МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

С. М. Цирульник, Ю. С. Хмелівський,
Н. А. Потапова, О. В. Зелінська

ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

Практикум до виконання лабораторних робіт:
навчальний посібник для студентів
галузі знань 12 Інформаційні технології

Вінниця
2025

УДК 004.42(075.8)
Т 384

Затверджено на засіданні Вченої ради
Донецького національного університету імені Василя Стуса
(протокол № 2 від 26 вересня 2025 р.)

Автори:
Цирульник С. М. – канд. техн. наук, доцент кафедри інформаційних технологій ДонНУ імені Василя Стуса;
Хмелівський Ю. С. – асистент кафедри інформаційних технологій ДонНУ імені Василя Стуса;
Потапова Н. А. – канд. екон. наук, доцент кафедри інформаційних технологій ДонНУ імені Василя Стуса;
Зелінська О. В. – канд. техн. наук, доцент кафедри інформаційних технологій ДонНУ імені Василя Стуса.

Рецензенти:
Іванчук Я. В. – д-р техн. наук, професор, професор кафедри комп'ютерних наук Вінницького національного технічного університету;
Федоров Є. Є. – д-р техн. наук, професор, професор кафедри статистики та прикладної математики Черкаського державного технологічного університету.

Т 384 Технології програмування: практикум до виконання лабораторних робіт: навчальний посібник для студентів галузі знань 12 Інформаційні технології / С. М. Цирульник, Ю. С. Хмелівський, Н. А. Потапова, О. В. Зелінська. Вінниця: ДонНУ імені Василя Стуса, 2025. 172 с.

У навчальному посібнику подано завдання та методику виконання лабораторних робіт з курсу «Технології програмування». Наведено теоретичні відомості з досліджуваної тематики, вказані індивідуальні завдання для виконання робіт та контрольні питання для оцінювання знань.

Для здобувачів усіх форм навчання факультету інформаційних та прикладних технологій галузі знань 12 Інформаційні технології.

УДК 004.42(075.8)

ЗМІСТ

ВСТУП

Створення сучасного програмного забезпечення є трудомістким процесом, який залежить від знань та вмінь розробника в галузі програмування, алгоритмізації, методів та підходів розробки програмних продуктів. Вивчення патернів програмування та їх реалізація в конкретних прикладних задачах є базовим підґрунтям для розробки рішень у проєктуванні складних інформаційних систем. Складність такого процесу обумовлена різними класами завдань, взаємодією розробників, знаннями технології, що дає змогу зменшити час на розробку та отримати ефективні архітектурні рішення. Тому необхідним є отримання практичних навичок та використання технологічних принципів і підходів з розробки програмних продуктів, які зможуть повною мірою впливати на їх якість та споживчу цінність.

Навчальний посібник орієнтований на формування у студентів практичних навичок та систематизації теоретичних знань з курсу «Технології програмування». В посібнику представлено 12 лабораторних робіт, виконання яких орієнтоване на засвоєння таких складників: створення консольної програми в середовищі Microsoft Visual Studio на мові програмування C#, створення діалогових програм за допомогою Windows Forms мовою C#; перетворення типів даних для введення даних користувачем, їх обробка та вивід результатів за допомогою Windows Forms; робота з елементом CheckedListBox, NumericUpDown для створення генератора паролів та ін. Розкриття суті методики виконання лабораторних робіт супроводжується викладенням теоретичних основ з вказаної тематики. Подано індивідуальні завдання та контрольні питання.

Цей посібник призначений для здобувачів усіх форм навчання галузі знань 12 Інформаційні технології.

ЛАБОРАТОРНА РОБОТА № 1
СТВОРЕННЯ КОНСОЛЬНОЇ ПРОГРАМИ З ВИКОРИСТАННЯМ МЕТОДІВ ВВЕДЕННЯ-ВИВЕДЕННЯ

Мета: ознайомитися зі створенням консольної програми за допомогою середовища розробки Microsoft Visual Studio на мові програмування C#; навчитись використовувати операції вводу-виводу, застосовувати математичні, логічні та операції порівняння.

1. Теоретичні відомості

1.1. Базові елементи програмування в C# та .NET

Структура рішення .NET[1]

Рішення містить один або кілька проєктів, ресурси, необхідні цим проєктам, можливо, додаткові файли, які не входять у проєкти. Один із проєктів рішення повинен бути виділений і призначений стартовим проєктом. Виконання рішення починається зі стартового проєкту.

Стартовий проєкт повинен мати точку входу – клас, що містить статичну процедуру з ім'ям Main, якій автоматично передається керування в момент запуску рішення на виконання.

Проєкт формується з класів, які можуть бути розміщені в одному або кількох просторах імен. Використання просторів імен дає змогу впорядкувати великі проєкти, що містять значну кількість класів, об'єднуючи споріднені елементи в спільні групи. У випадку командної роботи над проєктом зазвичай кожен учасник створює власний простір імен. Окрім впорядкування, це дає змогу надавати класам однакові назви без потреби турбуватися про їх унікальність. У різних просторах імен можуть існувати однойменні класи. Проєкт – це основна одиниця, із якою працює програміст. Він вибирає тип проєкту, а Visual Studio (або інше середовище розробки) створює скелет проєкту відповідно до обраного типу.

Методи[2]

Метод – це фрагмент коду, що об'єднує послідовність інструкцій. Виконання цих інструкцій розпочинається тоді, коли програма викликає метод і передає йому необхідні аргументи. У мові C# усі команди виконуються саме в межах методів. Головним методом кожної програми є Main, який виступає точкою входу. Під час запуску застосунку його виклик здійснюється загальномовним середовищем виконання (CLR).

[1] https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/inheritance
[2] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods

Методи оголошуються в класі або структурі, з визначенням рівня доступу (public, private тощо), необов'язкових модифікаторів, типу значення, що повертається, назви методу та будь-яких параметрів методу. Ці частини разом є заголовком методу.

Параметри методу записують у дужках і розділяють комами. Порожні дужки вказують, що параметри методу не потрібні.

Властивості[3]

Властивість є членом, який надає гнучкий механізм для читання, запису або обчислення значення приватного поля. Властивості можна використовувати так, ніби вони є публічними членами даних, але вони насправді є спеціальними методами, які називаються «accessors». Це дає змогу легко отримувати доступ до даних і допомагає забезпечити безпеку та гнучкість методів.

Властивості дають змогу класу визначати загальнодоступний спосіб отримання та встановлення значень під час приховування коду реалізації або перевірки.

Статичні класи і члени статичних класів[4]

Статичний клас загалом такий же, як і нестатичний клас, але є одна відмінність: не можна створювати екземпляри статичного класу.

Статичний клас може використовуватися як звичайний контейнер для наборів методів, які працюють на вхідних параметрах.

Нестатичний клас може містити статичні методи, поля, властивості або події. Статичний член викликається для класу навіть у тому випадку, якщо не створено екземпляр класу. Доступ до статичного члена завжди виконується по імені класу, а не примірника. Статичні методи і властивості не можуть звертатися до нестатичних полів в типі, що їх містить, і вони не можуть звертатися до змінної примірника об'єкта, якщо він не передається явно в параметрі методу.

Клас Console[5]

Консоль – це вікно операційної системи, в якому користувачі взаємодіють з операційною системою або текстовим консольним додатком, здійснюючи введення тексту за допомогою клавіатури комп'ютера і зчитуючи текстові дані з термінала комп'ютера. Клас Console надає базову підтримку для додатків, що зчитують символи з консолі та записують символи у консоль.

[3] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/objects

[4] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members

[5] https://docs.microsoft.com/en-us /dotnet/api/system.console?view=netframework-4.8

Зокрема, у класі Console визначені такі властивості:

BackgroundColor – повертає або встановлює колір фону тексту, що виводиться (тип ConsoleColor).

ForegroundColor – повертає або встановлює колір тексту, що виводиться (тип ConsoleColor).

InputEncoding – повертає або встановлює значення кодування тексту, що вводиться.

OutputEncoding – повертає або встановлює значення кодування тексту, що виводиться.

Title – повертає або встановлює текст заголовка вікна консолі.

Деякі статичні методи класу Console, специфічні для консольного введення-виведення:

Clear – очищає буфер консолі і вікно консолі від тексту.

Read – читає наступний символ зі стандартного потоку введення.

ReadKey – отримує інформацію про натиснуту клавішу (об'єкт класу ConsoleKeyInfo).

ReadLine – повертає наступний рядок тексту зі стандартного потоку введення.

Write – здійснює виведення інформації в стандартний потік виведення.

WriteLine – здійснює виведення інформації в стандартний потік виведення, доповнюючи рядок символом «\n» (переводить текст на наступний рядок).

Клас Encoding[6]

Клас Encoding представляє кодування символів.

Кодування – це процес перетворення набору символів Юнікоду в послідовність байтів. На відміну від декодування, це процес перетворення послідовності закодованих байтів у набір символів Юнікоду.

