

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

з дисципліни

“Основи програмної інженерії”

для студентів спеціальностей

121 “Інженерія програмного забезпечення” та

122 “Комп’ютерні науки та інформаційні технології”

(всіх форм навчання)

Методичні вказівки до виконання лабораторних робіт з дисципліни “Основи програмної інженерії” для студентів спеціальності 121 “Інженерія програмного забезпечення” та 122 “Комп’ютерні науки та інформаційні технології” (всіх форм навчання) / А. О. Олійник, Є. М. Федорченко, О. І. Качан. – Запоріжжя: ЗНТУ, 2016. – 120 с.

Автори: Андрій Олександрович Олійник, к.т.н., доцент
Євген Миколайович Федорченко, ст. викладач
Олександр Іванович Качан, ст. викладач

Рецензент: С. О. Субботін, д.т.н., професор

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

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “16” серпня 2016 р.

ЗМІСТ

Вступ	7
1 Лабораторна робота № 1 Знайомство з Visual Studio C#.....	8
1.1 Мета роботи	8
1.2 Основні теоретичні відомості.....	8
1.2.1 Кнопки – Button	10
1.2.2 Чекбокси – CheckBox	10
1.2.3 Радіокнопки – RadioButton	11
1.2.4 Блок групування – GroupBox.....	11
1.2.5 Поле введення – TextBox.....	17
1.2.6 Розширене поле вводу – RichTextbox	17
1.2.7 Список – ListBox.....	17
1.2.8 Список, що позначається, – CheckedListBox	17
1.2.9 Випадаючий список – ComboBox	18
1.3 Завдання до роботи.....	24
1.4 Зміст звіту.....	26
1.5 Контрольні запитання	26
2 Лабораторна робота № 2 Другорядні елементи оболонки програми.....	28
2.1 Мета роботи	28
2.2 Основні теоретичні відомості.....	28
2.2.1 Мітка – Label	28
2.2.2 Мітка – LinkLabel	29
2.2.3 Бігунок – TrackBar	29
2.2.4 Індикатор прогресу – ProgressBar	29
2.2.5 Регулятор числових значень – NumericUpDown	29
2.2.6 Список – ListView.....	33
2.2.7 Дерево – TreeView	34
2.3 Завдання до роботи.....	38
2.4 Зміст звіту.....	40
2.5 Контрольні запитання	40
3 Лабораторна робота № 3 Робота з рядками.....	42
3.1 Мета роботи	42
3.2 Основні теоретичні відомості.....	42
3.2.1 Особливості типу System.String	42

3.2.2 Створення рядків	43
3.2.3 System.Object.ToString()	44
3.2.4 Маніпулювання рядками	44
3.2.5 Пошук підстроки	50
3.2.6 Розбиття рядків	52
3.2.7 Клас StringBuilder	53
3.2.8 Регулярні вирази	55
3.2.8.1 Застосування регулярних виразів	56
3.2.8.2 Основи синтаксису регулярних виразів. Класи символів (Character classes)	57
3.2.8.3 Квантифікатори (Quantifiers)	57
3.2.8.4 Закінчення й початки рядків	58
3.2.8.5 Границя слова	58
3.2.8.6 Варіації й групування. Правила побудови регулярних виразів	59
3.2.9 Використання регулярних виразів: Regex	61
3.2.10 Використання Match колекцій	62
3.3 Завдання до роботи	63
3.4 Зміст звіту	64
3.5 Контрольні запитання	65
4 Лабораторна робота № 4 Робота з файлами	66
4.1 Мета роботи	66
4.2 Основні теоретичні відомості	66
4.2.1 Поняття потоків	66
4.2.2 Атрибути відкриття файлів	67
4.2.3 Діалоги відкриття та збереження файлів	67
4.3 Завдання до роботи	70
4.4 Зміст звіту	71
4.5 Контрольні запитання	71
5 Лабораторна робота № 5 Обробка подій миші	73
5.1 Мета роботи	73
5.2 Основні теоретичні відомості	73
5.2.1 Обробка повідомлень миші	73
5.2.2 Види подій	73
5.2.3 Параметри подій	74
5.3 Завдання до роботи	75
5.4 Зміст звіту	76

5.5 Контрольні запитання	76
6 Лабораторна робота № 6 Обробка подій клавіатури.....	78
6.1 Мета роботи	78
6.2 Основні теоретичні відомості.....	78
6.2.1 Робота з клавіатурою.....	78
6.2.2 Повідомлення клавіатури.....	78
6.2.3 Клас Keyeventargs	80
6.2.4 Таймер і час	81
6.2.5 Компонент Timer	81
6.2.6 Компонент – DateTimePicker	82
6.2.7 Структура DateTime	83
6.2.8 Формат рядку часу.....	83
6.3 Завдання до роботи.....	84
6.4 Зміст звіту.....	85
6.5 Контрольні запитання	85
7 Лабораторна робота № 7 Робота з зображеннями	86
7.1 Мета роботи	86
7.2 Основні теоретичні відомості.....	86
7.2.1 Особливості GDI+.....	86
7.2.2 Малювання об'єктів	86
7.2.3 Малювання олівцем.....	87
7.2.4 Малювання тексту й графічних примітивів	90
7.3 Завдання до роботи.....	92
7.4 Зміст звіту.....	92
7.5 Контрольні запитання	92
8 Лабораторна робота № 8 Створення довідки.....	94
8.1 Мета роботи	94
8.2 Основні теоретичні відомості.....	94
8.2.1 Створення довідки.....	94
8.2.2 Підготовка HTML-сторінок.....	94
8.2.3 Створення змісту	109
8.2.4 Створення покажчика.....	113
8.2.5 Створення пошуку.....	115
8.2.6 Створення вкладки "Вибране".....	116
8.2.7 Тестування та налагодження довідки	118
8.2.8 Додавання готового файлу довідки до додатка	119

8.3 Завдання до роботи.....	121
8.4 Зміст звіту.....	121
8.5 Контрольні запитання	121
Література.....	123

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ Visual Studio C#.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. Звичайно, в дані методичні вказівки неможливо було внести весь матеріал, необхідний для виконання та захисту лабораторних робіт. Тому тут містяться основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт. Таким чином для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та проробити весь матеріал, наведений в переліку рекомендованої літературі. При цьому не варто обмежуватись лише наведеним списком.

Для одержання заліку з кожної роботи студент здає викладачу цілком оформлений звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

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 A4 (210 × 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок або вміщують у канцелярський файл.

Під час співбесіди при захисті лабораторної роботи студент повинний виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів оформленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

1 ЛАБОРАТОРНА РОБОТА № 1 ЗНАЙОМСТВО З VISUAL STUDIO C#

1.1 Мета роботи

Вивчити основні можливості та принципи роботи з середовищем розробки ПЗ в Microsoft Visual Studio C#.

1.2 Основні 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 проекту C# у середовищі Visual Studio обирається один з 10 можливих типів проектів, у тому числі Windows Application, Class Library, Web Control Library, ASP.NET, Application і ASP.NET Web Service. На підставі зробленого вибору автоматично створюється каркас проекту.

Visual Studio.NET — це не тільки середовище для розробки програм мовою C#. Visual Studio.NET дозволяє створювати програми на мові VB, C#, C++, формувати Setup (встановлювальний пакет) ваших програм и багато іншого. Для того щоб реально побачити, як створюється новий проект в Visual Studio .NET, виберіть пункт меню File/New/Project... Після його виклику з'явиться вікно, аналогічне зображеному на рис. 1.1.

Тут можна обрати потрібну вам мову програмування (у лівій частині вікна) або якийсь спеціальний майстер створення програм цей список може поповнюватися інструментами незалежних розроблювачів. Виберіть пункт Visual C# Project.

У правій частині вікна потрібно вказати тип створюваного вами проекту. Це може бути Windows-Програма (Windows Application), програма для Інтернету (ASP.NET), консольну програму (Console Application) і деякі інші. Виберіть у лівій частині вікна пункт Windows Application.

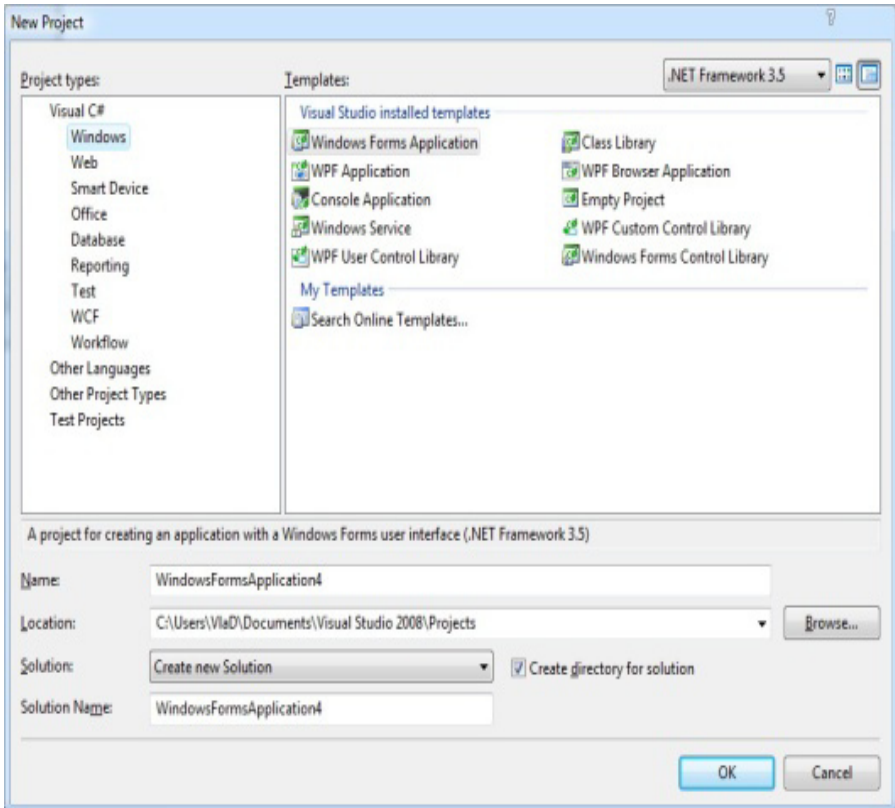


Рисунок 1.1 – Вікно створення нового проекту

Крім того, ви можете вказати назву створюваного проекту й шлях до каталогу, у якому він буде розташовуватися. Натисніть OK.

Тепер ви побачите основні частини візуального середовища розробки проекту. Вони зображені на рис. 1.2. У центрі перебуває головне вікно для створення візуальних форм і написання коду. Праворуч розміщується вікно Solution Explorer для керування вашими проектами, Class View для огляду всіх класів і вікно властивостей Properties Explorer. Зліва розміщується вікно Toolbox, яке містить елементи керування C#. У випадку відсутності цих вікон, їх можна знайти у меню View.

Розглянемо основні елементи керування.

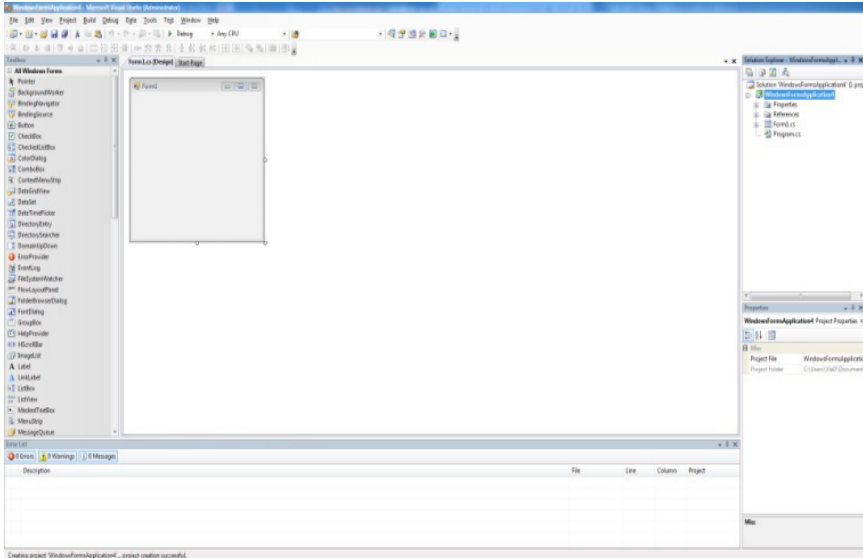
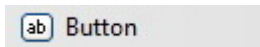


Рисунок 1.2 – Головне вікно середовища Visual Studio .NET

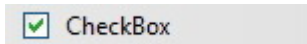
1.2.1 Кнопки – Button



Кнопкою називається елемент керування, вся взаємодія користувача з яким обмежується однією дією – натисканням. Усе, що вам необхідно зробити при роботі із кнопкою, – це помістити її в потрібному місці форми й призначити їй відповідний оброблювач.

Оброблювач призначається для події Click.

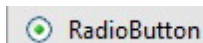
1.2.2 Чекбокси – CheckBox



Перше, що необхідно сказати про чекбокси це те, що вони є кнопками відкладеного дії, тобто їхнє натискання не повинне запускати яку-небудь негайну дію. З їхньою допомогою користувачі вводять параметри, які позначаються після, коли дія буде запущена іншими елементами керування. Елемент Checkbox може мати 3 стани – позначений, непозначений й змішаний.

Найчастіше цей елемент застосовується для визначення значень, які можуть мати тільки два стани.

1.2.3 Радіокнопки – RadioButton



– За своїми властивостям небагато схожі на чекбоксы. Їхня головна відмінність полягає в тому, що група чекбоксів дозволяє обрати будь-яку комбінацію параметрів, радіокнопки ж 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 сполучити взаємовиключаючі параметри).

1.2.4 Блок групування – GroupBox



– Блок групування допомагає візуально об'єднати кілька елементів керування в одну групу.

Наприклад, об'єднати групу радіокнопок (див. рис. 1.3)

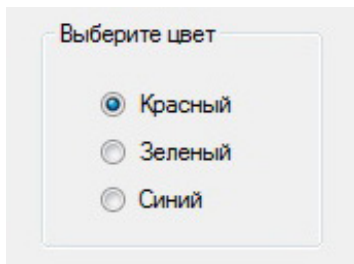


Рисунок 1.3 – Приклад використання блоку угруповання для об'єднання радіокнопок

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 проект за назвою TestButtons. Збережіть його в створену для власних проектів папку. Змініть деякі

властивості створеної форми: Name = «TestButtonsForm» Text=«Тест для кнопок».

