МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни «Основи програмної інженерії»

для студентів спеціальності
121 «Інженерія програмного забезпечення»
усіх форм навчання

2020

Методичні вказівки до виконання лабораторних робіт з дисципліни «Основи програмної інженерії» для студентів спеціальності 121 «Інженерія програмного забезпечення» усіх форм навчання /Уклад.: Каплієнко Т.І., Качан О.І., Федорченко Є.М. – Запоріжжя: НУ «Запорізька політехніка», 2020. – 88 с.

Укладачі: Т.І. Каплієнко, к.т.н., доцент кафедри ПЗ,
О.І. Качан, старший викладач кафедри ПЗ,
Є.М. Федорченко, старший викладач кафедри ПЗ.

Рецензент: А.О. Олійник, к.т.н., доцент кафедри ПЗ.

Відповідальний
за випуск: С.О. Субботін, зав. каф. ПЗ, д.т.н., професор.

Затверджено
на засіданні кафедри
"Програмні засоби"

Протокол № 1 від 18.08.2020 р.

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Методичні вказівки представляють собою необхідний засіб для успішного вивчення теоретичного матеріалу та практичного освоєння основ Visual Studio C# в рамках дисципліни «Основи програмної інженерії».

Мета роботи – підвищити рівень знань студентів з методики створення програм на алгоритмічній мові C#, проектування і розробки користувацьких інтерфейсів та їх модулів до різних програм. Лабораторний курс складається із сьомі робіт та двох самостійних завдань щодо індивідуального виконання за методичними вказівками для самостійних робіт. Кожна робота виконується та здається студентом індивідуально. Студенти, що не підготовлені до роботи, а також, які не мають вірно оформленого звіту, до занять не допускаються.

Вимоги до оформлення та змісту звіту, а також контрольні запитання, надано в кожній лабораторній роботі. Студент повинен знати мету роботи, теоретичні відомості, методику проєктування та розробки необхідних програм, а також продемонструвати результат виконання роботи за комп'ютером на робочому місці.

Студент, який не здав попередньої роботи, не допускається до виконання наступних.

Звіт має містити:

– титульний аркуш (на ньому вказують назву міністерства, назву університету, назву кафедри, номер, вид і тему роботи, виконавця та особу, що приймає звіт, рік);

– тему та мету роботи;

– завдання до роботи;

– лаконічний опис теоретичних відомостей;

– результати виконання лабораторної роботи;

– змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 × 297 мм). Текст розміщують тільки з однієї сторони листа. Розмір елементів тексту – 14пт. Міжрядковий інтервал – 1,5. Поля сторінки згори та знизу – 20мм, ліворуч – 30мм, праворуч – 10мм. Аркуші скріплюють за допомогою канцелярських скріпок або вміщують у канцелярський файл, але з'єднання листів стиплером категорично не допускається.

1 ЛАБОРАТОРНА РОБОТА №1
ЗНАЙОМСТВО З VISUAL STUDIO C#

Мета роботи: вивчити основні можливості та принципи роботи з середовищем розробки ПЗ у Microsoft Visual Studio C#.

1.1 Короткі теоретичні відомості

Програма на C# містить у собі один або декілька файлів. Кожний файл може містити одне або кілька просторів імен. Кожний простір імен може містити вкладені простори імен і типи, такі як класи, структури, інтерфейси, перерахування й делегати – функціональні типи. При створенні нового проєкту C# у середовищі Visual Studio обирається один з 10 можливих типів проєктів, у тому числі Windows Application, Class Library, Web Control Library, ASP.NET, Application та ASP.NET Web Service. На підставі зробленого вибору автоматично створюється каркас проєкту.

Visual Studio.NET – це не тільки середовище для розробки програм мовою C#. Visual Studio.NET дозволяє створювати програми на мові VB, C#, C++, формувати встановлювальні пакети Setup для створених програм та багато іншого. Для створення нового проєкту обирається послідовність пунктів меню: **`File->New->Project...`** Потім з'являється вікно, аналогічне зображеному на рисунку 1.1.

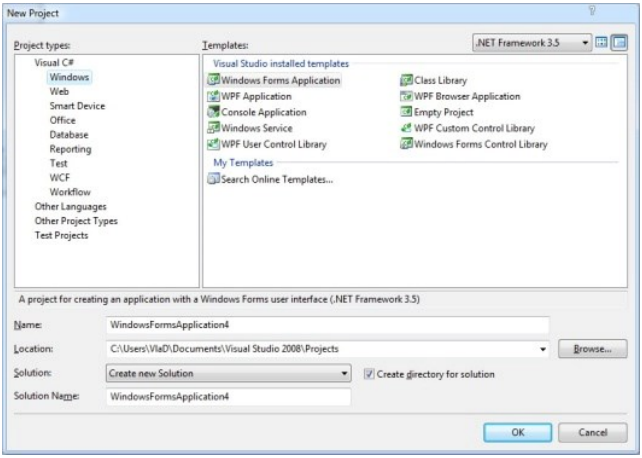


Рисунок 1.1 – Вікно створення нового проєкту

На цьому етапі обирається потрібна мова програмування або спеціальний майстер створення програм. Далі обирається пункт Visual C# Project.

У правій частині вікна потрібно вказати тип створюваного проєкту. Це може бути Windows-програма (Windows Application), програма для Інтернету (ASP.NET), консольна програма (Console Application) або деякі інші. Ліворуч вікна обирається пункт Windows Application. Крім того, обов'язково вказується назва створюваного проєкту та шлях до робочого каталогу, де він буде розташован. Натискається кнопка «ОК».

На рисунку 1.2 зображено основні частини візуального середовища розробки проєкту.

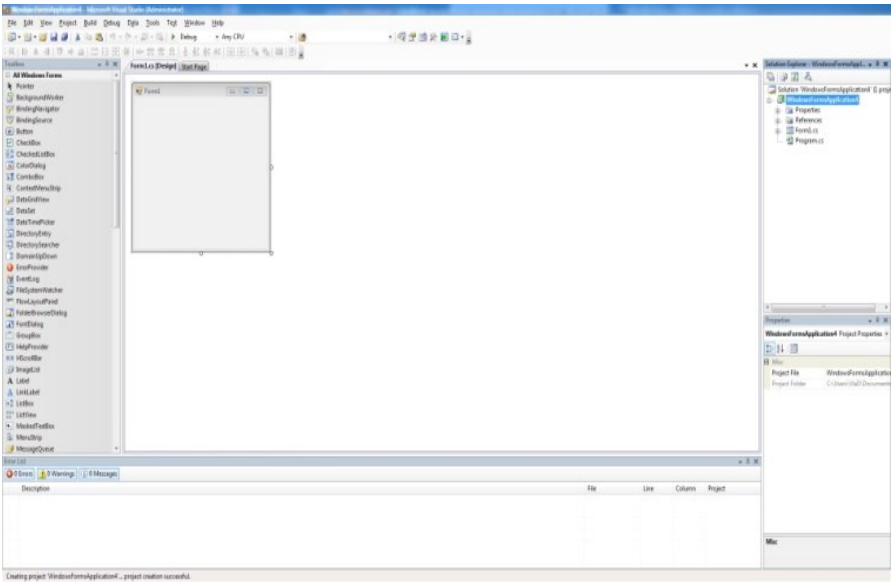
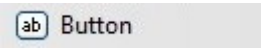


Рисунок 1.2 – Головне вікно середовища Visual Studio.NET

У центрі розташоване головне вікно для створення візуальних форм та написання коду. Праворуч розміщається вікно Solution Explorer для керування вашими проєктами, Class View для огляду всіх класів та вікно властивостей Properties Explorer.

Надалі буде розглянуто основні елементи керування.

1.1.1 Кнопки – `Button`

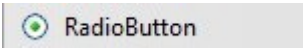


– кнопкою називається елемент керування, вся взаємодія користувача з яким обмежується однією дією – натисканням. Усе, що необхідно зробити при роботі із кнопкою – це помістити її в потрібному місці форми та призначити їй відповідний оброблювач. Оброблювач призначається для події **`Click`**.

1.1.2 Чекбокси – `CheckBox`

– поперше, що необхідно сказати про чекбокси це те, що вони є кнопками відкладеної дії, тобто їхнє натискання не повинне запускати яку-небудь негайну дію. З їхньою допомогою користувачі вводять параметри, які позначаться після, коли дія буде запущена іншими елементами керування. Елемент **`CheckBox`** може мати 3 стани – позначений, непозначений та змішаний. Найчастіше цей елемент застосовується для визначення прапорців, які можуть мати тільки два стани.

1.1.3 Радіокнопки – `RadioButton`



– за своїми властивостям вони небагато схожі на чекбокси. Їхня головна відмінність полягає в тому, що група чекбоксів дозволяє обрати будь-яку комбінацію параметрів, радіокнопки дають можливість обрати тільки один параметр. Із цієї відмінності виникають і всі інші. Наприклад, у групі не може бути менше двох радіокнопок. Крім того, у радіокнопок не може бути змішаного стану (не можна сполучати взаємовиключаючі параметри).

1.1.4 Блок групування – `GroupBox`

– блок групування допомагає не тільки візуально об’єднати кілька елементів керування в одну групу. Блок групування повертає один з декілька станів згрупованих елементів, особливо таких як **`RadioButton`**. Якщо радіокнопки не будуть згруповані у **`GroupBox`**, то неможливим буде отримати стан від групи радіокнопок (рис. 1.3).

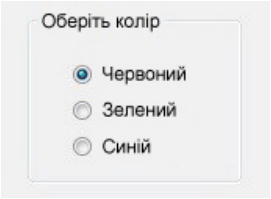


Рисунок 1.3 – Приклад використання блоку групування для об’єднання радіокнопок

Розроблювана у прикладі програма буде виконувати наступні функції: радіокнопки задають текст повідомлення, яке буде виводитися по натисканню на звичайну кнопку. Чекбокс повинен визначати – виводити повідомлення чи ні.

Створіть новий Windows Forms проєкт за назвою **`Testbuttons`**. Збережіть його в створену для власних проєктів папку. Змінить деякі властивості створеної форми:

```
Name = «Testbuttonsform»
Text = «Тест для кнопок»
```

Тепер додайте на форму один елемент керування **`GroupBox`**, три елементи **`RadioButton`**, один елемент **`CheckBox`** та один елемент **`Button`** (рис. 1.4). Усі три радіокнопки повинні бути поміщені в один **`GroupBox`**. Інакше вони не будуть зв’язані між собою.

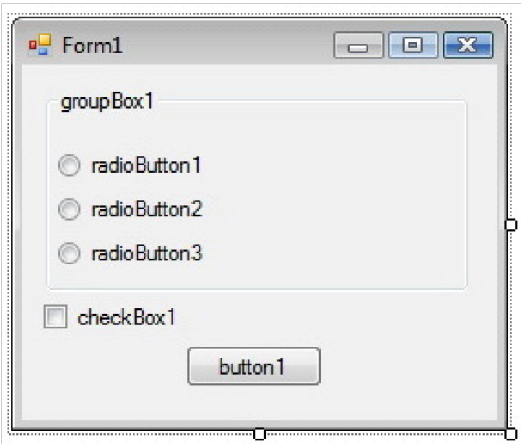


Рисунок 1.4 – Проєктування форми для застосунку Testbuttons

Тепер змініть деякі властивості доданих елементів:

button1: Text – показати повідомлення
groupbox1: Text – оберіть текст повідомлення
radiobutton1: Text – перше повідомлення
radiobutton2: Text – друге повідомлення
radiobutton3: Text – третє повідомлення
checkbox1: Text – показувати повідомлення
Checked – True

Можливо, для того щоб додати вашій формі більш гідного виду, вам доведеться змінити розміри деяких елементів керування.

Тепер звернемося до коду нашої програми, який створило середовище Visual Studio.NET:

```
using System;
using System.Drawing;
using System.Collections;
using System.Componentmodel;
using System.Windows.Forms;
using System.Data;
namespace Testbuttons {
//<summary>
// summary description for Form1.
//</summary>
public Class Testbuttonsform: System.Windows.Forms.Form {
  private System.Windows.Forms.Button button1;
  private System.Windows.Forms.Groupbox groupbox1;
  private System.Windows.Forms.Radiobutton radiobutton1;
  private System.Windows.Forms.Radiobutton radiobutton2;
  private System.Windows.Forms.Radiobutton radiobutton3;
  private System.Windows.Forms.Checkbox checkbox1;
//<summary>
// Required designer variable.
//</summary>
private system.componentmodel.container components = null;
public Testвuttonsforм () {
//
// Required fcr Windows Form Designer support
//
Initializecomponent 0;
//
// TODO:  Add any constructor code after Initializecomponent
// call
}

//<summary>
// Clean up any resources being used.
//</summary>
protected override void Dispose ( bool disposing ) {
```

```
      if (disposing) {
        if (components != null) {
          components.Dispose();
        }
      }
      base.Dispose ( disposing );
    }

      #region Windows Form Designer generated code
      //<summary>
      // Required method for Designer support - do not motify
      // the contents of this method with the code editor.
      //</summary>
      private void Initializecomponent 0 (
        this.buttonl = new System.Windows.Forms.Button ();
        this.groupboxl = new System.Windows.Forms.Groupbox();
        this.radiobutton3 = new System.Windows.Forms.Radiobutton ();
        this.radiobutton2 = new System.Windows.Forms.Radiobutton ();
        this.radiobutton1 = new System.Windows.Forms.Radiobutton ();
        this.checkboxl = new System.Windows.Forms.Checkbox();
        this.groupboxl.Suspendlayout(); this.Suspendlayout 0;
      //
      // buttonl
      //
      this.buttonl.Location = new System.Drawing.Point(80, 240)
      this.buttonl.Name = "button1";
      this.button1.Size = new System.Drawing.Size(128, 23);
      this.button1.Tabindex = 0;
      this.buttonl.Text = "Показати  повідомлення";
      //
      // groupboxl
      //
      this.groupboxl.Controls.Addrange(new
System.Windows.Forms.Control[]
      { this.radiobutton3, this.radiobutton2, this.radiobuttonl});
      this.groupboxl.Location = new System.Drawing.Point (18, 16);
      this.groupboxl.Name =  "groupbox1";
      this.groupbox1.Size = new System.Drawing.Size(280, 112);
      this.groupbox1.Tabindex = 1; this.groupbox1.Tabstop = false;
      this.groupboxl.Text =   "Виберіть текст повідомлення";
      //
      // radiobutton3
      //
      this.radiobutton3.Location = new System.Drawing.Point (16, 80);
      this.radiobutton3.Name = "radiobutton3";
      this.radiobutton3.Size = new System.Drawing.Size(240, 24);
      this.radiobutton3.Tablndex = 2;
      this.radiobutton3.Text =   "третє повідомлення";
      //
      // radiobutton2
      //
      this.radiobutton2.Location = new System.Drawing.Point (16, 52);
      this.radiobutton2.Name = "radiobutton2";
```

```
      this.radiobutton2.Size = new System.Drawing.Size (240, 24);
      this.radiobutton2.Tablndex = 1;
      this.radioButton2.Text = "друге повідомлення";
      //
      // radiobutton1
      //
      this.radiobutton1.Location = new System.Drawing.Point (16, 24);
      this.radiobuttonl.Name = "radiobuttonl";
      this.radiobutton1.Size = new System.Drawing.Size (240, 24);
      this.radiobuttonl.Tablndex = 0;
      this.radiobuttonl.Text = "перше повідомлення";
      //
      // checkbox1
      //
      this.checkboxl.Checked = true;
      this.checkboxl.Checkstate =
System.Windows.Forms.Checkstate.Checked;
      this.checkboxl.Location = new System.Drawing.Point(24, 136);
      this.checkboxl.Name = "checkboxl";
      this.checkboxl.Size = new System.Drawing.Size(256, 24);
      this.checkboxl.Tablndex = 2;
      this.checkboxl.Text = "Показувати повідомлення";
      //
      // Testbuttonsform
      //
      this.Autoscalebasesize = new System.Drawing.Size(5, 131);
      this.Clientsize = new System.Drawing.Size(292, 273);
      this.Controls.Addrange(new System.Windows.Forms.Control[] f
this.checkboxl, this.groupboxl, this.buttonl});
      this.Name = "Testbuttonsform";
      this.Text = "Тест для кнопок";
      this.groupboxl.Resumelayout(false);
      this.Resumelayout(false);
    }
      attendregion
      //<summary>
      // The main entry point for the application.
      //</summary>
      [Stathread]
      Static voidmain() {
        Application.Run (newform1 ());
      }
    }
  }
```

Код, який було створено студентом може відрізнятися від наданого у прикладі в залежності від версії C#.

У деяких випадках необхідно буде змінити у програмному коді ім’я форми, з якої буде запускатися застосунок. У попередньому прикладі **`Form1`** було перейменовано у **`Testbuttonsform`**. Знайдіть у коді рядок:

```
Application.Run(new Form1());
```

і при необхідності (якщо програма не запускається), замініть його на:

```
Application.Run(new Testbuttonsform());
```

Тепер програма працездатна. Якщо ви строго додержувалися усіх інструкцій, то програма повинна побудуватися без помилок. Відкомпілюйте її та запустіть на виконання. Поки програма не здатна виконувати будь-які дії. Щоб наділити її функціональністю, додайте функцію-оброблювач для кнопки **button1**. Використовуйте ім'я за замовчуванням для цієї функції – **button1_Click**, яке було вже створене середовищем Visual Studio.NET. При цьому у функцію **Initializecomponent** додається рядок:

```
this.button1.Click += new
System.Eventhandler(this.buttonl_Click);
```

та з'явиться тіло самої функції:

```
private voidbutton1_Click(object Sender, System.Eventargs e) {
}
```

Додайте до тіла функції **button1_Click** наступний код:

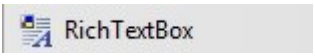
```
//вводимо строкову змінну
//для зберігання обраного повідомлення string strmessage="";
//визначаємо яка саме радіокнопка відзначена
//і обираємо відповідно до цього
//текст повідомлення
//перевіряємо першу радіокнопку
if (radioButton1.Checked == true) {
//якщо відмічена саме ця кнопка
//то копіюємо текст кнопки в змінну
  strmessage = radioButton1.Text;
}
//перевіряємо другу радіокнопку
else if (radioButton2.Checked == true) {
//якщо відзначена саме ця кнопка
//то копіюємо текст кнопки в змінну
  strmessage = radioButton2.Text;
}
//перевіряємо третю радіокнопку
else if (radioButton3.Checked == true) {
//якщо відзначена саме кнопка
//то копіюємо текст кнопки в змінну
  strmessage = radioButton3.Text;
}
//перевіряємо, чи встановлений чекбокс,
//що дозволяє виведення повідомлення
//якщо так, то виводимо обране повідомлення на екран
if (checkBox1.Checked == true) {
  MessageBox.Show("Ви вибрали " + strmessage);
}
```

Відкомпілюйте та запустіть програму на виконання. Оберіть першу радіокнопку (перше повідомлення). Натисніть кнопку «Показати повідомлення». На екрані з'явиться напис: **«Ви обрали перше повідомлення»**. Обравши іншу радіокнопку, ви одержите інший текст повідомлення. Тепер зніміть прапорець «Показати повідомлення». Натисніть кнопку «Показати повідомлення». На екрані нічого не повинне з'явитися.