Клас Encoding загалом призначений для перетворення між різними кодуваннями і Unicode.

Клас Convert[7]

Статичні методи класу Convert використовуються в основному для підтримки перетворення в базові типи даних у .NET Framework і з них. Підтримуються такі базові типи: Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime, String. До того ж клас Convert включає методи для підтримки інших типів перетворень.

Клас Convert містить статичні методи, які можна викликати для перетворення цілочисельних значень у недесяткові рядкові представлення, а також

[6] https://docs.microsoft.com/en-us/dotnet/api/system.text.encoding?view=netframework-4.8

[7] https://docs.microsoft.com/en-us/dotnet/api/system.convert?view=netframework-4.8

для перетворення рядків, що представляють недесяткові числа, в цілочисельні значення. Кожен із цих методів перетворення включає аргумент base, який дає змогу вказати систему числення: двійкову (основа 2), вісімкову (основа 8), шістнадцяткову (основа 16), а також десяткову (основа 10). Існує набір методів для перетворення кожного з CLS-сумісних цілочисельних типів у рядок, а другий – для перетворення рядка в кожен із примітивних цілочисельних типів.

Клас Math[8]

Клас Math надає константи і статичні методи для логарифмічних, тригонометричних та інших спільних математичних функцій.

1.2. Створення нового проєкту в MS VS

Створення нового проєкту можливо декількома способами. По-перше, на стартовій сторінці можна вибрати (активувати мишею) пункт «Створити проєкт». Другий варіант – застосувати можливості головного меню: «Файл → Створити → Проєкт» (або поєднання клавіш: «Ctrl + Shift + N»). В обох випадках відкривається вікно «Створити проєкт» (рис. 1.1).

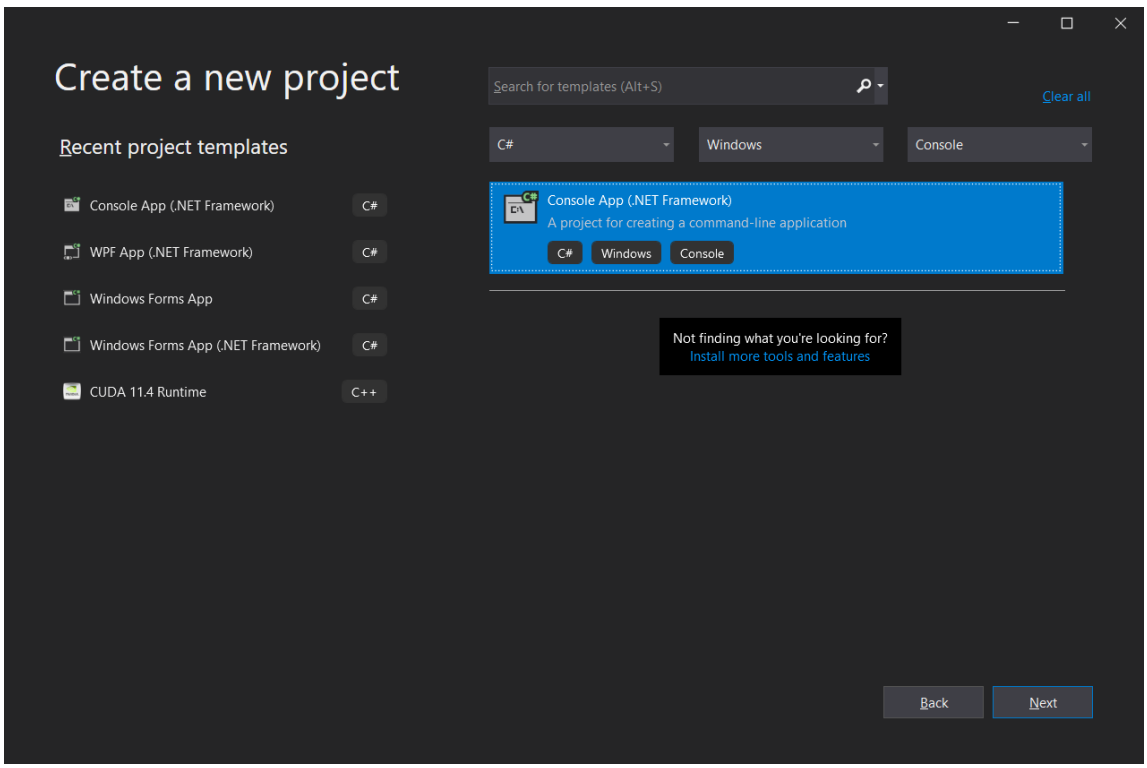


Рис. 1.1. Вікно вибору типу проєкту

У цьому вікні на лівій панелі встановлені шаблони, виберіть мову Visual C#. На центральній панелі виберіть вид програми Консольний додаток.

[8] https://docs.microsoft.com/ en-us/dotnet/api/system.math?view=netframework-4.8

Для введення з клавіатури використовують методи **ReadLine(), Read(), ReadKey().** Це статичні методи класу **Console.** Перший з них зчитує рядок, набраний на клавіатурі, другий – код окремого символа, третій – код натиснутої клавіші. Для виведення даних використовують метод **Console. WriteLine().**

1.3. Створення проєкту для виконання арифметичних операцій

1. Роботу над реалізацією програми почніть зі створення проєктного рішення з проєктом консольного застосування, назву яких утворіть за форматом:

назва проєкту:

TASK1*_Номер_варіанта_Прізвище_виконавця*

назва рішення:

Expressions*_Номер_варіанта_Прізвище_виконавця*

2. Протестуйте роботу програми.

3. З метою затримки відображення вікна консолі до моменту натискання користувачем довільної клавіші у методі Main класу Program проєкту консольного застосування опишіть оператора виклику статичного методу ReadKey класу Console з простору імен System із логічним значенням true у якості аргументу, наприклад (тут і далі курсивом позначені раніше створені рядки коду):

```
        static void Main(string[] args) {
            Console.ReadKey(true);
        }
```

4. Текст умови індивідуального варіанта задачі на виконання арифметичної операції з дійсними числами використайте як коментар до методу Main класу Program проєкту, наприклад:

```
    class Program {
     //Дано значення температури в градусах Цельсія Tc.
     //Визначити значення температури в градусах Фаренгейта TF
    //Tc = (TF – 32 )*5/9
        static void Main(string[] args) {
```

5. У тілі методу Main перед оператором виклику методу ReadKey класу Console опишіть:

а) оператор присвоєння статичній властивості Title класу Console значення рядка з назвою проєкту, наприклад:

```
            Console.Title = "TASK1 _26_Tsyrulnyk";
```

б) оператор присвоєння статичній властивості BackgroundColor класу Console значення White перерахування ConsoleColor, наприклад:

```
            Console.BackgroundColor = ConsoleColor.White;
```

в) оператор виклику статичного методу Clear класу Console, наприклад:

```
            Console.Clear();
```

г) оператор присвоєння статичній властивості ForegroundColor класу Console значення Black перерахування ConsoleColor, наприклад:

```
Console.ForegroundColor = ConsoleColor.Black;
```

д) оператор виклику статичного методу WriteLine класу Console з метою відображення рядка тексту, що вказує на призначення програми, наприклад:

```
Console.WriteLine ("Обчислення температури " + "у градусах Фаренгейта").
```

Збережіть зміни програмного коду проєкту. Протестуйте роботу програми.

6. Якщо текст у вікні консолі відображується неправильно, то:

а) додайте у метод Main перед оператором відображення рядка з умовою задачі оператор присвоєння статичній властивості OutputEncoding класу Console значення властивості Unicode класу Encoding з простору імен System.Text, наприклад:

```
Console.OutputEncoding = Encoding.Unicode;
Console.InputEncoding = Encoding.Unicode;
```

б) відкрийте вікно властивостей вікна консолі, виберіть закладку «Шрифт», вкажіть на використання шрифту Consolas і натисніть кнопку «ОК»;

в) закрийте вікно консолі і повторно запустіть програму на виконання.

7. Проаналізуйте умову індивідуального варіанта задачі на виконання арифметичної операції, сформуйте тестовий набір значень, що буде використовуватись у якості вхідних даних програми, і вручну виконайте обчислення для отримання очікуваного результату роботи програми.