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

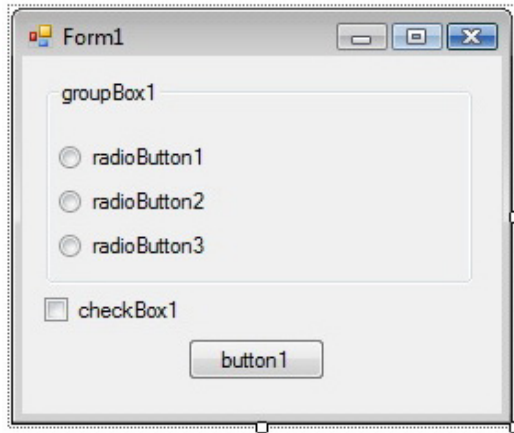


Рисунок 1.4 – Проектування форми для додатка TestButtons

Тепер змініть деякі властивості доданих елементів:

button1:	Text – показати повідомлення
groupBox1:	Text – виберіть 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 давайте 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 TestButtonsForm ()
{
//
// 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.Dispose();
}
}
base.Dispose ( disposing );
}

#region Windows 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.button1 = new System.Windows.Forms.Button ();
this.groupBox1 = 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.checkBox1 = new System.Windows.Forms.CheckBox();
this.groupBox1.SuspendLayout();
this.SuspendLayout ();
//
//button1
//
this.button1.Location = new System.Drawing.Point(80, 240);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(128, 23);
this.button1.TabIndex = 0;
this.button1.Text = "Показати повідомлення";
//
// groupBox1
//
this.groupBox1.Controls.AddRange(new System.Windows.Forms.Control[] {
this.radioButton3,
this.radioButton2,
this.radioButton1});
this.groupBox1.Location = new System.Drawing.Point (18, 16);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(280, 112);
this.groupBox1.TabIndex = 1;
this.groupBox1.TabStop = false;
this.groupBox1.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.TabIndex = 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.TabIndex = 1;
this.radioButton2.Text = "друге повідомлення";
//
// radioButton1
//
this.radioButton1.Location = new System.Drawing.Point (16, 24);
this.radioButton1.Name = "radioButton1";
this.radioButton1.Size = new System.Drawing.Size (240, 24);
this.radioButton1.TabIndex = 0;
this.radioButton1.Text = "перше повідомлення";
//
// checkBox1
//

```

```

this.checkBox1.Checked = true;
this.checkBox1.CheckState = System.Windows.Forms.Checkstate.Checked;
this.checkBox1.Location = new System.Drawing.Point(24, 136);
this.checkBox1.Name = "checkBox1";
this.checkBox1.Size = new System.Drawing.Size(256, 24);
this.checkBox1.TabIndex = 2;
this.checkBox1.Text = "Показувати повідомлення";
//
// TestButtonsForm
//
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.checkBox1,
    this.groupBox1,
    this.button1});
this.Name = "TestButtonsForm";
this.Text = "Тест для кнопок";
this.groupBox1.ResumeLayout(false);
this.ResumeLayout(false);
}
#endregion
// <summary>
// The main entry point for the application.
//</summary>
[StathRead]
Static void main() {
Application.Run (new Form1 ());
}
}
}

```

Ваш код може відрізнятись від даного – в залежності від версії C#.

В деяких випадках необхідно змінити у програмі ім'я форми, з якої буде запускатися додаток. Як ви пам'ятаєте, ми перейменували Form1 в TestButtonsForm. Знайдіть у коді рядок

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

і замініть 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. Відкопілюйте її й запустіть. Поки програма не здатна виконувати які-небудь дії. Щоб наділити її функціональністю, додайте функцію-оброблювач для кнопки button1. Використовуйте ім'я за замовчуванням для цієї функції – button1_Click, створене

середовищем Visual Studio.NET. При цьому у функцію InitializeComponent додається рядок:

```
this.button1.Click += new System.EventHandler
(this.button1_Click);
```

і з'явиться тіло самої функції:

```
private void button1_Click(object Sender, System.EventArgs e)
{
}
```

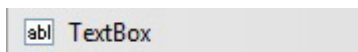
Додайте в тіло функції button1_Click наступний код:

```
//вводимо строкову змінну
// для зберігання обраного повідомлення
string strmessage="";
//визначаємо яка саме радіокнопка відзначена
//і обираємо відповідно до цього
//текст виводимого 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
    //те 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 установлений чекбокс,
//що дозволяє вивід повідомлення
//якщо так, то виводимо обране повідомлення на екран
if (checkBox1.Checked == true)
{
    MessageBox.Show("Ви вибрали " + strmessage);
}
```

Відкомпілюйте й запустіть програму. 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

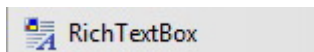
повідомлення. Тепер заберіть прапорець Показувати повідомлення. Натисніть кнопку Показати повідомлення. На екрані нічого не повинне з'явитися. Якщо програма працює в строгій відповідності з даним описом, виходить, ви все зробили вірно.

1.2.5 Поле введення – TextBox



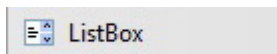
– Цей елемент керування є основним, що призначений для введення користувачем текстових даних. Використовувати TextBox можна в однорядковому або багаторядковому режимі. Однак даний елемент керування має обмеження – до 64 кб тексту. Якщо необхідно обробляти більші обсяги інформації, краще використовувати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

1.2.6 Розширене поле вводу – Richtextbox



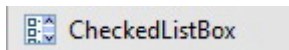
– Даний елемент керування дає можливість користувачеві вводити й обробляти більші обсяги інформації (більш 64 кб). Крім того, RichTextBox дозволяє редагувати колір тексту, шрифт, додавати зображення. RichTextBox включає всі можливості текстового редактора Microsoft Word.

1.2.7 Список – ListBox



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

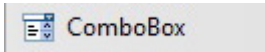
1.2.8 Список, що позначається, – CheckedListBox



– 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 кілька елементів списку, виставивши напроти його прапорець.

1.2.9 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 в списку.

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

Створіть новий Windows Forms проект за назвою TestLists. Збережіть його в створену для наших проектів папку. Перейменуйте файл Form1.cs в TestListsForm.cs. Тепер додайте на вашу форму наступні елементи керування (рис. 1.5):

- GroupBox, і помістіть в нього CheckedListBox;
- ComboBox;
- два елементи Button.

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 усередину GroupBox елементів. Це зробить ваші програми більш привабливими.

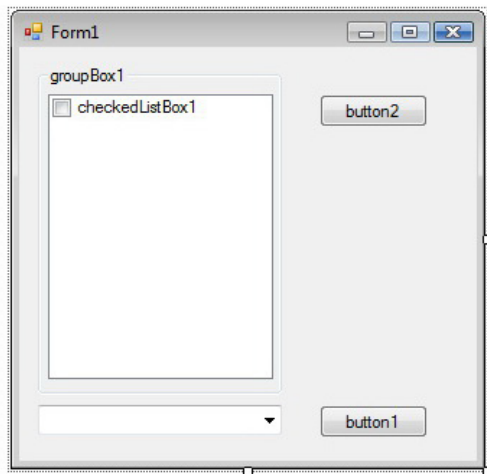


Рисунок 1.5 – Проектування форми програми TestLists

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: Text – «Вилучити»

Елементи керування ComboBox і 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. Давайме 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 в запропонований список три прізвища: Іванов, Петров, Сидоров (рис. 1.6).

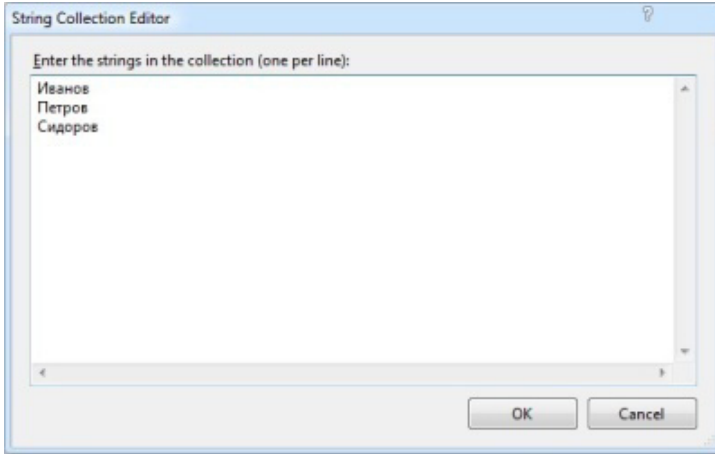


Рисунок 1.6 – Редактор списку рядків

І наостанок, додайте оброблювачі для кнопок Додати й Вилучити, два рази клацнувши лівою кнопкою миші по кожній із кнопок.

Підготовчий етап до написання програми завершений. Збережіть зроблені вами зміни. Тепер давайте 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 () {
//
// 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.Dispose();
    }
    base.Dispose ( disposing ) ;
}

#region Windows 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.TabIndex = 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.TabIndex = 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.TabIndex = 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.TabIndex = 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,
this.groupBox1) ;
this.Name = "TestListsForm";
this.Text = "Робота зі списками";
this.groupBox1.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, Sys-
tem.EventArgs e)
{

}

private void buttonadd_Click (object sender,   System.EventArgs
e)
{

}
}
}

```

Зараз необхідно додати оброблювачі для кнопки «Додати» і «Вилучити». Як відомо з вихідних даних, кнопка «Додати» повинна зтягати рядок, обраний у комбобоксі, у список учасників. Для цього змініть функцію `buttonadd_Click` так, як показано нижче:

```

private void buttonadd_click(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 `memberlist.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'єкта
memberlist.Items.RemoveAt(memberlist.CheckedIndices[0] ) ;
// при видаленні зі списку останнього позначеного елемента
// 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 вилюченим, 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 в списку `CheckedIndices` буде залишатися хоч один елемент.

1.3 Завдання до роботи

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

1.3.2 Вивчити основні принципи роботи з Visual Studio.

1.3.3 Виконати наступні завдання:

Загальні завдання (виконується всіма студентами):

1. Створити новий проект з двома елементами `TextBox` та одним `Button`. Зробити так, щоб при натисканні кнопки `Button` вводимі дані в одному з елементів `TextBox` повторювалися в іншому.
2. Створити новий проект з двома елементами `TextBox`. Зробити так щоб вводимі дані в одному з них повторювалися в іншому у реальному часі.

Індивідуальні завдання (номер завдання узгоджується з викладачем):

1. Створити новий проект 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 проект з потрібними елементами. Зробити так, щоб обраний елемент випадаючого списку автоматично вписувався до текстового блоку, тільки при натисканні на кнопку. При цьому передбачити можливість: дозволяти вивід чи ні.
3. Створити новий проект з потрібними елементами. Зробити так, щоб вводимі дані у текстовий блок 1 дублювалися до інших блоків 2 та 3, у режимі реального часу. Передбачити можливість керування дублювання до блоку 2 та блоку 3 (використовувати CheckBox).
4. Створити новий проект з потрібними елементами. Зробити так, щоб при натисканні на кнопку, що керує процесом, вводимі дані у текстовий блок 1 дублювалися до інших блоків 2 та 3. Передбачити можливість керування дублювання до блоку 2 та блоку 3 (використовувати RadioButton).
5. Створити новий проект з потрібними елементами. Реалізувати множення (*) двох чисел та виведення результату на екран, у реальному часі.
6. Створити новий проект з потрібними елементами. Реалізувати ділення (/) двох чисел та виведення результату на екран, у реальному часі.
7. Створити новий проект з потрібними елементами. Реалізувати суму (+) двох чисел та виведення результату на екран, у реальному часі.
8. Створити новий проект з потрібними елементами. Реалізувати різницю (-) двох чисел та виведення результату на екран, у реальному часі.
9. Створити новий проект 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 проект 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. Передбачити функцію запам'ятовування результатів та виводу їх у будь який момент часу.

1.3.4 Оформити звіт з роботи.

1.3.5 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Завдання до роботи.

1.4.3 Короткі теоретичні відомості.

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

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

1.5 Контрольні запитання

1.5.1 Які основні функції виконує Visual Studio?

1.5.2 Які типи даних підтримуються у C#?

1.5.3 Що таке структура у C#?

1.5.4 Які властивості елементу Button?

1.5.5 Які властивості елементу CheckBox?

1.5.6 Які властивості елементу RadioButton?

1.5.7 Наведіть спільні та відмінні властивості елементів CheckBox та RadioButton.

1.5.8 Які властивості елементу GroupBox?

1.5.9 Які властивості елементу TextBox?

1.5.10 Які властивості елементу RichTextBox?

1.5.11 Порівняйте елементи TextBox та RichTextBox.

1.5.12 Списки ListBox, CheckedListBox, ComboBox: загальна характеристика.

1.5.13 Які властивості елементу ListBox?

1.5.14 Які властивості елементу CheckedListBox?

1.5.15 Які властивості елементу ComboBox?

2 ЛАБОРАТОРНА РОБОТА № 2 ДРУГОРЯДНІ ЕЛЕМЕНТИ ОБОЛОНКИ ПРОГРАМИ

2.1 Мета роботи

Вивчити основні прийоми взаємодії з другорядними елементами програми.

2.2 Основні теоретичні відомості

2.2.1 Мітка – Label

A Label – Елемент керування Label призначений для створення підписів до інших елементів керування або для виводу інформаційних повідомлень прямо на поверхні форми. Наприклад, ви можете поєднувати мітки з полями введення (див. рис. 2.1).

Форма TestListsForm розширена за рахунок додавання елемента керування Label з написом «Виберіть прізвище учасника або введіть нове». Це підвищує рівень сприйняття програми користувачем.

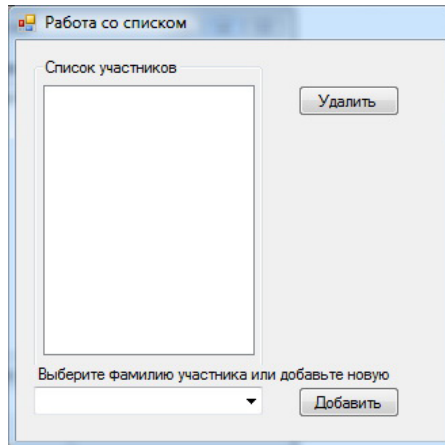


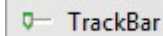
Рисунок 2.1 – Додавання елемента керування Label до форми додатка TestLists

2.2.2 Мітка – LinkLabel



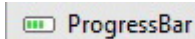
– Linklabel являє собою гіперпосилання, якими наповнений Інтернет. Розроблювачі Visual Studio .NET представили цей елемент керування як різновид мітки (елемент керування Label).

2.2.3 Бігунок – TrackBar



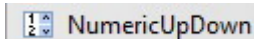
– Типовим прикладом застосування елемента TrackBar є регулятор рівня гучності в панелі Windows. TrackBar може використовуватися в різних режимах: у горизонтальнім або вертикальнім положенні, із включеними рисками або без. Розглянемо далі використання елемента керування TrackBar на прикладі.

2.2.4 Індикатор прогресу – ProgressBar



– Найчастіше ProgressBar використовують для відображення ступеня завершеності тієї або іншої операції.

2.2.5 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 і елемент керування управляють індикатором прогресу. Додаткова умова: бігунок і NumericUpDown повинні працювати синхронно. Тобто, при зміні значення одного елемента, значення іншого повинне змінюватися автоматично на ту ж величину.

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

– TrackBar

- ProgressBar
- NumericUpDown.

Змініть властивості елементів керування. Властивості елемента TrackBar1:

Maximum — 100

TickStyle — Both

При цьому TrackBar змінить свій вид. 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 вибрали 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 в крайньому лівому положенні, його властивості Value буде рівно 0, а при знаходженні бігунка в крайньому правому положенні властивість Value буде мати значення 100.

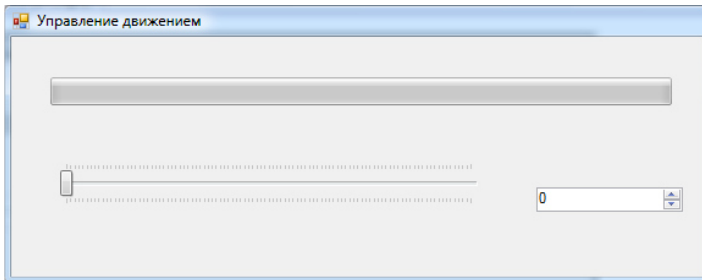


Рисунок 2.2 – Проектування форми програми TestIndicator

Властивості об'єкта numericUpDown1 залишимо за замовчуванням. Елемент керування NumericUpDown також має властивості Minimum і Maximum. І за замовчуванням, властивість Minimum дорівнює 0, властивість Maximum дорівнює 100. Це відповідає параметрам, встановленим для об'єкта trackBar1. Дуже важлива властивість компонента NumericUpDown – DecimalPlaces. Вона визначає кількість знаків після коми. 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».

Гляньте на код програми. Він не має особливого значення, а лише відбиває візуальний вміст форми у кодї програми мовою C#.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
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.Dispose();
                }
            }
            base.Dispose ( disposing );
        }

        # region Windows Form designer generated code

        #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 додається 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 елемента `numericUpDown1`. Для цього клацніть два рази покажчиком миші по імені події `ValueChanged` у вікні властивостей. У код програми додається 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 й запустіть програму. Спробуйте змінити положення бігунка. При цьому індикатор прогресу й 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).

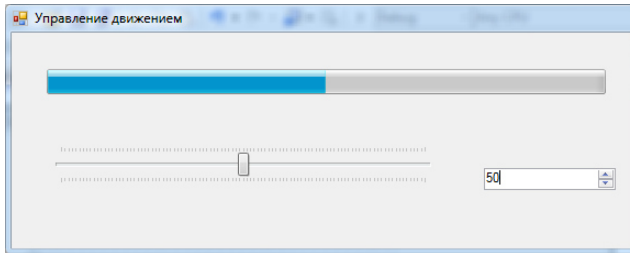
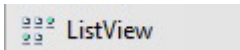


Рисунок 2.3 – Вікно програми «Керування рухом»

2.2.6 Список – ListView



– Елемент керування ListView призначений для відображення списків даних. Компонент ListView відрізняється від ListBox розширеним списком можливостей: установка піктограм для кожного елемента списку, відображення даних у декількох колонках, кілька стилів вистави елементів списку. Типовим прикладом використання елемента ListView є права сторона програми «Провідник», яка входить у стандартну поставку Microsoft Windows (рис. 2.4).

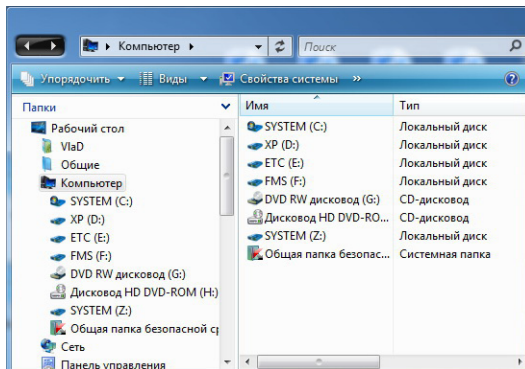
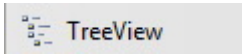


Рисунок 2.4 – Провідник

2.2.7 Дерево – TreeView



– Компонент TreeView призначений для відображення даних у вигляді дерева. Тобто елементи представлення починаються з кореня дерева й відображаються вглиб. Прикладом може служити ліва сторона програми «Провідник», яка відображає дерево каталогів (див. рис. 2.4).

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

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

Для цього створіть нове C# Windows-програму під назвою Viewapp. Додайте на форму наступні елементи (рис. 2.5):

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

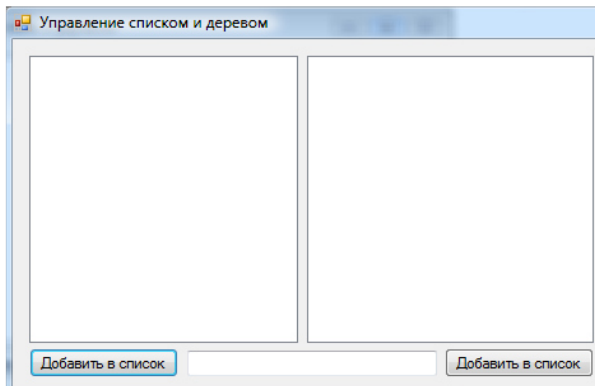


Рисунок 2.5 – Проектування форми програми Viewapp

Змініть властивості доданих компонентів:

button1: text — «додати в список»

button2: text — «додати в дерево»

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;
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 button1;
        private System.Windows.Forms.Button button2;
        //<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
            //
        }

        // <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
#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 в список. 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.SelectedIndices collection =
listView1.SelectedIndices;

    // 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 й запустіть програму. Виконайте описану нижче послідовність дій: наберіть у поле введення «П'ять». Натисніть кнопку Додати в список. Наберіть у поле введення «Три». Натисніть кнопку Додати в список. Наберіть у поле введення «Один». Натисніть кнопку Додати в список. Елементи розташуються в списку так, як показано на

рис. 2.6. Кожний новий елемент буде доданий у початок списку. При цьому всі попередні змістяться вниз.

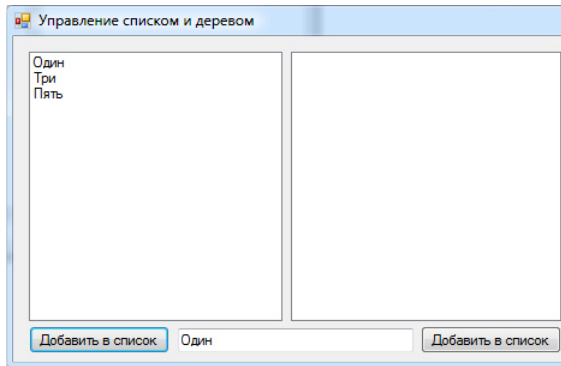


Рисунок 2.6 – Додавання елементів у список

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