1.1.5 Поле введення – `TextBox`

– цей елемент керування є основним, що призначений для введення користувачем текстових даних. Використовувати **`TextBox`** можна в однорядковому або багаторядковому режимі. Однак даний елемент керування має обмеження до 64 Кб тексту. Якщо необхідно обробляти більші обсяги інформації, то краще використовувати елемент **`RichTextBox`**.

1.1.6 Розширене поле вводу – `RichTextBox`



– даний елемент керування дає можливість користувачеві вводити та обробляти більші обсяги інформації (більше ніж 64 Кб). Крім того, **`RichTextBox`** дозволяє редагувати колір тексту, шрифт, додавати зображення. **`RichTextBox`** включає всі можливості текстового редактору Microsoft Word.

1.1.7 Список – `ListBox`

– найпростіший варіант списку. Він дозволяє обирати один або декілька елементів, що зберігаються у списку. Крім того, **`ListBox`** має можливість відображати дані у декількох колонках. Це дозволяє представляти дані в більшому обсязі та не стомлювати користувача скролюванням.

1.1.8 Список, що позначається – `CheckedListBox`

– список, що позначається є різновидом простого списку. Його додаткові переваги полягають у наявності чекбоксів поруч із кожним елементом списку. Користувач має

можливість відзначити один або декілька елементів списку, виставивши напроти нього прапорець.

1.1.9 Випадаючий список – `ComboBox`

– цей варіант списку зручний тим, що не займає багато простору на формі. Постійно на формі представлене тільки одне значення цього списку. При необхідності користувач може розкрити список і вибрати інше його значення. Крім того, режим **`Dropdown`** дає користувачеві можливість вводити власне значення при відсутності необхідного значення у списку.

Розглянемо приклад використання списків. Створимо програму, призначену для обліку даних про учасників змагань. Програма буде містити два списки: **`ComboBox`** для введення інформації про учасників та **`CheckedListBox`** для зберігання та обробки даних. За допомогою списку **`ComboBox`** користувач буде обирати прізвища осіб, яких необхідно додати у список учасників. Дві кнопки на формі будуть додавати або видаляти учасників зі списку.

Створіть новий Windows Forms проєкт за назвою **`Testlists`**. Збережіть його у створену для проєктів папку. Перейменуйте файл **`Form1.cs`** у **`TestListsForm.cs`**. Тепер додайте на вашу форму наступні елементи керування (рис. 1.5):

– **`GroupBox`**, і помістіть до нього **`CheckedListBox`**;

– **`ComboBox`**;

– два елементи **`Button`**.

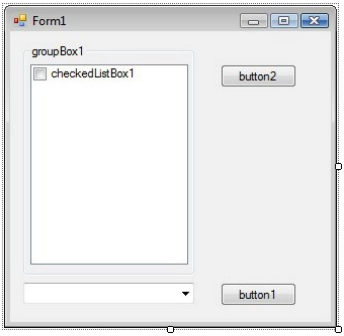


Рисунок 1.5 – Проєктування форми програми **`Testlists`**

У даному прикладі можна було обійтися й без елемента **GroupBox**, оскільки він призначений тільки для оформлення інтерфейсу програми. Однак візьміть собі за гарну звичку завжди поміщати списки у середину **GroupBox**-елементів. Це зробить ваші програми більш привабливими.

Змінить деякі властивості створеної форми:

```
Text – «робота зі списками»
```

Тепер змінимо властивості елементів керування:

```
GroupBox1:        Text – «список учасників»
CheckedListBox:   Name – memberlist
ComboBox1:        Name – peoplelist
                  Text – «»
Button1:          Name – buttonadd
                  Text – «Додати»
Button2:          Name – buttonadd
                  Text – «Вилучити»
```

Елементи керування **ComboBox** та **CheckedListBox** можуть бути проініціалізовані за допомогою дизайнера середовища Visual Studio.Net. Для зберігання елементів списків дані компоненти мають властивість **Items**. Властивість **Items** саме по собі є масивом рядків. Давайте проініціалізуємо елемент керування **ComboBox**, який має ім'я **peoplelist**, списком прізвищ передбачуваних учасників змагань. Для цього у вікні властивостей **peoplelist** оберіть властивість **Items**. Відкрийте вікно **String Collection Editor**, нажавши на кнопку із трьома крапками в полі **Items**. Додайте у запропонований список прізвища (рис. 1.6).

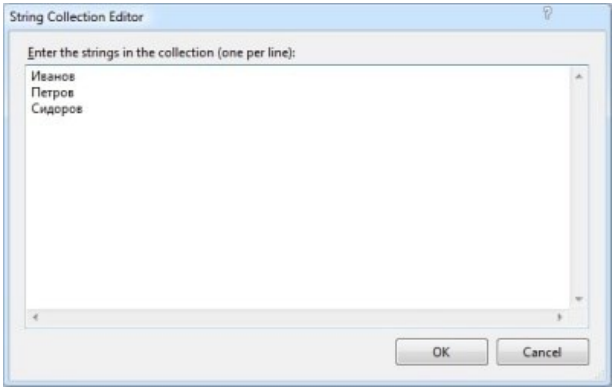


Рисунок 1.6 – Редактор списку рядків

Надалі додайте оброблювачі для кнопок «Додати» та «Вилучити», два рази клацнувши лівою кнопкою миші по кожній із кнопок.

Підготовчий етап до написання програми завершений. Збережіть зроблені вами зміни. Тепер звернемося до коду програми:

```
using System;
using System.Drawing;
using System.Collections;
using System.Componentmodel;
using System.Windows.Forms;
using System.Data;
namespace Testlists {
//<summary>
// Summary description for Form1.
//</summary>
public Class Form1: System.Windows.Forms.Form {
  private System.Windows.Forms.Groupbox groupbox1;
  private System.Windows.Forms.Checkedlistbox memberlist;
  private System.Windows.Forms.Combobox peoplelist;
  private System.Windows.Forms.Button buttonadd;
  private System.Windows.Forms.Button buttondelete;
  //<summary>
  // Required designer variable.
  //</summary>
  private system.Componentmodel.Container components = null;
  public Testlistsform() {
  //
  // Reqiured for Windows Form Designer Support
  //
  Initializecomponent();
  //
  // TODO: Add any constructor code after Initializecomponent
call
  //
}
//<summary>
// Clean up any resurces being used.
//</summary>
protected override void Dispose( bool disposing ) {
  if ( disposing ) {
    if (components != null)
      components.Dispose();
  }
  base.Dispose( disposing );
    }
#region Widows Form Designer generated code
//<summary>
// Required method for Designer support - do not modify
// the contents of this method with the code editor.
//</summary>
private void Initializecomponent() {
```

```
      this.groupbox1 = new System.Windows.Forms.Groupbox();
      this.memberlist = new System.Windows.Forms.Checkedlistbox();
      this.peoplelist = new System.Windows.Forms.Combobox();
      this.buttonadd = new System.Windows.Forms.Button();
      this.buttondelete = new System.Windows.Forms.Button();
      this.groupbox1.Suspendlayout();
      this.Suspendlayout();
      //
      // groupbox1
      //
      this.groupbox1.Controls.Addrange(new Sys-
tem.Windows.Forms.Control() {
        this.memberlist();
        this.groupbox1.Location = new System.Drawing.Point (8, 8);
        this.groupbox1.Name = "groupbox1";
        this.groupbox1.Size = new System.Drawing.Size{184, 216);
        this.groupbox1.Tabindex = 0;
        this.groupbox1.Tabstop = false;
        this.groupbox1.Text = "Список учасників";
      //
      // memberlist
      //
        this.memberlist.Location = new System.Drawing.Point(8, 24);
        this.memberlist.Name = "memberlist";
        this.memberlist.Size = new System.Drawing.Size(168, 184);
        this.memberlist.Tablndex = 0;
      //
      // peoplelist
      //
        this.peoplelist.Items.Addrange(new object[] {
          "Іванов", "Петров", "Сидоров"});
        this.peoplelist.Location = new System.Drawing.Point(8, 232);
        this.peoplelist.Name = "peoplelist";
        this.peoplelist.Size = new System.Drawing.Size(184, 21);
        this.peoplelist.Tablndex = 1;
      //
      // buttonadd
      //
        this.buttonadd.Location = new System.Drawing.Point(200, 232);
        this.buttonadd.Name = "buttonadd";
        this.buttonadd.Size = new System.Drawing.Size(80, 23);
        this.buttonadd.Tablndex = 2;
        this.buttonadd.Text = "Додати";
        this.buttonadd.Click += new System.Eventhandler(this-
      buttonadd_Click);
      //
      // buttondelete
      //
      this.buttondelete.Location - new System.Drawing.Point(200, 32);
      this.buttondelete.Name = "buttondelete";
      this.buttondelete.Size = new System.Drawing.Size(80, 23);
      this.buttondelete.Tablndex = 3;
      this.buttondelete.Text = "Вилучити";
```

```
      this.buttondelete.Click += new System.Eventhandler (this
.buttondelete_Click);
      //
      // Testlistsform
      //
      this.Autoscalebasesize = new System.Drawing.Size(5, 13);
      this.Clientsize = new System.Drawing.Size(292, 273);
      this.Controls.Addrange(new System.Windows.Forms.Control[] (
this.buttondelete, this.buttonadd, this.peoplelist, chis.groupboxlj);
      this.Name = "Testlistsform";
      this.Text = "Робота зі списками";
      this.groupboxl.Resumelayout(false);
      this.Resumelayout(false);
    }
      #endregion
      //<summary>
      // The main entry point for the application.
      //</summary>
      [Stathread] static void Main() {
        Application.Run(new Form1());
      }
      private void buttondelete_Click(object sender, System.Eventargs
e) {  }
      private void buttonadd_Click(object sender, System Eventargs e)
{  }
    }
  }
```

Зараз необхідно додати оброблювачі для кнопки «Додати» та «Вилучити». Як відомо з початкових даних, кнопка «Додати» повинна затягати рядок, обраний у комбобоксі, до списку учасників. Для цього змініть функцію **buttonadd_Click** так, як показано нижче:

```
      private void buttonadd_ciick(object sender, System.Eventargs e)
      {
      //працюємо зі списком для введення прізвищ
      //перевіряємо чи обраний елемент у списку
      if (peoplelist.Text.Length != 0) {
      //якщо елемент обраний, то переносимо його в список учасників
        memberlist.Items.Add(peoplelist.Text);
      } else {
      //якщо елемент не обраний
      //то видаємо інформаційне повідомлення
        MessageBox.Show("Оберіть елемент в списку для введення або
введіть новий.");
      }
    }
```

Опис роботи функції наведений разом з її кодом. Функція **memberUst.Items.Add** додає новий елемент у список **memberlist**. При цьому, параметром функції є значення властивості **peoplelist.Text**, яке обирає користувач. Тепер залишилося

реалізувати видалення елементів зі списку. Для цього введіть код для функції **buttondelete_Click**:

```
      private void buttondelete_Click(object sender, System.Eventargs
e) {
      //поки список позначених елементів не порожній
      while (memberlist.CheckedIndices.Count > 0) {
      //видаляємо із загального списку учасників по одному елементу
      //при цьому список позначених елементів автоматично оновлюється
      //таким чином, щораз нульовий елемент із CheckedIndices
      //буде містити індекс першого позначеного в списку об'єкту
        memberlist.Items.RemoveAt(memberlist.CheckedIndices[0]);
      //при видаленні зі списку останнього позначеного елементу
      //CheckedIndices.Count стане рівним нулю
      //і цикл автоматично завершиться
      }
    }
```

Функція **CheckedListBox.Items.RemoveAt** видаляє зі списку елемент по його індексу. При цьому елементи списку, що йдуть за вилученим, зменшують свій індекс на одиницю. Це обов'язково потрібно враховувати при подальшому обході списку.

Клас **CheckedListBox** містить властивість **CheckedIndices**, яка являє собою масив індексів усіх позначених елементів списку. Цей масив теж змінюється, якщо зі списку був вилучений позначений елемент. А оскільки ми видаляємо зі списку тільки позначені елементи, то **CheckedIndices** буде змінюватися завжди – місце вилученого елемента займе наступний за ним. Цикл продовжить працювати доти, поки у списку **CheckedIndices** буде залишатися хоч один елемент.

1.2 Завдання до роботи

1.2.1 Ознайомитися з короткими теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

1.2.2 Вивчити основні принципи роботи з Visual Studio.

1.2.3 Виконати наступні загальні завдання:

– створити новий проєкт з двома елементами **TextBox** та одним **Button**. Зробити так, щоб при натисканні кнопки **Button** вводимі дані в одному з елементів **TextBox** повторювалися в іншому;

– створити новий проєкт з двома елементами **TextBox**. Зробити так, щоб вводимі дані в одному з них повторювалися в іншому у реальному часі.

1.2.4 Виконати наступні індивідуальні завдання (номер завдання відповідає порядковому номеру варіанту):

1) створити новий проєкт з потрібними елементами. Зробити так, щоб обраний елемент випадаючого списку автоматично вписувався до текстового блоку в режимі реального часу. При цьому, передбачити можливість – дозволяти вивід чи ні;

2) створити новий проєкт з потрібними елементами. Зробити так, щоб обраний елемент випадаючого списку автоматично вписувався до текстового блоку але тільки при натисканні на кнопку. При цьому, передбачити можливість – дозволяти вивід чи ні;

3) створити новий проєкт з потрібними елементами. Зробити так, щоб вводимі дані у текстовий блок №1 дублювалися до інших блоків №2 та №3 в режимі реального часу. Передбачити можливість керування дублюванням до блоку №2 та блоку №3 (використовувати елемент **`CheckBox`**);

4) створити новий проєкт з потрібними елементами. Зробити так, щоб при натисканні на кнопку, яка керує процесом, вводимі дані у текстовий блок №1 дублювалися до інших блоків №2 та №3. Передбачити можливість керування дублюванням до блоку №2 та блоку №3 (використовувати **`RadioButton`**);

5) створити новий проєкт з потрібними елементами. Реалізувати множення (**`*`**) двох чисел та виведення результату на екран, у реальному часі;

6) створити новий проєкт з потрібними елементами. Реалізувати ділення (**`/`**) двох чисел та виведення результату на екран, у реальному часі;

7) створити новий проєкт з потрібними елементами. Реалізувати додавання (**`+`**) двох чисел та виведення результату на екран, у реальному часі;

8) створити новий проєкт з потрібними елементами. Реалізувати віднімання (**`-`**) двох чисел та виведення результату на екран, у реальному часі;

9) створити новий проєкт з потрібними елементами. Реалізувати будь-яку арифметичну операцію (**`* / + -`**) двох чисел, що задані користувачем у елементах **`TextBox`**. Виконання арифметичних дій (**`* / + -`**) реалізувати у реальному часі;

10) створити новий проєкт з потрібними елементами. Реалізувати будь-яку арифметичну операцію (*** / + -**) двох чисел, що задані користувачем у елементах **TextBox**. Виконання арифметичних дій (*** / + -**) реалізувати так, щоб вони виконувалися при натисканні на керуючу процесом кнопку.

1.2.5 Оформити звіт з роботи.

1.2.6 Відповісти на контрольні питання.

1.3 Зміст звіту

1.3.1 Тема та мета роботи.

1.3.2 Завдання до роботи.

1.3.3 Короткі теоретичні відомості.

1.3.4 Копії екрану та тексти розроблених програм, що відображають результати виконання лабораторної роботи.

1.3.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), котрі відображують результати виконання роботи та їх критичний аналіз.

1.4 Контрольні запитання

1.4.1 Які типи даних підтримуються у C#?

1.4.2 Що таке структура у C#?

1.4.3 Які властивості елементу **Button**?

1.4.4 Які властивості елементу **CheckBox**?

1.4.5 Які властивості елементу **RadioButton**?

1.4.6 Наведіть спільні та відмінні властивості елементів **CheckBox** та **RadioButton**.

1.4.7 Які властивості елементу **GroupBox**?

1.4.8 Які властивості елементу **TextBox**?

1.4.9 Які властивості елементу **RichTextBox**?

1.4.10 Порівняйте елементи **TextBox** та **RichTextBox**.

1.4.11 Списки **ListBox**, **CheckedListBox**, **ComboBox**: загальна характеристика.

1.4.12 Які властивості елементу **ListBox**?

1.4.13 Які властивості елементу **CheckedListBox**?

1.4.14 Які властивості елементу **ComboBox**?

2 ЛАБОРАТОРНА РОБОТА №2
ДРУГОРЯДНІ ЕЛЕМЕНТИ ОБОЛОНКИ ПРОГРАМИ

Мета роботи: вивчити основні прийоми взаємодії з другорядними елементами програми.

2.1 Короткі теоретичні відомості

2.1.1 Мітка – `Label`

– елемент керування **`Label`** призначений для створення підписів до інших елементів керування або для виводу інформаційних повідомлень безпосередньо на поверхні форми. Наприклад, можна поєднувати мітки з полями введення (рис. 2.1).

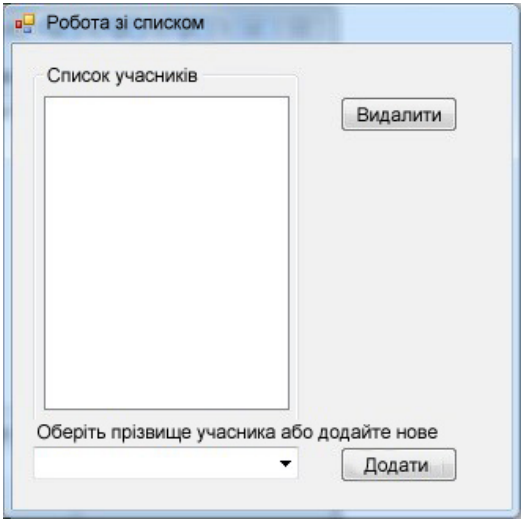


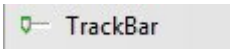
Рисунок 2.1 – Додавання елемента керування **`Label`** до форми застосунку **`Testlists`**

Форма **`Testlistsform`** розширена за рахунок додавання елемента керування **`Label`** з написом «Оберіть прізвище учасника або додайте нове». Це підвищує рівень сприйняття програми користувачем.

