Практична робота №5. Рядки і функції

Мета. Вивчення основних принципів роботи з рядками, і функції. Формування навичок застосування вивченого теретичного матеріалу для роз'язування практичних завдань.

Постановка завдання: у середовищі Microsoft Visual Studio виконати завдання №1, 2, 3, 4, використуючи інструкції до виконання, наведені в теоретичної частині цієї лабораторної роботи, а також завдання до самостійного виконання згідно з варіантом.

Теоретичні відомості

Рядки в C#

Рядки в C# успадковані від одного базового класу – System.String, у якому реалізовані методи для операцій над рядками. Деякі властивості та методи класу String представлені в таблиці нижче.

Length	Кількість символів в рядку (!властивість, без круглих дужок).
Concat()	Дозволяє з'єднати декілька рядків або змінних типу *object*.
CompareTo()	Порівнює два рядки. Якщо вони рівні, то результат виконання функції дорівнює нулю. При додатньому 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 іншого.
Remove() *Replace()*	Видаляють або замінюють символи в рядку.
ToUpper() *ToLower()*	Перетворять всі символи рядка в рядкові 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()	Підрядок основного рядка, що починається з певного символу і що має задану довжину.
Trim()	Видаляє пропуски або набір заданих символів на початку і кінці основного рядка.
ToCharArray()	Створює масив символів і розміщує в ньому символи початкового рядка.

!!!Зверніть увагу.

1. Тип String є типом-посиланням. Проте, при використанні операцій порівняння відбувається порівняння значень рядків (самих текстів), а не їх адрес в оперативній пам'яті.
2. Операція "+" для об'єктів string перевизначена через метод Concat(). Це одначає, що ***кожне використання цієї операції призводить то створення нового рядка в оперативній пам'яті.***

Завдання 1

Створити ASP.Net Web-застосування, в якому реалізувати функцію *MakeLine.*

Функція має створювати рядок чисел, які є значеннями таблички множення на 5, 6, 9, 7.

Довжина таблички множення має задаватись змінною count.

Рядок розділення може бути – “ “, “,”, “.”, “<-“.

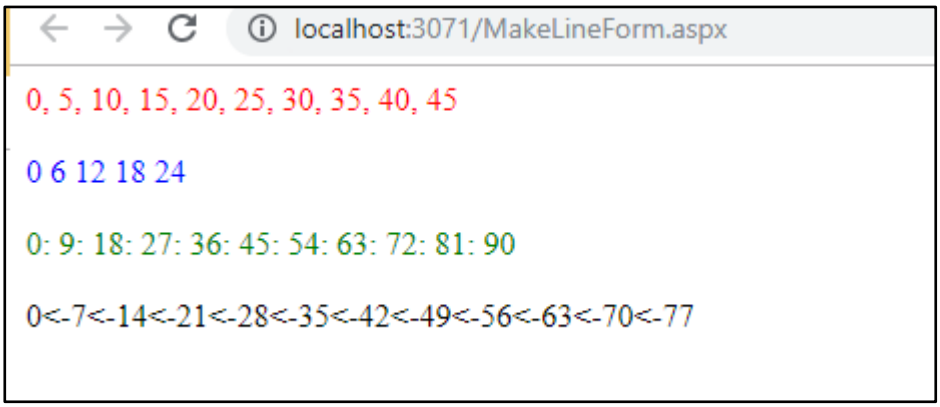
Результати обчислень заносяться в масив рядків.

Після цього, використовуючи функцію *Join*, значення всіх елементів масиву повинні об'єднуватись в рядок.

Виведення рядка на форму здійснюється в функції `Page_Load`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Example1
{
 public partial class MakeLineForm : System.Web.UI.Page
 {
  protected void Page_Load(object sender, EventArgs e)
  {
   Response.Write("<div style='color:red'>");
   Response.Write(MakeLine(5, ", ",10));
   Response.Write("</div></br>");

   Response.Write("<div style='color:blue'>");
```

```
   Response.Write(MakeLine(6, " ",5));
   Response.Write("</div></br>");

   Response.Write("<div style='color:green'>");
   Response.Write(MakeLine(9, ": ",11));
   Response.Write("</div></br>");