```
ListView.SelectedIndexCollection collection =  
listView1.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 у початок списку.

```
else  
idx = collection[0];
```

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

```
listView1.Items.Insert(idx, textBox1.Text);
```

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

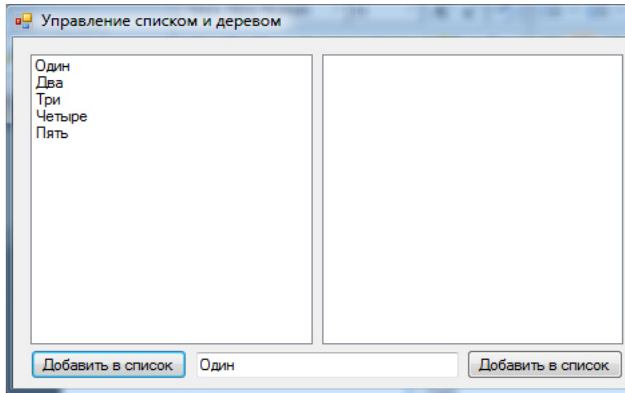


Рисунок 2.7 – Результати роботи зі списком

2.3 Завдання до роботи

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

2.3.2 Виконати наступні завдання:

Загальні завдання:

1. Возведення числа у регульовану ступінь, за допомогою `NumericUpDown`. При цьому за допомогою `ProgressBar` демонструвати ступінь виконання операції (для цього возведення у ступінь доцільно виконувати за допомогою операції множення з використання операторів циклу), результат вивести на екран. Врахувати оформлення програми (при необхідності використовувати додаткові візуальні елементи з відповідними властивостями).

2. Створити програму для керування двома списками ListView, надавати можливість видаляти елементи списку, додавати та перемішувати з першого до другого і навпаки. Врахувати оформлення програми.

Індивідуальні завдання:

1. Реалізувати програму, що обчислює значення виразу
$$\sum_{i=1}^N \sum_{j=1}^K \frac{1}{i+j^2}$$
, де N та K – числа, що задає користувач.

Врахувати оформлення програми, демонструвати ступінь виконання операції.

2. Реалізувати рух елементу Label по 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. Врахувати оформлення програми.

3. Возведення числа у регульовану ступінь, за допомогою NumericUpDown, при цьому за допомогою ProgressBar демонструвати ступінь виконання операції, та за допомогою TrackBar керувати часовою затримкою між возведеннями, результат вивести на екран. Врахувати оформлення програми.

4. Реалізувати програму для обчислення факторіалу довільної величини, за схемою індивідуального завдання №3 до лабораторної роботи № 2. Врахувати оформлення програми.

5. Реалізувати програму для обчислення факторіалу довільної величини, за схемою загального завдання №1 до лабораторної роботи № 2. Врахувати оформлення програми.

6. Реалізувати обчислення функції $\sin(x)$, де $0 < x < 2 \cdot \pi$ з кроком $\pi/6$. Результат вивести у ListView. Врахувати оформлення програми.

7. Реалізувати обчислення функції $\cos(x)$, де $0 < x < \pi$ з кроком $\pi/9$. Результат вивести у ListView. Врахувати оформлення програми.

8. Реалізувати програму, що обчислює значення виразу
$$\sum_{j=1}^N \frac{1}{(j^2)!}$$
, де N – число, що задає користувач. Врахувати

оформлення програми, демонструвати ступінь виконання операції.

9. Реалізувати програму яка введе число інтерпретує у текстову форму (словами), виконувати обробку помилок, число не повинно бути більше 1000. Врахувати оформлення програми.
10. Реалізувати програму для створення таблиці множення, з керуванням максимального множника, з виводом у елемент ListView. Візуалізувати прогрес роботи програми елементом ProgressBar. Врахувати оформлення програми.

2.3.3 Оформити звіт з роботи.

2.3.4 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Завдання до роботи.

2.4.3 Короткі теоретичні відомості.

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

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

2.5 Контрольні запитання

2.5.1 Які властивості елементу Label?

2.5.2 Які властивості елементу LinkLabel?

2.5.3 Порівняйте елементи Label та LinkLabel.

2.5.4 Поясніть призначення елементу TrackBar.

2.5.5 Які властивості елементу TrackBar?

2.5.6 В яких випадках доцільним є використання елементу ProgressBar?

2.5.7 Які властивості елементу ProgressBar?

2.5.8 Навіщо призначений елемент NumericUpDown?

2.5.9 Які властивості елементу NumericUpDown?

2.5.10 Які властивості елементу ListView?

2.5.11 Які властивості елементу TreeView?

2.5.12 Наведіть спільні та відмінні риси елементів ListView та TreeView.

2.5.13 Як здійснюється додавання елементу у ListView?

2.5.14 Як здійснюється додавання вузлів у TreeView?

2.5.15 Як здійснюється пошук елементів у ListView?

3 ЛАБОРАТОРНА РОБОТА № 3 РОБОТА З РЯДКАМИ

3.1 Мета роботи

Навчитися працювати з рядками у Visual Studio.

3.2 Основні теоретичні відомості

C# забезпечує вбудовану підтримку роботи зі строками. Більше того, C# оброблює строки як об'єкти, що інкапсують всі методи маніпулювання, сортування та пошуку.

3.2.1 Особливості типу System.String

C# оброблює строки як вбудовані типи, які є гнучкими, потужними й зручними. Кожен об'єкт рядка – 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`. В такий спосіб фактично виконується оголошення об'єкту типу `System.String`, що є одним з вбудованих типів, що підтримуються .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 в .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. `Comparable` 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 нову 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()` і т.д.

3.2.2 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 риси (\) і використовуються для вказівки перекладу рядка або вставки символу табуляції. Оскільки похила риска вліво самостійно використовується в деяких синтаксисах рядків, типу URL або шляхів каталогу, то в такому рядку похилій рисі вліво повинен передувати інший символ похилої риси вліво.

Рядки можуть також бути створені за допомогою дослівного запису рядка. Такі рядки повинні починатися із символу (@), який повідомляє конструктор `String`, що рядок повинен використовуватися дослівно, навіть якщо він включає службові символи. У дослівнім визначенні рядка похилі риси вліво й 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="@\\"Myfileem\\MyDirectory\\MyFile.txt";
```

У першому рядку використовується звичайний літерал рядка, так що символи похилої риси вліво (\) повинні дублюватися. Це означає, що для відображення (\) потрібно записати (\\). У другому рядку використовується дослівний літеральний рядок, так що додаткова похила риса вліво не потрібна.

Наступний приклад ілюструє багаторядкові рядки:

```
string stringOne = "Line one\nline Two";
string stringTwo = @"Line One
line Two";
```

Такі оголошення рядків також є взаємозамінними.

3.2.3 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 в рядок.

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

Виклик методу ToString() в об'єкта myint поверне строкове представлення числа 10.

Клас System.String в .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 CLR створює екземпляр нового рядка з використанням безпечного коду. При створенні рядка на основі покажчика застосовується «небезпечний» код, що вкрай небажано при розробці додатків .NET.

3.2.4 Маніпулювання рядками

Клас string забезпечує різні вбудовані методи для порівняння, пошуку й 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 налаштувань;

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

Copy() – створює дублікат вихідного рядка;

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 іншою;
 CopyTo() – копіює певне число символів рядка 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 підстроку, відділену від основного масиву певним символом;
 StartsWith() – визначає, чи починається рядок з певної послідовності символів;
 Substring() – повертає підстроку із загального масиву символів;
 ToCharArray() – копіює символи з рядка в масив символів;
 ToLower() – перетворює рядок до нижнього регістру;
 ToUpper() – перетворює 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() і 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 = @"С# являє собою інструмент "+
        "швидкого створення додатків 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 врахування
    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'єкта
    textBox1.Text = textBox1.Text + "str7.Equals(str6): " +
        str7.Equals(str6) + "\r\n";
    //використовуючи статичний метод Equals
```

```

    textBox1.Text = textBox1.Text + "str7 і str6 півні?: " +
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 рядка по його індексу
    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 підстроки в рядок
    textBox1.Text = textBox1.Text + "Перше входження слова інструмент
у рядок str3 має індекс " + str3.IndexOf("інструмент")+ "\r\n";
    // вставляємо нове слово в рядок
    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 регіструstr1: 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 півні?: 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 платформи Чи закінчується цей рядок словом "інструмент"? False
str3: C# являє собою інструмент швидкого створення додатків для
.NET платформи Чи ракінчується 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 рядка str3 використовувалася дослівне представлення рядку. Слід зазначити, що можна розривати рядок у коді програми для переносу його на іншу лінію. 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 врахування
регістру";
textBox1.Text = textBox1.Text + "str1: " + str1 + " і str2: " +
str2 + ", результат: " + result + "\r\n";
```

Відповідно й результат буде:

```
Порівнюємо без врахування регістру str1: абвг, str2:АБВГ, ре-
зультат: 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 в інший. Різниця полягає лише в записі коду програми:

```
string str6 = string.Copy (str5);
string Str7 = Str6;
```

І str6, і str7 будуть у результаті мати те значення, яке записано в 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'єкта 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, і 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 рядка str7, використовуючи число 4 в операторі ([]), оскільки елементи рядку в C#, як і у C++, починаються з нулевого індексу (табл. 3.1).

Таблиця 3.1 – Індксація 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 рядка `str7` є символ «А» індекс, що має, 4.

3.2.5 Пошук підстроки

Тип `string` має перевантажений метод `Substring()` для вилучення підстроки з рядку. Один з методів приймає як параметр індекс елементу, починаючи з якого слід витягти підстроку. Другий метод приймає й початковий, і кінцевий індекс, щоб вказати, де почати й 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 підстроку, 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 s1 на підстроку
// від нульового символу до ix
s1 = s1.Substring(0,ix);

// скидаємо ix на пробіл
// між "один" і "два"
ix = s1.LastIndexOf(" ");
// встановлюємо s4 на підстроку після
string s4 = s1.Substring (ix+1) ;
//встановлюємо s1 на підстроку від 0 до ix
//одержуємо тільки слово "один"
s1 = s1.Substring(0,ix);

// намагаємося одержати індекс останнього пробілу
// але цього разу функція LastIndexOf поверне -1
ix = s1.LastIndexOf(" ");
// встановлюємо s5 на підстроку 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 і 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 й кінця підстроки. Початком рядка в нас буде початок вихідного рядка, 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 й передачі в неї значення $(-1 + 1 = 0)$, нам 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
```

3.2.6 Розбиття рядків

По суті попередній приклад, робив розбір заданого рядка (речення) на слова, зберігав знайдені слова й виводив їх на екран. Більш ефективний розв'язок проблеми, проілюстрованої в попередньому прикладі, полягає в тому, щоб використовувати метод Split() класу string. Метод Split() розбиває рядок у підстроки. Для виділення підстрок 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 підстроки на основі роздільників
// і зберігаємо результат
foreach (string substring in s1.Split(delimiters))
{
    output += ctr++; //номер підстроки
    output += ": ";
    output += substring; //підстрока
    output += "\r\n"; //перехід на нову лінію
}
// вивід 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 й пробіл.

Далі в програмі оголошуються дві константи, що визначають розділові символи.

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

Ці константи поєднуються в один масив.

```
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 й символу коми. Отже, як тільки метод Split() виявить у вихідному рядку один із цих символів, те 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 рядка. Як я вже відзначав раніше, між словами «Три» і «Рядок» розташовуються відразу два розділові символи — і 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 в 0 символів-порожній рядок, який метод Split і поміщає у вихідний масив.

3.2.7 Клас StringBuilder

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

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

Chars – Індексатор класу;

Length – визначає довжину об'єкта StringBuilder;

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

Append() – додає об'єкт заданого типу в кінець StringBuilder;

AppendFormat() – заміщає або встановлює новий формат StringBuilder;

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

Insert() – вставляє об'єкт деякого типу в зазначену позицію;

Remove() – видаляє зазначений символ;

Replace() – заміщає 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. Візьмемо 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 = ',';
// створимо масив розділових символів
// і ініціалізуємо його двома елементами
char[] delimiters = { cspace, ccomma };
StringBuilder output = new StringBuilder();
int ctr = 1;
// виділяємо підстроки 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);
}
// вивід результату
textBox5.Text = output.ToString();
```

Як видно, зміни відбулися лише в нижній частині коду програми. Робота з розбору рядка відбувається як завжди, змінився лише вивід результатів. Для виводу 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. Ми передаємо як формат рядку два значення, розділені символом двокрапки. Перше значення – це номер підстроки, друге – сама підстрока.

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

При виводі результату нам не буде потрібно проводити додаткове форматування тексту, оскільки ми відразу затирали результат у потрібному форматі, проте необхідно перетворити формат рядку `StringBuilder` до звичайного формату рядків для їх виводу у текстовому полі:

```
textBox5.Text = output.ToString();
```

Результат роботи програми залишається таким ж, що й раніше:

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

3.2.8 Регулярні вирази

Регулярні вирази – це один зі способів пошуку підстрок (відповідностей) у рядках. Здійснюється за допомогою перегляду рядку в пошуках деякого шаблону. Загальновідомим прикладом можуть бути символи «*» і «?», що використовуються в командному рядку DOS. Перший з них заміняє нуль або більше довільних символів, другий же – один довільний символ. Так, використання шаблону пошуку типу «text?.*» знайде файли `textf.txt`, `textl.asp` і інші аналогічні, але не знайде `text.txt` або `text.htm`. Якщо в DOS використання регулярних виразів було вкрай обмеженим, то в інших програмних застосування, зокрема в операційних системах і 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).

3.2.8.1 Застосування регулярних виразів

Як правило, за допомогою регулярних виразів виконуються три дії:

- перевірка наявності відповідної до шаблону підстроки;
- пошук і видача користувачеві відповідних до шаблону підстрок;
- заміна відповідних до шаблону підстрок.

У C# робота з регулярними виразами виглядає наступним чином:

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

re – це об'єкт типу Regex. У конструкторі йому передається зразок пошуку й опції. Нижче представлена таблиця 3.2, що задає різні варіанти 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. Часто це значно спрощує життя.
x	Виключає зі зразка неприкриті незначні символи (пробіли, табуляція і т.ін.).
r	Виконує пошук праворуч-ліворуч.