2.1.2 Мітка – `LinkLabel`

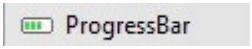
– є гіперпосиланням, якими наповнений Інтернет. Розробники Visual Studio.NET представили цей елемент керування як різновид мітки (елемент керування **`Label`**).

2.1.3 Бігунок – `TrackBar`



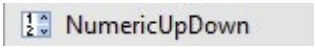
– типовим прикладом застосування елементу **`TrackBar`** є регулятор рівня гучності в панелі Windows. **`TrackBar`** може використовуватися в різних режимах: у горизонтальнім або вертикальнім положенні, із рисками або ні.

2.1.4 Індикатор прогресу – `ProgressBar`



– найчастіше **`ProgressBar`** застосовують для відображення ступеня завершеності тієї або іншої операції.

2.1.5 Регулятор числових значень – `NumericUpDown`



– дозволяє без допомоги клавіатури вводити чисельні значення в поле для введення. Даний елемент керування має три можливості для введення даних: клацання мишкою на покажчики нагору-вниз, використання кнопок нагору-вниз на клавіатурі або введення даних у поле введення.

Для найшвидшого засвоєння інформації про роботу з вищевказаними компонентами, розглянемо приклад. Напишемо програму, у якій бігунок та елемент керування керують індикатором прогресу. Додаткова умова: бігунок та **`NumericUpDown`** повинні працювати синхронно. Тобто, при зміні значення одного елементу, значення іншого повинне змінюватися автоматично на відповідне значення.

Створіть новий **`Windows Forms`** проєкт за назвою **`TestIndicator`**. Перейменуйте файл **`Forms1.cs`** у **`TestIndicatorForm.cs`**. Тепер додайте на форму наступні елементи керування (рис. 2.2):

– **`TrackBar`**;

– **ProgressBar**;
– **NumericUpDown**.

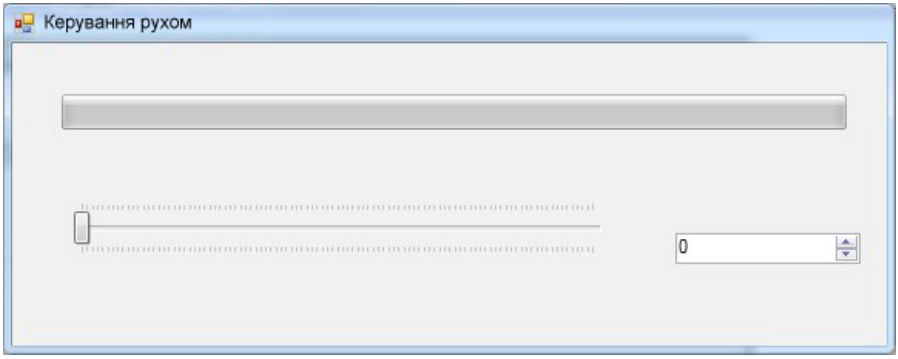


Рисунок 2.2 – Проєктування форми програми **TestIndicator**

Змініть властивості елементів керування. Властивості елементу **TrackBar1**:

`Maximum` – **100**
`TickStyle` – **Both**

При цьому **TrackBar** змінить свій вигляд. Бігунок прийме прямокутну форму, а смужки з'являться й зверху, і знизу від нього. Це результат зміни властивості `TickStyle`. Дана властивість визначає місце розташування рисок елемента керування. У цьому випадку ми обрали значення **Both** (по обидва боки). Крім того, можливі розташування тільки зверху, тільки знизу або взагалі без рисок. Властивості `Minimum` і `Maximum` задають мінімальне й максимальне значення, для яких може змінюватися **TrackBar**. У цьому випадку ми задали максимальне значення 100, а мінімальне 0 (залишили за замовчуванням). Тобто, коли бігунок буде перебувати у крайньому лівому положенні, його значення `Value` буде дорівнювати 0, а при знаходженні бігунка у крайньому правому положенні значення `Value` буде дорівнювати 100.

Властивості об'єкту **NumericUpDown1** залишимо за замовчуванням. Елемент керування **NumericUpDown** також має властивості `Minimum` і `Maximum`. І за замовчуванням, властивість `Minimum` дорівнює 0, а властивість `Maximum` дорівнює 100. Це

відповідає параметрам, встановленим для об'єкту **TrackBar1**. Дуже важлива властивість компонента **NumericUpDown** – DecimalPlaces. Вона визначає кількість знаків після десяткової крапки. У прикладі цю властивість необхідно залишити за замовчуванням – 0, однак при необхідності одержання більшої точності, ніж ціле значення, слід встановлювати значення цієї властивості відповідно до заданої точності.

Змініть значення властивості Text форми на «Керування рухом». Проаналізуйте код програми. Програма відображає візуальний вміст форми у коді мовою C#:

```
using System;
using System.Drawing;
using System.Collections;
using System.Componentmodel;
using System.Windows.Forms;
using namespace TestIndicator
{
//<summary>
// Required description for Form1
//</summary>
public class Form1: System.Windows.Forms.Form {
  private System.Windows.Forms.Trackbar trackbar1;
  private System.Windows.Forms.Progressbar progressbar1;
  private System.Windows.Forms.Numericupdown numericupdown1;
//<summary>
// Required designer variable.
//</summary>
  private System.Componentmodel.Container components = null;
  public Form1 () {
// Required for Windows Form Designer support
    Initializecomponent();
//TODO: Add any constructor code after Initializecomponent call
}
//<summary>
// Clean up any resources being used.
//</summary>
protected override void Dispose ( bool disposing ) {
  if ( disposing ) {
    if (components!= null) {
      components.Disposed();
    }
  }
  base.Dispose ( disposing );
}
# region Windows Form designer generated cede
# endregion
//<summary>
// The main entry point for the application.
```

```
      //</summary>
      [Statread] static void Main () {
        Application. Run (new Form1 () );
      }
    }
  }
```

Елементи **NumericUpDown1** і **TrackBar1** є керуючими, а елемент **ProgressBar1** – керованим. Задамо оброблювачі подій для керування індикатором прогресу. Компонент **TrackBar** має подію `Scroll`, яка призначена для обробки переміщення покажчика бігунка. Створіть функцію-оброблювач для події `Scroll`, клацнувши два рази покажчиком миші по імені події у вікні властивостей. У код програми додасться функція з іменем **TrackBar1_Scroll**. Змініть її код так, як показано нижче:

```
      private void trackBar1_Scroll(object Sender, System.Eventargs
e)    {
        int Value = trackBar1.Value;
        numericUpDown1.Value = Value;
        progressBar1.Value = Value;
      }
```

Тепер, при русі курсору бігунка, буде змінюватися положення індикатору прогресу й значення елемента **NumericUpDown1**. Однак, це ще не повна синхронність роботи елементів, тому що керування повинно відбуватися з двох елементів: бігунка й числового ітератору (**NumericUpDown**), а у нас зараз керування можливе лише від бігунка. Додамо оброблювач події **ValueChanged** для елементу **NumericUpDown1**. Для цього клацніть два рази покажчиком миші по імені події **ValueChanged** у вікні властивостей. У код програми додасться функція з іменем **NumericUpDown1_ValueChanged**. Змініть її вміст аналогічно функції **TrackBar1_Scroll**.

```
      private void numericUpDown1_ValueChanged (object sender,
System.Eventargs e) {
        int Value = (int)numericUpDown1.Value;
        trackBar1.Value = Value;
        progressBar1.Value = Value;
      }
```

Відкомпілюйте та запустіть програму. Спробуйте змінити положення бігунка. При цьому індикатор прогресу та числовий ітератор змінять свої значення на відповідні величини. Спробуйте керувати індикатором прогресу за допомогою числового ітератору. Ефект буде аналогічний роботі з бігунком. Спробуйте знайти середнє положення всіх елементів керування (рис. 2.3).

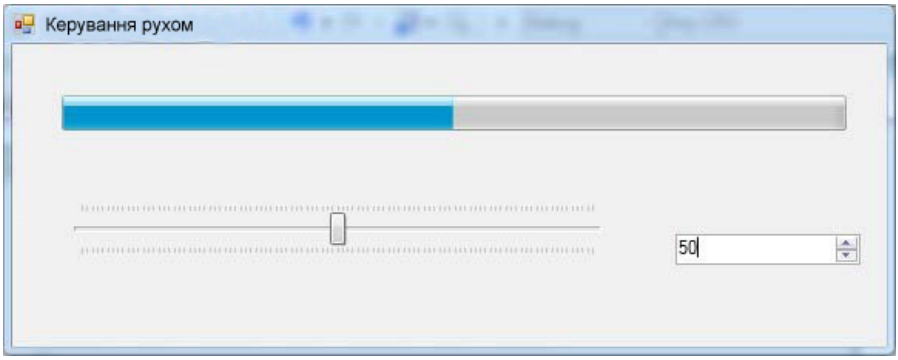


Рисунок 2.3 – Вікно програми «Керування рухом»

2.1.6 Список – `ListView`

– елемент керування, призначений для відображення списків даних. Компонент **`ListView`** відрізняється від **`ListBox`** розширеним списком можливостей: установка піктограм для кожного елементу списку, відображення даних у декількох колонках, декілька стилів розміщення елементів списку. Типовим прикладом використання елемента **`ListView`** є бік праворуч програми «Explorer», яка входить до стандартної поставки Microsoft Windows (рис. 2.4).

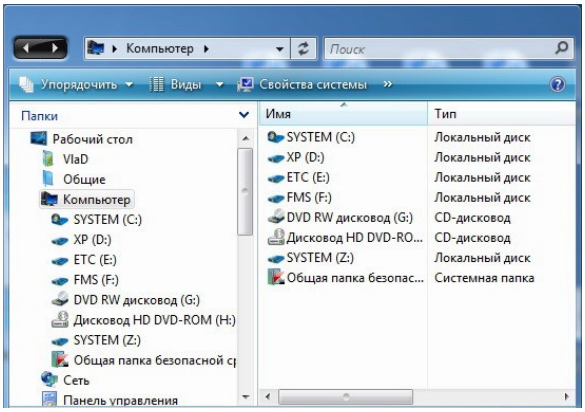


Рисунок 2.4 – Explorer

2.1.7 Дерево – `TreeView`

– призначений для відображення даних у вигляді дерева. Елементи представлення починаються з кореня дерева та відображаються вглиб. Прикладом може бути бік ліворуч програми «Explorer», яка відображає дерево каталогів (рис. 2.4).

Ліворуч вікна розташовується вміст усього диска комп'ютера. Усі елементи списку мають загальний корінь «Desktop», тобто всі елементи є підпунктами одного загального кореня. Праворуч відображається вміст поточної виділеної папки.

Як приклад, створимо програму, яка буде здатна додавати й видаляти елементи у дерево та у список. Головне вікно програми буде містити поле введення даних, дві кнопки: «Додати до списку» та «Додати до дерева», компонент **`ListView`** та компонент **`TreeView`**. У поле введення користувач може увести будь-який рядок. При натисканні на відповідну кнопку, вміст поля переноситься до **`ListView`** або до **`TreeView`**.

Для цього створіть нову C#-Windows-програму під назвою **`ViewApp`**. Додайте на форму наступні елементи (рис. 2.5):

- **`ListView`**;
- **`TreeView`**;
- **`Button`** – два елементи;
- **`TextBox`**.

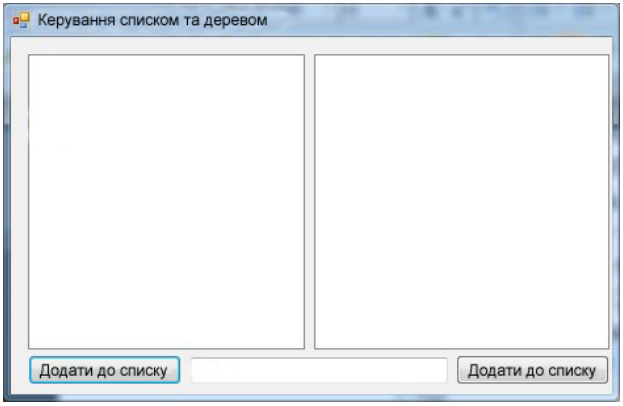


Рисунок 2.5 – Проєктування форми програми **`ViewApp`**

Змініть властивості доданих компонентів:

Button1: text – «додати до списку»

Button2: text – «додати до дерева»

TextBox1:text – «»

Звернемося до коду програми (не наведена ділянка коду, сгенерована дизайнером під час візуальної побудови форми):

```
using System;
using System.Drawing;
using System.Collections;
using System.Componentmodel;
using System.Windows.Forms;
using System.Data;
using namespace Viewapp {
//<summary>
// Summary description for Form1.
//</summary>
public class Form1: System.Windows.Forms.Form {
  private System.Windows.Forms.Listview listview1;
  private System.Windows.Forms.Treeview treeview1;
  private System.Windows.Forms.Textbox textbox1;
  private System.windows.Forms.Button buttonl;
  private System.Windows.Forms.Button button2;
//<summary>
// Required designer variable.
//</summary>
  private system.Componentmodel.container components = null;
  public Forml() {
//
// Required for Windows Form Designer support
//
  Initializecomponent ();
//
// TODO: Add any constructor code after initiaiizecomponent
call
//
}
//<summary>
// clean up any resources being used.
//</summary>
protected override void Dispose ( booi disposing ) {
  if ( disposing ) {
    if (components != null) {
      components.Dispose();
    }
  }
  base. Dispose ( disposing );
}
#region Windows Form Designer generated code
#endregion
//<summary>
```

```
      // The main entry point for the application,
      //</summary>
      [Stathread] static void Main () {
        Application. Run (new Form1());
      }
    }
  }
```

Необхідно наділити код функціональністю. У першу чергу слід додати оброблювач натискання кнопки «Додати до списку». Для цього клацніть два рази курсором миші по кнопці на формі. Перед вами відкриється вікно коду, у якому буде додана функція **`Button1_Click`**. Додайте у функцію **`Button1_Click`** код, представлений нижче:

```
      private void button1_Click(object sender. System.Eventargs e) {
      // індекс виділеного елемента int idx;
      // одержуємо список усіх виділених об'єктів
        Listview.Selectedlndexcollection collection =
listview1.Selectedlndices;
      // якщо виділених об'єктів немає
        if (collection.Count == 0)
          idx = 0;
      // беремо нульовий індекс
      // якщо виділені об'єкти є
        else
      // беремо індекс нульового об'єкта списку
          idx = collection[0];
      // додаємо новий елемент у список
        listview1.Items.Insert(idx, textbox1.Text);
      }
```

Відкомпілюйте та запустіть програму. Елементи розташуються в списку так, як показано на рисунку 2.6. Кожний новий елемент буде доданий у початок списку. При цьому всі попередні змістяться вниз.

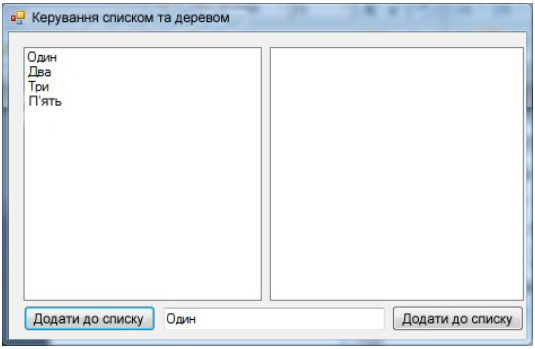


Рисунок 2.6 – Додавання елементів до списку

Тепер оберіть у списку пункт «Три». Наберіть у поле введення «Два». Натисніть кнопку «Додати до списку». Елемент «Два» додасться перед елементом «Три», але після елемента «Один» (там, де він і повинен стояти). Це відбулося тому, що кожний новий елемент додається перед виділеним елементом списку. Оберіть елемент «П'ять», наберіть у поле введення «Чотири». Натисніть кнопку «Додати до списку». Елемент «Чотири» додається між «Три» і «П'ять». Розглянемо код, який змушує програму працювати саме так.

```
ListView.SelectedIndexCollection collection =
ListViewl.SelectedIndices;
```

Клас **ListView** містить властивість `SelectedIndices` зі списком індексів усіх виділених елементів списку. Ви можете обрати відразу кілька елементів у списку, утримуючи натиснутою клавішу Shift або Ctrl.

```
if(collection.Count = 0 )
  idx = 0;
```

Цей код призначений для обробки ситуації, у якій жоден елемент списку не виділений. У такому випадку елемент буде доданий до початку списку.

Якщо ж у списку присутній хоча б один індекс, то новий елемент буде доданий перед найпершим виділеним елементом у списку. Індекс першого виділеного елементу можна одержати за допомогою виразу `collection[0]`.

```
else
  idx = collection[0];
ListView1.Items.Insert(idx, textbox1.Text);
```

Цей рядок коду додає новий елемент до списку. Властивість `ListView.Items` містить колекцію всіх елементів списку. Функція **Insert** дозволяє додати новий елемент до списку. Новий елемент може бути доданий у будь-яке місце списку. Перший параметр **idx** вказує позицію нового елементу у списку. Наступний параметр містить рядок для відображення у списку. Після виконання всіх вищеописаних операцій вікно програми буде виглядати як показано на рисунку 2.7.

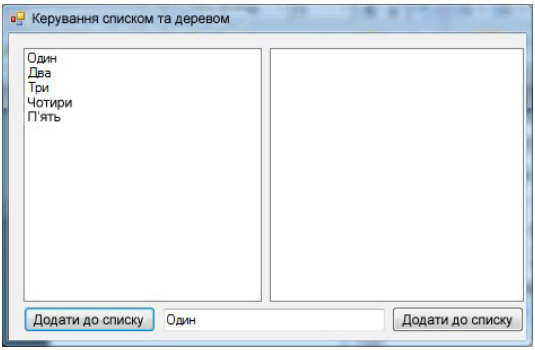


Рисунок 2.7 – Результати роботи зі списком

2.2 Завдання до роботи

2.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

2.2.2 Виконати наступні загальні завдання:

– возведення числа у регульовану ступінь за допомогою **`NumericUpDown`**. При цьому, за допомогою **`ProgressBar`** продемонструвати ступінь виконання операції (возведення у ступінь доцільно виконувати за допомогою операції множення з використанням операторів циклу), результат вивести на екран. Зробити оформлення інтерфейсу програми (при необхідності використовувати додаткові візуальні елементи з відповідними властивостями);

– створити програму для керування двома списками **`ListView`**, надавати можливість видаляти елементи списку, додавати та переміщувати з першого до другого та навпаки. Зробити оформлення інтерфейсу програми.