8. У тілі методу Main перед оператором виклику методу ReadKey класу Console опишіть:

а) оператори оголошення змінних типу double, що призначені для збереження значень вхідних даних програми, та їх ініціалізації значеннями з тестового набору, наприклад:

```
double  tf, tc = 10;
```

б) оператор відображення значень змінних, що зберігають вхідні дані, наприклад:

```
Console.Write("\tПочаткова температура: ");
string str = Console.ReadLine();
tc = Convert.ToDouble(str);
```

в) оператор обчислення значення виразу згідно з умовою задачі та збереження результату у змінній типу double, наприклад:

```
tf = tc * 1.8 + 32;
```

г) оператор відображення результату рішення задачі, наприклад:

```
Console.WriteLine("Температура у градусах Фаренгейта: " + tf);
```

9. Збережіть зміни програмного коду проєкту. Протестуйте роботу програми (рис. 1.2).

```
Обчислення температури у градусах Фаренгейта
        Початкова температура: 10
Температура у градусах Фаренгейта: 50
```

Рис. 1.2. Зразок результатів виконання поточного завдання

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

namespace TASK1_26_Tsyrulnyk
{
    class Program
     //Дано значення температури в градусах Цельсія Tc.
     //Визначити значення температури в градусах Фаренгейта TF
     //Tc = (TF – 32)*5/9
    {
        static void Main(string[] args)
        {
            Console.Title = "Expressions_26_Tsyrulnyk";
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.OutputEncoding = Encoding.Unicode;
            Console.WriteLine("Обчислення температури " + "у градусах
Фаренгейта");
            double  tf, tc = 10;
            Console.Write("\tПочаткова температура: ");
            string str = Console.ReadLine();
            tc = Convert.ToDouble(str);
            tf = tc * 1.8 + 32;
            Console.WriteLine("Температура у градусах Фаренгейта: "+ tf);
            Console.ReadKey();
        }
    }
}
```

1.4. Приклади програм з використанням методів класу System

Приклад 1

Користувач вводить з клавіатури обсяг пам’яті в мегабайтах. Перевести його у байти і кілобайти.

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

```
using System.Threading.Tasks;
namespace TASK2_26_Tsyrulnyk {
    class Program {
        static void Main(string[] args) {
            Console.Title = "Expressions_26_Tsyrulnyk";
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.OutputEncoding = Encoding.Unicode;

           long k, b;
           Console.WriteLine("Введіть обсяг пам’яті (в Мб)");
           long m = Convert.ToInt32(Console.ReadLine());
           k = m << 10;
           b = k << 10;
           Console.WriteLine("Обсяг пам’яті в кб: " + k);
           Console.WriteLine("Обсяг пам’яті в байтах :" + b);
           Console.ReadKey();
        }
    }
}
```

```
Введіть обсяг пам'яті (в Мб)
10
Обсяг пам'яті в кб: 10240
Обсяг пам'яті в байтах :10485760
```

Рис. 1.3. Результат виконання завдання (приклад 1)

Приклад 2

Нехай A, B і C – довжини сторін певного трикутника. Потрібно перевірити твердження: «Трикутник із такими сторонами є рівностороннім».

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace TASK3_26_Tsyrulnyk
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = " Expressions_26_Tsyrulnyk ";
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.OutputEncoding = Encoding.Unicode;
            Console.WriteLine("Перевірка факту існування "
                + "трикутника за його сторонами");
```

```
            double a, b, c;
            Console.Write("\tВведіть значення a = ");
            a = Convert.ToDouble(Console.ReadLine());
            Console.Write("\tВведіть значення b = ");
            b = Convert.ToDouble(Console.ReadLine());
            Console.Write("\tВведіть значення c = ");
            c = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("\n\ta = {0}, b = {1}, c = {2} ", a, b, c);
            bool truth = (a + b) > c && (a + c) > b && (b + c) > a;
            Console.WriteLine("\nРезультат: {0}", truth);
            Console.ReadKey();
        }
    }
}
```

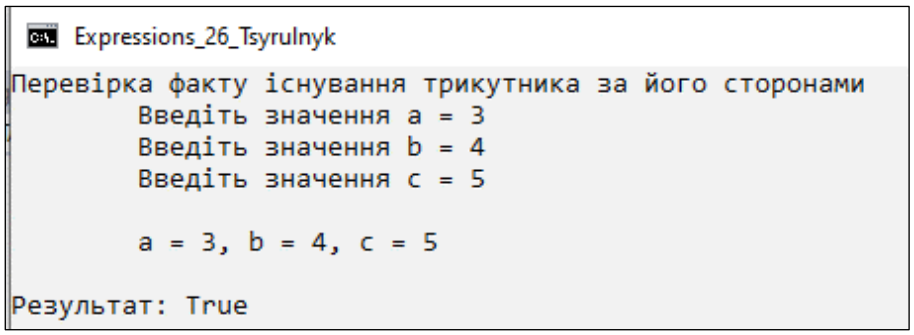


Рис. 1.4. Результат виконання завдання (приклад 2)

Приклад 3

Вивести у консольне вікно поточну дату та час.

```
using System;
using System.Text;
namespace TASK_time
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = " Expressions_26_Tsyrulnyk ";
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.OutputEncoding = Encoding.Unicode;
            DateTime dat = DateTime.Now;
            Console.WriteLine("\nToday is {0:d} at {0:T}.", dat);
            Console.ReadKey();

        }
    }
}
```

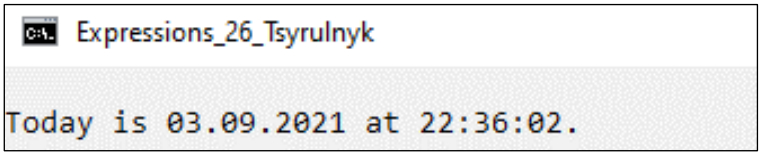


Рис. 1.5. Результат виконання завдання (приклад 3)

Приклад 4

Знайти найбільше з п’яти чисел.

```
using System;
using System.Text;

namespace TASK_MAX
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.OutputEncoding = Encoding.Unicode;
            int a1 = -8, a2 = 13, a3 = 64, a4 = 87, a5 = -61;
            int b1, b2, b3, b4;
            b1 = Math.Max(a1, a2);
            b2 = Math.Max(a3, a4);
            b3 = Math.Max(b1, b2);
            b4 = Math.Max(a5, b3);
            Console.WriteLine("Найбільше з п’яти чисел: " + b4);
            Console.ReadKey();
        }
    }
}
```

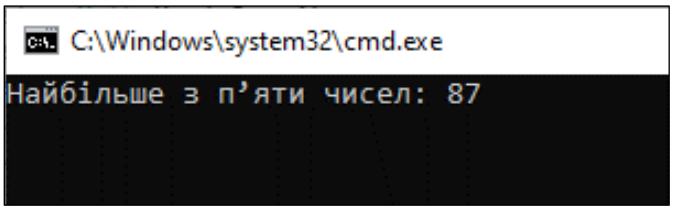


Рис. 1.6. Результат виконання завдання (приклад 4)

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Проаналізувати умову індивідуального варіанта завдання 1 на виконання арифметичних операцій з цілими числами, сформувати тестовий набір значень, що буде використовуватись як вхідні дані програми, і вручну виконати обчислення для отримання очікуваного результату роботи програми. Скласти проєкт програми мовою C#. Оформити звіт з аналізом створеного коду та представленням результатів виконання програми.

Варіанти завдання 1

1. Дано відстань *L* у метрах. Знайти кількість повних кілометрів у ній і залишок у метрах.

2. Дана маса *M* у грамах. Знайти кількість повних кілограмів у ній і залишок у грамах.

3. Дана маса *M* в кілограмах. Знайти кількість повних тонн у ній і залишок у кілограмах.

4. Дано розмір файла у байтах. Знайти кількість повних мегабайтів і гігабайтів, які займає цей файл.

5. Є два додатні цілі числа A і B (A > B). На відрізку довжиною A розміщується найбільша можлива кількість відрізків довжиною B (без накладання). Потрібно знайти кількість таких відрізків N та довжину залишку.

6. Задано двозначне число. Вивести окремо десятки та одиниці.

7. Задано двозначне число. Обчислити суму та добуток його цифр.

8. Задано двозначне число. Сформувати число, отримане перестановкою його цифр.

9. Задано тризначне число. Вивести першу цифру (сотні).

10. Задано тризначне число. Вивести останню цифру (одиниці), а потім середню (десятки).

11. Задано тризначне число. Обчислити суму та добуток його цифр.

12. Задано тризначне число. Вивести число, утворене читанням цифр справа наліво.

13. Задано тризначне число. Якщо закреслити першу цифру і приписати її справа, отримати нове число. Вивести його.

14. Задано тризначне число. Якщо закреслити останню цифру і приписати її зліва, отримати нове число. Вивести його.

15. Задано тризначне число. Поміняти місцями цифри сотень і десятків (наприклад, 123 → 213).

16. Задано тризначне число. Поміняти місцями цифри десятків і одиниць (наприклад, 123 → 132).

17. Дано ціле число > 999. Знайти цифру в розряді сотень.

18. Дано ціле число > 999. Знайти цифру в розряді тисяч.

19. З початку доби минуло N секунд. Знайти кількість повних хвилин.

20. З початку доби минуло N секунд. Знайти кількість повних годин.

21. З початку доби минуло N секунд. Знайти кількість секунд з початку останньої хвилини.

22. З початку доби минуло N секунд. Знайти кількість секунд з початку останньої години.

23. З початку доби минуло N секунд. Знайти кількість повних хвилин з початку останньої години.

24. Користувач вводить час у секундах, що минув від початку робочого дня. Визначити, скільки повних годин залишилось до завершення 8-годинного робочого дня.