Комбінація прапорів m і s дає дуже зручний режим роботи, що враховує кінці рядків, що й дозволяє пропустити всі незначні символи, включаючи 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;
```

3.2.8.2 Основи синтаксису регулярних виразів. 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[а-і]ржа» відповідає словам «баржа» і «біржа», тобто словам, що починаються з «б», за яким ідуть «а» або «і», що й закінчуються на «ржа».

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

3.2.8.3 Квантифікатори (Quantifiers)

Якщо невідомо, скільки саме знаків повинна містити шукана підстрока, можна використовувати спецсимволи, що називаються квантифікаторами (quantifiers).

Наприклад, можна написати «hel+o», що буде означати слово, що 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 можна побачити в нижчеподаній таблиці 3.3.

Таблиця 3.3 – Опис квантифікаторів.

Символ	Опис
*	Відповідає 0 або більшій кількості входжень попереднього виразу. Наприклад, «zo*» відповідає «z» і «zoo».
+	Відповідає 1 або більшій кількості попередніх виразів. Наприклад, «zo+» відповідає «zo» і «zoo», але не «z».
?	Відповідає 0 або 1 попередніх виразів. Наприклад, «do(es)?» відповідає «do» в «dorf» або «does».
{n}	n – позитивне ціле. Відповідає точній кількості входжень. Наприклад, «o{2}» не знайде «o» в «Bob», але знайде два «o» в «food».

$\{n,\}$	n – позитивне ціле. Відповідає входженню, повтореному не менш n разів. Наприклад, $\{2,\}$ не знаходить "o" в "Bob", проте знаходить усі "o" в "fooooood". $\{1,\}$ еквівалентно $\{o^+\}$. $\{0,\}$ еквівалентно $\{o^*\}$.
$\{n,m\}$	n і m – позитивні цілі числа, де $n \leq m$. Відповідає мінімум n і максимум m входжень. Наприклад, $\{1,3\}$ знаходить три перші "o" в "fooooood". $\{0,1\}$ еквівалентно $\{o^?\}$. Пробіл між комою й цифрами неприпустимий

3.2.8.4 Закінчення й початки рядків

Перевірка початку або кінця рядка проводиться за допомогою метасимволів $\wedge$ і $\$$. Наприклад, $\langle \text{thing} \rangle$ відповідає рядку, що починається з $\langle \text{thing} \rangle$. $\langle \text{thing}\$ \rangle$ відповідає рядку, що закінчується на $\langle \text{thing} \rangle$. Ці символи працюють тільки при включеній опції 's'. При виключеній опції 's' знаходять тільки кінець і початок тексту. Але й у цьому випадку можна знайти кінець і початок рядка, використовуючи escape-послідовності $\backslash A$ и $\backslash Z$.

3.2.8.5 Границя слова

Для завдання границь слова використовуються метасимволи $\backslash b$ і $\backslash B$.

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

У цьому випадку зразок в re відповідає не тільки «мені» у рядку «дайте мені», але й «мені» у рядку «відбулася заміна». 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 $\backslash b$ позначає символ `backspace` (стирання).

Наведені нижче в таблиці 3.4 метасимволи не змушують машину регулярних виразів просуватися по рядкові або захоплювати символи. Вони просто відповідають певному місцю рядку. 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 (ignores the m option).
\Z	Кінець рядка, або перед \n наприкінці рядку (ігнорує опцію m).
\z	Точно кінець рядку (ігнорує опцію m).
\G	Початок поточного пошуку (часто це в одному символі за кінцем останнього пошуку).

Продовження таблиці 3.4

Символ	Значення
\b	На границі між \w (алфавітно-цифровими) і \W (неалфавітно-цифровими) символами. Повертає true на перших і 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].

3.2.8.6 Варіації й 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» підстрок для подальшого використання й 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 й знайде послідовність «like apples», оскільки «apples» — один із трьох перерахованих варіантів. Дужки також помістять «apples» в \$1 як зворотне посилання для подальшого використання. В основному це має сенс при заміні.

Правила побудови регулярних виразів:

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

2) рядок символів позначає рядок цих символів;

3) множина можливих символів (група) поміщається 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 і т.ін;

4) усі символи, зокрема і спеціальні, можна позначати за допомогою '\';

5) альтернативні послідовності розділяються символом '|'. Проте усередині квадратних дужок це звичайний символ.

Деякі приклади використання регулярних виразів наведено у табл. 3.5.

Таблиця 3.5 – Приклади використання

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

3.2.9 Використання регулярних виразів: Regex

.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 в 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». Ми оголосили його як s1. Далі ми оголосили 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, що йдуть підряд комой і пробіл. Рядок, що задає регулярний вираз, передається в якості параметру конструктору об'єкту 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 для вас способом, за допомогою класу String Builder, формується вихідний рядок.

Метод Split класу Regex перевантажений і може використовуватися у двох варіантах. Перший з них був показаний 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 нам не довелося створювати екземпляр об'єкта `Regex` У цьому випадку для пошуку використовувався статичний метод `Regex.Split`, який ухвалює два параметри – рядок, у якому буде проводитися пошук, і рядок з регулярним виразом.

Крім цього, метод `Split` перевантажений ще двома варіантами, які дозволяють обмежити кількість раз використання методу `Split` і задати початкову позицію в рядку для пошуку.

3.2.10 Використання `Match` колекцій

Простір імен містить два класи, які дозволяють здійснювати ітераційний пошук у рядку й 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`) і значення (`Value`). Давайте розглянемо 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. (\\S) означає будь-який не пробільний символ. Знак (+), що стоїть після (\\S), означає, що може бути будь-яка кількість не пробільних символів. (\\s) у нижньому регістрі-пробільний символ. У цілому, цей вираз означає: «Знайти всі набори, які починаються з не пробільного символу й закінчуються пробільним».

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

3.3 Завдання до роботи

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

3.3.2 Виконати наступні завдання:

Загальні завдання:

1. Розробити архіватор: символи строки, що повторюються, замінити на послідовність – {символЧислоПовторювань}, наприклад: «fehhh eгуaaa» повинна перетворитися на строку виду «feh3 eгуa3». Також реалізувати зворотну функцію програми.
2. 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».

Індивідуальні завдання:

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.

3.3.3 Оформити звіт з роботи.

3.3.4 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Завдання до роботи.

3.4.3 Короткі теоретичні відомості.

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

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

3.5 Контрольні запитання

3.5.1 Які основні функції роботи зі строками?

3.5.2 Які основні функції порівняння строк?

3.5.3 Які основні функції модифікатори строк?

3.5.4 Які основні функції пошуку підстроки?

3.5.5 Що таке клас `StringBuilder`?

3.5.6 Що таке «регулярний вираз»?

3.5.7 Наведіть переваги та недоліки роботи з регулярними виразами.

3.5.8 Які основи роботи з регулярними виразами?

3.5.9 Що таке «квантифікатор»?

3.5.10 Наведіть приклад використання варіацій та групування у регулярних виразах.

3.5.11 Користуючись чим можна знайти потрібну підстроку у строці?

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

3.5.13 Який клас є головним для обробки регулярних виразів?

3.5.14 Як здійснюється видалення частини строки?

3.5.15 Поясніть призначення `Match` колекцій.

4 ЛАБОРАТОРНА РОБОТА № 4 РОБОТА З ФАЙЛАМИ

4.1 Мета роботи

Вивчити основні принципи роботи з файлами в середовищі Visual Studio.

4.2 Основні теоретичні відомості

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

4.2.1 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 в порівнянні з файлами. Потоки дозволяють записувати й 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 в буфер, і лише при виклику певних команд переноситься у файл.

Основними класами для роботи з файлами й потоками у C# є File, FileStream и StreamReader. Клас File призначений для створення, відкриття, видалення, зміни атрибутів файлу. Клас FileStream призначений для читання й запису інформації у файл. 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 і проініціалізувати його відкритим файлом. Оскільки всі методи класу File є статичними (не прив'язані до об'єктів), немає необхідності створювати екземпляр класу File. От типовий приклад ініціалізації об'єкта FileStream:

```
FileStream mystream = File.Open("3:\Myfile.txt", FileMode.Open,
FileAccess.Read);
```

У якості додаткового інструменту для роботи з текстовими файлами розроблювачами C# були зроблені класи `StreamReader` і `StreamWriter`. Вони дозволяють читати й писати дані з потоку построково, посимвольно, відразу все. `StreamReader` і `StreamWriter` зв'язуються з потоком за допомогою конструктору ініціалізації:

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

4.2.2 Атрибути відкриття файлів

При відкритті файлу завжди необхідно вказувати режим відкриття файлу й 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 у файл.

4.2.3 Діалоги відкриття та збереження файлів

На рис. 4.1 наведено приклад діалогу для відкриття або запису файлу. Для роботи з діалогами відкриття й збереження файлів використовуються компоненти `OpenFileDialog` і `SaveFileDialog`. Вони багато в чому схожі. Для відображення діалогу відкриття файлу вам необхідно лише створити об'єкт класу `OpenFileDialog` і викликати його метод `ShowDialog`. Після закриття діалогу властивість `FileName` зберігає ім'я обраного файлу й повний шлях до нього.

Як *приклад*, створимо найпростіший текстовий редактор, що дозволяє читати текстові файли, редагувати інформацію й зберігати її у файл.

Створіть новий додаток з іменем `Fileapp`. Перейменуйте властивість `Text` форми в «Текстовий редактор». Помістіть на форму компонент `Textbox` і змініть його властивості:

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

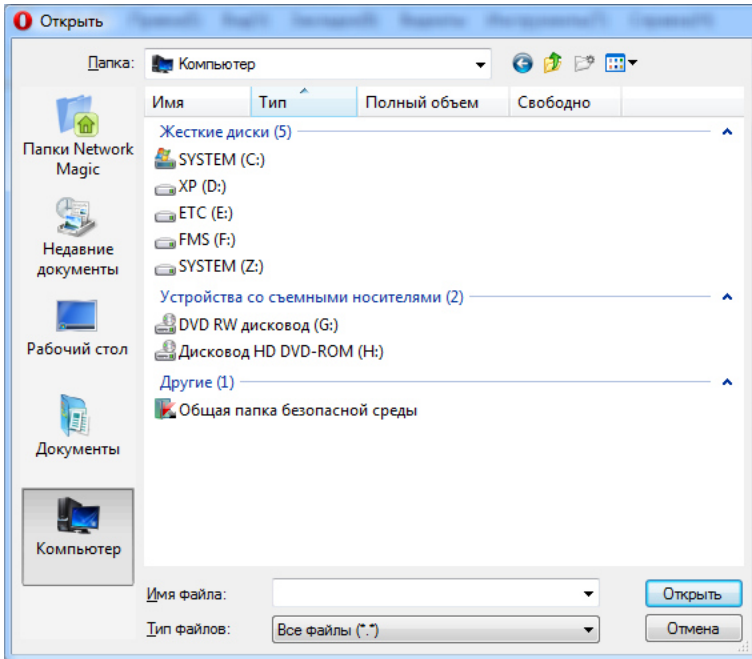


Рисунок 4.1 – Діалог відкриття файлу

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 установлене в Fill, то елемент займе всю площу форми.

Помістіть на форму компонент MainMenu. Створіть у ньому один пункт «Файл» із двома підпунктами «Відкрити» і «Зберегти». Змініть властивість Name пунктів «Відкрити» і «Зберегти» на menuItemOpen і menuItemSave. Створіть оброблювачі для пунктів меню «Відкрити» і «Зберегти». Залиште для них імена за замовчуванням, просто клацнувши два рази покажчиком миші по відповідних до пунктів меню. При цьому у код програми повинні були додатися методи menuItemOpen_Click і menuItemSave_Click.

Додайте на форму компоненти OpenFileDialog і SaveFileDialog. Для обох встановіть властивість Filter як «Text files (*.txt)!*.txt». Це

означає, що в діалозі будуть показуватися тільки файли з розширенням «txt».

Замініть оброблювачі відкриття й збереження файлів так, як показано нижче:

```
private void menuItemOpen_Click (object sender, Sys-
tem.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, File-
Access.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, Sys-
tem.EvantdArgs e) {
    // показуємо діалог вибору файлу
    SaveFileDialog1.ShowDialog();
    // одержуємо ім'я файлу
    string filename = SaveFileDiaiog1.FileName;
    // відкриваємо файл для запису й асоціюємо з кім потік
    FileStream.stream = File.Open (filename, FileMode.Create, File-
Access.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();
    }
}
```

Докладний опис роботи методів подано у коментарях. Робота із читанням файлу відбувається в 6 етапів:

– відкриття файлу;

- асоціація файлу з потоком;
- асоціація потоку з StreamReader;
- читання даних;
- перенесення даних в TextBox;
- закриття файлу.

Запис файлу також проходить в 6 етапів:

- відкриття файлу;
- асоціація файлу з потоком;
- асоціація потоку з StreamWriter;
- запис даних;
- звільнення потоку;
- закриття файлу.

Запустіть програму. 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 у файл. Ви можете переконалися в цьому, скориставшись програмою «Блокнот».

4.3 Завдання до роботи

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

4.3.2 Вивчити основні принципи роботи з файлами.

4.3.3 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 кожен сеанс роботи програми, нову сесію дописувати у початок файлу. Під час роботи програми реалізувати завантаження збереженої сесії.
2. Зберігати кожен сеанс 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.
6. Зберігати кожну сесію роботи програми, нову сесію дописувати у початок файлу використовуючи діалог вибору файлу. Під час роботи програми реалізувати завантаження збереженої сесії використовуючи діалог вибору файлу.

4.3.4 Оформити звіт з роботи.

4.3.5 Відповісти на контрольні питання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Завдання до роботи.

4.4.3 Короткі теоретичні відомості.

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

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

4.5 Контрольні запитання

4.5.1 Що таке файловий потік?

4.5.2 Які класи є основними для роботи з файлами й потоками у C#?

4.5.3 Поясніть призначення та використання об'єкту типу FileStream.

4.5.4 Які атрибути існують відкриття файлів?

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

4.5.6 Як прочитати файл?

4.5.7 Як записати файл?

4.5.8 Які відмінності роботи з файлами у Visual Studio та VC++?

4.5.9 Як керувати переміщенням каретки у файлі?

4.5.10 Що таке діалоги відкриття та збереження файлів?

4.5.11 Які компоненти використовуються для роботи з діалогами відкриття й збереження файлів?

4.5.12 Яка властивість файлового діалогу зберігає ім'я обраного файлу й повний шлях до нього?

4.5.13 Як налаштувати діалог вибору файлу для вибору декількох файлів одночасно?

4.5.14 Як здійснюється очищення файлового потоку?

4.5.15 Як здійснюється асоціація потоку з StreamReader?

5 ЛАБОРАТОРНА РОБОТА № 5 ОБРОБКА ПОДІЙ МИШІ

5.1 Мета роботи

Навчитися основним принципам обробки подій маніпулятора в середовищі Visual Studio.

5.2 Основні теоретичні відомості

5.2.1 Обробка повідомлень миші

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

5.2.2 Види подій

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

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

MouseDown – обробка натискання яких-небудь із кнопок униз;

MouseEnter – викликається при влученні покажчика миші в область форми;

MouseOver – викликається при зависанні покажчика миші у вікні форми;

MouseLeave – викликається при покиданні курсору миші області форми;

MouseHover – викликається при наведенні покажчика миші на об'єкт

MouseMove – викликається при русі миші в області форми;

MouseUp – викликається при відпусканні кнопки миші.

Для прикладу програми обробки повідомлень миші будемо використовувати програму Grapheditorapp. Відкрийте у вікні дизайнера форму Form2.

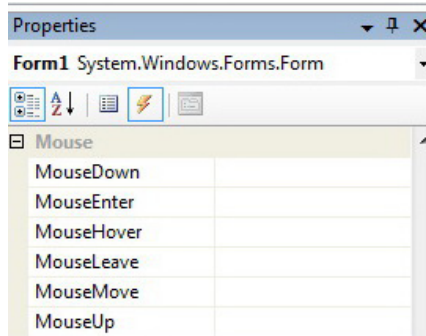


Рисунок 5.1 – Mouse events

5.2.3 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 в нижчеподаному коді:

```
private void Form2_MouseDown (object sender, Sys-
tem.Windows.Forms.MouseEventArgs e)
{
    string text;
    MouseButton button;
    button = e.Button;
    if (button == MouseButton.Left)
    {
        text = "ліву";
    }
    else if (button == MouseButton.Right)
    {
        text = "праву";
    }
    else
    {
        text = "середню";
    }
    string message = "Ви нажали " + text + " кнопку миші в
координатах:\n" +
        "x:= " + e.X.ToString() + "\n" +
```