2.2.3 Виконати наступні індивідуальні завдання (номер завдання відповідає порядковому номеру варіанту):

1) реалізувати програму, що обчислює значення суми будь-якого на вибір студента арифметичного ряду, але унікального серед обраних завдань іншими студентами групи, де аргументи задаються користувачем. Зробити оформлення інтерфейсу програми, продемонструвати ступінь виконання операції;

2) реалізувати рух елементу **Label** по віконній формі програми. Форма руху – за графіком синуса. Зміну положення мітки можна виконувати за допомогою команди: **Label1.Location** = new Point(x,y). Для зміни положення через певний проміжок часу доцільно використовувати об'єкт **Timer**. Зробити оформлення інтерфейсу програми;

3) реалізувати приклад возведення числа у регульовану ступінь за допомогою **NumericUpDown**, при цьому за допомогою **ProgressBar** продемонструвати ступінь виконання операції, та за допомогою **TrackBar** керувати часовою затримкою між возведеннями. Результат виводити на екран у реальному часі. Зробити оформлення інтерфейсу програми;

4) реалізувати програму для обчислення факторіалу довільної величини за схемою індивідуального завдання п.3 до даної лабораторної роботи. Зробити оформлення інтерфейсу програми;

5) реалізувати програму для обчислення факторіалу довільної величини за схемою загального завдання до даної лабораторної роботи. Зробити оформлення інтерфейсу програми;

6) реалізувати обчислення функції $\sin(x)$ для аргументу з діапазону $0 < x < 2\pi$ з кроком $\pi/6$. Результат вивести у **ListView**. Зробити оформлення інтерфейсу програми;

7) реалізувати обчислення функції $\cos(x)$ для аргументу з діапазону $0 < x < \pi$ з кроком $\pi/9$. Результат вивести у **ListView**. Зробити оформлення інтерфейсу програми;

8) реалізувати програму, що обчислює значення добутку будь-якого на вибір студента ряду геометричної прогресії, але унікального серед обраних завдань іншими студентами групи, де аргументи задаються користувачем. Зробити оформлення інтерфейсу програми, продемонструвати ступінь виконання операції;

9) реалізувати програму, яка інтерпретує введене число у текстову форму (словами), дотримуватись технології обробки помилок. Число не повинно бути більше 1000. Зробити оформлення інтерфейсу програми;

10) реалізувати програму для створення таблиці множення з керуванням максимального множника та з виводом у елемент

ListView. Візуалізувати прогрес роботи програми елементом **ProgressBar**. Зробити оформлення інтерфейсу програми.

2.2.4 Оформити звіт з роботи.

2.2.5 Відповісти на контрольні питання.

2.3 Зміст звіту

2.3.1 Тема та мета роботи.

2.3.2 Завдання до роботи.

2.3.3 Короткі теоретичні відомості.

2.3.4 Копії екрану та тексти розроблених програм, що відображають результати виконання лабораторної роботи.

2.3.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

2.4 Контрольні запитання

2.4.1 Які властивості елементу **Label**?

2.4.2 Які властивості елементу **LinkLabel**?

2.4.3 Порівняйте елементи **Label** та **LinkLabel**.

2.4.4 Поясніть призначення елементу **TrackBar**.

2.4.5 Які властивості елементу **TrackBar**?

2.4.6 У яких випадках доцільним є використання елементу **ProgressBar**?

2.4.7 Які властивості елементу **ProgressBar**?

2.4.8 Навіщо призначений елемент **NumericUpDown**?

2.4.9 Які властивості елементу **NumericUpDown**?

2.4.10 Які властивості елементу **ListView**?

2.4.11 Які властивості елементу **TreeView**?

2.4.12 Наведіть спільні та відмінні риси елементів **ListView** та **TreeView**.

2.4.13 Як здійснюється додавання елементу у **ListView**?

2.4.14 Як здійснюється додавання вузлів у **TreeView**?

2.4.15 Як здійснюється пошук елементів у **ListView**?

3 ЛАБОРАТОРНА РОБОТА №3
РОБОТА З РЯДКАМИ

Мета роботи: навчитися працювати з рядками у Visual Studio C#.

3.1 Короткі теоретичні відомості

Інструментальне середовище Visual Studio C# забезпечує вбудовану підтримку роботи зі строками. Більше того, C# оброблює строки як об'єкти, що інкапсулюють усі методи маніпулювання, сортування та пошуку.

3.1.1 Особливості типу `Sytem.String`

Інструментальне середовище Visual Studio C# оброблює строки як вбудовані типи, які є гнучкими, потужними та зручними. Кожен об'єкт рядку – це незмінна послідовність символів **`Unicode`**. Інакше кажучи, ті методи, які змінюють рядки, насправді повертають змінену копію, а первісні рядки залишаються неушкодженими.

Оголошення рядку у C# виконується за допомогою ключового слова **`string`**. У такий спосіб фактично виконується оголошення об'єкту типу **`System.String`**, що є одним із вбудованих типів, який підтримується .NET Framework бібліотекою класів. C#-рядок – це об'єкт типу **`System.String`**.

Оголошення класу **`System.String`** наступне:

```
public sealed class String: Icomparable, Icloneable,
Iconvertible, Ienumerable
```

Таке оголошення говорить про те, що клас запечатаний та неможливо успадкувати свій клас від класу **`String`**. Клас також реалізує чотири системні інтерфейси – `Icomparable`, `Icloneable`, `Iconvertible` і `Ienumerable`, які визначають функціональні можливості **`System.String`** за рахунок його додаткового використання з іншими класами у .NET Framework.

Інтерфейс `Icomparable` визначає тип, що реалізує його як тип, значення якого можуть упорядковуватися. Рядки, наприклад, можуть бути розташовані за абеткою, будь-який рядок можна порівняти з іншими рядками, щоб визначити, який з них повинен

стояти першим в упорядкованому списку. `Icomparable`-класи реалізують метод **`CompareTo`**.

`Ienumerable`-інтерфейс дозволяє використовувати інструкцію **`foreach`**, щоб перебирати елементи рядка як набір символів.

`Icloneable`-об'єкти можуть створювати нові екземпляри об'єктів з тими ж самими значеннями, як і первісний варіант. У цьому випадку можливо клонувати рядок таким чином, щоб створити новий з тим же самим набором символів, як і в оригіналі. `Icloneable`-класи реалізують метод **`Clone()`**.

`Iconvertible`-класи реалізують методи для полегшення перетворення об'єктів класу до інших вбудованих типів, наприклад **`ToInt32(), ToDouble(), ToDecimal()`** та інших.

3.1.2 Створення рядків

Найбільш загальний спосіб створення рядків полягає в тому, щоб установити рядок символів, відомий як строковий літерал, заданий користувачем у змінну типу **`string`**:

```
string newstring = "Новий рядок";
```

Зазначений рядок може містити службові символи типу «**`\n`**» або «**`\t`**», які починаються з похилої риски (**`\`**) та використовуються для вказівки переведення рядку або вставки символу табуляції. Оскільки похила риска ліворуч самостійно використовується в деяких синтаксисах рядків типу URL або шляхів каталогу, то в такому рядку похилій рисі ліворуч повинен передувати інший символ похилої риси ліворуч.

Рядки можуть також бути створені за допомогою дослівного запису рядка. Такі рядки повинні починатися із символу (**`@`**), який повідомляє конструктору **`String`**, що рядок повинен використовуватися дослівно, навіть якщо він містить службові символи. У дослівнім визначенні рядка похилі риси ліворуч та символи, які ідуть за ними, просто розглядаються як додаткові символи рядка. Таким чином, наступні два визначення еквівалентні:

```
string stringone = "\\\\Mysystem\\Mydirectory\\MyFile.txt";
string stringtwo = @"\\Mysystem\MyDirectory\MyFile.txt";
```

У першому рядку використовується звичайний літерал рядка, так що символи похилої риси ліворуч (**`\`**) повинні дублюватися. Це означає, що для відображення (**`\`**) потрібно записати (**`\\`**). У другому

рядку використовується дослівний літеральний рядок, так що додаткова похила риса ліворуч не потрібна.

Наступний приклад ілюструє багаторядкові конструкції:

```
string stringone = "Line One\nLine Two";
string stringtwo = @"Line One
Line Two";
```

Такі оголошення рядків також є взаємозамінними.

3.1.3 System.Object.Tostring()

Інший спосіб створити рядок полягає в тому, щоб викликати в об'єкті метод **ToString()** і встановити результат змінної типу **string**. Усі вбудовані типи мають цей метод, що дозволяє спростити завдання перетворення значення (часто числового значення) до строкового виду. У наступному прикладі викликається метод **ToString()** для типу **int**, щоб зберегти його значення у рядок.

```
int myint = 10;
string intstring = myint.ToString();
```

Виклик методу **ToString()** у об'єкті **myint** поверне строкове представлення числа 10.

Клас **System.String** у .NET підтримує множину перевантажених конструкторів, які забезпечують різноманітні методи для ініціалізації строкових значень різними типами. Деякі із цих конструкторів дають можливість створювати рядок у вигляді масиву символів або у вигляді покажчика на символи. При створенні рядка у вигляді масиву символів створюється екземпляр нового рядка з використанням безпечного коду. При створенні рядка на основі покажчика застосовується «небезпечний» код, що вкрай небажано при розробці застосунків .NET.

3.1.4 Маніпулювання рядками

Клас **string** забезпечує різні вбудовані методи для порівняння, пошуку та керування строковими значеннями. Нижче наведено неповний список можливостей цього класу:

Empty – властивість, що визначає, чи порожній рядок;

Compare() – функція порівняння двох рядків;

CompareOrdinal() – порівнює рядки в незалежності від регіональних настроювань;

Concat() – створює новий рядок із двох і більш вихідних рядків;

Copy() – створює дублікат вихідного рядку;

Equals() – визначає, чи містять два рядки однакові значення;

Format() – форматує рядок, використовуючи строго заданий формат;

Intern() – повертає посилання на існуючий екземпляр рядка;

Join() – додає новий рядок у будь-яке місце вже існуючого рядка;

Chars – індексатор символів рядка;

Length – кількість символів у рядку;

Clone() – повертає посилання на існуючий рядок;

CompareTo() –порівнює один рядок з іншим;

CopyTo() – копіює певне число символів рядку в масив Unicode-символів;

EndsWith() – визначає, чи закінчується рядок певною послідовністю символів;

Insert() – вставляє новий рядок у вже існуючий;

LastIndexOf() – повертає індекс останнього входження елемента в рядок;

PadLeft() – вирівнює рядок по правому краю, пропускаючи всі пробільні символи або інші (спеціально задані);

PadRight() – вирівнює рядок по лівому краю, пропускаючи всі пробільні символи або інші (спеціально задані);

Remove() – видаляє необхідне число символів з рядку;

Split() – повертає підрядок, відділений від основного масиву певним символом;

StartsWith() – визначає, чи починається рядок з певної послідовності символів;

SubString() – повертає підрядок із загального масиву символів;

ToCharArray() – копіює символи з рядка в масив символів;

ToLower() – перетворить рядок до нижнього регістру;

ToUpper() – перетворить рядок до верхнього регістру;

Trim() – видаляє всі входження певних символів на початку й наприкінці рядка;

TrimEnd() – видаляє всі входження певних символів наприкінці рядка;

TrimStart() – видаляє всі входження певних символів на початку рядка.

Розглянемо приклад використання рядків. Для цього напишемо програму, що використовує методи **Compare(), Concat(), Copy(), Insert()** та багато інших. Виведення результатів організуємо за допомогою елементу **TextBox1**:

```
      private void button1_Click(object sender, EventArgs e) {
        string str1 = "абвг";
        string str2 = "АБВГ";
        string str3 = @"C# являє собою інструмент "+
        "швидкого створення застосунків для .NET платформи";
        int result;
      //методи порівняння рядків
      //використовуємо статичну функцію Compare для порівняння
        result = string.Compare (str1, str2);
        TextBox1.Text = "порівнюємо str1: " + str1 + " і str2: " +
str2 + ", результат: " + result + "\r\n";
      //використовуємо функцію Compare з додатковим параметром
      //для ігнорування регістру рядка
        result = string.Compare(str1, str2, true);
        TextBox1.Text = TextBox1.Text + "Порівнюємо без враховування
регістру";
        TextBox1.Text = TextBox1.Text + "str1: " + str1 + " і str2: "
+ str2 + ", результат: " + result + "\r\n";
      //методи об'єднання рядків
      //використовуємо функцію для рядків
        string str4 = string.Concat(str1, str2);
        TextBox1.Text = TextBox1.Text + "Створюємо str4 шляхом " +
"об'єднання str1 і str2:  " + str4 + "\r\n";
      //використовуємо перевантажений оператор
      //для об'єднання рядків
        string str5 = str1 + str2;
        TextBox1.Text = TextBox1.Text + "str5 = str1 + str2:  " +
str5 + "\r\n";
      //використовуємо метод Copy для копіювання рядку
        string str6 = string.Copy(str5);
        TextBox1.Text = TextBox1.Text + "str6 скопійована з str5: " +
str6 + "\r\n";
      //використовуємо перевантажений оператор копіювання
        string str7 = str6;
        TextBox1.Text = TextBox1.Text + "str7 = str6: " + str7 +
"\r\n";
      //кілька способів порівняння
      //використовуючи метод Equals самого об'єкту
        TextBox1.Text = TextBox1.Text + "str7.Equals(str6): " +
str7.Equals(str6) + "\r\n";
      //використовуючи статичний метод Equals
```

```
        TextBox1.Text = TextBox1.Text + "str7 і str6 рівні?: " +
string.Equals(str7, str6) + "\r\n";
      //використовуючи оператор ==
        TextBox1.Text = TextBox1.Text + "str7==str6? " + (str7 ==
str6) + "\r\n";
      //визначення довжини рядка
        TextBox1.Text = TextBox1.Text + "Рядок str7 має довжину " +
str7.Length + "символів" + "\r\n";
      //визначення символу рядку за його індексом
        TextBox1.Text = TextBox1.Text + "П'ятим елементом у рядку
str7 є символ " + str7[4] + "\r\n";
      //порівняння кінця рядка із вхідним екземпляром
        TextBox1.Text = TextBox1.Text + "str3: " + str3 + "чи
Закінчується цей рядок словом \"інструмент\"? " +
str3.EndsWith("інструмент") + "\r\n";
      //порівняння кінця рядка із вхідним екземпляром
        TextBox1.Text = TextBox1.Text + "str3: " + str3 + "чи
Закінчується цей рядок словом \"платформа\"? " +
str3.EndsWith("платформа") + "\r\n";
      //пошук першого входження підрядка у рядку
        TextBox1.Text = TextBox1.Text + "Перше входження слова
інструмент у рядок str3 має індекс " + str3.IndexOf("інструмент")+
"\r\n";
      //вставляємо нове слово до рядку
        string str8 = str3.Insert(str3.IndexOf("додатків"),
"потужних");
        TextBox1.Text = TextBox1.Text + "str8: " + str8;
      }
```

Результат роботи програми буде наступний:

```
порівнюємо str1: абвг і str2: АБВГ, результат: -1
Порівнюємо без враховування регістру str1: абвг і str2: АБВГ,
результат: 0
Створюємо str4 шляхом об’єднання str1 і str2:  абвгАБВГ str5 =
str1 + str2:  абвгАБВГ str6 скопійована з str5: абвгАБВГ str7 = str6:
абвгАБВГ str7.Equals(str6): True str7 і str6 рівні?: True str7==str6?
True
Рядок str7 має довжину 8символів П'ятим елементом у рядку str7
є символ А
str3: C# являє собою інструмент швидкого створення додатків для
.NET платформичи Закінчується цей рядок словом "інструмент"? False
str3: C# являє собою інструмент швидкого створення додатків для
.NET платформичи Закінчується цей рядок словом "платформа"?
False Перше входження слова інструмент у рядок str3 має індекс 15
str8: C# являє собою інструмент швидкого створення
потужнихдодатків для .NET платформи
```

Можна використовувати різні способи оголошення рядків. Для оголошення рядку **str3** використовувалося дослівне представлення рядку. Слід зазначити, що можна розривати рядок у коді програми для переносу його на іншу строку. При цьому необхідно поєднувати

розірвані частини рядка оператором (+). Такий рядок буде сприйматися як єдине ціле.

```
string str3 = @"C# представляє собою інструмент " + "швидкого створення додатків для .NET платформи";
result = string.Compare(str1, str2);
```

У цьому випадку використовується чутлива до регістру функція порівняння двох чисел. Функція порівняння завжди повертає різні значення в залежності від результату порівняння:

– значення менше нуля, якщо перший рядок менше другого;

– 0, якщо рядки рівні;

– значення більше нуля, якщо перший рядок більше другого. У нашому випадку результат буде наступний:

```
порівнюємо str1: абвг і str2: АБВГ, результат: -1
```

Букви нижнього регістру мають менше значення, ніж верхнього, звідси й результат.

У наступній функції **`Compare`** ми використовуємо порівняння без врахування регістру. Про це свідчить додатковий третій параметр функції – `true`.

```
result = string.Compare(str1, str2, true);
TextBox1.Text = TextBox1.Text + "Порівнюємо без враховування регістру";
TextBox1.Text = TextBox1.Text + "str1: " + str1 + " і str2: " + str2 + ", результат: " + result + "\r\n";
```

Відповідно й результат буде:

```
Порівнюємо без враховування регістру str1: абвг, str2:Aebr, результат: 0
```

Функція порівняння без врахування регістру спочатку приводить обидва рядки до загального регістру, а потім здійснює посимвольне порівняння рядків. У підсумку ми одержуємо послідовність дій: абвг, АБВГ, АБВГ = АБВГ ? «ТАК» – результат 0.

Для об'єднання рядків ми використовували дві можливості класу **`string`**. Одна з них – це використання статичної функції **`Concat()`**:

```
string str4 = string.Concat (str1, str2);
```

Другий спосіб – використання оператора (**+**):

```
string str5 = str1 + str2;
```

Оператор (**+**) класу **`string`** перевантажений таким чином, що виконує дію, аналогічну функції **`Concat()`**. Однак, використання запису `str1 + str2` краще читається, тому програмісти звичайно віддають перевагу застосуванню операторів виклику функцій.

Аналогічне порівняння можна провести між функцією **Copy()** та оператором (**=**). Вони виконують ті самі дії – копіюють вміст одного рядка до іншого. Різниця полягає лише в записі коду програми:

```
string str6 = string.Copy(str5);
string str7 = str6;
```

І `str6` та `str7` будуть у результаті мати те значення, яке записано у рядку `str5`.