25. Користувач вводить із клавіатури обсяг пам'яті в байтах. Перевести його у кілобайти і терабайти.

26. Користувач вводить із клавіатури обсяг пам'яті в мегабайтах. Перевести його у байти і гігабайти.

27. Користувач вводить із клавіатури обсяг пам'яті в кілобайтах. Перевести його в байти і терабайти.

28. Користувач вводить із клавіатури обсяг пам'яті в гігабайтах. Перевести його в байти і біти.

29. Користувач вводить із клавіатури розмір одного фільму в гігабайтах і швидкість інтернет-з'єднання в бітах за секунду. Порахувати, за скільки годин, хвилин і секунд скачається фільм.

30. Користувач вводить із клавіатури кількість студентів, які склали іспит, і кількість боржників. Порахувати, скільки у групі відсотків двієчників, і скільки відсотків студентів, які склали іспит.

31. Користувач вводить із клавіатури час початку і час завершення телефонної розмови (години, хвилини і секунди). Порахувати вартість розмови, якщо вартість хвилини – 30 копійок.

32. Є прямокутник розміру A × B. Розмістити на ньому найбільшу кількість квадратів зі стороною C (без накладання). Знайти кількість квадратів і площу залишку.

33. Користувач вводить час у секундах від початку доби. Вивести поточний час у годинах, хвилинах і секундах. Також: дано число K (1–365). Визначити, який це день тижня, якщо відомо, що 1 січня було вівторком.

34. Користувач вводить обсяг флешки (у ГБ). Визначити, скільки на неї можна записати фільмів певного розміру.

Завдання 2. Розробити програмний проєкт мовою **C#** для обчислення функції $y = f(x)$ згідно з варіантом, наведеним у таблиці 1. Перше із заданих

у таблиці значень необхідно оголосити як константу, а друге – ввести з клавіатури. Після реалізації програми потрібно підготувати звіт з аналізом написаного коду та демонстрацією отриманих результатів.

Таблиця 1.1 – Варіанти функцій для виконання завдання 2

№ вар.	Функція Y = F(x)	Значення
1	$y = a\sin^2 b + b\cos^2 a; a = \sqrt[3]{\lvert b + c\rvert}; b = \sqrt{x}$	$x = 1.52;\ c = 5$
2	$y = a^2 + b^2; a = \ln\lvert x\rvert; b = e^k + a$	$x = 5.3;\ k = 3$
3	$y = e^x + 5.8^c; c = a^2 + \sqrt{b}; a = b^3 + \ln\lvert b\rvert$	$x = 2.5;\ b = 7$
4	$y = \sqrt[3]{\lvert a - b\rvert}; a = \lg x; b = \sqrt{(x^2 + t^2)}$	$x = 1.7;\ t = 3$
5	$y = a^3/b^{\cdot 2}; a = e^{\sqrt{\lvert x\rvert}}; b = (\sin(p^2 + x^3))$	$x = 2.1;\ p = 2$
6	$y = p^2 + t^4; p = x^2 - \sqrt{\lvert x\rvert}; t = \sqrt[3]{x + a^2}$	$x = 4;\ a = 3.7$
7	$y = c^3/\cos c; c = a^2 + b^2; a = \sqrt{\lvert x\rvert} + e^{\sqrt{b}}$	$x = -11;\ b = 12.5$
8	$y = \sin^3(a + b); a = t^3 + \sqrt{b}; b = \lg^2\lvert x\rvert$	$x = 10.9;\ t = 2$
9	$y = \mathrm{arctg}^3 x^2; x = p + k; k = \sqrt{p + t^2}$	$t = 4.1;\ p = 3$
10	$y = \cos^2(a + \sin b); a = \sqrt{\lvert x\rvert}; b = x^4 + m^2$	$m = 2;\ x = 1.1$
11	$y = \sin^3 a + \cos^2 x; a = c + k^2; c = \mathrm{arctg}\lvert x\rvert$	$k = 7.2;\ x = 5$
12	$y = e^{\sqrt{\lvert x\rvert}} + \cos x; x = a + c^3; a = \sin^5 b$	$b = 3;\ c = 1.7$
13	$y = a\cos x - b\sin x; x = \sqrt[3]{a - b}; a = t^2 b$	$t = 2.2;\ b = 3$
14	$y = \sqrt{x}\sin a + \sqrt{b}\cos x; a = \lg\lvert x\rvert; b = x + p^3$	$x = 11;\ p = 2.6$
15	$y = \lg a / \lg b; a = \sqrt{x^2 + b^2}; x = e^b + N$	$N = 9.1;\ b = 3$

3. Контрольні запитання

1. Як створити проєктне рішення з проєктом консольного застосування?
2. Як додати в рішення ще один проєкт консольного застосування?
3. У який спосіб можна призначити проєкт стартовим?
4. Що таке простір імен?
5. Для чого призначене ключове слово «using»?
6. Засоби яких просторів імен були використані в процесі виконання запропонованих завдань?
7. Що таке клас?
8. Які елементи може містити клас? Яке їх призначення?
9. Що таке метод?
10. У який спосіб здійснюється виклик методів класу?
11. В чому полягає особливість використання статичних елементів класу?
12. В чому полягає особливість використання статичних класів?
13. У який спосіб здійснюється доступ до значень властивостей класу?
14. Що таке точка входу в додаток?
15. Що є точкою входу консольного застосування?

16. Засоби якого класу призначені для підтримки роботи з вікном консолі?

17. Як змінити текст рядка заголовка вікна консолі?

18. Як встановити потрібний колір фону консолі?

19. Як вказати колір переднього плану консолі?

20. Яку команду використовують для виведення даних на екран?

21. В чому відмінність між методами Write та WriteLine класу Console?

22. У яких випадках використовується метод Write класу Console?

23. У який спосіб можна здійснити форматування тексту, що виводиться?

24. Як вказати ширину поля виведення значення і спосіб вирівнювання значення?

25. Що таке керуючі послідовності і з якою метою вони використовуються?

26. Що визначає властивість OutputEncoding класу Console?

27. Що визначає властивість InputEncoding класу Console?

28. Використання яких шрифтів дає змогу правильно реалізувати введення та виведення тексту українською мовою? Яке кодування потрібно під час цього задати?

29. Які засоби для роботи з вікном консолі надає вікно його властивостей?

30. Що таке тип даних? Що визначає тип даних?

31. Які ви знаєте базові типи даних?

32. Як оголосити змінну?

33. Що таке ініціалізація змінної?

34. Як можна описати оператор введення даних із клавіатури?

35. Як перетворити рядкове значення до значень базових типів даних?

36. Як отримати текстове представлення значень базових типів даних?

37. Як отримати шістнадцяткове представлення значень цілочислових типів даних?

38. Як отримати двійкове представлення значень цілочислових типів даних?

39. Як можна проаналізувати значення елементів програми під час її виконання?

40. Який клас містить константи і методи для виконання математичних обчислень?

ЛАБОРАТОРНА РОБОТА № 2
СТВОРЕННЯ ДІАЛОГОВОЇ ПРОГРАМИ. WINDOWS FORMS. ЕЛЕМЕНТИ LABEL, BUTTON, TEXBOX

Мета заняття: навчитися створювати діалогові програми за допомогою Windows Forms мовою C#; навчитися працювати з елементами Label, Button, TexBox Windows Forms у середовищі Microsoft Visual Studio.

1. Теоретичні відомості

1.1. Вступ до C# Windows Forms

На рис. 2.1 продемонстровано процес створення нового проєкту у Microsoft Visual Studio. Для цього у параметрах треба обрати мову C# (#1) та шаблон Windows Forms (#2).