```

"в:" + e.Y.ToString();
MessageBox.Show(message);
}

```

Параметр функції `MouseEventArgs` містить усю інформацію про кнопку, що послала повідомлення. Властивість `MouseEvetArgs.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-якій місці дочірнього вікна. На екрані з'явиться повідомлення, аналогічне тому, яке зображено на рис. 5.2.

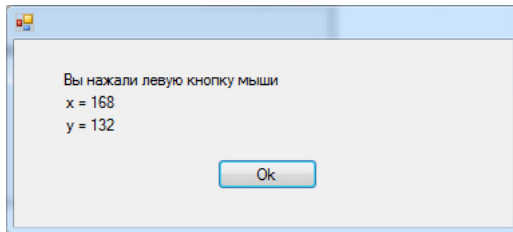


Рисунок 5.2 – Повідомлення про натискання лівої кнопки миші

Точно так само можна обробляти інші повідомлення від миші.

5.3 Завдання до роботи

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

5.3.2 Вивчити основні принципи роботи Visual Studio з подіями маніпулятора.

5.3.3 Виконати наступні завдання:

1. 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.
2. 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.
3. Реалізувати обробку подій потрійного натискання кнопки миші. Обрати елементи керування програмою (свій вибір обґрунтувати), та за допомогою потрійного натискання миші реалізувати зміну положення елемента, та зміну його стилю.

5.3.4 Оформити звіт з роботи.

5.3.5 Відповісти на контрольні питання.

5.4 Зміст звіту

5.4.1 Тема та мета роботи.

5.4.2 Завдання до роботи.

5.4.3 Короткі теоретичні відомості.

5.4.4 Результати виконання роботи.

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

5.5 Контрольні запитання

5.5.1 Які події миші оброблюються C#?

5.5.2 Які параметри подій миші?

5.5.3 Як отримати поточні координати курсору?

5.5.4 Чим відрізняється подія MouseHover від MouseMove?

- 5.5.5 Як обробити подвійне натискання правої кнопки миші?
- 5.5.6 Чим відрізняється подія MouseUp відMouseDown?
- 5.5.7 Як реалізувати перетягування об'єкту мишею?
- 5.5.8 Які константи є у MouseButtons?
- 5.5.9 Як можливо обробляти не основні кнопки миші?
- 5.5.10 Як програмно емулювати натискання кнопки миші?

6 ЛАБОРАТОРНА РОБОТА № 6 ОБРОБКА ПОДІЙ КЛАВІАТУРИ

6.1 Мета роботи

Навчитися основним принципам обробки подій клавіатури в середовищі Visual Studio.

6.2 Основні теоретичні відомості

6.2.1 Робота з клавіатурою

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

6.2.2 Повідомлення клавіатури

Для обробки повідомлень із клавіатури в Windows Forms програмах передбачено чотири події: KeyUp, KeyPress, KeyDown, PreviewKeyDown.

Подія KeyUp посилає при відпусканні кнопки на клавіатурі.

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

Виконується перед подією KeyDown при натисканні кнопки, коли елемент управління має фокус.

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

- Text – «Інформація про натиснуті клавіші»;
- KeyPreview – True.

Властивість `Text` задає заголовок вікна. Властивість `KeyPreview` визначає властивість, що дозволяє формі вловлювати повідомлення клавіатури від дочірніх елементів керування формою. Якщо властивість `KeyPreview` форми встановлена в `False`, то форма не буде одержувати повідомлення від клавіатури, якщо активізовано один з елементів форми. Інакше кажучи, якщо на формі присутня компонент `TextBox` і курсор миші перебуває в його полі, то при натисканні клавіші на клавіатурі форма про це довідатися не зможе. Тому, якщо ви збираєтесь обробляти клавіатуру в класі форми, вам необхідно виставляти властивість `KeyPreview` в `true`. Додайте на форму елемент керування `TextBox`.

Змініть деякі властивості елемента `TextBox`:

- `Text` — «»;
- `ReadOnly` — `True`;
- `TabStop` — `False`.

Форма повинна виглядати так, як показано на рис. 6.1.

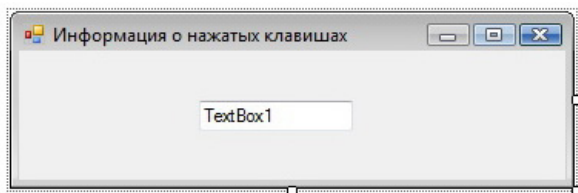


Рисунок 6.1 – Проектування форми програми `Keyboardapp`

Додайте у програму оброблювач події `KeyDown`. Для цього у вікні властивостей форми клацніть два рази покажчиком миші по події `KeyDown` у вікні властивостей форми. У код програми ввімкнеться оброблювач події `KeyDown` з іменем `Form1_KeyDown`. Додайте до оброблювача код, представлений нижче:

```
private void Form1_KeyDown(object sender, System.Windows.Forms.KeyEventArgs e) {
    // очищаємо поле textbox1.Text = "";
    // перевіряємо чи натиснута клавіша Ctrl
    // якщо так, те записуємо у поле слово Ctrl
    if(e.Control) {
        textbox1.Text += "Ctrl+";
    }
    // перевіряємо чи натиснута клавіша Shift
    // якщо так, те записуємо у поле слово Shift
```

```

if (e.Shift){
    textBox1.Text += "Shift+";
}
// перевіряємо чи натиснута клавіша Alt
// якщо так, те 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();

```

6.2.3 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` рівно `false`. Якщо ви не прагнете подальшої обробки натискання кнопки, виставте прапор `Handled` в `true`.

Для виводу на екран інформації про натиснуті керуючі клавіші ми перевіряємо значення властивостей `Alt`, `Ctrl` і `Shift`. Якщо одна із клавіш натиснута, то в поле `Textbox` додається відповідне слово.

Для того щоб вивести на екран інформацію про основну натиснуту клавішу, ми повинні попередньо вилучити з поля `KeyData` системну інформацію. Тому ми скидаємо в 0 прапори 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 само віддаляються коди клавіш `Alt` і `Shift`.

Метод `ToString` повертає словесний опис кожної клавіші. Код звичайної клавіші виводиться однією буквою; код системної клавіші виводиться словом, відповідним до клавіші. Запустіть програму. Спробуйте натискати кнопки на клавіатурі, поєднуючи їх з керуючими клавішами. На екрані буде відображатися інформація у вигляді рядка, що містить усі натиснуті клавіші (рис. 6.2).

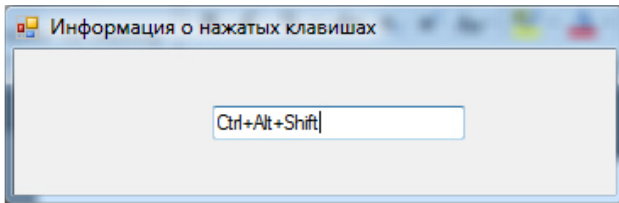


Рисунок 6.2 – Інформація про натиснуті клавіші

6.2.4 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 в певний момент часу.

6.2.5 Компонент Timer

Робота з таймером в Windows Forms програмах заснована на все тому ж механізмі подій. Ви встановлюєте таймер на певну частоту, і операційна система буде розсилати вашому додатку події оповіщення із зазначеною частотою.

Компонент `Timer` дозволяє легко й просто працювати з часом. Основними властивостями компонента `Timer` є:

- `Interval` – задає період приймання повідомлень таймером у міль секундах;
- `Enabled` — визначає стан Включення/Вимикання таймера;

Для роботи з таймером вам необхідно лише помістити на форму компонентів Timer, встановити його властивість Interval на заданий інтервал часу й обробити подія Elapsed.

6.2.6 Компонент – DateTimePicker

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

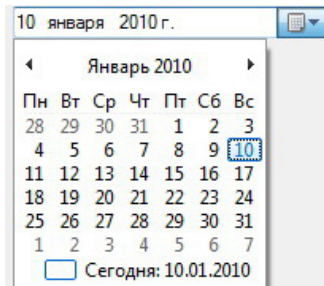


Рисунок 6.3 – Елемент керування DateTimePicker

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

Основні властивості компонента DateTimePicker наступні:

- Format – дозволяє встановити один зі стандартних форматів відображення часу або вказати свій;
- ShowUpDown – встановлює тип елемента із правої сторони поля відображення. Якщо встановити False – відображається ComboBox, що відкриває календар, якщо True – відображається 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.

6.2.7 Структура DateTime

Структура DateTime призначена для зберігання й 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, мілі секунда). Крім того, DateTime має набір методів для обробки тимчасових інтервалів. Наприклад, можна скласти два тимчасові значення, відняти, конвертувати в інший формат і т.д.

6.2.8 Формат рядку часу

Дуже важливою особливістю структури DateTime є можливість перекладу тимчасового значення в строковий формат. При цьому формат рядка, що вертається, може бути заданий динамічно. Метод DateTime.ToString повертає рядок часу на підставі формату аргументу. От деякі константи, що визначають формат рядка часу:

- dd – два знаки дня місяця. Якщо день складається з однієї цифри – спереду ставиться незначний 0;
- dddd – день тижня;
- MM – номер місяця (1–12);
- MMMM – назва місяця;
- yyy – 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. Приклади формату часу наведено у табл. 6.1.

Таблиця 6.1 – Формат часу

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

Як видно з таблиці, формат часу може бути дуже зручно підбудований під конкретне завдання.

6.3 Завдання до роботи

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

6.3.2 Вивчити основні принципи роботи з клавіатурою та часом.

6.3.3 Виконати наступні завдання:

1. Реалізувати програму «клавіатурний тренажер». У програмі на екран видається певний символ, який треба ввести, та таймер для його вводу. Передбачити 5 ступенів важкості – регулювання відведеного часу та кількості літер, слів для вводу. Також передбачити таблицю найкращих результатів, з можливістю автоматичного запису та зчитування з файлу. Крім того реалізувати зберігання та завантаження прогресу користувача.
2. Реалізувати програму «будильник». Надати можливість виставлення сигналу на конкретний час, дату, день тижня, та коротке повідомлення. Інформацію про виставлені режими роботи зберігати у файлі.
3. Реалізувати «конвертер». Фіксувати кожне натискання клавіш клавіатури, та виводити на екран відповідний код натиснутої клавіші. Зберігати лог роботи програми у файлі у вигляді: код натиснутої клавіші – назва клавіші.

6.3.4 Оформити звіт з роботи.

6.3.5 Відповісти на контрольні питання.

6.4 Зміст звіту

6.4.1 Тема та мета роботи.

6.4.2 Завдання до роботи.

6.4.3 Короткі теоретичні відомості.

6.4.4 Результати виконання роботи.

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

6.5 Контрольні запитання

6.5.1 Назвіть основні події клавіатури.

6.5.2 Що містить клас EventArgs?

6.5.3 Як отримати інформацію о натисканні керуючих клавіш?

6.5.4 Для чого використовують метод ToString?

6.5.5 Назвіть основні властивості компоненту Timer.

6.5.6 Назвіть основні властивості компоненту DateTimePicker.

6.5.7 Для чого призначена структура DateTime?

6.5.8 Для чого призначена структура TimeSpan?

6.5.9 Вкажіть формат для виводу дати «2002-21-11 GMT+2 14:48:12.87»?

6.5.10 Як отримати поточний час?

7 ЛАБОРАТОРНА РОБОТА № 7 РОБОТА З ЗОБРАЖЕННЯМИ

7.1 Мета роботи

Навчитися працювати із зображення за допомогою C#.

7.2 Основні теоретичні відомості

7.2.1 Особливості GDI+

Для малювання об'єкту у Windows Forms програмі мови C# містить дуже багатий набір методів. Простір імен Drawing містить множину об'єктів, які полегшують програмістові роботу із графікою. Спеціально для .NET платформи розроблювачі Microsoft розробили GDI+ бібліотеку, значно підвищивши можливості GDI (Graphic Device Interface). GDI+ включає можливості малювання найпростіших об'єктів (лінії, еліпси), малювання різних об'єктів 2D графіки, відображення файлів різних графічних форматів (bmp, jpeg, gif, wmf, ico, tiff) і багато чого іншого. Ми розглянемо роботу з найпростішими об'єктами для малювання: лініями, окружностями, шрифтами.

7.2.2 Малювання об'єктів

Малювання графічних примітивів розглянемо на прикладі програми.

Для малювання лінії вам необхідно буде два рази клацнути мишею в різних частинах вікна. При цьому лінія намалюється від координати, у якій відбулося перше клацання мишею в координату, де відбулося друге клацання мишею. Це нетипово для графічного редактора. Звичайно лінія тягнеться від координати першого клацання миші за курсором доти, поки користувач не відпустить кнопку. Але така функціональність може ускладнити код програми. Для наших навчальних цілей ми обійдемося двома клацаннями курсору миші.

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

Малювання тексту. При клацанні миші рядок тексту буде відображатися праворуч від курсору.

Особливий інтерес 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 ми будемо малювати лінії між передостанньою координатою курсору й останньої. Для цього нам доведеться постійно запам'ятовувати координату останнього положення курсору миші.

7.2.3 Малювання олівцем

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

У нас у програмі вже присутній оброблювач події MouseDown. Вам необхідно додати оброблювачі для подій MouseMove і MouseUp. Зробіть це, клацнувши два рази по відповідних до полів на закладці подій у вікні властивостей. Той код, який у вас був присутній в оброблювачі події MouseDown, нам більше не знадобиться. Ви можете вилучити його або закоментувати.

Змініть код оброблювачів подій так, як представлено нижче:

```
private void Form2_MouseDown (object sender, System.Windows.Forms.MouseEventArgs e)
{
    Form1 arentorm = (Form1)MdiParent;
    switch(parentorm.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 прапор 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, Sys-
tem.Windows.Forms.MouseEventArgs e)
    {
        drawPen = false;
    }
    private void Form2_Mousemove (object sender, Sys-
tem.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 в клас Form2 нових полів і методів. У коді програми використовуються неоголошені поки змінні PreviousPoint, drawPen і неоголошені методи DrawLine, DrawEllipse і Text. Останні закоментовані й не заважають поки компіляції програми. А от змінні необхідно оголосити. Давайте оголосимо в програмі змінні PreviousPoint і 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 в 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 в true і не скидається в false доти, поки користувач не відпустить кнопку нагору. Оброблювач Form2_MouseMove малює на екрані лінії тільки тоді, коли змінна drawPen перебуває в true. Малювання відбувається в такий спосіб:

1. Створюється об'єкт Pen, який ініціалізується чорним

кольором (Color.Black) і товщиною лінії, що дорівнює трьом.

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

2. Виходять 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'єкта Graphics. У якості координат лінії передаються попередня й 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 яким рухався курсор миші,

Запустіть програму. Створіть нове вікно. Оберіть режим малювання олівцем. Спробуйте намалювати у вікні будь-яку криву лінію, утримуючи курсор миші у натиснутому стані (рис. 7.1). Якщо у вас вийшло намалювати криву – ви все зробили правильно, якщо немає – уважно перечитайте наведений опис.

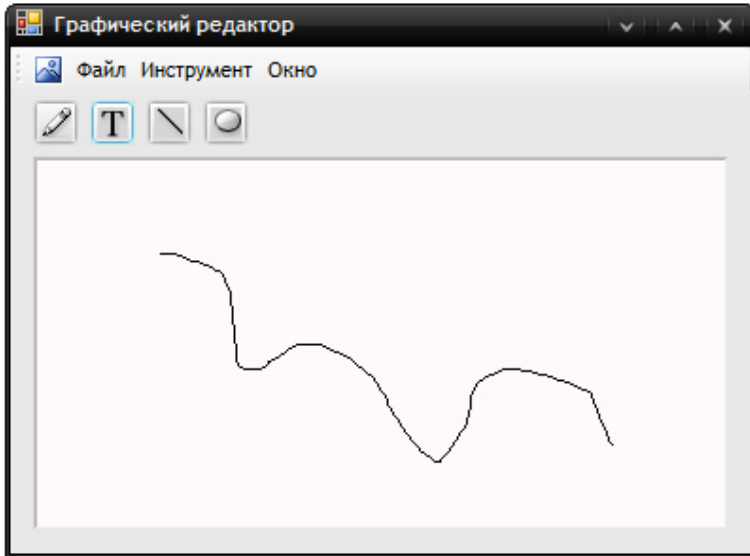


Рисунок 7.1 – Малювання олівцем

7.2.4 Малювання тексту й графічних примітивів

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

Для цього додайте в клас 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);
FirstClick = false;
}
else{
Firstclick = true;
}

void DrawEllipse(Point point) {
// якщо один раз уже клацнули
if (FirstClick== 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);
FirstClick = false;
}
else{
FirstClick = 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("Програмуванн на C#", titlefont, new Solid-
Brush(Color.Red), point.X, point.Y).
}
```

Методи DrawLine і DrawEllipse використовують змінну FirstClick Додайте оголошення цієї змінної в клас Form2.