Клас **string** у C# забезпечує три способи перевірки рівності двох строк. Перший з них – це використання методу **Equals()**.

```
str7.Equals(str6);
```

У цьому випадку для об'єкту `str7` перевіряється рівність йому об'єкту `str6`. Другим варіантом перевірки рівності рядків є використання статичної функції **string.Equals()**.

```
string.Equals(str7, str6);
```

Третій варіант – це використання перевантаженого оператора (**==**):

```
str7 == str6;
```

Кожен із цих викликів повертає `True`, якщо рядки рівні та `False`, якщо рядки не рівні.

Властивість **Length** повертає число символів рядку. А оператор (**[]**) повертає символ рядку, що має відповідний індекс. Наприклад:

```
TextBox1.Text = "П'ятим елементом у рядку str7 є символ" +
str7[4];
```

У цьому рядку програми одержується п'ятий елемент рядку str7, використовуючи число 4 в операторі (**[]**), оскільки елементи рядку в C#, як і у C++, починаються з нульового індексу (табл. 3.1).

Таблиця 3.1 – Індексація у рядках

Рядок	а	б	в	г	А	Б	В	Г
Індекс елементу	0	1	2	3	4	5	6	7
Порядковий номер елементу	1	2	3	4	5	6	7	8

З цієї таблиці видно, що п'ятим символом рядку str7 є символ «А», індекс якого дорівнює 4.

3.1.5 Пошук підрядка

Тип **string** має перевантажений метод **Substring()** для вилучення підрядка із рядка. Один з методів приймає як параметр індекс елементу, починаючи з якого слід витягти підрядок. Другий метод приймає й початковий і кінцевий індекс, щоб вказати, де почати та де закінчити пошук.

Метод **Substring** можна розглянути на наступному прикладі. Програма виводить слова рядка в порядку, зворотному послідовності їх запису:

```
      // оголошуємо рядок для обробки
      string s1 = "Один Два Три Чотири";
      // одержуємо індекс останнього пробілу
      int ix = s1.LastIndexOf(" ");
      // одержуємо останнє слово в рядку
      string s2 = s1.Substring(ix+1);
      // встановлюємо s1 на підрядок, що починається
      // с 0-ого індексу, що й закінчується останнім пробілом
      s1 = s1.Substring(0,ix);
      // знову шукаємо індекс останнього пробілу
      ix = s1.LastIndexOf(" ");
      // встановлюємо s3 на останнє слово рядку
      string s3 = s1.Substring(ix+1);
      // скидаємо si на підрядок
      // від нульового символу до ix
      s1 = s1.Substring(0,ix);
      // скидаємо ix на пробіл
      // між "один" і "два"
      ix = s1.LastIndexOf(" ");
      // встановлюємо s4 на підрядок після
      string s4 = s1.Substring(ix+1);
      // встановлюємо s1 на підрядок від 0 до ix
      // одержуємо тільки слово "один"
      s1 = s1.Substring(0,ix);
      // намагаємося одержати індекс останнього пробілу
      // але цього разу функція Lastlndexof поверне -1
      ix = s1.LastIndexOf(" ");
      // встановлюємо s5 на підрядок починаючи з
      // оскільки ix = 1, s5 встановлюється на початок s1
      string s5 = s1.Substring(ix+1);
      // Виводимо результат
      TextBox3.Text = "s2: " + s2 + "\r\n" + "s3: " + s3 + "\r\n";
      TextBox3.Text = TextBox3.Text + "s4: " + s4 + "\r\n" + "s5: " +
s5 + "\r\n";
      TextBox3.Text = TextBox3.Text + "s1: " + s1 + "\r\n";
```

Спочатку оголошуємо рядок та ініціалізуємо його необхідними параметрами:

```
string s1 = "Один Два Три Чотири";
```

Потім обчислюємо позицію останнього пробілу в рядку. Це необхідно для того, щоб визначити початок останнього слова рядку:

```
int ix = s1.LastIndexOf(" ");
```

У цьому випадку значення ix буде дорівнювати 12. Слово «Чотири» починається з позиції 13. Тепер, коли ми знаємо початок останнього слова рядку, можна витягти його:

```
string s2 = s1.Substring(ix+1);
```

У підсумку **s2** буде рівно «Чотири». Далі ми обрізаємо вихідний рядок **s1** на слово «Чотири». Для цього необхідно викликати функцію **Substring** із двома параметрами – початку та кінця підрядку. Початком рядку в нас буде початок вихідного рядку, а кінцем – індекс останнього пробілу:

```
s1 = s1.Substring(0,ix);
```

Новий рядок буде мати вигляд «Один Два Три». Тепер ми повторюємо ту ж послідовність дій, що й раніше для повного рядку. Одержуємо індекс останнього пробілу, обираємо останнє слово, обрізаємо рядок. Робимо це доти, поки в рядку не залишиться одне слово «Один». Коли ми спробуємо одержати з цього рядка індекс символу пробілу, то функція поверне значення -1:

```
ix = s1.LastIndexOf(" ");
```

Тому, при виклику функції та передачі до неї значення (-1 + 1 = 0), назад повернеться повний вихідний рядок, а саме слово «Один»:

```
string s5 = s1.Substring(ix+1);
```

Результат роботи програми буде наступний:

```
s2: Чотири
s3: Три
s4: Два
s5: Один
s1: Один
```

3.1.6 Розбиття рядків

По суті попередній приклад робив розбір заданого рядку (речення) на слова, зберігав знайдені слова та виводив їх на екран. Більш ефективний розв'язок проблеми, проілюстрованої у попередньому прикладі полягає в тому, щоб використовувати метод **Split()** класу **string**. Метод **Split()** розбирає рядок на підрядки. Для виділення підрядків з вихідного рядку необхідно передати методу **Split()** розділовий символ як параметр. При цьому

результат повернеться у вигляді масиву рядків. Розглянемо роботу методу **Split()** на прикладі:

```
// рядок для аналізу
string s1 = "Один,Два,Три, Рядок для розбору";
// задаємо розділові символи для аналізу
const char cspace = ' ';
const char ccomma = ',';
// створюємо масив розділових символів
// і ініціалізуємо його двома елементами
char[] delimiters = {cspace, ccomma};
string output = "";
int ctr = 1;
// виділяємо підрядки на основі роздільників
// і зберігаємо результат
foreach (string substring in s1.Split(delimiters)) {
  output += ctr++;
  //номер підрядку
  output += ": ";
  output += substring;
  //підрядок
  output += "\r\n";
  //перехід на нову лінію
}
// виведення результату
TextBox4.Text = output;
```

Результатом роботи програми буде текст:

```
1: Один
2: Два
3: Три
4:
5: Рядок
6: для
7: розбору
```

Зверніть увагу, що рядок під номером 4 є порожнім – це не помилка:

```
string s1 = "Один,Два,Три, Рядок для розбору";
```

Ми оголосили рядок **s1**, що містить шість слів. У рядку використовується два типи розділових символів. Один з них символ пробілу, інший – символ (**,**). Між словами «Три» і «Рядок» розташовуються відразу два розділові символи, кома та пробіл.

Далі у програмі оголошуються дві константи, що визначають розділові символи:

```
const char cspace = ' ';
const char ccomma = ',';
```

Ці константи поєднуються до одного масиву:

```
char[] delimiters = {cspace, ccomma};
```

Метод **Split()** може приймати в якості параметру як один символ, так і масив розділових символів. У нашому випадку ми використовуємо як параметр масив розділових символів із двох елементів – символу пробілу та символу коми. Отже, як тільки метод **Split()** виявить у вихідному рядку один із цих символів, то відразу ж помістить у вихідний масив послідовність символів від попереднього роздільника до поточної позиції:

```
foreach (string substring in s1.Split(delimiters))
```

Інструкція `foreach` у цьому випадку буде застосовуватися до всіх елементів результуючого масиву рядків, який поверне функція **Split()**. Результуючий масив буде складатися із семи елементів, одним з яких виявиться порожній рядок. Порожній рядок з'явився через характерну особливість аналізованого рядку. Як було вже відзначено раніше, між словами «Три» та «Рядок» розташовуються відразу два розділові символи – кома і пробіл. Коли метод **Split()** дійде до аналізу символу пробілу між «Три» і «Рядок», він визначить пробіл як розділовий символ. Однак, відразу перед ним перебуває ще один розділовий символ. Отже, між двома розділовими символами розташовується рядок довжиною у **0** символів – порожній рядок, який метод **Split** і поміщує до вихідного масиву.

3.1.7 Клас `StringBuilder`

Клас **StringBuilder** використовується для створення та редагування рядків з динамічного набору даних, наприклад, з масиву байтових значень. Найбільш важливими членами класу **StringBuilder** є:

Capacity – визначає число символів, які здатний зберігати та обробляти **StringBuilder**;

Chars – індексатор класу;

Length – визначає довжину об'єкту **StringBuilder**;

Maxcapacity – визначає максимальне число символів, які здатний зберігати та обробляти **StringBuilder**;

Append() – додає об'єкт заданого типу до кінця **StringBuilder**;

Appendformat() – заміщує або встановлює новий формат **StringBuilder**;

Ensurecapacity() – гарантує, що **StringBuilder** має ємність не менш зазначеної в параметрі;

Insert() – вставляє об'єкт деякого типу до зазначеної позиції;

Remove() – видаляє об'єкт із зазначеної позиції;

Replace() – заміщує всі екземпляри зазначених символів на нові символи.

Дуже важливою особливістю класу **StringBuilder** є те, що при зміні значень в об'єкті **StringBuilder** відбувається зміна значень у вихідному рядку, а не в її копії. Розглянемо приклад використання класу **StringBuilder** для роботи з рядками. Наприклад, для виводу результатів у попередньому прикладі замість класу **string** клас **StringBuilder**:

```
// рядок для аналізу
string s1 = "Один,Два,Три, Рядок для розбору";
// задаємо розділові символи для аналізу
const char cspace = ' ';
const char ccomma = ',';
// створюємо масив розділових символів
// і ініціалізуємо його двома елементами
char[] delimiters = { cspace, ccomma };
StringBuilder output = new StringBuilder();
int ctr = 1;
// виділяємо підрядки на основі роздільників
// і зберігаємо
foreach (string substring in s1.Split(delimiters)) {
//Appendformat додає рядок певного формату
  output.AppendFormat("{0}: {1}\r\n", ctr++, substring);
}
// виведення результату
TextBox5.Text = output.ToString();
```

Як можна переконатись, зміни відбулися лише у нижній частині коду програми. Робота з розбору рядка відбувається як завжди, змінилось лише виведення результатів. Для виведення результатів використовувався клас **StringBuilder**:

```
StringBuilder output = new StringBuilder();
```

Для об'єктів класу **StringBuilder** завжди потрібно явно викликати конструктор, оскільки цей тип не відноситься до вбудованих типів. При збереженні результатів обробки рядку використовується метод **AppendFormat()**. Даний метод дозволяє відформатувати рядок необхідним форматом. У прикладі передається як формат рядку два значення, розділені символом двокрапки. Перше значення – це номер підрядка, друге – саме підрядок:

```
output.AppendFormat("{0}: {1}\r\n", ctr++, substring);
```

При виведенні результату не потрібно проводити додаткове форматування тексту, оскільки відразу результат заноситься у потрібному форматі, проте необхідно перетворити формат рядку **StringBuilder** до звичайного формату рядків щодо їх виведення у текстовому полі:

```
TextBox5.Text = output.ToString();
```

Результат роботи програми залишається таким же, як раніше:

```
1: Один
2: Два
3: Три
4:
5: Рядок
6: для
7: розбору
```

3.1.8 Регулярні вирази

Регулярні вирази – це один зі способів пошуку підрядків (відповідностей) у рядках. Здійснюється за допомогою перегляду рядку в пошуках деякого шаблону. Загальновідомим прикладом можуть бути символи «*****» та «**?**», що використовуються у командному інтерпретаторі операційної системи **DOS** або **UNIX**, чи в командному рядку інтерпретації **DOS** новітніх операційних систем. Перший із символів «*****» замінює нуль або більше довільних символів, другий же «**?**» замінює тільки один довільний символ. Так, використання шаблону пошуку типу «**text?.***» знайде файли **textf.txt**, **textl.asp** та інші аналогічні, але не знайде **text.txt** або **text.htm**. Якщо у командному інтерпретаторі операційної системи **DOS** або **UNIX**, чи в командному рядку інтерпретації **DOS** новітніх операційних систем використання регулярних виразів було вкрай обмеженим, то в інших програмних застосуваннях, зокрема в сучасних операційних системах та мовах програмування, вони знайшли широке використання.

Застосування регулярних виразів дає значне збільшення продуктивності, оскільки бібліотеки, що інтерпретують регулярні вирази, як правило, пишуться на низькорівневих високопродуктивних мовах (C, C++, Assembler).

3.1.8.1 Застосування регулярних виразів

Як правило, за допомогою регулярних виразів виконуються три дії:

− перевірка наявності відповідного до шаблону підрядка;

− пошук та видача користувачеві відповідних до шаблону підрядків;

− заміна відповідних до шаблону підрядків.

У C# робота з регулярними виразами виглядає наступним чином:

```
Regex re = new Regex("зразок","опції");
MatchCollection mc = re.Matches("рядок для пошуку");
icountmatchs = inc.Count;
```

re – це об'єкт типу **Regex**. У конструкторі йому передається зразок пошуку та опції. Нижче наведено таблицю 3.2, де є опис різних варіантів пошуку.

Таблиця 3.2 – Значення регулярних виразів

Символ	Значення
i	Пошук без врахування регістру.
m	Багаторядковий режим, що дозволяє знаходити збіги на початку або кінці рядка, а не всього тексту.
n	Знаходить тільки явно іменовані або нумеровані групи у формі (**?<name>...**).
c	Компілює. Генерує проміжний Msil-Код, перед виконанням змінюється у машинний код.
s	Дозволяє інтерпретувати кінець рядка як звичайний символ-роздільник.
x	Виключає із зразку неприкриті незначні символи (пробіли, табуляцію та інші).
r	Виконує пошук праворуч – ліворуч.

Комбінація прапорців **m** та **s** дає дуже зручний режим роботи, що враховує кінці рядків та дозволяє пропустити всі незначні символи, включно символ кінця рядка.

Варто відзначити, що для можливості використання регулярних виразів необхідно використовувати модуль **RegularExpressions**, що входить до пакету **System.Text**:

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

3.1.8.2 Класи символів (Character classes)

Виразом може бути один символ або послідовність символів, укладених у круглі або квадратні дужки.

Використовуючи квадратні дужки, можна вказати групу символів (це називають класом символів) для пошуку. Наприклад, конструкція «**б[ai]ржа**» відповідає словам «**баржа**» і «**біржа**», тобто словам, що починаються з «**б**», за яким ідуть «**a**» або «**i**» та закінчуються на «**ржа**».

Можливо й зворотне, тобто можна вказати символи, які не повинні бути у знайденому підрядку. Так, вираз **'[^1-6]'** знаходить усі символи крім цифр від **1** до **6**. Слід згадати, що усередині класу символів **'\b'** позначає символ **backspace** (стирання).

3.1.8.3 Квантифікатори (Quantifiers)

Якщо невідомо, скільки саме знаків повинен містити шуканий підрядок, можна використовувати спецсимволи, що називаються квантифікаторами (**Quantifiers**).

Наприклад, можна написати «**hel+o**», що буде означати слово, яке починається з «**he**», з наступними за ним одним або декількома «**l**», що й закінчується на «**o**». Слід розуміти, що квантифікатор ставиться до попереднього виразу, а не до окремого символу.

Повний список квантифікаторів наведено у таблиці 3.3.

Таблиця 3.3 – Опис квантифікаторів

Символ	Опис
*****	Відповідає **0** або більшій кількості входжень попереднього виразу. Наприклад, **'zo*'** відповідає "**z**" та "**zoo**".
+	Відповідає **1** або більшій кількості попередніх виразів. Наприклад, **'zo+'** відповідає "**zo**" та "**zoo**", але не "**z**".
?	Відповідає **0** або **1** попередніх виразів. Наприклад, **'do(es)?'** відповідає "**do**" у "**dorf**" або "**does**".
{n}	**n** – позитивне ціле. Відповідає точній кількості входжень. Наприклад, **'o{2}'** не знайде "**o**" у "**Bob**", але знайде два "**o**" у "**food**".
{n,}	**n** – позитивне ціле. Відповідає входженню, повтореному не менш **n** разів. Наприклад, **'o{2,}'** не знаходить "**o**" у "**Bob**", проте знаходить усі "**o**" у "**fooooood**"; **'o{1,}'** еквівалентно **'o+'**; **'o{0,}'** еквівалентно **'o*'**.
{n,m}	**n** та **m** – позитивні цілі числа, де **n<=m**. Відповідає мінімум **n** та максимум **m** входжень. Наприклад, **'o{1,3}'** знаходить три перші "**o**" у "**fooooood**". **'o{0,1}'** еквівалентно **'o?'**.

3.1.8.4 Закінчення та початки рядків

Перевірка початку або кінця рядку проводиться за допомогою метасимволів **^** та **$**. Наприклад, «**^thing**» відповідає рядку, що починається з «**thing**», а «**thing$**» відповідає рядку, що закінчується на «**thing**». Ці символи працюють тільки при опції **'s'**. Без опції **'s'** знаходять тільки кінець та початок тексту. Але можна знайти кінець та початок рядку, використовуючи **escape**-послідовності **\A** та **\Z**.

3.1.8.5 Границя слова

Для завдання меж слова призначені метасимволи **'\b'** та **'\B'**:

```
Regex re = new Regex("мен","ms");
```

У цьому випадку зразок у **re** відповідає не тільки «мені» у рядку «дайте мені», але й «мені» у рядку «відбулася замена». Щоб уникнути цього, можна зробити зразок маркером границі слова:

```
Regex re = new Regex("\bмен","ms");
```

Тепер буде знайдено тільки «мен» на початку слова. Не варто забувати, що усередині класу символів **'\b'** – (стирання).

Наведені нижче у таблиці 3.4 метасимволи не змушують машину регулярних виразів просуватися по рядку або захоплювати символи. Вони просто відповідають певному місцю рядка. Наприклад, визначає, що поточна позиція – початок рядку. **'FTP'** повертає тільки ті «**FTP**», що перебувають на початку рядку.

Таблиця 3.4 – Значення метасимволів.