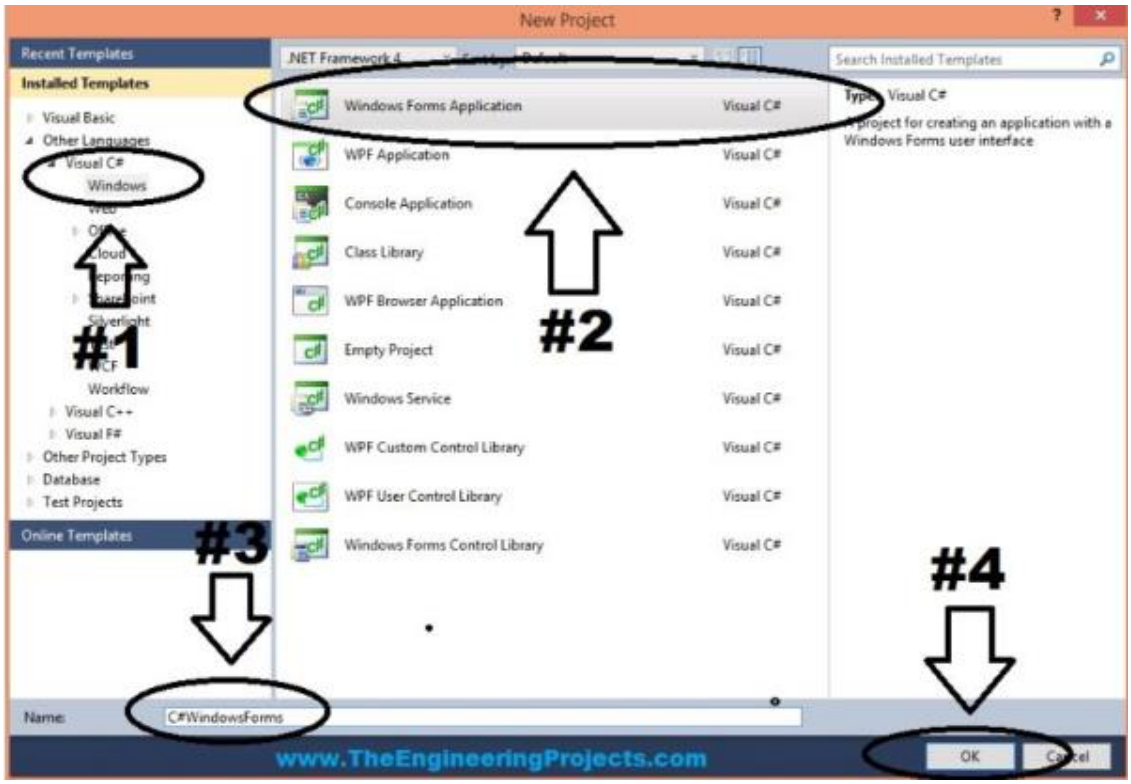


Рис. 2.1. Створення проєкту з Windows Forms у C#

Після підтвердження кнопкою «OK» (#4) відкриється новий проєкт (рис. 2.2).

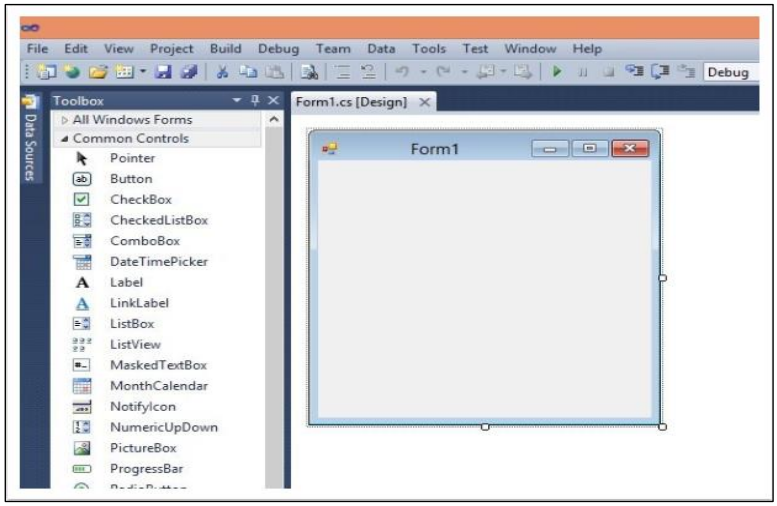


Рис. 2.2. Вікно Visual Studio з новим проєктом у Windows Forms

Створений проєкт складається з трьох основних частин. У центрі розташована форма з назвою Form1. Ліворуч знаходиться панель інструментів, що містить доступні компоненти для роботи з Windows Forms. Праворуч – «Провідник рішень», під яким розташована «Панель властивостей».

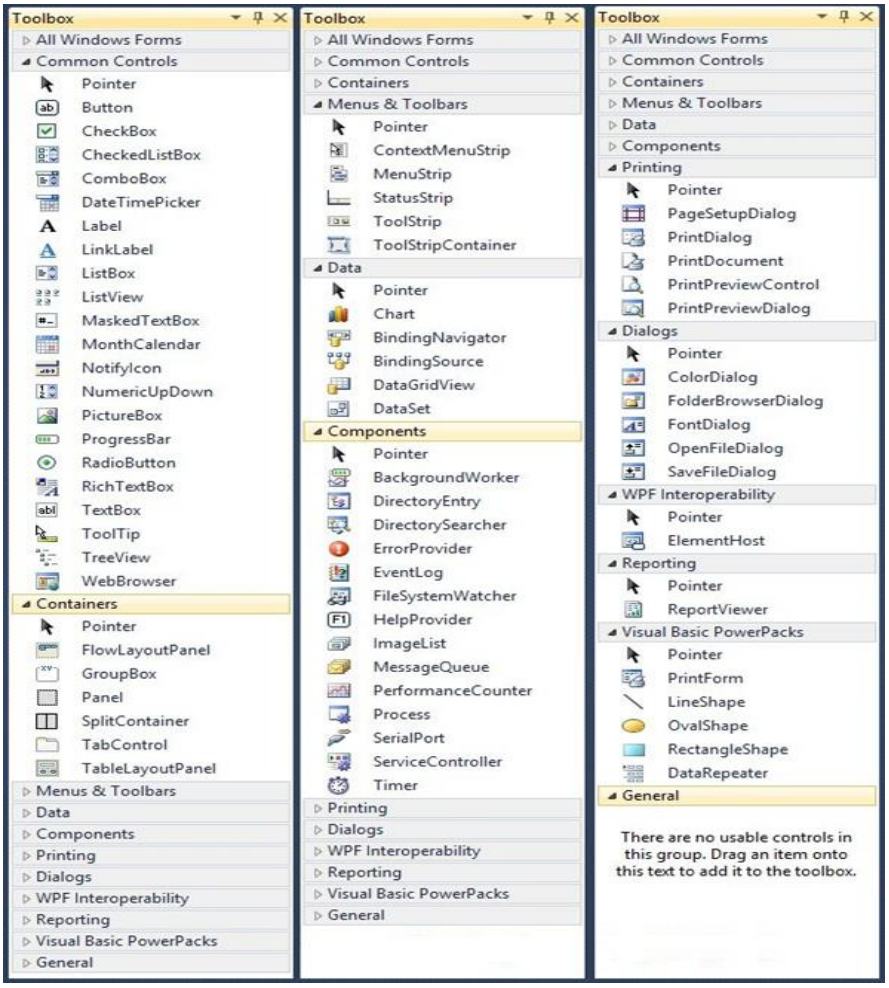


Рис. 2.3. Компоненти панелі інструментів C# Windows Forms

Усі компоненти можна застосовувати у проєкті, однак найчастіше використовуються кнопки та текстові поля з розділу «Загальні елементи керування». На рис. 2.4 подано зображення Провідника рішень у формах C# Windows.

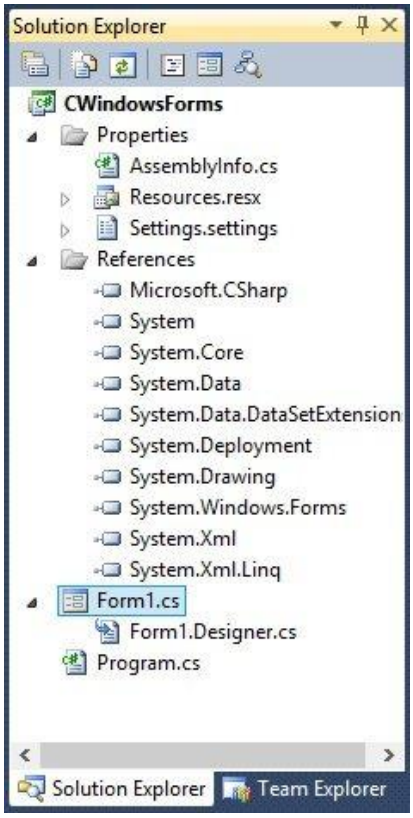


Рис. 2.4. Провідник рішень у формах C# Windows

Провідник рішень показує всі файли, які використовуються у вашому C# Windows Forms Project. У папці «Properties» зберігається файл AssemblyInfo.cs з інформацією про ім'я та авторські права. У розділі «References» зібрані бібліотечні файли системи. У файлі Form1.Designer.cs міститься опис дизайну форми Form1. У файлі Program.cs розташований код для її виконання. Весь новий програмний код проєкту буде додаватися саме у цей файл.

На рис. 2.5 наведено приклад вікна «Властивості» у Windows Forms. Під час вибору будь-якого компонента відображаються його характеристики, які можна змінювати відповідно до потреб.

Зокрема, атрибут Text визначає назву форми. За замовчуванням це Form1, але його можна змінити, наприклад, на Introduction to C# Windows Forms (рис. 2.6).