   Response.Write("<div style='color:braun'>");
   Response.Write(MakeLine(7, "< ",12));
   Response.Write("</div></br>");
  }

  private static string MakeLine(int number, string separator, int count)
  {
   string[] sArr = new string[count];

   for (int i = 0; i < count; i++)
    sArr[i] = String.Format("{0}", i * number);

   return String.Join(separator, sArr);
  }
 }
                                                    }
```



Мал. 1. Використанням функції Join. Результат роботи програми

При роботі з рядками буває необхідно розділити рядок на підрядки, які відокремлені один від одного вказаними символами.

Завдання 2

Створити рядок символів, в якому є декілька розділових символів.

За допомогою функції *Split* рядок розділити на підрядки та вивести їх на екран.

Розділові символи задати, як масив символів.

Застосувати функцію *Trim*, щоб переконатися, що заданий рядок не містить лише пропуски.

Файл SplitWebForm.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="SplitWebForm.aspx.cs"
Inherits="Example2.SplitWebForm" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
```

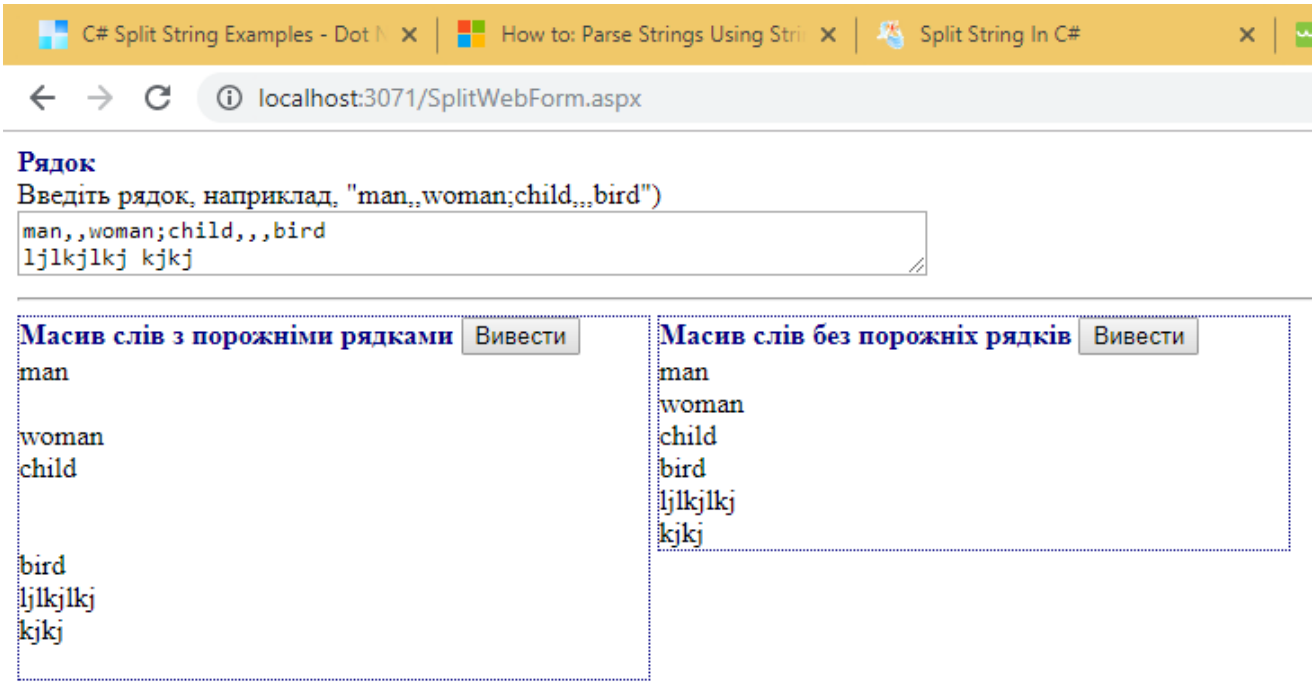
```
<head runat="server">
 <title></title>
</head>
<body>
 <form id="form1" runat="server">
 <div>
  <span style="font-weight: bold; color: navy">Рядок </span><br />
  Введіть рядок, наприклад, "man,,woman;child,,,bird")<br />

  <asp:TextBox ID="txtString" runat="server" TextMode="MultiLine" Width="500px"></asp:TextBox>
  <hr />

  <div style="vertical-align:top;border: 1px dotted navy; width: 350px; display: inline-block">
  <span style="font-weight: bold; color: navy">Масив слів з порожніми рядками</span>

  <asp:Button ID="btnArrayWithEmptyWord" runat="server" Text="Вивести"
   OnClick="btnArrayWithEmptyWord_Click" /><br />

  <asp:Literal ID="ltrArrayWithEmptyWord" runat="server"></asp:Literal>
  </div>

  <div style="vertical-align:top;border: 1px dotted navy; width: 350px; display: inline-block">
  <span style="font-weight: bold; color: navy">Масив слів без порожніх рядків</span>

  <asp:Button ID="btnArrayWithOutEmptyWord" runat="server" Text="Вивести"
   OnClick="btnArrayWithOutEmptyWord_Click" /><br />

  <asp:Literal ID="ltrArrayWithOutEmptyWord" runat="server"></asp:Literal>
  </div>
 </div>
 </form>
</body>
</html>
```