Символ	Значення
^	Початок рядку.
$	Кінець рядку, або перед **\n** наприкінці рядку (див. опцію **m**).
\A	Початок рядку (ігнорує опцію **m**).
\Z	Кінець рядку, або перед **\n** наприкінці рядку (ігнорує **m**).
\z	Точно кінець рядку (ігнорує опцію **m**).
\G	Початок поточного пошуку (часто це в одному символі за кінцем останнього пошуку).
\b	На границі між **\w** (алфавітно-цифровими) та **\W** (неалфавітно-цифровими) символами. Повертає **true** на перших та останніх символах слів, розділених пробілами.
\B	Не на **\b**-границі.
\w	Слово. Те ж, що й **[a-za-z_0-9]**.
\W	Усі, крім слів. Те ж, що й **[Aa-za-Z_0-9]**.
\s	Будь-яке порожнє місце. Те ж, що й **[\f\n\r\t\v]**.
\S	Будь-яке непусте місце. Те ж, що й **[\f\n\r\t\v]**.
\d	Десяткова цифра. Те ж, що й **[0-9]**.
\D	Не цифра. Те ж, що й **[0-9]**.

3.1.8.6 Варіації та групування. Правила побудови регулярних виразів

Символ **'|'** можна використовувати для перебору декількох варіантів. Використання його разом з дужками – **'(…|...|...)'** – дозволяє створити групи варіантів. Дужки використовуються для «захвату» підрядків щодо подальшого використання та збереження їх у вбудованих змінних **$1, $2,..., $9**.

```
Regex re = new Regex("like (apples|pines|bananas)");
Matchcollection mc = re.Matches("I like apples a lot");
```

Такий приклад буде працювати та знайде послідовність «**like apples**», оскільки «**apples**» – один із трьох перерахованих варіантів. Дужки також помістять «**apples**» до **$1** як зворотне посилання для подальшого використання. В основному це має сенс при заміні.

Правила побудови регулярних виразів:

– будь-який символ позначає себе самого, якщо він не є метасимволом. Якщо потрібно скасувати дію метасимволу, то треба поставити перед ним **'\'**;

– рядок символів позначає рядок цих символів;

– множина можливих символів (група) поміщується у квадратні дужки **'[]'**, це значить, що в даному місці може стояти один із зазначених у дужках символів. Якщо першим символом у дужках є символ **'^'**, то це означає, що жоден із зазначених символів не може стояти в даному місці виразу. Усередині групи можна вживати символ **'-'**, що позначає діапазон символів. Наприклад, **a-z** – одна з малих букв латинського алфавіту, **0-9** – цифра та інше;

– усі символи, зокрема і спеціальні, можна позначати за допомогою **'\'**;

– альтернативні послідовності розділяються символом **'|'**. Проте усередині квадратних дужок це звичайний символ.

Деякі приклади використання регулярних виразів наведено у таблиці 3.5.

Таблиця 3.5 – Приклади використання

Регулярний вираз	Опис
s/(\S+)(\s+)(\S+)/$3$2$1/	Перестановка двох перших слів.
m/(\w+)\s*=\s*(.*?)\s*$/	Пошук пар **name=value**. Тут ім'я зберігається – в **$1**, а значення – в **$2**.
m/(\d{4})-(\d\d)-(\d\d)/	Читання дати у форматі YYYY-MM-DD. YYYY – в **$1**, MM – в **$2**, DD – в **$3**.
m/^.*(\\|\/)	Виділення шляху з імені файлу. Наприклад, у "**Y:\KS\regExp\!.Net\Compilation\ms-6D(1).tmp**" – такий вираз знайде: "**Y:\KS\regExp\!.Net\Compilation**".
("(\\"|\\\\\|[^"])*"|/*./|//[^\r]*|#\S+|\b(new|static char|const)\b)**	У застосуванні до файлу з текстом програми на C++, виділяє коментарі, рядки та ідентифікатори "**new**", "**static char**" та "**const**".
<\s*a("[^"]*"|[^>])*>	Виділяє тег **<a href="…">** у HTML-коді.

3.1.9 Використання регулярних виразів Regex

У програмуванні .NET-технологія забезпечує об'єктно-орієнтований підхід до обробки регулярних виразів.

Бібліотека класів для обробки регулярних виразів ґрунтується на просторі імен **System.Text.Regularexpressions**. Головним класом для обробки регулярних виразів є клас **Regex**, який являє собою компілятор регулярних виразів. Крім того, **Regex** забезпечує безліч корисних статичних методів. Використання **Regex** проілюстроване у наступному прикладі:

```
string s1 = "Один,Два,Три, Рядок для розбору";
Regex theregex = new Regex(" |, |,");
StringBuilder sbuilder = new StringBuilder();
int id = 1;
foreach (string substring in theregex.Split(s1)) {
  sbuilder.AppendFormat("{0}: {1}\r\n", id++, substring);
}
TextBox1.Text = sbuilder.ToString();
```

Результат роботи програми буде представлений як:

```
1: Один
2: Два
3: Три
4: Рядок
5: для
6: розбору
```

Як видно у цьому прикладі рядком для аналізу є: «Один, Два, Три, Рядок для розбору». Оголошено як **s1**. Далі було оголошено регулярний вираз, який задає зразок-роздільник для пошуку:

```
Regex theregex = new Regex(" |, |,");
```

У якості зразку використовується вираз, об'єднаний оператором **АБО**. Шукається або знак пробілу, або кома, або якщо йдуть підряд кома та пробіл. Рядок, що задає регулярний вираз, передається в якості параметра конструктору об'єкта **theregex**.

Клас **Regex** містить метод **Split**, який по своїй дії нагадує метод **string.Split**. Він повертає масив рядків як результат пошуку регулярного виразу в рядку. У тілі циклу вже звичним способом, за допомогою класу **StringBuilder**, формується вихідний рядок.

Метод **Split** класу **Regex** перевантажений та може використовуватися у двох варіантах. Перший з них було надано у попередньому прикладі. Другий спосіб – використання виклику статичного методу:

```
string s1 = "Один, Два, Три, Рядок для розбору";
StringBuilder sbuilder = new StringBuilder();
int id = 1;
foreach (string substring in Regex.Split(s1," |, |,")) {
  sbuilder.AppendFormat("{0}:{1}\r\n", id++, substring);
}
TextBox2.Text = sbuilder.ToString();
```

Результатом роботи програми є той же вихідний масив рядків. Відмінність цього прикладу від попереднього полягає в тому, що не довелося створювати екземпляр об'єкту **Regex**. У цьому випадку для пошуку використовувався статичний метод **Regex.Split**, який ухвалює два параметри – рядок, у якому буде проводитися пошук та рядок з регулярним виразом.

Крім цього, метод **Split** перевантажений ще двома варіантами, які дозволяють обмежити кількість раз використання методу **Split** та задати початкову позицію у рядку для пошуку.

3.1.10 Використання `Match`-колекцій

Простір імен містить два класи, які дозволяють здійснювати ітераційний пошук у рядку та повертати результат у вигляді колекції. Колекція повертається у вигляді об'єкту типу **`MatchCollection`**, що містить об'єкти типу **`Match`**. Об'єкт **`Match`** містить у собі дві дуже важливі властивості, які визначають його довжину (`Length`) та значення (`Value`). Надалі буде розглянуто приклад використання ітераційного пошуку:

```
string s1 = "Це рядок для пошуку";
// знайти будь-який пробільний символ
// наступний за непробільним
Regex thereg = new Regex(@"(\S+)\s");
// одержати колекцію результату пошуку
MatchCollection thematches = thereg.Matches(s1);
// перебір усієї колекції
foreach (Match thematch in thematches) {
  TextBox3.Text = TextBox3.Text + "thematch.Length: " +
thematch.Length + "\r\n";
  if (thematch.Length != 0) {
    TextBox3.Text = TextBox3.Text + "thematch: " +
thematch.ToString() + "\r\n";
  }
}
```

Результат роботи програми:

```
thematch.Length: 3
thematch: Це
thematch.Length: 6
thematch: рядок
thematch.Length: 4
thematch: для
```

Розглянемо програму докладніше. Створюємо простий рядок для пошуку:

```
string s1 = "Це рядок для пошуку";
```

Потім формуємо регулярний вираз:

```
Regex thereg = new Regex(@"(\S+)\s");
```

Це приклад найпростішого регулярного виразу. Метасимвол (**`\S`**) означає будь-який не пробільний символ. Знак (**`+`**), що стоїть після (**`\S`**) означає, що може бути будь-яка кількість не пробільних символів. Метасимвол (**`\s`**) у нижньому регістрі – пробільний символ. У цілому, цей вираз означає: «Знайти всі набори, які починаються з не пробільного символу але закінчуються пробільним».

Результат програми відображає лише три перші слова вихідного рядку. Четверте слово не увійшло в результуючу колекцію, тому що після нього немає пробільного символу. Якщо додати пробіл наприкінці речення, то слово «пошуку» також буде знайдено та виведено в якості результату.

3.2 Завдання до роботи

3.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

3.2.2 Виконати наступні загальні завдання:

– розробити архіватор: символи строки, що повторюються, заміняти на послідовність – {СимволЧислоПовторювань}, наприклад: «fehhh eryaaa» повинна перетворитися на строку виду «feh3 erya3». Також реалізувати зворотну функцію програми;

– розробити аналізатор: рядок, що вводиться, інтерпретується програмою, яка виконує потрібні дії, задані користувачем у рядку. Реалізувати прості арифметичні операції (**/ * – +**). Наприклад, при введенні строки «2 плюс 5» або «2 + 5», результатом виконання програми повинно бути – «7».

3.2.3 Виконати наступні індивідуальні завдання (номер завдання відповідає порядковому номеру варіанту):

1) продемонструвати роботу таких функцій: Empty, Length, Split;

2) продемонструвати роботу таких функцій: Compare, Clone, StartsWith;

3) продемонструвати роботу таких функцій: CompareOrdinal, CompareTo, Substring;

4) продемонструвати роботу таких функцій: Concat, CopyTo, ToCharArray;

5) продемонструвати роботу таких функцій: Copy, EndsWith, ToLower;

6) продемонструвати роботу таких функцій: Equals, Insert, ToUpper;

7) продемонструвати роботу таких функцій: Format, LastIndexOf, Trim;

8) продемонструвати роботу таких функцій: Intern, PadLeft, TrimEnd;

9) продемонструвати роботу таких функцій: Join, PadRight, TrimStart;

10) продемонструвати роботу таких функцій: Chars, Remove, Insert.

3.2.4 Оформити звіт з роботи.

3.2.5 Відповісти на контрольні питання.

3.3 Зміст звіту

3.3.1 Тема та мета роботи.

3.3.2 Завдання до роботи.

3.3.3 Короткі теоретичні відомості.

3.3.4 Копії екрану та тексти розроблених програм, що відображають результати виконання лабораторної роботи.

3.3.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

3.4 Контрольні питання

3.4.1 Які основні функції роботи з рядками?

3.4.2 Які основні функції порівняння рядків?

3.4.3 Які основні функції модифікаторів рядка?

3.4.4 Які основні функції пошуку підрядка?

3.4.5 Що таке клас **`StringBuilder`**?

3.4.6 Що таке «регулярний вираз»?

3.4.7 Наведіть переваги та недоліки роботи з регулярними виразами.

3.4.8 Які основи роботи з регулярними виразами?

3.4.9 Що таке «квантифікатор»?

3.4.10 Наведіть приклад використання варіацій та групування у регулярних виразах.

3.4.11 Користуючись чим можна знайти потрібний підрядок у рядку?

3.4.12 Які метасимволи використовуються у регулярних виразах для завдання границь слова?

3.4.13 Який клас є головним для обробки регулярних виразів?

3.4.14 Як здійснюється видалення частини рядка?

3.4.15 Поясніть призначення **`Match`**-колекцій.

4 ЛАБОРАТОРНА РОБОТА №4
РОБОТА З ФАЙЛАМИ

Мета роботи: вивчити основні принципи роботи з файлами у середовищі Visual Studio C#.

4.1 Короткі теоретичні відомості

Користувач повинен мати можливість зберегти результати своєї роботи на диск, а потім прочитати їх. Інакше усі напрацювання будуть загублені при виході з програми. Файли можуть бути записані у каталоги, а каталоги – вкладені один в одного. Мова C# дає змогу програмістам легко та просто зберігати і зчитувати дані з диску.

4.1.1 Поняття потоків

В основі роботи з файлами лежить поняття потоків. Потік асоціюється з файлом і надає набір методів для доступу до файлу через потік. Потоки мають розширені функціональні можливості у порівнянні з файлами. Потоки дозволяють записувати та зчитувати структури даних, масиви, інші потоки. Хоча потік і асоціюється з файлом, не всі дані з потоку прямо попадають у файл. Уся інформація з потоку заноситься до буферу, і лише при виклику певних команд переноситься до файлу.

Основними класами для роботи з файлами та потоками у Visual Studio C# є **`File`**, **`Filestream`** та **`StreamReader`**. Клас **`File`** призначений для створення, відкриття, видалення, зміни атрибутів файлу. Клас **`Filestream`** призначений для зчитування та запису інформації до файлу. Об’єкти цих класів працюють у парі один з одним. Механізм їх взаємодії дуже простий і зрозумілий.

Для роботи з текстовими файлами необхідно створити об’єкт типу **`Filestream`** та проініціалізувати його відкритим файлом. Оскільки всі методи класу **`File`** є статичними (не прив’язані до об’єктів), то немає необхідності створювати екземпляр класу **`File`**. Типовий приклад ініціалізації об’єкту **`Filestream`**:

```
Filestream mystream = File.Open("D:\MyFile.txt", Filemode.Open, Fileaccess.Read);
```

У якості додаткового інструменту для роботи з текстовими файлами розробниками Visual Studio C# було створено класи

StreamReader та **StreamWriter**. Вони дозволяють читати та писати дані з потоку построково, посимвольно або відразу все. **StreamReader** та **StreamWriter** зв'язуються з потоком за допомогою конструктору ініціалізації:

```
StreamReader reader = new StreamReader(mystream);
StreamWriter writer = new StreamWriter(mystream);
```

4.1.2 Атрибути відкриття файлів

При відкритті файлу завжди необхідно вказувати режим відкриття файлу та права доступу до файлу. У цьому випадку режим відкриття встановлений як **Filemode.Open**, що означає відкрити файл, якщо він існує. Права доступу встановлені **Fileaccess.Read**, що означає можливість тільки читати файл. Функція **Open** повертає об'єкт типу **Filestream**, за допомогою якого надалі відбуваються читання або запис до файлу.

4.1.3 Діалоги відкриття та збереження файлів

На рисунку 4.1 наведено приклад діалогу для відкриття або запису файлу.

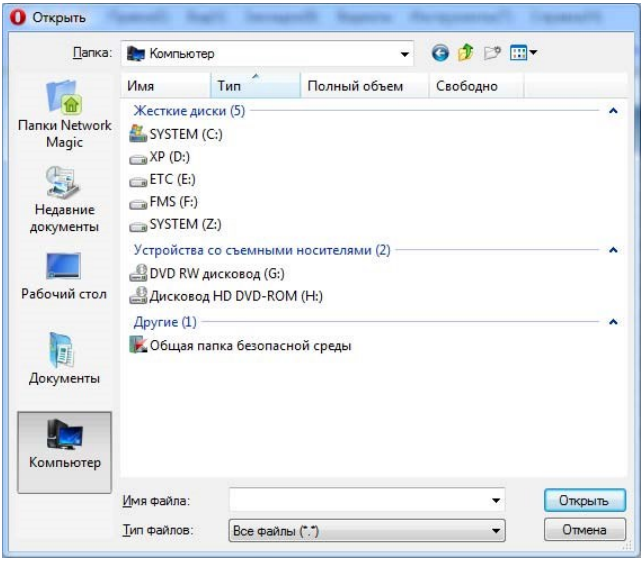


Рисунок 4.1 – Діалог відкриття файлу

Для роботи з діалогами відкриття та збереження файлів використовуються компоненти **Openfiledialog** та **Savefiledialog**. Для відображення діалогу відкриття файлу необхідно лише створити об'єкт класу **Openfiledialog** та викликати його метод **Showdialog**. Після закриття діалогу властивість `Filename` зберігає ім'я обраного файлу та повний шлях до нього.

Як приклад, створимо найпростіший текстовий редактор, що дозволяє читати текстові файли, редагувати інформацію та зберігати її у файлі.

Створіть новий застосунок з іменем **Fileapp**. Перейменуйте властивість `Text` форми у «Текстовий редактор». Помістіть на форму компонент **TextBox** та змініть його властивості:

```
Text – «»;
Multiline – True;
Dock – Fill.
```

При цьому **TextBox** повинен розтягтися на весь екран. Властивість `Multiline` дозволяє елементу вводити текст у кілька рядків, а властивість `Dock` визначає положення елемента на формі. Якщо властивість `Dock` встановлене у `Fill`, то елемент займе всю площу форми.

Помістіть на форму компонент **Mainmenu**. Створіть у ньому один пункт «Файл» із двома підпунктами «Відкрити» та «Зберегти». Змініть властивість `Name` пунктів «Відкрити» та «Зберегти» на **menuitemopen** і **menuitemsave**. Створіть оброблювачі для пунктів меню «Відкрити» та «Зберегти». Залиште для них імена за замовчуванням, клацнувши два рази покажчиком миші по відповідним пунктам меню. При цьому до коду програми повинні були додатися методи **menuitemopenclick** та **menuitemsaveclick**.

Додайте на форму компоненти **Openfiledialog** та **Savefiledialog**. Для обох встановіть властивість `Filter` як «**Text files (*.txt)!*.txt**». Це означає, що в діалозі будуть показуватися тільки файли з розширенням «**txt**».

Замінить оброблювачі відкриття та збереження файлів так, як показано нижче:

```
      private void menuitemopenclick (object sender, System.Eventargs
e) {
      // показуємо діалог вибору файлу
        openfiledialog1.Showdialog();
      // одержуємо ім'я файлу
        string filename = openfiledialog1.Filename;
      // відкриваємо файл для читання й асоціюємо з ним потік
        Filestream stream = File.Open(filename, Filemode.Open,
Fileaccess.Read);
      // якщо файл відкритий
        if(stream != null) {
      // створюємо об'єкт Streamreader і асоціюємо
      // його з відкритим потоком
         StreamReader reader = new StreamReader(stream);
      // читаємо весь файл і записуємо в TextBox
         TextBox1.Text = reader.Readtoend();
      // закриваємо файл
         stream.Close();
        }
      }
      private void menuItemSave_Click (object sender,System.EvoidArgs
e) {
      // показуємо діалог вибору файлу
        savefiledialog1.Showdialog();
      // одержуємо ім'я файлу
        string filename = savefilediaiog1.Filename;
      // відкриваємо файл для запису й асоціюємо з нім потік
        Filestreem.stream = File.Open(filename, Filemode.Create,
Fileaccess.Write);
      // якщо файл відкритий
        if(stream != null) {
      // створюємо об'єкт Streamwriter і асоціюємо
      // його з відкритим потоком
         StreamWriter writer = new StreamWriter(stream);
      // записуємо дані в потік
         writer.Write(TextBox1.Text);
      // переносимо дані з потоку у файл
         writer.Flush();
      // закриваємо файл
         stream.Close();
        }
      }
```

Робота із читанням файлу відбувається у 6 етапів:

– відкриття файлу;

– асоціація файлу з потоком;

– асоціація потоку із **`StreamReader`**;

– читання даних;

– перенесення даних до **`TextBox`**;

– закриття файлу.