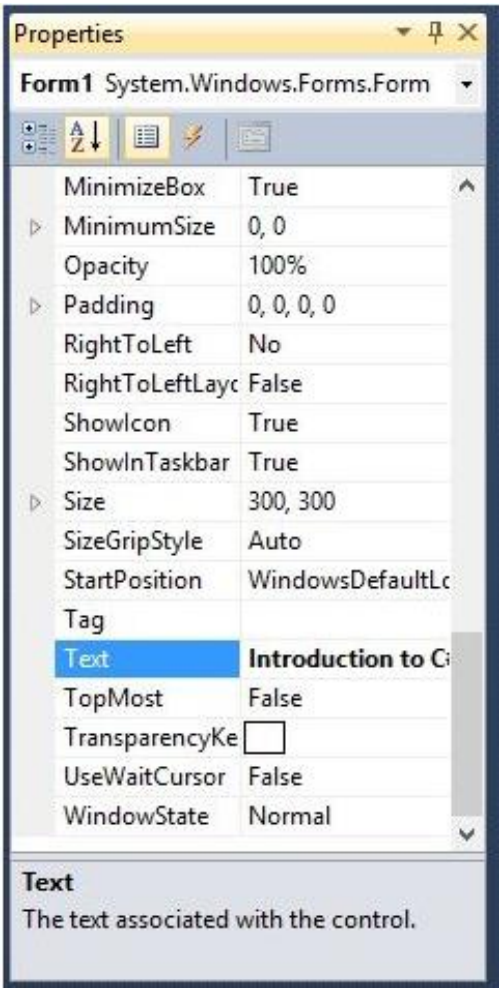


Рис. 2.5. Properties panel in C# Windows Forms

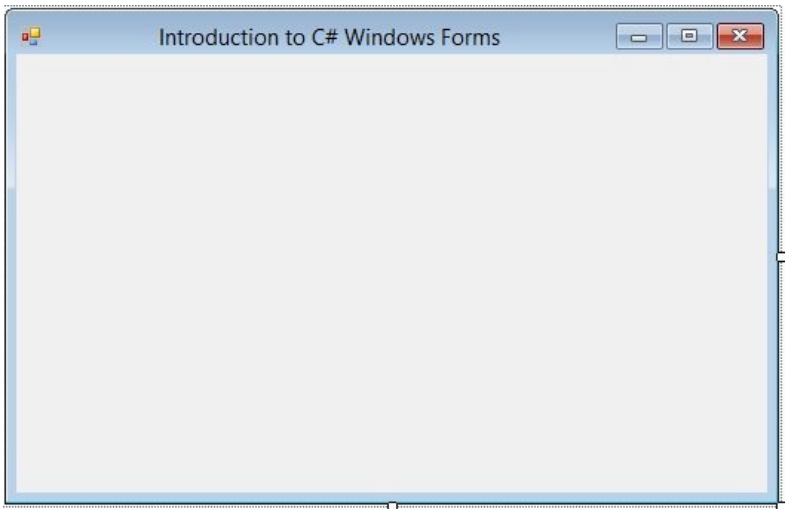


Рис. 2.6. Змінений заголовок Windows Forms

Таблиця 2.1 – Основні властивості форми

Властивість	Опис
Name	Визначає ім'я класу Form. Це значення задається лише під час розробки
BackColor	Вказує колір фону форми
Enabled	Визначає, чи може форма приймати введення від користувача. Якщо значення **False,** усі елементи керування також стають недоступними
ForeColor	Задає колір тексту (передній план форми)
FormBorderStyle	Визначає вигляд, поведінку меж і заголовка форми. Можливі значення: ***None*** – відсутні межі, немає можливості мінімізувати чи розгортати, відсутні кнопки керування та довідки. ***FixedSingle*** – тонка межа; змінювати розмір не можна; форма може бути мінімізована, розгорнута; наявна кнопка довідки або керування. ***Fixed3D*** – об'ємна межа; змінювати розмір не можна; можливість мінімізації та розгортання; кнопка довідки або керування доступна. ***FixedDialog*** – тонка межа; розмір змінювати не можна; кнопка керування відсутня, але може бути кнопка довідки; доступна мінімізація та розгортання. ***Sizable*** – стандартний варіант, користувач може змінювати розмір; підтримує мінімізацію, розгортання та кнопку довідки. ***FixedToolWindow*** – тонка межа; змінювати розмір не можна; доступна лише кнопка закриття. ***SizableToolWindow*** – тонка межа; користувач може змінювати розмір; доступна тільки кнопка закриття
Location	Коли властивості **StartPosition** задано значення **Manual,** то властивість указує початкове положення форми відносно верхнього лівого кута екрану
MaximizeBox	Указує, чи є у форми кнопка **«MaximizeBox»**
MaximumSize	Встановлює максимальний розмір форми. Якщо вказати 0;0, обмеження не застосовується
MinimizeBox	Указує, чи у форми є кнопка **«MinimizeBox»**
MinimumSize	Визначає мінімально допустимий розмір форми, який користувач може задати
Opacity	Регулює прозорість у межах від 0 до 100 % (100 % – форма повністю непрозора)
Size	Приймає та встановлює початковий розмір форми
StartPosition	Указує на положення форми в момент її першого виведення на екран
Text	Указує заголовок форми
TopMost	Указує, чи завжди форма відображається поверх усіх форм, властивості **TopMost** яких не задано значення **True**
Visible	Указує, чи видима форма під час роботи
WindowState	Указує, чи форма є мінімізованою, розгорнутою до максимальних розмірів, або при першій появі їй зада ний розмір , що вказаний у властивості **Size**

1.2. Додавання елементів керування C#

Отже, перший C#-елемент керування, який будемо використовувати, – це поле Text Box, яке наявне в категорії **Common Controls** (Загальні елементи керування). Необхідно перетягнути це текстове поле з панелі інструментів до форми Windows, після чого воно буде розміщене у формі Windows, як показано на рис. 2.7.

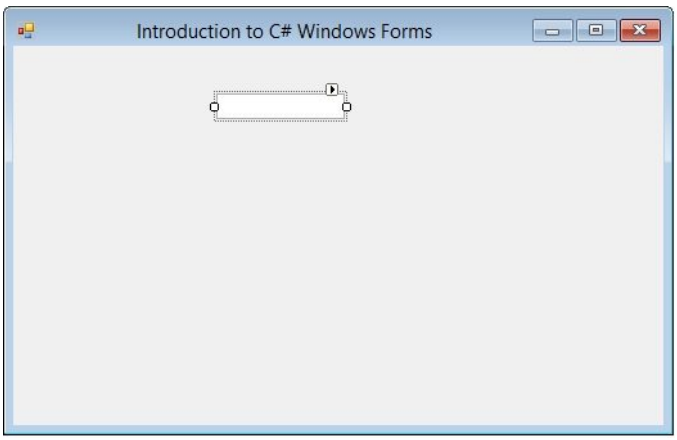


Рис. 2.7. Розташування елементу Text Box на Windows Forms

Якщо вибрати це текстове поле, тоді у розділі властивостей відкриються його властивості. З його властивостей ви можете змінити багато його атрибутів.

Два основні атрибути у властивостях будь-яких елементів керування C# – це ім’я (Name) та текст (Text).

Ім’я – це ім’я цього елементу управління, яким ми його називаємо, коли пишемо код для цього елементу управління.

Текст – це текст, який з’являється на цьому контролі для перегляду користувачем.

Перетягніть кнопку з панелі інструментів і розмістить її на формі Windows, як показано на рис. 2.8.

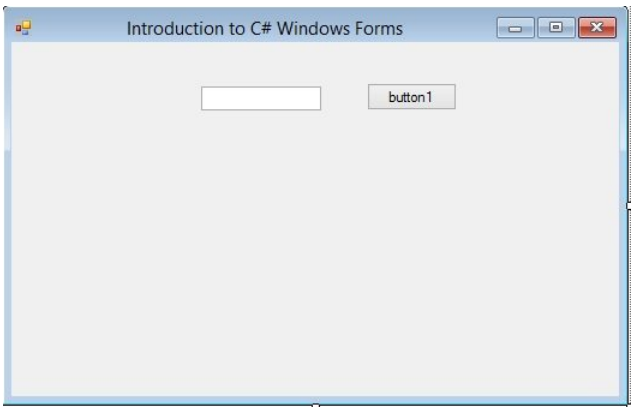


Рис. 2.8. Розташування елементу Button на Windows Forms

У розділі «Властивості» для Button змініть її текст на «Click Here», як показано на рис. 2.9.