Файл SplitWebForm.aspx.cs

```
using System;
namespace Example2
{
  public partial class SplitWebForm : System.Web.UI.Page
  {
    // Input string contain separators.
    char[] delimiter1 =
                new char[] {';',',', ' ', '\"', '\n','\t'};  // Split on these
    string[] array1;
    string[] array2;

    protected void Page_Load(object sender, EventArgs e)
    {

    }
    /// <summary>
    /// Виводить масив слів з пустими рядками
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void btnArrayWithEmptyWord_Click(object sender, EventArgs e)
    {

     //перевірка, чи рядок не є лише з пропусків і не є порожнім
     if(!String.IsNullOrEmpty(txtString.Text))
      {
```

```
        array1 = txtString.Text.Split(delimiter1, StringSplitOptions.None);

        foreach (string entry in array1)
        {
          ltrArrayWithEmptyWord.Text += entry + "<br>";
        }
      }
    }
    /// <summary>
    /// Виводить масив слів без прожніх рядків
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void btnArrayWithOutEmptyWord_Click(object sender, EventArgs e)
    {
      //перевірка, чи рядок не містить одні пропуски
      if (txtString.Text.Trim() != "")
      {
        array2 = txtString.
                    Text.
                    Split(delimiter1, StringSplitOptions.RemoveEmptyEntries);

        foreach (string entry in array2)
        {
          ltrArrayWithOutEmptyWord.Text += entry + "<br>";
        }
      }
    }
  }
}
```

Результат зображений на мал. 2.



Мал. 2. – Результат роботи програми з використанням функції Split

Використання класу System.Text.StringBuilder

Рядки в C# не змінюються. Операції по зміні рядка, не змінюють його початковий варіант, а лише повертають змінену копію рядка. Це можна побачити на наступному прикладі.

```
string s1 = "Приклад рядка";
string s2 = s1.ToUpper();

Console.WriteLine(s1); // Буде виведено - Приклад рядка
Console.WriteLine (s2); // Буде виведено – ПРИКЛАД РЯДКА
```

Рядок s1 не зазнає ніяких змін. Метод ToUpper() створює копію рядка s1 і застосовує до неї необхідні перетворення, тому на екран буде виведений як початковий рядок s1, так і змінений рядок s2. Результатом виконання операції конкатенації рядків є також новий рядок.

Створення копій (чи нових рядків) при виконанні операції пов'язано з додатковими накладними витратами пам'яті комп'ютера. Для уникнення таких ситуацій використовують клас StringBuilder. Всі операцій з об’єктом цього класу виконується саме з цим об’єктом, що дещо нагадує роботу з рядками в C++.

Наступний код додає один рядок до іншої і виводить результат у вікно браузера. При цьому змінюється оригінал рядка, копія не створюється:

```
StringBuilder sb =new StringBuilder(@"Я вчусь в коледжі університету “Крок” ");
sb.Append(", на відділенні комп’ютерних наук");
txtSpeciality.Text = sb.ToString();
```

Метод ToString() для перетворює тип `StringBuilder` в string.

Таблиця деяких методів класу `StringBuilder`

Append	Додавання заданого рядка в кінець рядка об'єкту
AppendFormat	Додавання заданого форматованого рядка (рядка, що містить символи, які управляють) 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*, що зменшить витрати, пов'язані із створенням копії рядка при виконанні кожної операції.

Функції C#

Визначення функцій в C# близьке до їх визначенням в мові C++. Проте є відмінності. Мова C# є повністю об'єктно-орієнтованою мовою програмування, тому функції можуть визначатись лише всередині класів. В C# для функцій використовують термін "метод".

Синтаксис опису методів:

[атрибути][модифікатори] {void | тип результату функції}
ім'я методу ([список формальних аргументів])
{
…….
}

Ім'я методу і список формальних аргументів представляють сигнатуру методу. Квадратні дужки, як це прийнято, показують, що їх вміст може бути опущен при описі методу.