Запис файлу також проходить у 6 етапів:
− відкриття файлу;
− асоціація файлу з потоком;
− асоціація потоку із **`StreamWriter`**;
− запис даних;
− звільнення потоку;
− закриття файлу.

Запустіть програму на виконання. Оберіть пункт «Відкрити». У діалозі, що відкрився (рис. 4.1) оберіть текстовий файл. Після натискання кнопки «ОК» дані з файлу відобразяться у вікні програми. Змініть текст файлу. Натисніть меню «Зберегти». У вікні, що відкрилося, виберіть нове ім'я файлу, щоб не затерти старий файл. Після натискання кнопки «ОК» дані із програми перенесуться до файлу. Ви можете переконатися у цьому, скориставшись програмою «Блокнот».

4.2 Завдання до роботи

4.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи використовуючи ці методичні вказівки, а також рекомендовану літературу.

4.2.2 Вивчити основні принципи роботи з файлами.

4.2.3 Виконати завдання, аналогічні завданням лабораторної роботи №3 з доповненням: додати зберігання та завантаження останнього виду програми у файлі. Порядок зберігання та завантаження інформації у файли реалізувати одним з наступних способів:

1) зберігати кожну сесію роботи програми, нову сесію дописувати у початок файлу. Під час роботи програми реалізувати завантаження збереженої сесії;

2) зберігати кожну сесію роботи програми, нову сесію дописувати у кінець файлу. Під час роботи програми реалізувати завантаження збереженої сесії;

3) зберігати кожну сесію роботи програми, нову сесію дописувати у задану позицію файлу, що вводиться у налаштуваннях програми та зберігається у окремому файлі. Під час роботи програми реалізувати завантаження збереженої сесії;

4) зберігати кожну сесію роботи програми, нову сесію дописувати у початок файлу використовуючи діалог вибору файлу. Під час роботи програми реалізувати завантаження збереженої сесії;

5) зберігати кожну сесію роботи програми, нову сесію дописувати у кінець файлу. Під час роботи програми реалізувати завантаження збереженої сесії використовуючи діалог вибору файлу.

4.2.4 Оформити звіт з роботи.

4.2.5 Відповісти на контрольні питання.

4.3 Зміст звіту

4.3.1 Тема та мета роботи.

4.3.2 Завдання до роботи.

4.3.3 Короткі теоретичні відомості.

4.3.4 Копії екрану та тексти розроблених програм, що відображають результати виконання лабораторної роботи.

4.3.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

4.4 Контрольні питання

4.4.1 Що таке файловий потік?

4.4.2 Які класи є основними для роботи з файлами та потоками у Visual Studio C#?

4.4.3 Поясніть призначення та використання об’єкту типу **`Filestream`**.

4.4.4 Які атрибути існують щодо відкриття файлів?

4.4.5 Як обмежити типи файлів для показу у діалозі відкриття файлів?

4.4.6 Як прочитати файл?

4.4.7 Як записати файл?

4.4.8 Які відмінності роботи з файлами у Visual Studio C# та GNU C++?

4.4.9 Як керувати переміщенням каретки у файлі?

4.4.10 Що таке діалоги відкриття та збереження файлів?

4.4.11 Які компоненти використовуються для роботи з діалогами відкриття та збереження файлів?

4.4.12 Яка властивість файлового діалогу зберігає ім’я обраного файлу та повний шлях до нього?

4.4.13 Як налаштувати діалог вибору файлу щодо вибору декількох файлів одночасно?

4.4.14 Як здійснюється очищення файлового потоку?

4.4.15 Як здійснюється асоціація потоку із **`StreamReader`**?

5 ЛАБОРАТОРНА РОБОТА №5
ОБРОБКА ПОДІЙ МИШІ

Мета роботи: навчитися основним принципам обробки подій маніпулятора типу миша у середовищі Visual Studio C#.

5.1 Короткі теоретичні відомості

5.1.1 Обробка повідомлень миші

Миша стала невід'ємним атрибутом при роботі у Windows. Тому в будь-якій програмі необхідно надавати користувачеві можливість виконати будь-яку дію за допомогою миші. Виключення становить введення символів із клавіатури.

5.1.2 Види подій

Для обробки повідомлення миші у Microsoft Visual C# є ряд подій, які посилаються програмі при виконанні певних подій. Події виконуються, якщо ви пересунете курсор миші, клацніть якою-небудь кнопкою або зробите усі ці дії одночасно.

Для обробки повідомлень від миші у форми існують наступні події (рис. 5.1).

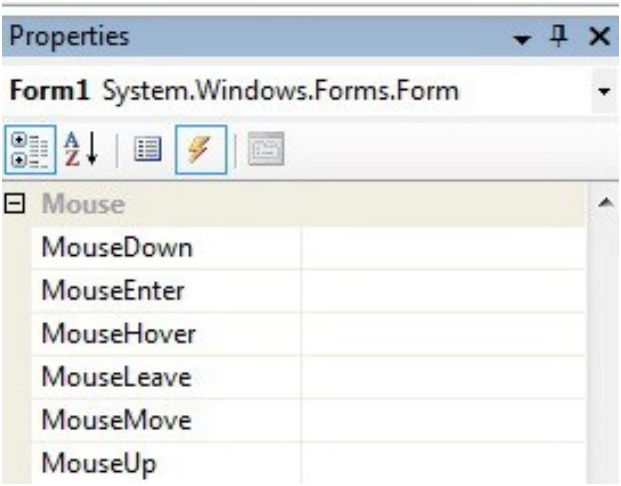


Рисунок 5.1 – Події миші

MouseDown – обробка натискання яких-небудь із кнопок униз;

MouseEnter – викликається при влученні покажчика миші у область форми;

MouseHover – викликається при зависанні покажчика миші у вікні форми;

MouseLeave – викликається при покиданні курсору миші області форми;

MouseMove – викликається при переміщенні миші у області форми;

MouseUp – викликається при відпусканні кнопки миші.

5.1.3 Параметри подій

Створіть для форми **Form2** оброблювач події **MouseDown**. Для цього клацніть два рази курсором миші по події **MouseDown** у вікні властивостей. У коді програми з'явиться функція-оброблювач **Form2_MouseDown**. Реалізуйте її так, як показано у наступному коді:

```
      private void Form2_MouseDown (object sender,
System.Windows.Forms.MouseEventArgs e) {
        string text;
        MouseButtons button;
        button = e.Button;
        if (button == MouseButtons.Left) {
          text = "ліву";
        }
        else if (button == MouseButtons.Right) {
          text = "праву";
        }
        else {
          text = "середню";
        }
        string message = "Ви нажали " + text + " кнопку миші у
координатах:\n" + "x:= " + e.X.Tostring() + "\n" + "y:=" +
e.Y.Tostring();
        Messagebox.Show(message);
      }
```

Параметр функції **MouseEventArgs** містить усю інформацію про кнопку, що послала повідомлення. Властивість **MouseEventArgs.Button** зберігає інформацію про тип кнопки (ліва, права, середня). Блок інструкцій **if ... else** обирає із трьох можливих варіантів ту кнопку, яка передалася в якості параметру. На підставі отриманої інформації про кнопку формується текст повідомлення. Наприклад, якщо ви натиснете ліву кнопку миші, то

повідомлення буде містити рядок «Ви нажали ліву кнопку миші». У кінець рядка повідомлення дописуються координати вікна, у яких було зроблено клацання покажчиком миші.

Запустіть програму. Створіть нове вікно, використовуючи меню Файл/Створити. Клацніть курсором у будь-якому місці дочірнього вікна. На екрані з'явиться повідомлення, аналогічне тому, яке зображено на рисунку 5.2.

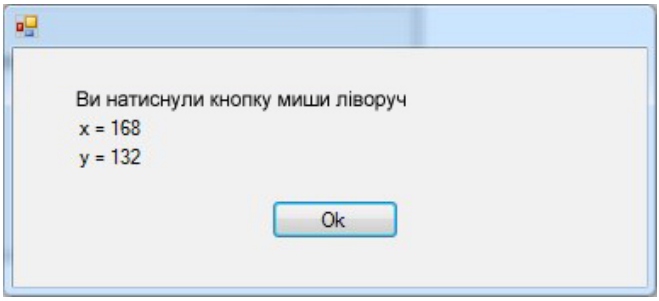


Рисунок 5.2 – Повідомлення про натискання лівої кнопки миші

5.2 Завдання до роботи

5.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

5.2.2 Вивчити основні принципи роботи Visual Studio з подіями маніпулятора.

5.2.3 Виконати наступні завдання:

– реалізувати кнопку, чуттєву до руху миші. При її наведенні вона повинна рухатися у бік протилежний, або будь який інший бік, від курсору. Передбачити демонстрування координат курсору та кнопки, також додати можливість регулювання швидкості реакції кнопки. Кожний раз при натисканні кнопки виводити повідомлення про перемогу, та можливість спробувати ще;

– реалізувати програму, де певний елемент управління можна переміщувати за допомогою миші у певні 4 області форми, де цей елемент притягується до різних сторін форми. Кожна область до певної сторони, без повторів. Стан об'єкту виводити на екран – координати, та сутність області де він знаходиться;

– реалізувати обробку подій потрійного натискання кнопки миші. Обрати елементи керування програмою (свій вибір обґрунтувати) та за допомогою потрійного натискання миші реалізувати зміну положення елементу і зміну його стилю.

5.2.4 Оформити звіт з роботи.

5.2.5 Відповісти на контрольні питання.

5.3 Зміст звіту

5.3.1 Тема та мета роботи.

5.3.2 Завдання до роботи.

5.3.3 Короткі теоретичні відомості.

5.3.4 Результати виконання роботи.

5.3.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

5.4 Контрольні питання

5.4.1 Які події миші оброблюються C#?

5.4.2 Які параметри подій миші?

5.4.3 Як отримати поточні координати курсору?

5.4.4 Чим відрізняється подія **`MouseHover`** від **`MouseMove`**?

5.4.5 Як обробити подвійне натискання правої кнопки миші?

5.4.6 Чим відрізняється подія **`MouseUp`** від **`MouseDown`**?

5.4.7 Як реалізувати перетягування об’єкту мишею?

5.4.8 Які константи є у **`MouseButtons`**?

5.4.9 Як можливо обробляти не основні кнопки миші?

5.4.10 Як програмно емулювати натискання кнопки миші?

6 ЛАБОРАТОРНА РОБОТА №6
ОБРОБКА ПОДІЙ КЛАВІАТУРИ

Мета роботи: навчитися основним принципам обробки подій клавіатури у середовищі Visual Studio C#.

6.1 Короткі теоретичні відомості

6.1.1 Робота з клавіатурою

Усі комерційні застосунки повинні мати можливість виконати будь-яку команду, як за допомогою миші так і за допомогою клавіатури.

6.1.2 Повідомлення клавіатури

Для обробки повідомлень із клавіатури в Windows Forms-програмах передбачено три події:

– **`KeyUp`**;

– **`KeyPress`**;

– **`KeyDown`**.

Подія **`KeyUp`** посилає при відпусканні кнопки на клавіатурі.

Подія **`KeyPress`** посилає перший раз при натисканні кнопки на клавіатурі разом з подією **`KeyDown`** але потім може посилати необмежене число раз, якщо користувач утримує клавішу в натиснутому стані. Частота посилань події **`KeyPress`** залежить від налаштувань операційної системи. Подія **`KeyDown`** посилає при натисканні клавіш на клавіатурі.

Для прикладу створимо програму, яка буде обробляти натискання клавіш та виводити на екран інформацію про те, яка клавіша була натиснута. Для цього створіть нову Windows Forms-програму з іменем `Keyboardapp`. Змініть властивості створеної форми:

– `Text` – «Інформація про натиснуті клавіші»;

– `KeyPreview` – `True`.

Властивість `Text` задає заголовок вікна. Властивість `KeyPreview` дозволяє формі перехоплювати повідомлення клавіатури від дочірніх елементів керування форми. Якщо властивість

KeyPreview форми встановлена у False, то форма не буде одержувати повідомлення від клавіатури. Якщо на формі присутній компонент **TextBox** і курсор миші перебуває в його полі, то при натисканні клавіші на клавіатурі форма про це довідатися не зможе. Тому, якщо необхідно обробляти події клавіатури у класі форми, то необхідно виставляти властивість KeyPreview у True. Додайте на форму елемент керування **TextBox**. Змініть деякі властивості елементу **TextBox**:

- Text – «»;
- ReadOnly – True;
- TabStop – False.

Форма повинна виглядати так, як показано на рисунку 6.1.

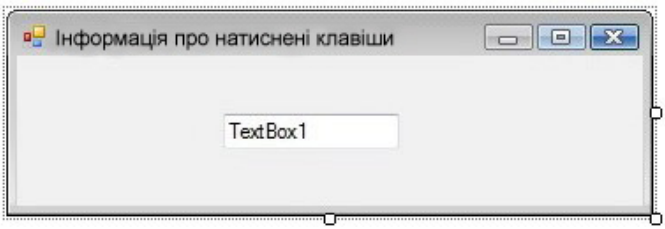


Рисунок 6.1 – Проєктування форми програми Keyboardapp

Додайте у програму оброблювач події **KeyDown**. Для цього у вікні властивостей форми клацніть два рази покажчиком миші по події **KeyDown**. У код програми буде додано оброблювач події **KeyDown** з іменем Form1_Keydown. Додайте до оброблювача код, наданий нижче:

```
      private void Form1_Keydown(object sender, System.Windows.Forms.
Keyeventargs e) {
      // очищуємо поле TextBox1.Text = “”;
      // перевіряємо чи натиснута клавіша Ctrl
      // якщо так, то записуємо у поле слово Ctrl
      if (e.Control) {
        TextBox1.Text += "Ctrl+";
      }
      // перевіряємо чи натиснута клавіша Shift
      // якщо так, то записуємо у поле слово Shift
      if (e.Shift) {
        TextBox1.Text += "Shift+";
      }
      // перевіряємо чи натиснута клавіша Alt
      // якщо так, то записуємо у поле слово Alt
```

```
if (e.Alt) {
  TextBox1.Text += "Alt+";
}
// копіюємо KeyData натиснутої клавіші
Keys key = e.KeyData;
// витягаємо з даних про натиснуту клавішу
// коди системних кнопок, таких як
// Ctrl, Shift, Alt
key &= ~keys.Control;
key &= ~keys.Shift;
key &= ~keys.Alt;
// виводимо отримане словосполучення
TextBox1.Text += key.ToString();
```

6.1.3 Клас `KeyEventArgs`

Клас **`KeyEventArgs`** містить усю інформацію про натиснуту клавішу. Властивості, які звичайно використовуються при обробці натискання кнопки:

– `Alt` – `True`, якщо натиснута клавіша **`Alt`**;

– `Control` – `True`, якщо натиснута клавіша **`Ctrl`**;

– `Shift` – `True`, якщо натиснута клавіша **`Shift`**;

– `KeyCode` – код натиснутої клавіші;

– `KeyData` – сукупність кодів натиснутих клавіш;

– `KeyValue` – десяткове значення властивості **`KeyData`**;

– `Handled` – прапорець, що вказує, чи було повідомлення оброблено. За замовчуванням, значення `Handled` дорівнює `False`. Якщо ви не прагнете подальшої обробки натискання кнопки, виставте прапорець `Handled` у `True`.

Для виведення на екран інформації про натиснуті керуючі клавіші ми перевіряємо значення властивостей **`Alt`**, **`Ctrl`** та **`Shift`**. Якщо одна із клавіш натиснута, то до поля **`TextBox`** додається відповідне слово.

Для виведення на екран інформації про основну натиснуту клавішу, ми повинні попередньо вилучити з поля **`KeyData`** системну інформацію. Тому ми скидаємо в **`0`** прапорці керуючих клавіш властивості **`KeyData`**.

Інструкція `key & = ~keys.Control` дозволяє вилучити зі змінної **`key`** код керуючої клавіші **`Ctrl`** незалежно від того, є він там чи ні. Точно так вилучаються коди клавіш **`Alt`** та **`Shift`**.

Метод **`ToString`** повертає дослівний опис кожної клавіші. Код звичайної клавіші виводиться однією буквою, а код системної клавіші виводиться словом, відповідним до клавіші. Запустіть на виконання програму. Спробуйте натискати кнопки на клавіатурі, поєднуючи їх з керуючими клавішами. На формі буде відображатися інформація у вигляді рядка, що містить позначення натиснутої клавіші (рис. 6.2).

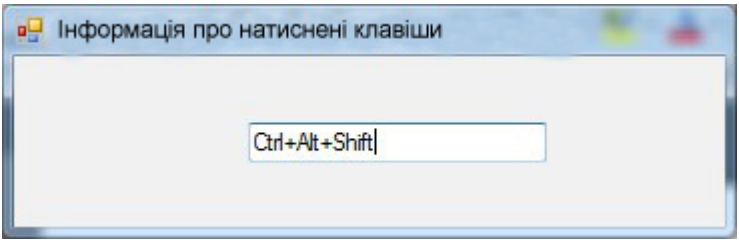


Рисунок 6.2 – Інформація про натисненні клавіші

6.1.4 Таймер і час

Дуже часто доводиться встановлювати залежність виконання яких-небудь дій у програмі від часу. Це може бути виконання періодично повторюваних дій або спрацьовування команди у певний момент часу.

6.1.5 Компонент `Timer`

Робота з таймером у WindowsForms-програмах заснована на все тому ж механізмі подій. Ви встановлюєте таймер на певну частоту та операційна система буде розсилати застосунку події оповіщення із зазначеною частотою.

Компонент **`Timer`** дозволяє легко та просто працювати з часом.

Основними властивостями компонента **`Timer`** є:

– `Interval` – задає період приймання повідомлень таймером у милісекундах;

– `Enabled` – визначає стан Увімкнення/Вимикання таймеру;

Для роботи з таймером необхідно лише помістити на форму компонент **`Timer`**, встановити його властивість `Interval` на заданий інтервал часу та обробити подію **`Elapsed`**.

