20. Дано ціле число *n*. Створити файл цілих чисел та записати в нього *n* перших додатних чисел (2, 4, 6, … ).
21. Записати за допомогою програми у файл три числа. Зчитати з файлу ці числа та перевірити чи з сторін, довжинами якого є ці числа, можна побудувати трикутник. Відповідь дописати в кінець файлу.
22. На диску *D* міститься файл *222.txt* з числами (записаними через пробіл). Знайти кількість чисел, що знаходяться у вказаному діапазоні [*a, b*]. Параметри *a* та *b* вводяться з клавіатури. Записати в кінець файлу цю кількість.
23. Створіть файл, який містить середній бал успішності студентів вашої групи (кількість студентів групи та їхні середні бали вводяться з клавіатури). Підрахуйте загальний середній бал групи.
24. Створіть файл дійсних чисел. Підрахуйте кількість елементів файлу, більших за *n* (кількість елементів файлу, *n* та самі елементи вводяться з клавіатури).
25. Знайти найбільший елемент у файлі цілих чисел та вивести його на екран (кількість елементів файлу та самі елементи вводяться з клавіатури).

***3. Розв’язати задачу з індивідуального завдання 1(б) при умові, що текстовий рядок імпортується з деякого наперед створеного файла input.txt, а результати роботи програми потрібно записати у новостворений під час виконання проекту файл output.txt. Номер варіанта для цього завдання визначається наступним чином: <номер прізвища студента в списку групи> +5, для номерів більше 20 формула інша <номер прізвища студента в списку групи>–20***

***(1 бал)***