Атрибути і модифікатори є дуже важливими складовими опису будь-якого методу, они будуть розглянуті вивченні класів, оскільки мають до цього найбезпосередніше відношення. Поки ж будемо використовувати лише два з них: ***private і public.***

Private означає, що даний метод є закритим, відповідно доступ до нього можуть отримати тільки методи того класу, в якому він оголошений.

Public – навпаки, означає, що доступ до даного методу є відкритим і загальнодоступним з будь-якої точки застосування.

При визначенні методу необхідно вказувати тип значення, ім’я методу, а також круглі дужки. Якщо метод не повертає значення, то вказується тип *void*.

Приклади опису функцій:

```
private void A()
{}

public int B()
{}

 public long Stepin(int a, int b)
 {
 long r;
 r = (long)Math.Pow(a, b);
 return (r);
```

```
}
```

Тут метод *A* є закритою процедурою, методи *B і Stepin* – відкритими функціями. У методів *A і B* невизначені формальні параметри, тоді як у *методу Stepin* два формальні параметри: *a і b*.

Як видно з цього прикладу, список формальних аргументів може бути порожнім.

Повний синтаксис оголошення формального аргументу виглядає таким чином:

[ref|out|params] тип_аргумента імя_аргумента

При цьому обов'язковою є вказівка типу аргументу, який може бути скалярним, масивом, класом, структурою, – будь-яким типом, допустимим в C#.

Для того, щоб передати в метод довільну кількість фактичних аргументів вказується ключове слово *params*. Використати його можна лише один раз і може бути лише останнім аргументом списку, що оголошується як масив довільного типу.

Всі аргументи методів можна розділити на три групи: *вхідні, вихідні, що оновлюються*.

Вхідні, необхідні для передачі інформації методу, їх значення в тілі методу доступні тільки для читання.

Вихідні є результати методу, позначаються ключовим словом *out*, в тілі методу цьому аргументу необхідно обов’язково присвоїти значення;

Оновлюванні, здатні виконувати обидві функції, позначаються за допомогою ключового слова *ref*.

Для передачі значень в метод в останніх двох типів використовується механізм передачі зв посиланням.

Як приклад змінимо *метод Stepin*:

```
public void Stepin(out long r,int a, int b)
{
 r = (long)Math.Pow(a, b);
}
```

Формальний параметр *r* використовується, як вихідний. У тілі методу йому присвоюється значення, яке надалі може бути використане в програмі, що його викликала.

Виклик методу здійснюється так

```
long s;
Stepin(out s, 2, 6);
Response.Write(s.ToString());
```

Перший параметр методу вказується з ключовим словом *out* і передається за посиланням, тобто будь-яка зміна параметру *r* в *методі Stepen* є, фатично, зміною змінної *s*.

Завдання 3

Створіть *метод Stepin* який знаходить суму квадратів для довільної кількості чисел.

Опис методу може бути таким:

```
public void Stepin(out long r, int a, params int[] b)
{
 r = 0;
 foreach (int i in b)
        r += (long)Math.Pow(i, a);
}
```

Виклик методу може бути таким:

```
int[] digits ={1,8,4};
Stepin(out s, 2, digits);
Console.WriteLine(s.ToString());
```

Для виклику методу необхідно сформувати масив цілих чисел, який потім необхідно передати як третій аргумент в метод.

Змінній *r* присвоюється сума квадратів чисел. Поскільки *r* є посиланням на *s*, то цей результат доступний програмі, яка викликала процедуру *Stepin*. Результатом роботи програми є число 81.

Іноді виникає необхідність не тільки передавати довільну кількість початкових даних для розрахунку, але і отримувати із визиваючої процедури довільну кількість змінних, що містять результати розрахунку.

Завдання 4

Створіть процедуру, яка підносить до степеня кожне число масиву, передає результат розрахунків в програму, яка викликала метод, та виводить результат на екран.

Процедура:

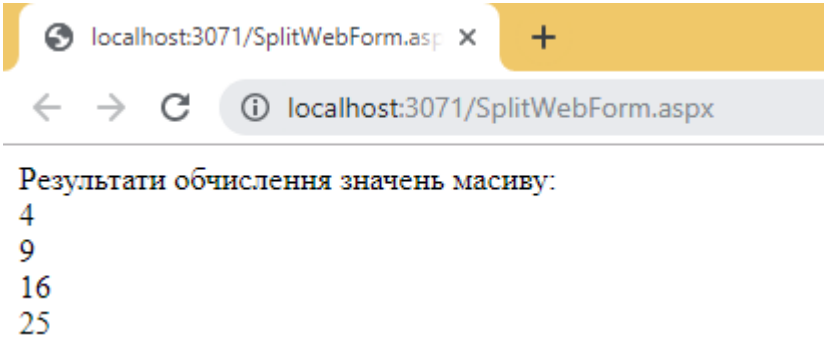
```
public void Stepin(out long[] r,int a, params int[] b)
{
 r=new long[0];
 Array.Resize(ref r,b.Length);
 int j=0;
 foreach (int i in b)
        r[j++]= (long)Math.Pow(i, a);
}
```