```
public bool drawPen;  
public bool FirstClick;  
public Point PreviousPoint;;
```

Методи DrawText і DrawEllipse малюють об'єкти по двом координатам крапок на екрані. При першій клацанні миші запам'ятовується перша координата й виставляється значення змінної FirstClick в true. При другій клацанні миші відбувається малювання лінії, і значення FirstClick скидається в 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 метод Drawstring рядок текст «Програмування на C#» виводиться на екран.

Тепер ви можете відкоментувати виклики методів DrawLine, DrawEllipse і DrawText у функції Form2_MouseDown.

Запустіть програму. Обираючи різні режими роботи програми, Ви можете створювати найпростіші графічні зображення (рис. 7.2).

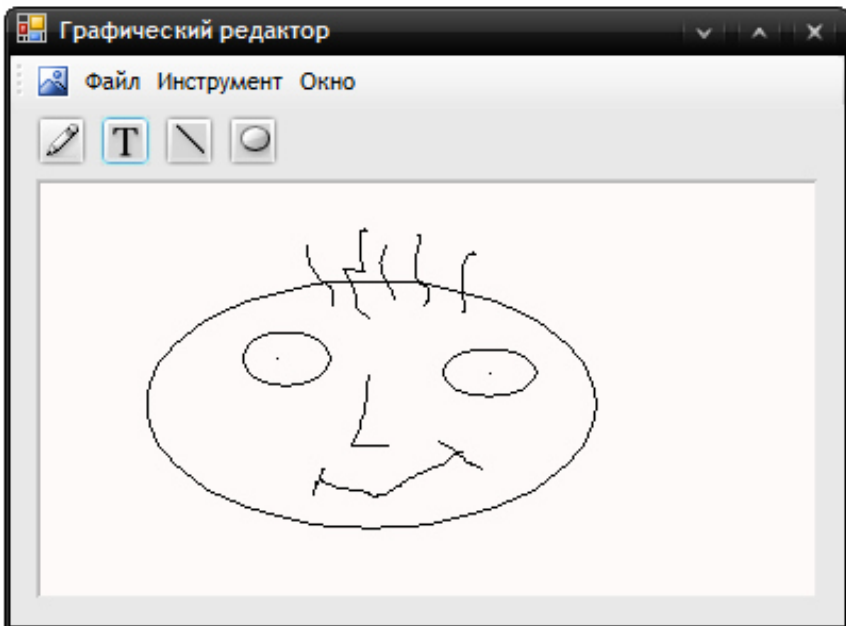


Рисунок 7.2 – Використання програми «Графічний редактор»

7.3 Завдання до роботи

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

7.3.2 Вивчити основні принципи робіт з зображеннями.

7.3.3 Виконати наступні завдання:

1. Реалізувати програму «Графічний редактор». Реалізувати малювання інструментами – лінія, еліпс, олівець, прямокутник та текст. Передбачити завантаження, збереження та створення зображень.
2. Реалізувати програму «Графічний переглядач». Дозволяє переглядати графічні зображення у формі програми, дає змогу обирати папку для переглядання та відповідний перегляд усіх зображень у папці.

7.3.4 Оформити звіт з роботи.

7.3.5 Відповісти на контрольні питання.

7.4 Зміст звіту

7.4.1 Тема та мета роботи.

7.4.2 Завдання до роботи.

7.4.3 Короткі теоретичні відомості.

7.4.4 Текст розробленого програмного забезпечення.

7.4.5 Результати виконання роботи.

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

7.5 Контрольні запитання

7.5.1 Як вивести пунктирну лінію на екран?

7.5.2 Як реалізувати градієнтну зафарбовування?

- 7.5.3 Для чого використовується клас Polygon?
- 7.5.4 Як задати товщину лінії?
- 7.5.5 Як намалювати стрілку?
- 7.5.6 Як задати Antialiasing для виводу тексту?
- 7.5.7 Як збільшити швидкість роботи технології GDI+?
- 7.5.8 Порівняти технологію GDI та GDI+.
- 7.5.9 Для чого використовуються матриці перетворення?
- 7.5.10 Використання alpha-прозорості у GDI+.

8 ЛАБОРАТОРНА РОБОТА № 8 СТВОРЕННЯ ДОВІДКИ

8.1 Мета роботи

Навчитись створювати довідку у C#.

8.2 Основні теоретичні відомості

8.2.1 Створення довідки

Довідка – неодмінний атрибут будь-якої, навіть самої невеликої програми. Добре написана довідка дає користувачам можливість швидко освоїти основи роботи з програмою, практично не вдаючись до допомоги додаткових підручників. Її зміст може бути зовсім маленьким - таким як довідка до блокнота Windows, або величезним – таким як MSDN (бібліотека, яка є, по суті, гігантську довідкову систему).

Файл довідки являє собою сукупність HTML-сторінок, скомпільованих у один додаток формату *. chm або *. hlp. Ієрархічна структура утворюється системою гіперпосилань між сторінками та їх деревовидним розташуванням. У папці C: \ WINDOWS \ Help знаходяться довідкові файли більшості додатків операційної системи Windows.

Як додаток, до якого ми будемо складати довідку, візьмемо Notepad C#. Оскільки програма включає в себе прості і всім відомі операції з текстом, ми не будемо їх описувати, а як матеріал скористаємося їх кодом (рис. 8.1).

8.2.2 Підготовка HTML-сторінок

Вихідні сторінки довідки найкраще створювати в одному з пакетів верстки web-сторінок, наприклад, Microsoft Front Page. Довідка буде складатися з 36 сторінок, пов'язаних між собою. В якості фонових кольорів на всіх сторінках я вибрав світло-блакитний (Тег <body bgcolor="#66CCFF">). Для копіювання коду я звернув всі

методи і обробники в проєкті в блоки, а при вставці в документ вони автоматично розгорталися (рис. 8.2).

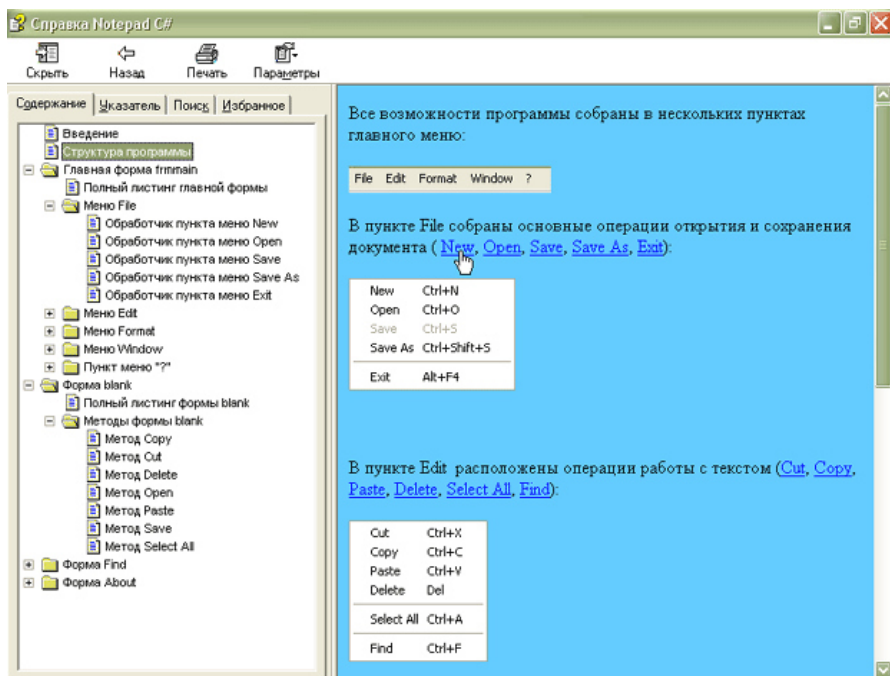


Рисунок 8.1 – Готова довідка програми Notepad C#

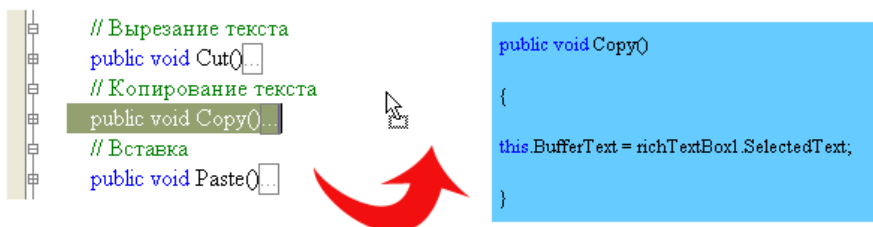
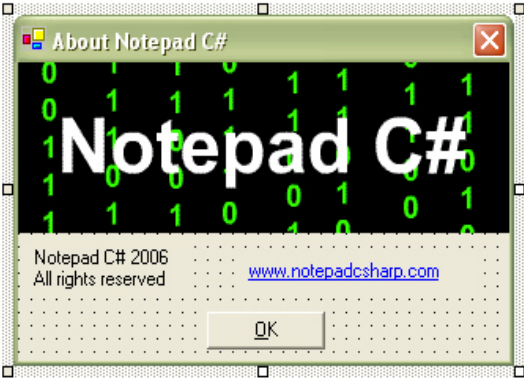


Рисунок 8.2 – Копіювання блоків коду

Таблиця 8.1 містить дані про сторінки довідки. Бажано самим створити HTML-сторінки з усіма посиланнями, зазначеними в цій таблиці

Таблиця 8.1 – Вихідні HTML-сторінки довідки

Назва файлу	Заголовок сторінки	Зміст сторінки	
About Form.htm	Властивості форми About	Властивості форми About:	
		Форма About, властивість	Значення
		Name	About
		FormBorderStyle	FixedSingle
		MaximizeBox	False
		MinimizeBox	False
		Size	318; 214
		Text	About Notepad C#
		Зовнішній вигляд форми в режимі дизайну:	
			
		Головне вікно програми Повний лістинг форми About. (Посилання на FullCodeAbout.htm)	
About Progra	Оброблювач пункту	<pre>private void mnuAbout_Click(object sender, System.EventArgs e) {</pre>	

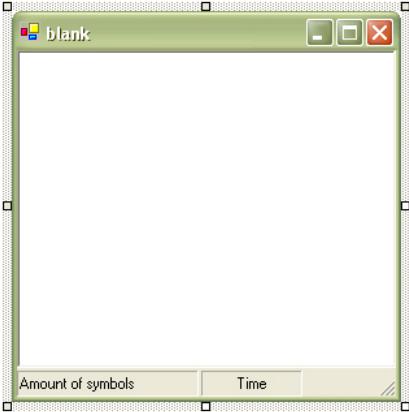
m.htm	меню About Program	<pre>About frm = new About(); frm.Show(); }</pre> При виборі цього пункту меню з'являється форма About
-------	--------------------------	---

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки								
ArrangeIcons.htm	Оброблювач пункту меню ArrangeIcons	<pre>private void mnuArrangecons_Click(object sender, System.EventArgs e) { this.LayoutMdi (MdiLayout.ArrangeIcons); }</pre>								
blank.htm	Властивості форми blank	Властивості форми blank: <table><tr><th>Форма blank, властивість</th><th>Значення</th></tr><tr><td>Name</td><td>blank</td></tr><tr><td>Text</td><td>blank</td></tr><tr><td>WindowState</td><td>Maximized</td></tr></table> Зовнішній вигляд форми в режимі дизайну:  При виборі цього пункту меню з'являється форма	Форма blank, властивість	Значення	Name	blank	Text	blank	WindowState	Maximized
Форма blank, властивість	Значення									
Name	blank									
Text	blank									
WindowState	Maximized									

		About (посилання на AboutForm).
--	--	---------------------------------

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки								
blank.htm	Властивості форми blank	Властивості форми blank: <table><tr><th>Форма blank, властивість</th><th>Значення</th></tr><tr><td>Name</td><td>blank</td></tr><tr><td>Text</td><td>blank</td></tr><tr><td>WindowState</td><td>Maximized</td></tr></table> Зовнішній вигляд форми в режимі дизайну:  Повний лістинг форми blank. (Посилання на FullCodeblank.htm)	Форма blank, властивість	Значення	Name	blank	Text	blank	WindowState	Maximized
Форма blank, властивість	Значення									
Name	blank									
Text	blank									
WindowState	Maximized									
blank Copy.htm	Метод Copy форми blank	<pre>public void Copy() { this.BufferText = richTextBox1.SelectedText; }</pre>								
blank Cut.htm	Метод Cut форми blank	<pre>public void Cut() { this.BufferText = richTextBox1.SelectedText; richTextBox1.SelectedText = ""; }</pre>								
blank Delete.htm	Метод Delete форми	<pre>public void Delete() { richTextBox1.SelectedText = ""; this.BufferText = ""; }</pre>								

	blank	
--	-------	--

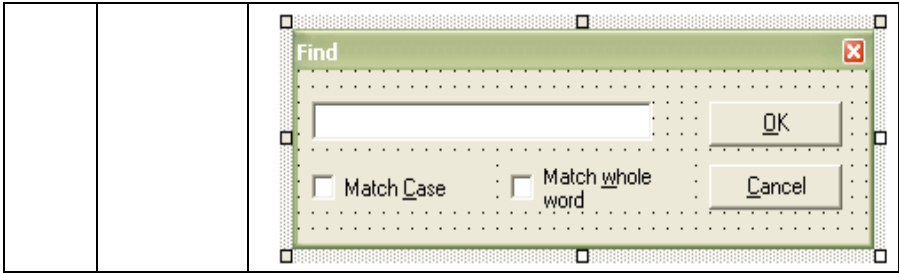
Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки
blank Open.htm	Метод Open форми blank	<pre>public void Open(string OpenFileName) { /// Вставте сюди код з проекту }</pre>
blankPaste.htm	Метод Paste форми blank	<pre>public void Paste() { richTextBox1.SelectedText = this.BufferText; }</pre>
blankSave.htm	Метод Save форми blank	<pre>public void Save(string SaveFileName) { /// Вставте сюди код з проекту }</pre>
blankSelectAll.htm	Метод Select All форми blank	<pre>public void SelectAll() { richTextBox1.SelectAll(); }</pre>
Cascade.htm	Оброблювач пункту меню Cascade	<pre>private void mnuCascade_Click(object sender, System.EventArgs e) { this.LayoutMdi(MdiLayout.Cascade); }</pre>
Color.htm	Оброблювач пункту меню Color	<pre>private void mnuColor_Click(object sender, System.EventArgs e) { /// Вставте сюди код з проекту }</pre>
Copy.htm	Оброблювач пункту меню Copy	<pre>private void mnuCopy_Click(object sender, System.EventArgs e) { blank frm = (blank)this.ActiveMdiChild; frm.Copy(); } В коді викликається метод Copy форми blank</pre>

Cut.htm	Оброблювач пункту меню Cut	<pre>private void mnuCut_Click(object sender, System.EventArgs e) { blank frm = (blank)this.ActiveMdiChild; frm.Cut(); }</pre> В кодї викликається метод Cut форми blank
---------	----------------------------	---

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки																		
Delete.htm	Оброблювач пункту меню Delete	<pre>private void mnuDelete_Click(object sender, System.EventArgs e) { blank frm = (blank)this.ActiveMdiChild; frm.Delete(); }</pre> В коде вызывается метод Delete формы blank																		
Exit.htm	Оброблювач пункту меню Ex	<pre>private void mnuExit_Click(object sender, System.EventArgs e) { this.Close(); }</pre>																		
Find.htm	Оброблювач пункту меню Save As	<pre>private void mnuFind_Click(object sender, System.EventArgs e) { /// Вставьте сюда код з проекту }</pre> При выборе этого пункта меню появляется форма FindForm																		
FindForm.htm	Оброблювач пункту меню Select All	Властивості форми FindForm: <table border="1"> <thead> <tr> <th>Форма</th><th>FindForm,</th><th>Значення</th></tr> </thead> <tbody> <tr> <td>властивість</td><td></td><td></td></tr> <tr> <td>Name</td><td></td><td>FindForm</td></tr> <tr> <td>FormBorderStyle</td><td></td><td>FixedToolWindow</td></tr> <tr> <td>Size</td><td></td><td>328; 112</td></tr> <tr> <td>Text</td><td></td><td>Find</td></tr> </tbody> </table> Зовнішній вигляд форми в режимі дизайну:	Форма	FindForm,	Значення	властивість			Name		FindForm	FormBorderStyle		FixedToolWindow	Size		328; 112	Text		Find
Форма	FindForm,	Значення																		
властивість																				
Name		FindForm																		
FormBorderStyle		FixedToolWindow																		
Size		328; 112																		
Text		Find																		



Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки	
Font.htm	Структура програми	<pre>private void mnuFont_Click(object sender, System.EventArgs e) { /// Вставте сюди код з проекту }</pre>	
frmmain.htm	Оброблювач пункту меню Tile Horizontal	Властивості форми frmmain:	
		Form1,форма, властивість	Значення
		Name	frmmain
		Icon	 Code\Glava2\NotepadCSharp\Icon\README.ICO
		IsMdiContainer	True
		Menu	mainMenu1
		Text	Notepad C#
		WindowState	Maximized
		Зовнішній вигляд форми в режимі дизайну:	
FullCodeAbout.htm	Оброблювач пункту меню Tile	Повний лістинг форми About: /// Вставте сюди код з проекту	

	Vertical	
FullCodeBlank.htm	Оброблювач пункту меню Arrange Icons	Повний лістинг форми blank: /// Вставте сюди код з проекту

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки
FullCodeFindForm.htm	Властивості форми blank	Повний лістинг форми FindForm: /// Вставте сюди код з проекту
FullCodefrmmain.htm	Властивості форми blank	Повний лістинг форми frmmain: /// Вставте сюди код з проекту
index.htm	Метод Copy форми blank	Програма Notepad C# була зроблена в ході вивчення елементів програмування на мові C# для Windows. За своєю функціональністю вона нагадує щось середнє між блокнотом і текстовим редактором WordPad. Деякі фрагменти коду були приведені в чистому вигляді, без блоків обробки винятків. У результаті при виборі невідповідних дій - наприклад, операцій роботи з текстом за відсутності бланка, – можливі помилки. Оскільки вихідний код програми додається, Ви можете додати відповідні блоки. Також, використовуючи властивість Enabled і Visible елементів, можна отримати недоступність або невидимість елементів, робота з якими в певні моменти викликає виключення
New.htm	Метод Cut форми blank	Оброблювач пункту меню New private void mnuNew_Click(object sender, System.EventArgs e) { /// Вставте сюди код з проекту }