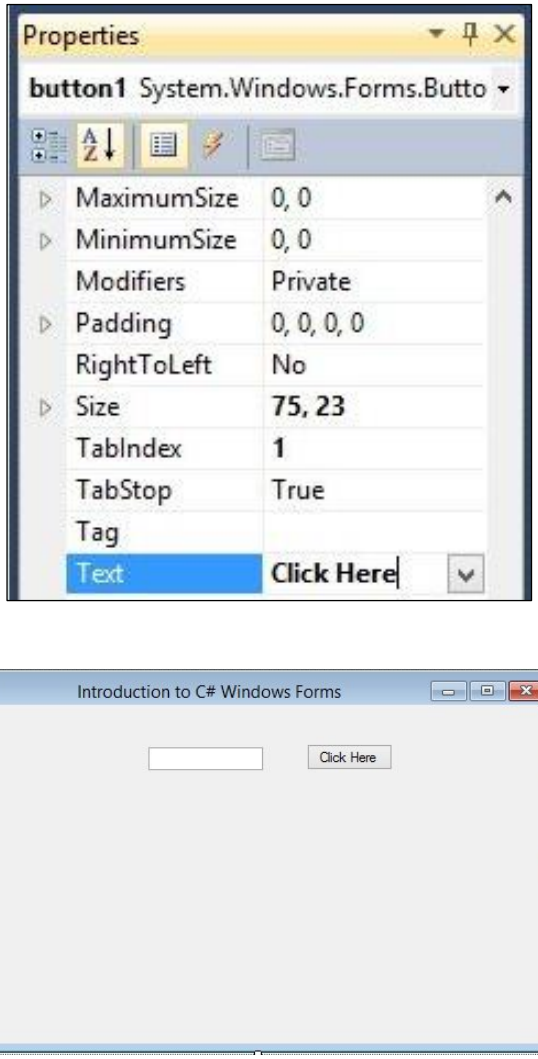


Рис. 2.9. Зміна тексту елементу Button на Windows Forms

1.3. Робота з кнопкою у формі C# Windows

Створіть новий Проєкт за зразком (рис. 2.10), він буде виглядати приблизно так:

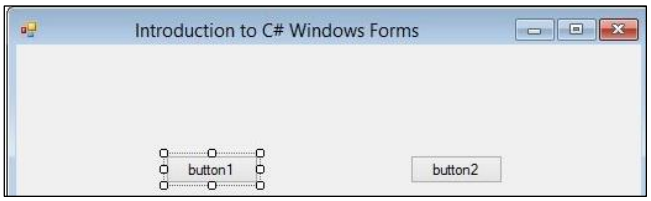


Рис. 2.10. Форма C# для Windows з 2 кнопками

Змініть текст Button1 на Click Here та текст Button2 на Submit. Також змініть назву цих кнопок, тож змініть Name першої кнопки на «ClickHere», а другої кнопки – «Submit».

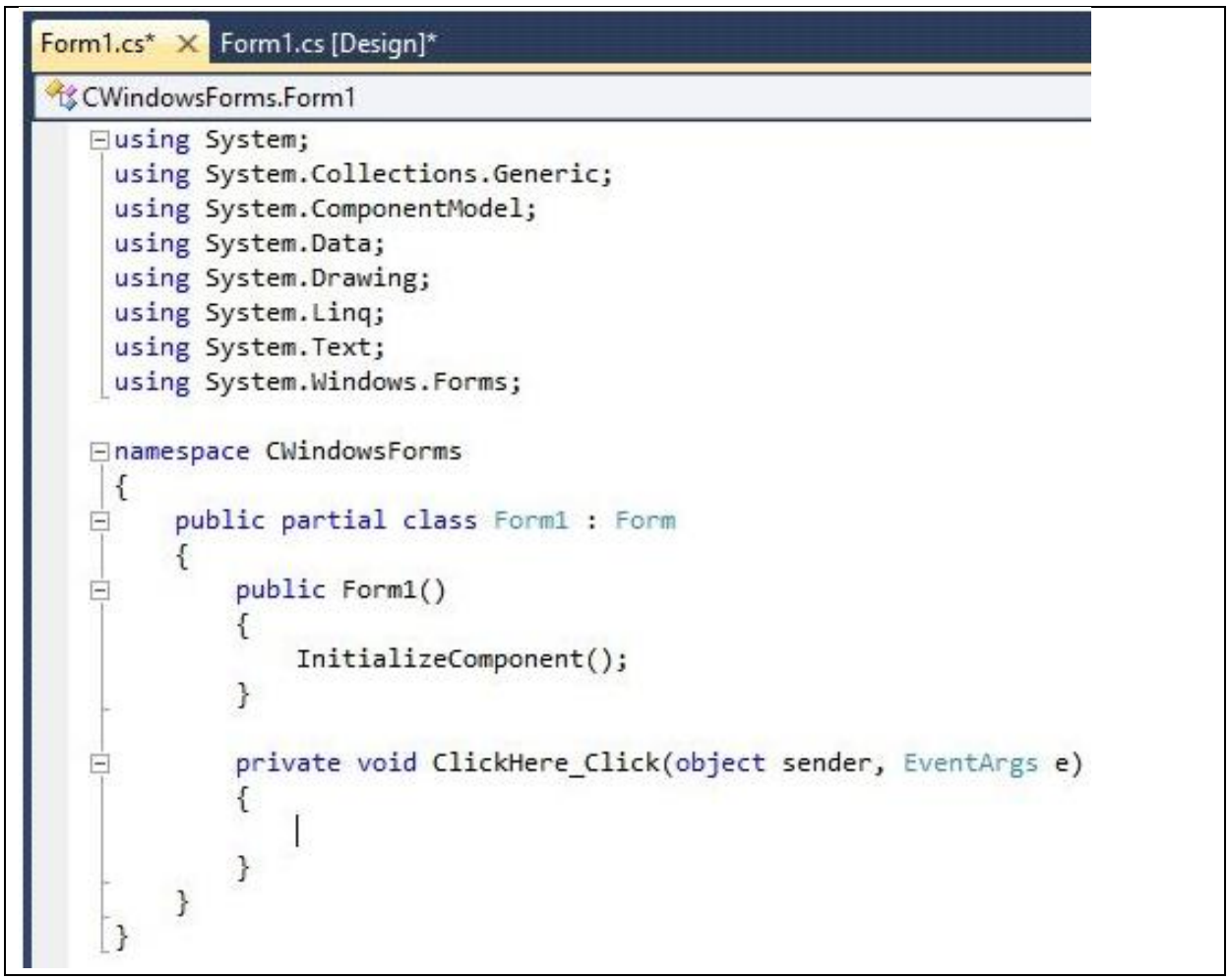


Рис. 2.11. Код обробки події Click для кнопки ClickHere

Додайте події Click для кожної з цих кнопок. Подія Click – це подія, яка повинна відбутися під час натискання на кнопку. Двічі клацніть першу кнопку, і вона відкриє код, як показано на рис. 2.11.

У наведеному коді є функція з назвою **ClickHere_Click,** яка є функцією події клацання для першої кнопки. **ClickHere** – це назва кнопки, яку змінили вище, а **_Click** – подія. Простими словами, ця подія відбудеться після натискання кнопки «ClickHere».

У фігурних дужках {} вставляється код для події клацання цієї кнопки. Додайте код у ці дужки:

```
MessageBox.Show ("It’s Click Here Buttom");
```

Коли буде натиснута кнопка «Click Here», відкриється вікно із повідомленням (рис. 2.12).

Аналогічно створіть обробку події для кнопки «Submit». Результат наведений на рис. 2.13.

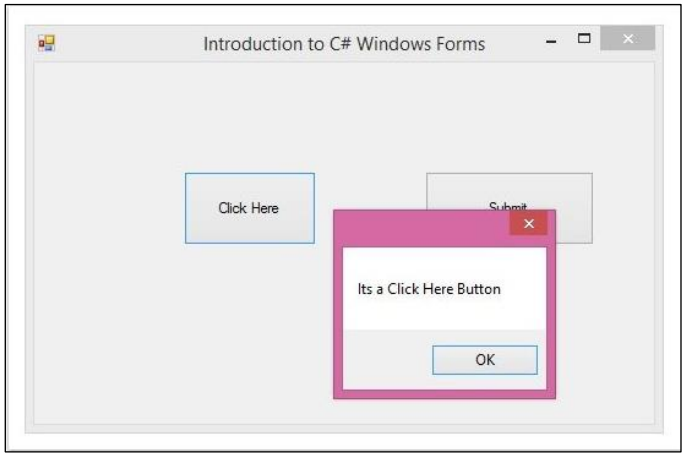


Рис. 2.12. Вікно повідомлення, що з'являється після натиснення кнопки «Click Here»

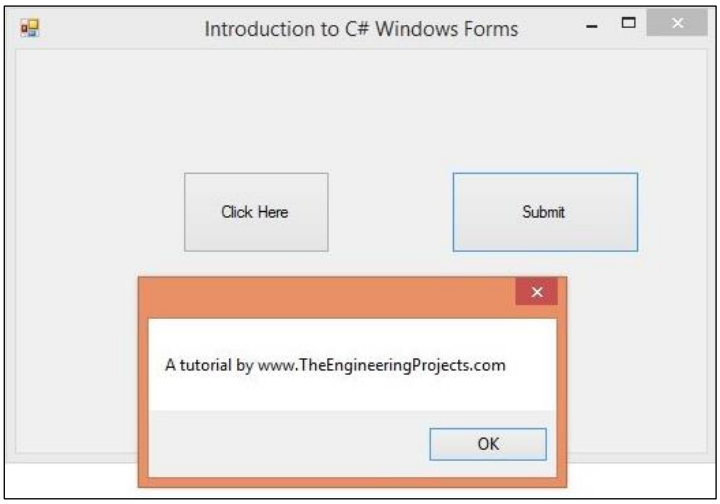


Рис. 2.13. Вікно повідомлення, що з'являється після натиснення кнопки «Submit»