Метод *Resize об'єкту Array* резервує пам’ять під елемети масиву *r*. Проте цей метод дозволяє змінювати кількість елементів масиву вказівник якого не вказує на null, тому перед викликом методу *Resize* створюється новий масив *r*, що складається з нуля елементів. Першим параметром методу *Resize* є вказівник на масив (ключове слово *ref*), тому всередині методу відбувається зміна самого масиву, а не його копії.

Виклик методу можна здійснити таким чином:

```
Response.Write("Результати обчислення значень масиву:<br/>");
 long [] result;
 int [] data={2,3,4,5};
 Stepin(out result, 2, data);
 foreach (long i in result)
        Response.Write(i.ToString()+"<br/>");
```

Результат роботи програми зображений на малюнку.



Мал. 3. – Результат роботи методу

Звуваження. Наступний варант значно простіший. Він враховує, що всі масиви передаються за вказівкою.

Виклик процедури

```
Response.Write("Результати обчислення значень масиву:<br/>");

 int[] data = { 2, 3, 4, 5 };
 long[] result = new long[data.Length];

 Stepin(result, 2, data);

 foreach (long i in result)
        Response.Write(i.ToString() + "<br/>");
```

Процедура:

```
public void Stepin(long[] r, int a, int[] b)
{
    for (int i = 0; i < b.Length; i++)
        r[i] = (long)Math.Pow(b[i], a);
}
```

Підсумок

В об'єктно-орієнтованих мовах, таких як C#, основну роль відіграють типи посилання. Якщо методу передається вказівник на об'єкт, то всі поля цього об'єкту можуть змінюватись в методі, тобто код методу має повний доступ до всіх полів об'єкту. Це відбувається незалежно від того, що формально об'єкт не є вихідним і не має ключових слів ref і out, тобто використовує семантику виклику за значенням. Під час такого виклику сам вказівник на об'єкт залишається незмінним, але значення його полів можуть змінюватись. Така ситуація є типовою, тому при роботі з типами-посиланнями ключові слова ref і out рідко застосовуються в описі аргументів методу.

Варіанти індивідуальних завдань:

1. Дано рядок, що містить англійський текст. Знайти кількість слів, що розпочинаються з літери «с».
2. Дано рядок. Підрахувати, скільки в ньому літер «а», «е», «я».
3. Дано рядок. Визначити, скільки в ньому символів « , » і пропусків.
4. Дано рядок, що містить текст. Знайти довжину найкоротшого слова і найдовшого слова.
5. Дано рядок символів, серед яких є двокрапка (:). Визначити, скільки символів йому передує.
6. Дано рядок, що містить текст, що закінчується крапкою. Вивести на екран слова, що містять менше п'яти букв.
7. Дано рядок. Видалити з нього кожну літеру «і» і повторити кожну літеру «а».
8. Дано рядок. Підрахувати кількість літер «і» в останньому його слові.
9. Даний рядок. Підрахувати, скільки різних літер зустрічається в ньому. Вивести їх на екран.
10. Дано рядок символів, серед яких є відкрита та закрита дужка. Вивести на екран всі символи, розташовані усередині цих дужок.
11. Є рядок, що містить літери латинського алфавіту і цифри. Вивести на екран довжину найбільшої послідовності цифр, що йдуть підряд.
12. Дано рядок. Вказати ті слова, які містять хоча б одну літеру «ж».
13. Дано рядок. Знайти ті слова, які розпочинаються і закінчуються однією і тією ж літерою.
14. У рядку замінити всі знаки пропусків двокрапкою (:). Підрахувати кількість замін.
15. Визначити, скільки разів в рядку зустрічається задане слово.
16. У рядку є одна крапка з комою (;). Підрахувати кількість символів до крапки з комою і після неї.

Контрольні питання

1. Від якого базового класу походять всі рядки в C#?

2. Перерахуйте основні методи классу System.String?
3. Поясніть особливості використання классу System.Text.StringBuilder?
4. Дайте визначення функції
5. Як здійснюється виклик функції
6. Дайте визначення групам аргументів методів ?