Notepadhelp	Метод Delete	Оброблювач пункту меню Notepad C# Help private void mnunotepadhelp_Click(object sender, System.EventArgs e)

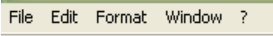
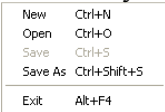
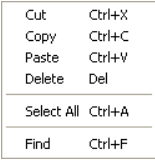
.htm	форми blank	<pre> { try { Process SysInfo = new Process(); SysInfo.StartInfo.ErrorDialog = true; SysInfo.StartInfo.FileName = "notepadhelp.chm"; SysInfo.Start(); } catch (Exception ex) { MessageBox.Show (ex.Message); } } </pre>
------	----------------	---

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки
Open. htm	Метод Open форми blank	Оброблювач пункту меню Open <pre> private void mnuOpen_Click(object sender, System.EventArgs e) { /// Вставте сюди код з проекту } </pre> У коді викликається метод Open форми blank
Paste.h tm	Метод Paste форми blank	Оброблювач пункту меню Paste <pre> private void mnuPaste_Click(object sender, System.EventArgs e) { blank frm = (blank)this.ActiveMdiChild; frm.Paste(); } </pre> У коді викликається метод Paste форми blank
Save.h tm	Метод Save форми blank	Оброблювач пункту меню Save <pre> private void mnuSave_Click(object sender, System.EventArgs e) { // Перемикаємо фокус на дану форму. blank frm = (blank)this.ActiveMdiChild; // Викликаємо метод Save форми blank frm.Save(frm.DocName); frm.IsSaved = true; } </pre> У коді викликається метод Save форми blank
SaveA s.htm	Метод Select All форми	Оброблювач пункту меню Save As <pre> private void mnuSaveAs_Click(object sender, System.EventArgs e) { </pre>

	blank	/// Вставте сюди код з проекту }
Select All.htm	Оброблювач пункту меню Cascade	Оброблювач пункту меню Select All private void mnuSelectAll_Click(object sender, System.EventArgs e) { blank frm = (blank)this.ActiveMdiChild; frm.SelectAll(); } У коді викликається метод Select All форми blank

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки
structure.htm	Оброблювач пункту меню Color	Всі можливості програми зібрані в кількох пунктах головного меню:  У пункті File зібрані основні операції відкриття і збереження документа:  У пункті Edit розташовані операції роботи з текстом:  Пункт Format дозволяє задати властивості шрифту, такі як колір, розмір, стиль:  Для упорядкування декількох відкритих документів використовується пункт меню Window:

		Arrange Icons Cascade Tile Horizontal Tile Vertical В останньому пункті "?" містяться пункти виклику цієї довідки і вікно з інформацією про програму: Notepad C# Help F1 About Program...
--	--	--

Продовження таблиці 8.1

Назва файлу	Заголовок сторінки	Зміст сторінки
TileHorizontal.htm	Оброблювач пункту меню Copy	Оброблювач пункту меню Tile Horizontal: <pre>private void mnuTileHorizontal_Click(object sender, System.EventArgs e) { this.LayoutMdi(MdiLayout.TileHorizontal); }</pre>
TileVertical.htm	Оброблювач пункту меню Cut	Оброблювач пункту меню Tile Vertical <pre>private void mnuTileVertical_Click(object sender, System.EventArgs e) { this.LayoutMdi(MdiLayout.TileVertical); }</pre>

Отже, в результаті у нас вийшли 36 HTML-файлів. На головній сторінці – structure.htm – розміщені посилання на ключові сторінку всієї групи.

На диску, прикладеному до книги, ви знайдете папку HelpPages з готовими файлами (Code \ Glava8 \ HELP \ HelpPages).

Створення проекту в програмі HTML Help Workshop

Приступимо тепер до розробки файлу довідки, що складається з підготовлених сторінок. При установці Microsoft Visual Studio .NET на комп'ютер також встановлюється програма для створення довідок - HTML Help Workshop. Для її запуску переходимо в меню Пуск \ Усі

програми \ папка HTML Help Workshop \ HTML Help Workshop. Подібно Visual Studio. NET, для роботи з цією програмою потрібно створити проект. У меню File вибираємо New \ Project, в який з'явився Майстрі New Project натискаємо кнопку "Далі", вказуємо розташування проекту та його назва (notepadhelp.hhp), за крок New Project -> Existing Files знову натискаємо "Далі" і завершуємо роботу Майстра. У результаті головне вікно програми прийме наступний вигляд (рис. 8.3):

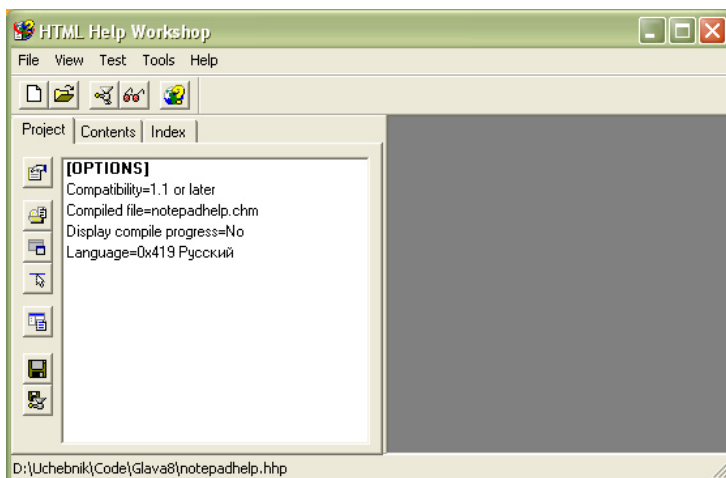


Рисунок 8.3 – Головне вікно HTML Help Workshop з новим проектом notepadhelp.hhp

Додаємо в проект html-сторінки, на бічній панелі інструментів клацаємо на кнопці (Add / Remove topic files ...), у вікні натискаємо на кнопку Add і вибираємо папку, яка містить сторінки (рис. 8.4).

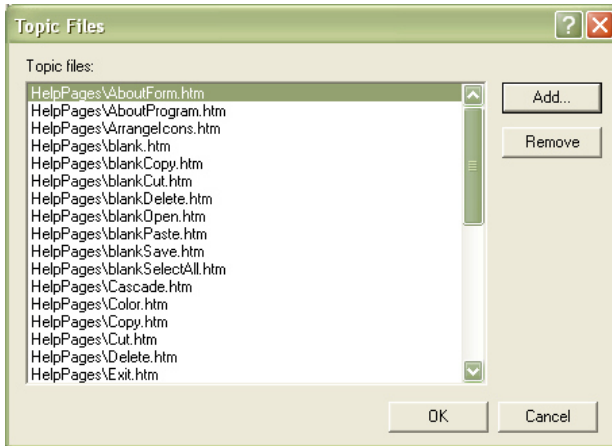


Рисунок 8.4 – Додавання HTML-сторінок в проект

У результаті у вікні Project під заголовком [Files] з'явиться список усіх приєднаних сторінок. Можна переглянути HTML-код кожної сторінки, натиснувши кнопку (View HTML Source). Зверніть увагу на заголовки сторінок: назви повинні бути видні російською мовою, тобто бути в кодуванні Windows – 1251 або 1252 (рис. 8.5). Якщо заголовки представлені в іншому кодуванні, виправте їх – вони знадобляться нам надалі.



Рисунок 8.5 – Кодування для заголовків

8.2.3 Створення змісту

Після того як до проекту були включені вихідні сторінки, можна приступати до створення змісту. Клацаємо на вкладку Contents, при цьому з'являється вікно – "Таблиця змісту не визначена" (Table Contents Not Specified) – при створенні проекту ми пропустили крок Майстри зі створенням файлу змісту. Зробимо це тепер – вибираємо пункт Create new contents file, вказуємо директорію для збереження (бажано зберігати файл змісту там же, де розташований файл проекту notepadhelp.hhp) і називаємо його ContentsNhelp.hhc. Клацаємо на кнопці додавання сторінки (Insert a page), в полі Entry title вводимо назву заголовка, яке буде відображатися у змісті, клацаємо на кнопці Add / Edit ... і вибираємо аналогічний заголовок HTML-сторінки з файлу проекту (рис. 8.6).

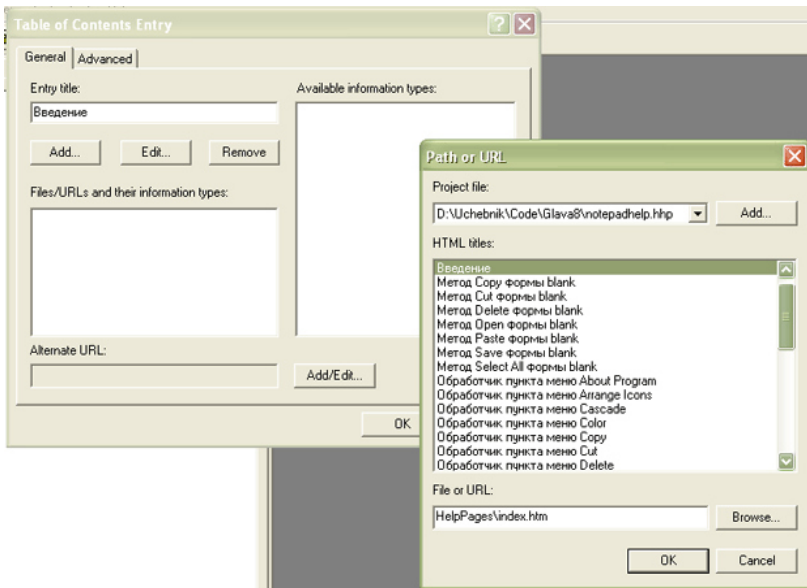


Рисунок 8.6 – Додавання сторінки в зміст довідки

Додаємо таким же чином сторінку "Структура програми". Змінюємо порядок розташування заголовків у змісті, клацаючи на

кнопках зі стрілками  і  на панелі інструментів. На Рисунку 8.1. помічаємо, що заголовок "Головна форма frmmain" представляє собою папку, яка містить заголовок "Повний лістинг головної форми", і папки с назвами пунктів меню, які, у свою чергу, теж містять вкладені заголовки. Для вставки заголовка-папки натискаємо на кнопку  (Insert a heading), з'являється вже знайоме вікно Table of Contents Entry. Вводимо назву заголовка і пов'язуємо з ним сторінку frmmain.htm. Додаємо під ним ще один звичайний заголовок "Повний лістинг головної форми" (пов'язаний з сторінкою FullCodefrmmain.htm), і він автоматично встановлюється з відступом від краю вікна. Розташування заголовка можна змінювати, виділивши його і, натискаючи на кнопки  (Move selection right) і  (Move selection left) (рис. 8.7).

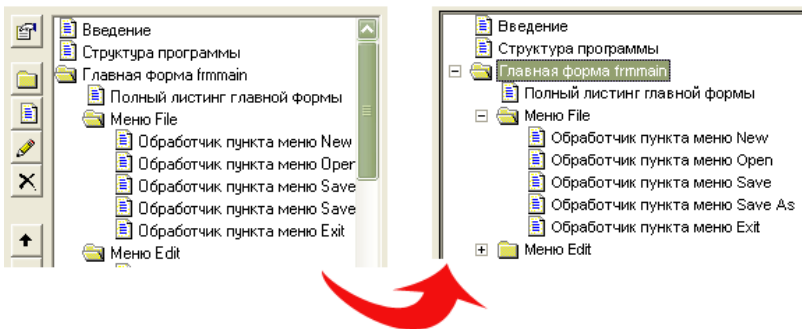


Рисунок 8.7 – Заголовки з відступом (ліворуч) перетворюються у вкладені заголовки в готовій довідці (праворуч)

Заголовки-папки можуть бути пов'язаними з HTML-сторінкою або просто бути контейнером для інших сторінок (рис. 8.8).

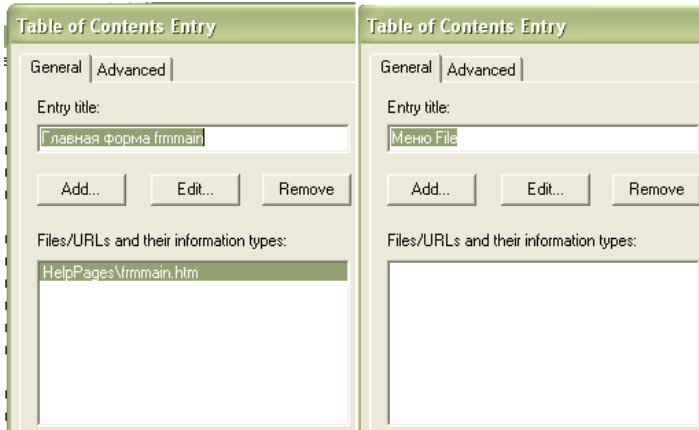


Рисунок 8.8 – Папка-заголовок "Головна форма frmmain" пов'язана з сторінкою frmmain.htm, папка-заголовок "Меню File" представляє собою просто контейнер для інших сторінок

У результаті повинна вийти наступна ієрархічна структура змісту (рис. 8.9):

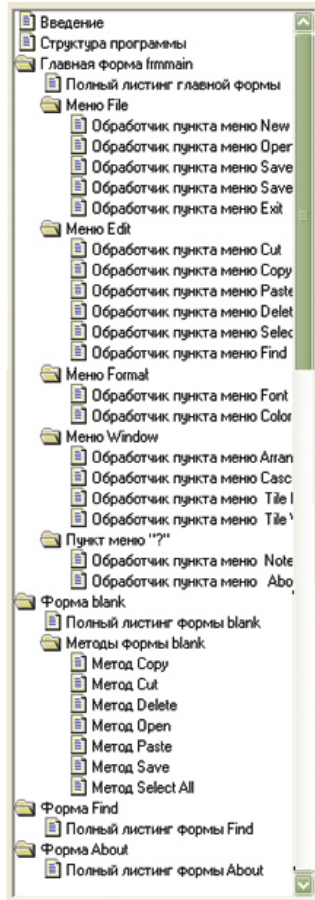


Рисунок 8.9 – Ієрархічна структура змісту

Скомпілюємо готовий файл довідки – натискаємо на кнопку (Compile HTML file) на панелі інструментів Toolbar або вибираємо пункт меню File / Compile З'являється діалогове вікно Create a compiled file, в якому додатково ставимо галочку Save all files before compiling (зберегти всі файли перед компіляцією). Після компіляції виводиться налагоджувальна інформація у вікні HTML Help Workshop, а у вказаній директорії з'являється файл notepadhelp.chm (рис. 8.10).

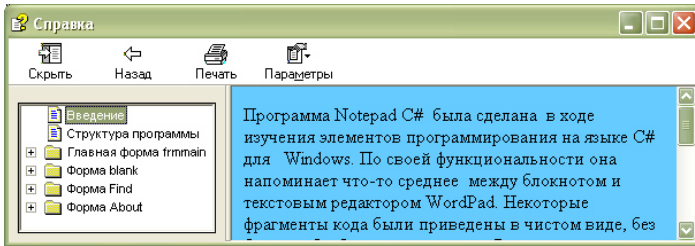


Рисунок 8.10 – Скомпільований файл довідки

8.2.4 Створення покажчика

На рис. 8.11 розташовується також вкладка "Покажчик", при перемиканні на яку з'являється можливість переглядати вміст довідки за ключовими словами (рис. 8.11).

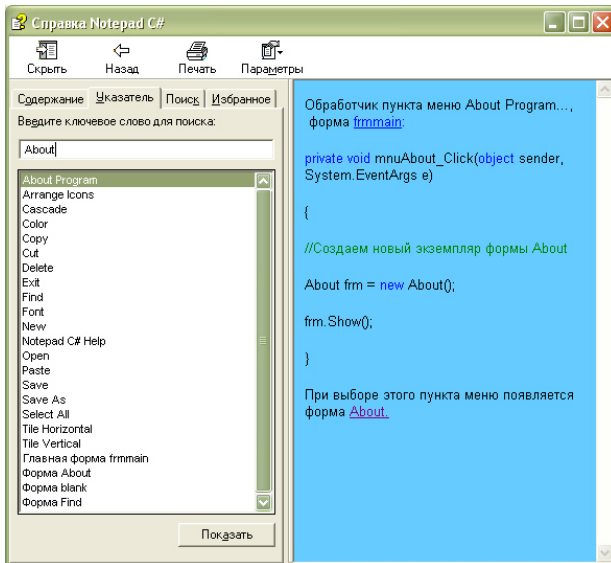


Рисунок 8.11 – Покажчик довідки

Для створення покажчика в проекті notepadhelp.hhp клацаємо на вкладці Index. З'являється вікно "Покажчик не визначено" (Index Not

Specified), аналогічне вікно, з яким ми стикалися при створенні змісту. Вибираємо створення нового файлу-індексу (Create a new index file), називаємо його NotepadIndex.hhk. Щоб додати ключового слова натискаємо на кнопку  (Insert a keyword), у вікні вводимо ключове слово і пов'язуємо з ним HTML-сторінку (рис. 8.12).

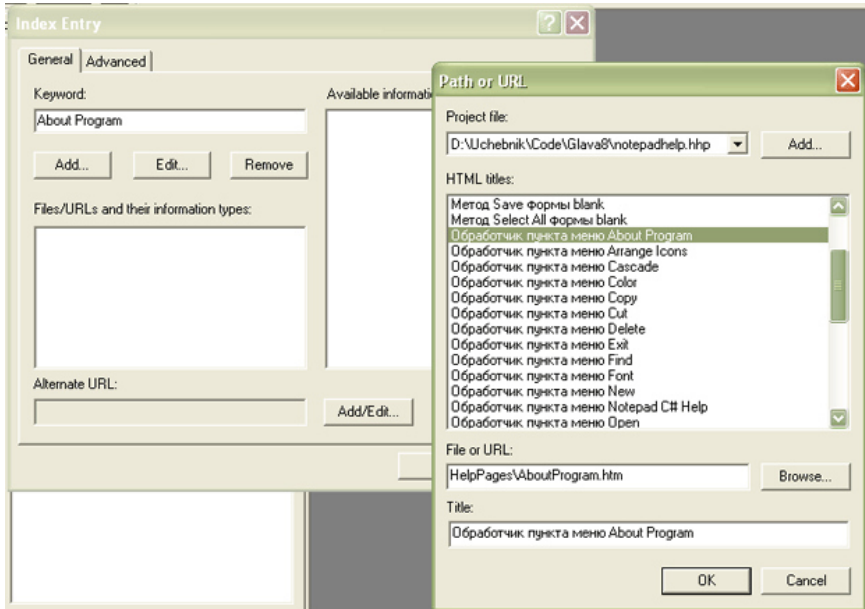


Рисунок 8.12 – Додавання ключового слова і зв'язування з ним HTML-сторінки

Додайте всі ключові слова, вказані на рис. 8.11. Для розташування елементів використовуйте стрілки  і  на панелі інструментів. При створенні списку надається можливість ієрархічного упорядкування слів за допомогою кнопок  і  — для великих списків це необхідно використовувати. Після завершення роботи проведіть сортування слів, натиснувши на кнопку  (Sort keywords alphabetically), і скопіюйте проект.

8.2.5 Створення пошуку

Іноді буває зручним переглянути всі сторінки, що містять ключове слово, – у великих довідках буває досить важко знайти потрібний розділ або заголовок. Для цього використовується вкладка "Пошук", де можна ввести ключове слово, за яким будуть виведені всі сторінки, що містять в тексті це слово хоча б один раз (рис. 8.13).

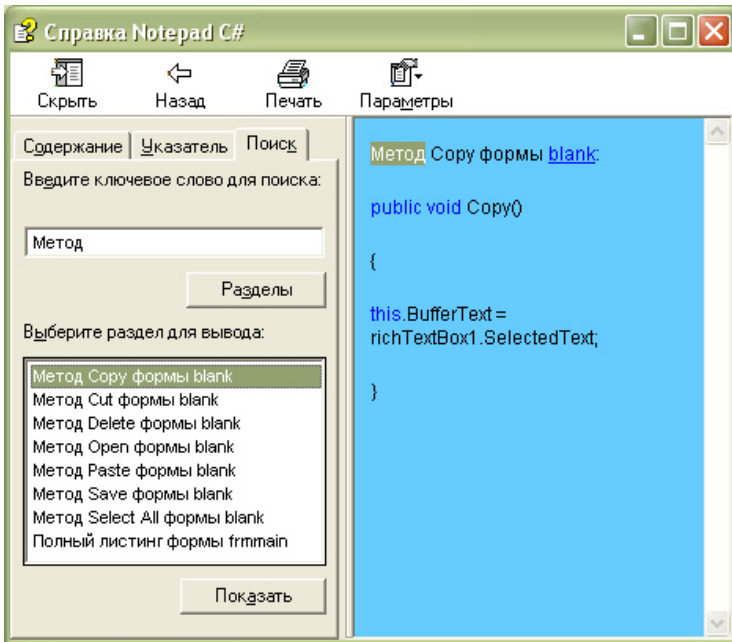


Рисунок 8.13 – Пошук за ключовим словом по всій довідці

У вікні "Виберіть розділ для висновку" відображаються сторінки з їх HTML-заголовків, тому тут дуже важлива правильна кодування.

Для створення пошуку в проєкті notepadhelp.hhp перемикається на вкладку Project і натискаємо на кнопку  (Change project options). У вікні на вкладці Compiler ставимо галочку в чекбоксі Compile full-text search information. Тепер при компіляції буде створюватися база

слів для пошуку. На цій же вкладці Project натискаємо на кнопку  (Add/Modify window definition) І у вікні на вкладці General в полі Window type вказуємо назву вкладки пошуку (рис. 8.14).

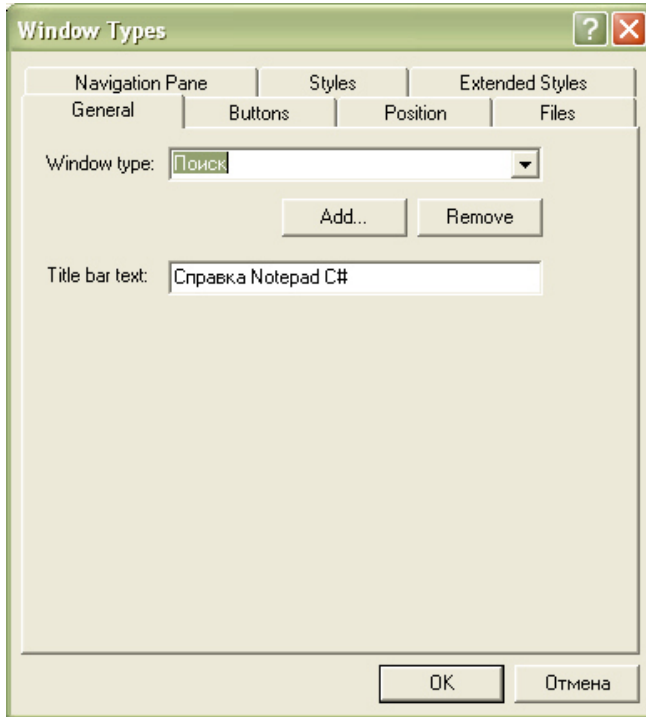


Рисунок 8.14 – Створення вкладки "Пошук"

У полі Title bar text також можна вказати заголовок довідки. Компілюємо проект, відкриваємо готовий файл notepad.chm – тепер з'явилася додаткова вкладка з пошуком.

8.2.6 Створення вкладки "Вибране"

При роботі з великою довідковою системою зручно організовувати власну міні-довідку, додаючи сторінки в "Вибране" (рис. 8.15).

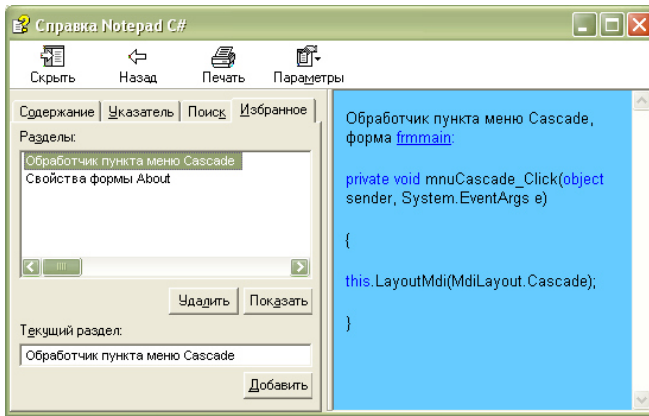


Рисунок 8.15 – Додавання сторінок в "Вибране"

Щоб додати цієї можливості в проєкті notepadhelp.hhp переходимо на вкладку Project і натискаємо на кнопку (Add / Modify window definition). У вікні переходимо на вкладку Navigation Pane, у групі Tabs ставимо галочку в чекбоксі Favorites tab (рис. 8.16).

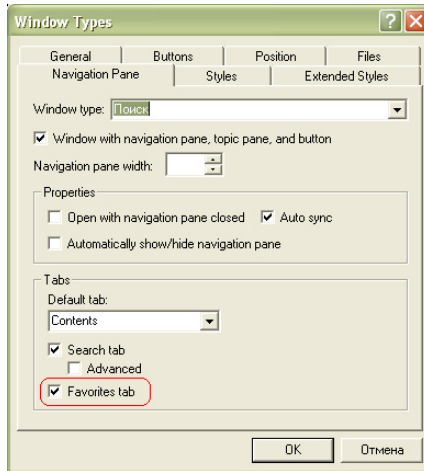


Рисунок 8.16 – Додавання вкладки "Вибране"

Закриваємо вікно властивостей і знову компілюємо проект.

8.2.7 Тестування та налагодження довідки

В кінці роботи ми повинні одержати один скомпільований файл. Chm, що містить в собі всі HTML-сторінки, малюнки на них та посилання. При компіляції проекту слід звертати увагу на вікно налагодження, в якому виводиться результати компіляції. Якщо процес пройшов успішно, виводиться наступна інформація:

```
Microsoft HTML Help Compiler 4.74.8702
```

```
Compiling d:\Uchebnik\Code\Glava8\HELP\notepadhelp.chm
```

```
Compile time: 0 minutes, 2 seconds
```

```
38      Topics
184     Local links
0       Internet links
1       Graphic
```

```
Created d:\Uchebnik\Code\Glava8\HELP\notepadhelp.chm, 77,290 bytes
Compression decreased file by 1,321,411 bytes.
```

Якщо програма стикається з помилкою – наприклад, з відсутністю частини сторінок, – вона продовжить компіляцію і в кінці виведе відповідний звіт:

```
Microsoft HTML Help Compiler 4.74.8702
```

```
Compiling d:\Uchebnik\Code\Glava8\HELP\notepadhelp.chm
```

```
NHC5003: Error: Compilation failed while compiling HelpPag-
es\Paste.htm.
NHC5003: Error: Compilation failed while compiling HelpPag-
es\Save.htm.
NHC5003: Error: Compilation failed while compiling HelpPag-
es\SaveAs.htm.
```

```
The following files were not compiled:
```

```
HelpPages\Paste.htm
HelpPages\Save.htm
HelpPages\SaveAs.htm
```

При компіляції великий довідкової системи бажано зберігати налагодження повідомлення у вигляді файлів (рис. 8.17).

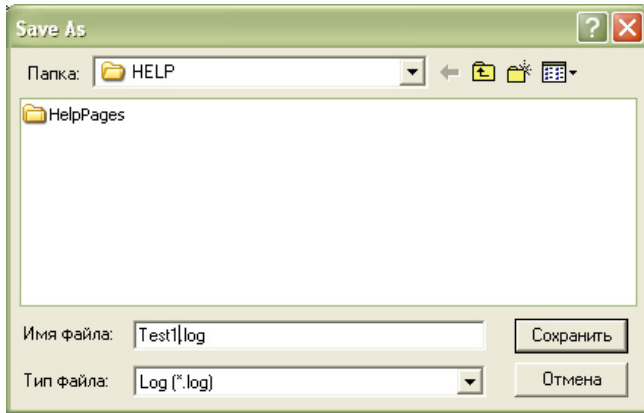


Рисунок 8.17 – Збереження отладочного повідомлення у вигляді файлу Test1.log

Якщо з різних причин вам не вдається скопіювати проект, найкраще відновити його наново – при наявності файлу змісту. Нhc і файлу покажчика. Нhk зробити це досить просто.

На диску, прикладеному до книги, ви знайдете файли notepadhelp.chm, notepadhelp.hhp, ContentsNhelp.hhc, notepadhelp.hhk і Test1.log в каталозі Code \ Glava8 \ HELP.

8.2.8 Додавання готового файлу довідки до додатка

Тепер, коли у нас є готовий файл довідки, нам залишається тільки додати його до додатка Notepad C#. Скопіюємо файл notepadhelp.chm в папку bin / Debug додатки. У головному меню форми frmmain додаємо новий пункт, встановлюючи такі властивості. Приклади наведено у табл. 8.2.

Таблиця 8.2 – Властивості

Властивість	Значення
Name	mnunotepadhelp
Shortcut	F1
Text	Notepad C# Help

У другій лекції ми проходили запуск програм з програми. Тут зовнішньої програмою буде файл `notepadhelp.chm`; переходимо в код форми і підключаємо відповідний простір імен:

```
using System.Diagnostics;
```

Створюємо обробник пункту меню `mnunotepadhelp`:

```
private void mnunotepadhelp_Click(object sender, System.EventArgs e)
{
    try
    {
        Process SysInfo = new Process();
        SysInfo.StartInfo.ErrorDialog = true;
        SysInfo.StartInfo.FileName = "notepadhelp.chm";
        SysInfo.Start();
    }
    catch (Exception ex)
    {
        MessageBox.Show (ex.Message);
    }
}
```

Запускаємо програму, у меню `File` вибираємо `Decompile`, у полі `Destination folder` вказуємо шлях до папки, в яку будуть поміщатися витягнуті HTML-файли, у полі `Compiled help file` – шлях до файлу `chm`.

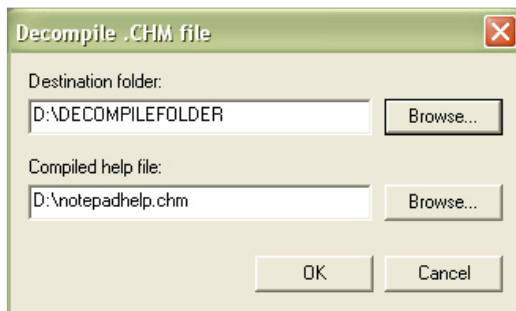


Рисунок 8.18 – Вікно `Decompile`

Після декомпіляції у зазначеній папці з'являться файли змісту `hhs`, покажчика `hhk` і вихідна папка з HTML-сторінками (рис. 8.19).



Рисунок 8.19 – Показчик hhk

8.3 Завдання до роботи

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

8.3.2 Вивчити основні принципи створення довідки.

8.3.3 Реалізувати довідку для програми “Notepad C #”.

8.3.4 Оформити звіт з роботи.

8.3.5 Відповісти на контрольні питання.

8.4 Зміст звіту

8.4.1 Тема та мета роботи.

8.4.2 Завдання до роботи.

8.4.3 Короткі теоретичні відомості.

8.4.4 Текст розробленого програмного забезпечення.

8.4.5 Результати виконання роботи.

8.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

8.5 Контрольні запитання

8.5.1 Що таке Довідка?

8.5.2 Проаналізуйте властивості форми About?

8.5.3 Властивості форми FindForm?

8.5.4 Тег <body bgcolor="...">)?

8.5.5 Як додавати сторінки в зміст довідки?

8.5.6 Властивості форми blank.

8.5.7 Що таке Decompile?

8.5.8 Властивості форми frmmain.

8.5.9 Оброблювач пункту меню Copy.

8.5.10 Оброблювач пункту меню Cut.

8.5.11 Оброблювач пункту меню Delete.

8.5.12 Метод Paste форми blank.

8.5.13 Метод Select All форми blank.

8.5.14 Що таке файл.

8.5.15 Як здійснюється закриття файлового потоку, та важливість закриття потоку?

8.5.16 Які класи для роботи з текстовою інформацією існують у C#?

8.5.17 Як створити покажчик?

8.5.18 Як створити пошук?

8.5.19 Як додавати готовий файл довідки до додатка?

8.5.20 Як програмно створити файл?

ЛІТЕРАТУРА

1. Лабор В.В. Си Шарп Создание приложений для Windows / В. В. Лабор. – Минск : Харвест, 2003. – 385 с.
2. Джонсон Б. Основы Microsoft Visual Studio .Net 2003 / Б. Джонсон. – Москва : Русская редакция, 2003. – 463 с.
3. Бишоп Д. С# в кратком изложении / Д. Бишоп. – Москва : БИНОМ Лаборатория знаний, 2005. – 467 с.