6.1.6 Компонент `DataTimePicker`

Елемент **`DataTimePicker`** являє собою універсальний візуальний компонент для представлення інформації про час. Він містить компонент календар (рис. 6.3) та дозволяє легко змінювати час у поле компонента.

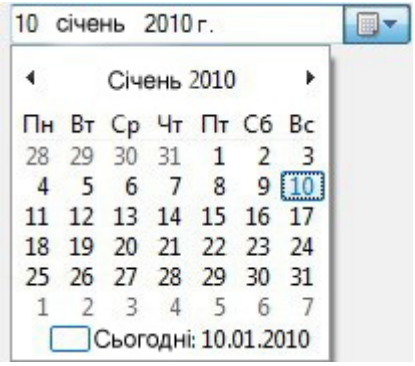


Рисунок 6.3 – Елемент керування **`DataTimePicker`**

Компонент **`DataTimePicker`** дозволяє робити тонке налаштування формату відображення часу. Це досягається за рахунок можливості завдання власного формату відображення.

Основні властивості компонента **`DataTimePicker`** наступні:

`Format` – дозволяє встановити один зі стандартних форматів відображення часу або вказати свій;

`ShowUpdown` – встановлює тип елементу з правої сторони поля відображення. Якщо встановити `False` – відображується **`ComboBox`**, що відкриває календар, якщо `True` – відображується **`NumericUpdown`**, що змінює активне поле відображення;

`CustomFormat` – рядок, що описує власний формат відображення часу;

`MaxDate` – максимально можливий час для введення;

`MinDate` – мінімально можливий час для введення;

`Value` – значення часу.

6.1.7 Структура `DateTime`

Структура **`DateTime`** призначена для зберігання та обробки змінних у форматі дати або часу. Структура **`DateTime`** настільки універсальна, що її використовують і при роботі з рядками, і при роботі із числами, і при роботі з базами даних. Структура **`DateTime`** має окремі властивості для кожної категорії часу (рік, місяць, число, година, хвилина, секунда, милісекунда). Крім того, **`DateTime`** має набір методів для обробки тимчасових інтервалів. Наприклад, можна скласти два тимчасові значення, відняти, конвертувати до іншого формату та інше.

6.1.8 Формат рядку часу

Дуже важливою особливістю структури **`DateTime`** є можливість перекладу тимчасового значення до строкового формату. При цьому формат рядка, що вертається, може бути заданий динамічно. Метод **`DateTime.ToString`** повертає рядок часу на підставі формату аргументу.

Деякі константи, що визначають формат рядка часу:

`dd` – два знаки дня місяця. Якщо день складається з однієї цифри, то спереду ставиться незначущий **`0`**;

`dddd` – день тижня;

`MM` – номер місяця (**`1–12`**);

`MMMM` – назва місяця;

`yyyy` – номер року;

`hh` – кількість годин (**`1–12`**);

`Hh` – кількість годин (**`1–24`**);

`mm` – кількість хвилин;

`ss` – кількість секунд.

Крім того, рядок формату може містити будь-які розділові символи для зручності представлення. Формат часу може бути дуже зручно налаштований під конкретне завдання.

Приклади формату часу наведено у таблиці 6.1.

Таблиця 6.1 – Формат часу

Формат	Значення
dd MMMM yyyy HH:mm:ss	21 листопада 2002 14:48:56
dd.MM.yyyy HH:mm	21.11.2002 14:48
Сьогодні **dd MMMM yyyy** року	Сьогодні 21 листопада 2002 року

6.2 Завдання до роботи

6.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

6.2.2 Вивчити основні принципи роботи з клавіатурою та часом.

6.2.3 Виконати наступні завдання:

– реалізувати програму «клавіатурний тренажер». У програмі на екран виводиться певний символ, який треба увести та таймер для його введення. Передбачити 5 ступенів важкості – регулювання відведеного часу та кількості літер, слів для введення. Також передбачити таблицю найкращих результатів з можливістю автоматичного запису та зчитування з файлу. Крім того, реалізувати зберігання та завантаження прогресу користувача;

– реалізувати програму «будильник». Надати можливість виставлення сигналу на конкретний час, дату, день тижня, та коротке повідомлення. Інформацію про виставлені режими роботи зберігати у файлі;

– реалізувати «конвертер». Фіксувати кожне натискання клавіш клавіатури та виводити на екран відповідний код натиснутої клавіші. Зберігати лог роботи програми в файлі у наступному вигляді: код натиснутої клавіші – назва клавіші.

6.2.4 Оформити звіт з роботи.

6.2.5 Відповісти на контрольні питання.

6.3 Зміст звіту

6.3.1 Тема та мета роботи.

6.3.2 Завдання до роботи.

6.3.3 Короткі теоретичні відомості.

6.3.4 Результати виконання роботи.

6.3.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

6.4 Контрольні питання

6.4.1 Назвіть основні події клавіатури.

6.4.2 Що містить клас **`KeyEventArgs`**?

6.4.3 Як отримати інформацію о натисканні керуючих клавіш?

6.4.4 Для чого використовують метод **`ToString`**?

6.4.5 Назвіть основні властивості компоненту **`Timer`**.

6.4.6 Назвіть основні властивості компоненту **`DateTimePicker`**.

6.4.7 Для чого призначена структура **`DateTime`**?

6.4.8 Для чого призначена структура **`TimeSpan`**?

6.4.9 Вкажіть формат для виводу дати «**`2002-21-11 GMT+2 14:48:12.87`**».

6.4.10 Як отримати поточний час?

7 ЛАБОРАТОРНА РОБОТА №7
РОБОТА З ЗОБРАЖЕННЯМИ

Мета роботи: навчитися працювати із зображеннями за допомогою Visual Studio C#.

7.1 Короткі теоретичні відомості

7.1.1 Особливості GDI+

Для малювання об'єкту у Windows Forms-програмі мова C# містить дуже багатий набір методів. Простір імен **`Drawing`** містить множину об'єктів, які полегшують програмісту роботу із графікою. Спеціально для .NET-платформи розробники Microsoft створили GDI+ бібліотеку, значно підвищивши можливості GDI (Graphics Device Interface). Бібліотека GDI+ містить інструменти малювання найпростіших об'єктів (ліній, еліпсів), малювання різних об'єктів 2D-графіки, відображення файлів різних графічних форматів (bmp, jpeg, gif, wmf, ico, tiff) та багато чого іншого.

7.1.2 Малювання об'єктів

Малювання графічних примітивів буде розглянуто на прикладі програми. Для малювання лінії необхідно буде два рази клацнути мишею у різних частинах вікна. При цьому, лінія намалюється від точки, у якій відбулося перше клацання мишею до точки, де відбулося друге клацання мишею. Це нетипово для графічного редактора. Звичайно лінія тягнеться від точки першого клацання миші за курсором доти, поки користувач не відпустить кнопку. Але така функціональність може ускладнити код програми.

Малювання еліпсів аналогічно малюванню ліній. Перше клацання миші буде задавати ліву верхню координату еліпса, а другий – праву нижню координату.

Щодо малювання тексту, при клацанні миші рядок тексту буде відображатися праворуч від курсору.

Особливу цікавість представляє малювання олівцем. Воно повинне відбуватися під час руху покажчика миші по області вікна з натиснутою лівою кнопкою миші. Для відображення шляху руху покажчика миші будемо малювати лінії між передостанньою

координатою курсору та останньою. Для цього доведеться постійно запам’ятовувати координату останнього положення курсора миші.

7.1.3 Малювання олівцем

Відкрийте з вікна **Solutionexplorer** форму **Form2**. Ця форма описує дочірнє вікно програми. Саме в класі **Form2** необхідно написати код, який дозволить малювати графічні примітиви.

У програмі вже присутній оброблювач події **MouseDown**. Необхідно додати оброблювачі для подій **MouseMove** та **MouseUp**. Зробіть це, клацнувши два рази по відповідних до полів на закладці подій у вікні властивостей. Той код, який був присутній в оброблювачі події **MouseDown**, більше не знадобиться. Можна вилучити його або закоментувати.

Змініть код оброблювачів подій так, як представлено нижче:

```
      private void Form2_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e) {
        Form1 parentform = (Form1)Mdiparent;
        switch (parentform.currenttool) {
          case Tools.LINE:
      //    DrawLine(new Point (e.X, e.Y));
            break;
          case Tools.ELLIPSE:
      //    DrawEllipse(new Point (e.x, e.Y));
            break;
          case Tools.TEXT:
      //    DrawText(new Point(e.X, e.Y));
            break;
          case Tools.PEN:
      // встановлюємо прапорець для початку малювання олівцем
            DrawPen = true;
            break;
        }
      // запам'ятовуємо першу крапку для малювання
        PreviousPoint.X = e.X;
        PreviousPoint.Y = e.Y;
      }
      private void Form2_MouseUp(object sender,
System.Windows.Forms.MouseEventArgs e) {
        DrawPen = false;
      }
      private void Form2_MouseMove (object sender,
System.Windows.Forms.MouseEventArgs e) {
      // якщо курсор ще не відпущений
        if (DrawPen) {
      // створюємо об’єкт Pen
          Pen blackpen = new Pen(Color.Black,3);
```

```
// одержуємо поточне положення курсору
    Point point = new Point(e.X, e.Y);
// створюємо об'єкт Graphics
    Graphics g = this.CreateGraphics();
// малюємо лінію
    g.DrawLine(blackpen, PreviousPoint, point);
// зберігаємо поточну позицію курсору
PreviousPoint = point;
  }
}
```

Коли ви введете цей код, програма не буде компілюватися. Ви повинні були помітити додавання до класу **Form2** нових полів та методів. У коді програми використовуються неоголошені поки змінні **PreviousPoint**, **DrawPen** та неоголошені методи **DrawLine**, **DrawEllipse** та **Text**. Останні закоментовані та не заважають поки компіляції програми. А от змінні необхідно оголосити. Давайте оголосимо у програмі змінні **PreviousPoint** та **DrawPen**:

```
public class Form2: System.Windows.Forms.Form {
  public bool DrawPen;
  public Point PreviousPoint;
}
```

Змінні слід оголосити членами класу **Form2.DrawPen** – це двійкова змінна, яка буде визначати, малювати олівцю чи ні. Оброблювач **Form2_MouseDown** виставляє значення змінної **DrawPen** у `True`, якщо в момент натискання кнопки миші встановлений інструмент «Олівець». Оброблювач **Form2_MouseUp** виставляє значення змінної назад в `False`. Таким чином, при натисканні кнопки миші прапор **DrawPen** встановлюється у `True` та не скидається у `False` доти, поки користувач не відпустить кнопку нагору. Оброблювач **Form2_MouseMove** малює на екрані лінії тільки тоді, коли змінна **DrawPen** перебуває у `True`. Малювання відбувається у такий спосіб:

1) створюється об'єкт **Pen**, який ініціалізується чорним кольором (**Color.Black**) та товщиною лінії, що дорівнює трьом:

```
Pen blackpen = new Pen (Color.Black, 3);
```

2) отримуються поточні координати курсору миші:

```
Point point = new Point(e.X, e.Y);
```

3) створюється об'єкт **Graphics** на базі поточної форми:

```
Graphics g = this.Creategraphics();
```

4) викликається метод **DrawLine** об'єкту **Graphics**. У якості координат лінії передаються попередня та поточна координати

курсору миші. Після промальовування лінії поточна координата курсору миші запам'ятовується у змінну, яка зберігає попередню координату:

```
PreviousPoint = point;
```

Таким чином, кожний рух миші з натиснутою кнопкою буде змушувати програму малювати лінію між двома сусідніми крапками.

У підсумку вийде крива шляху, по якім рухався курсор миші.

Запустіть програму на виконання. Створіть нове вікно. Оберіть режим малювання олівцем. Спробуйте намалювати у вікні будь-яку криву лінію, утримуючи курсор миші у натиснутому стані (рис. 7.1).

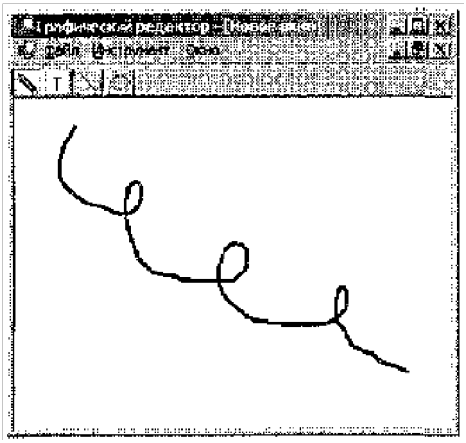


Рисунок 7.1 – Малювання олівцем

7.1.4 Малювання тексту та графічних примітивів

Однак наша програма ще не завершена. Ще треба реалізувати функції малювання ліній, еліпсів та написання тексту.

Для цього необхідно додати до класу **Form2** опис наведених нижче функцій:

```
void Drawline(Point point)
// якщо один раз вже клацнули
  if (Firstclick == true) {
// створюємо об'єкт Pen
    Pen blackpen = new Pen(Color.Black, 3);
// створюємо об'єкт Graphics
    Graphics g = this.Creategraphics();
// малюємо лінію
```

```
          g.Drawline(blackpen, Previouspoint, point);
          Firstciick = false;
        }
        else {
          Firstciick = true;
        }
      void Drawellipse(Point point) {
      // якщо один раз вже клацнули
        if (Firstciick== true) {
      // створюємо об'єкт Pen
          Pen blackpen = new Pen(Color.Black, 3);
      // створюємо об'єкт Graphics
          Graphics g = this.Creategraphics();
      // малюємо еліпс
          g.Drawellipse(BlackPen, Previouspoint.X, Previouspoint.Y,
point.X - Previouspoint.X, point.Y - Previouspoint.Y);
          Firstciick = false;
        }
        else {
          Firstciick = true;
        }
      }
      void Drawtext(Point point) {
      // створюємо об'єкт Graphics
        Graphics g = this.Creategraphics();
      // створюємо об'єкт Font
        Font titlefont = new Font("Lucida Sans Unicode", 15);
      // малюємо текст  червоним кольором
        g.Drawstring("Програмування на C#", titlefont, new
Solidbrush(Color.Red), point.X, point.Y);
      }
```

Методи **Drawline** та **Drawellipse** використовують змінну **Firstclick**. Додайте оголошення цієї змінної до класу **Form2**:

```
public bool drawpen;
public bool Firstclick;
public Point Previouspoint;
```

Методи **Drawtext** та **Drawellipse** малюють об'єкти по двом точкам на екрані. При першому клацанні миші запам'ятовується перша координата та виставляється значення змінної **Firstclick** у `True`.

При другому клацанні миші відбувається малювання лінії та значення **Firstclick** скидається у `False`.

Для малювання тексту використовується метод **Drawtext**. Для промальовування спочатку створюється об'єкт **Font** зі шрифтом типу **Lucida Sans Unicode** розміром **15** одиниць. Потім за допомогою методу **Drawstring** рядок тексту «Програмування на C#» виводиться на екран.

Тепер ви можете відкоментувати виклики методів **`Drawline`**, **`Drawellipse`** та **`Drawtext`** у функції **`Form2_MouseDown`**.

Запустіть програму на виконання. Обираючи різні режими роботи програми можна створювати найпростіші графічні зображення (рис. 7.2).

Рисунок 7.2 – Малювання інструментами

7.2 Завдання до роботи

7.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

7.2.2 Вивчити основні принципи роботи з зображеннями.

7.2.3 Програмно реалізувати малювання наступними інструментами: лінія, еліпс, олівець, прямокутник та текст.

7.2.4 Розробити завершені програмні рішення, які можна запустити на виконання як окремий застосунок:

а) створити простий але функціональний растровий графічний редактор (у якості аналогу застосунку використати вбудований до ОС Windows графічний редактор «Paint»);

б) створити нескладний але з необхідним набором функцій переглядач растрових зображень, які зберігаються у файлах різноманітних графічних форматів (у якості аналогів застосунку можна використати наступні переглядачі зображень: «FastStone», «XnView», «IrfanView», «ACDSee» тощо).

У процесі створення застосунків треба використовувати «дружній» користувацький інтерфейс (обов'язково із обробкою виключних ситуацій). Усі застосунки виконуються тільки із використанням алгоритмічної мови C#.

Варіантом роботи є персональна оригінальність застосунку, за що буде підвищення оцінки, але й зменшення її, якщо буде підозра на плагіат.

7.2.5 Оформити звіт з роботи.

7.2.6 Відповісти на контрольні питання.

7.3 Зміст звіту

7.3.1 Тема та мета роботи.

7.3.2 Завдання до роботи.

7.3.3 Короткі теоретичні відомості.

7.3.4 Текст розробленого програмного забезпечення.

7.3.5 Результати виконання роботи.

7.3.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

7.4 Контрольні питання

7.4.1 Як вивести пунктирну лінію на екран?

7.4.2 Як реалізувати градієнтне зафарбовування?

7.4.3 Для чого використовується клас Polygon?

7.4.4 Як задати товщину лінії?

7.4.5 Як намалювати стрілку?

7.4.6 Як задати Antialiasing для виводу тексту?

7.4.7 Як збільшити швидкодію роботи технології GDI+?

7.4.8 Порівняти технології GDI та GDI+.

7.4.9 Для чого використовуються матриці перетворення?

7.4.10 Використання Alpha-прозорості у GDI+.

ПЕРЕЛІК ПОСИЛАНЬ

1. Лабор, В. Си Шарп. Создание приложений для Windows [Текст]/ В. Лабор. – Минск: Харвест, 2003. – 385 с.

2. Джонсон, Б. Основы Microsoft Visual Studio .Net 2003 [Текст]/ Б. Джонсон. – М.: Русская редакция, 2003. – 463 с.

3. Бишоп, Д. C# в кратком изложении [Текст]/ Д. Бишоп. – М.: БИНОМ Лаборатория знаний, 2005. – 467 с.

4. Орлов, С. Технологии разработки программного обеспечения. Учебник для Вузов [Текст]/ С. Орлов. – СПб.: Питер, 2002. – 463 с.

5. Соммервил, И. Инженерия программного обеспечения [Текст]/ И. Соммервил. – М.: Издательский дом «Вильямс», 2002. – 623 с.

Додаток А

Приклад оформлення титульного аркушу звіту з лабораторної роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 1

дисципліна: "Основи програмної інженерії"

на тему: «**ЗНАЙОМСТВО З VISUAL STUDIO C#**»

Виконав:

студент групи КНТ-110 А.Б. Іваненко

Прийняв:

ст. викладач О.І. Качан

2021