1.4. Керування кнопками C#

Якщо необхідно змінити текст кнопки під час виконання або динамічно, тоді використовується код:

```
button1.Text = "TEP Button Example Text";
```

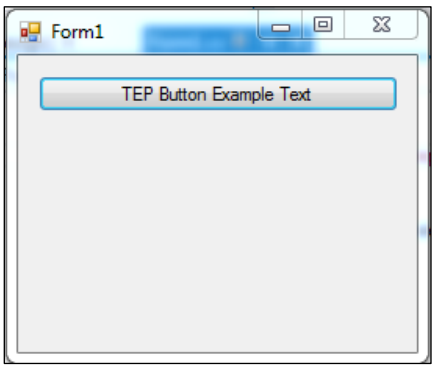


Рис. 2.14. Зміна тексту кнопки

Якщо потрібно показати зображення всередині кнопки, тоді використовується властивість Image. **FromFile** використовується для встановлення шляху зображення, яке ви хочете встановити.

```
Button1.Image = Image.FromFile("C:\\Users\\Jade\\Pictures\\brownImage.jpg");
```

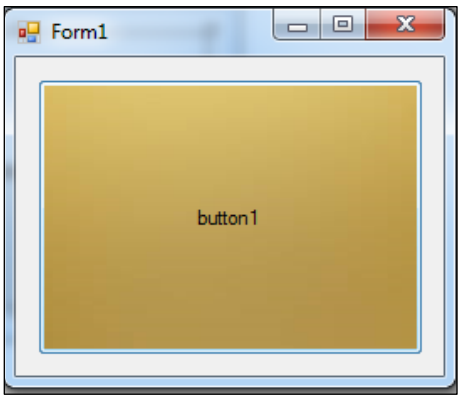


Рис. 2.15. Результат встановлення зображення brownImage.jpg для кнопки

Для встановлення кольору фону кнопки використовують властивість BackColor.

Можна використовувати вбудовані атрибути кольору для встановлення кольорів:

```
Button1.BackColor = Color.Aqua;
```

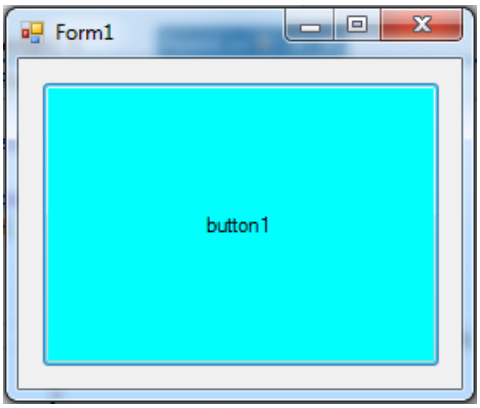


Рис. 2.16. Результат встановлення фону кнопки

Властивість **ForeColor** використовується для встановлення кольору тексту.

Приклад коду, який змінить колір переднього плану на чорний, та встановить колір тексту білий:

```
button1.ForeColor = Color.White;
button1.BackColor = Color.Black;
```

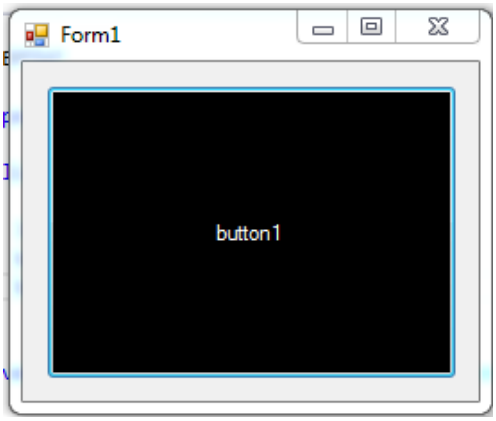


Рис. 2.17. Результат встановлення властивостей кольору фону та тексту кнопки

Властивість **Font** дає змогу змінити розмір тексту кнопки. Створіть **new Font object** і передайте **button.font.fontfamily,** після коми **ставиться** розмір тексту кнопки:

```
Button1.Font = new Font(button1.Font.FontFamily, 33);
```

або:

```
int newSize = 33;
button1.ForeColor = Color.White;
button1.BackColor = Color.Black;
button1.Font = new Font(button1.Font.FontFamily, newSize);
```

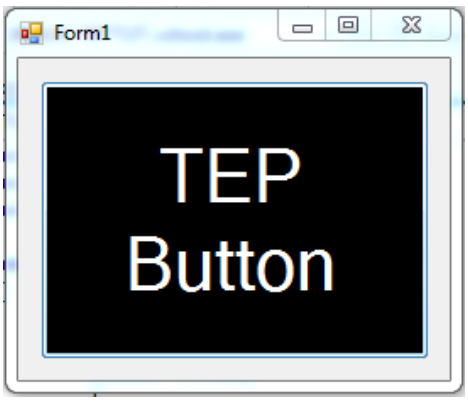


Рис. 2.18. Результат встановлення розміру тексту кнопки

Є кілька подій, які можемо використовувати для кнопок. Події – це деякі умовні перевірки на виконання конкретних функцій, коли користувач виконує щось конкретне. Перелік поширених подій, які використовуються для кнопки C#:

– Click Event;
– Text Changed Event;
– MouseHover Event;
– MouseLeave Event.

Подія Click відбудеться, коли кінцевий користувач один раз натисне кнопку. Для використання події **Click** для обробки цієї функції після назви кнопки необхідно написати **_Click.** Наприклад для кнопки з ім'ям button1 доведеться створити метод з іменем **button1_Click:**

```
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("C# Button Click Event Executed");
        }
```

Подія Text Changed відбувається під час зміни тексту кнопки. Це вбудований метод з ім'ям **_TextChanged.** У наступному коді є подія **button1_Text Changed,** яка генерує спливаюче повідомлення під час зміни тексту кнопки. Подія **Button1_Click** використовується для зміни тексту, коли користувач натисне кнопку. Логіка – це коли користувач натисне кнопку, а текст кнопки буде змінено і **button1_TextChanged** виконуватиметься:

```
        Private void button1_Click(object sender, EventArgs e)
        {
            button1.Text = "New Text";
        }
        private void button1_TextChanged(object sender, EventArgs e)
        {
            MessageBox.Show("C# Button TextChanged Event");
        }
```

Подія **MouseHover** відбувається, коли користувач наводить курсор миші на кнопку. Це вбудована подія з ім'ям **_MouseHover** після імені кнопки. У коді використовується типова назва кнопки. Коли користувач наведе курсор миші на кнопку «button1», він створить спливаюче повідомлення:

```
        Private void button1_MouseHover(object sender, EventArgs e)
        {
            MessageBox.Show("C# Button MouseHover Event");
        }
```

Подія **MouseLeave** відбувається, коли користувач залишить кнопку або перемістить курсор із меж кнопки. У наступному коді створена подія **_MouseLeave** із назвою кнопки за замовчуванням. Коли користувач забере курсор миші з кнопки, генерується спливаюче повідомлення.

```
Using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
```

```
        {
        }
        private void button1_MouseLeave(object sender, EventArgs e)
        {
            MessageBox.Show("C# Button MouseLeave Event");
        }
```

1.5. C#-керування мітками

Label (мітка) також використовується для відображення описового тексту, як-от примітки та попередження про використання програмного забезпечення. Щоб додати мітку з панелі інструментів, її потрібно перетягнути з вкладки дизайнера форми.

Label1 – це ім’я за замовчуванням першої мітки, яку використовують у програмі. Можна регулювати розмір форми, розтягуючи координати. Можна змінити текст на правій панелі під міткою властивості. Також можна змінити текст, використовуючи наведений код (рис. 2.19):

```
namespace Labl
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = "TEP C# Label Control Example Text";
        }
    }
}
```

Рис. 2.19. Зміна тексту мітки Label1 під час завантаження форми

Рис. 2.20. Результат зміни тексту Label1 під час завантаження форми

Властивість BackColor використовується для зміни кольору фону мітки, наприклад (рис. 2.21):

```
label1.BackColor = Color.Aqua;
```

Результат коду:

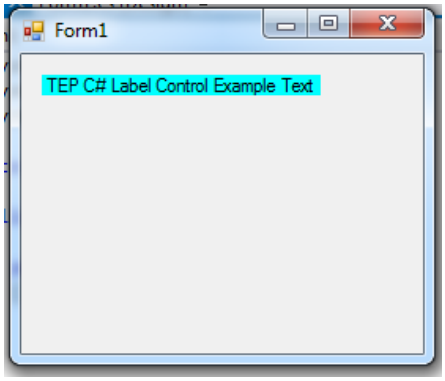


Рис. 2.21. Встановлення фону кольору Label1

Властивість ForeColor використовується для зміни кольору тексту, наприклад (рис. 2.22):

```
label1.ForeColor = Color.BlueViolet;
```

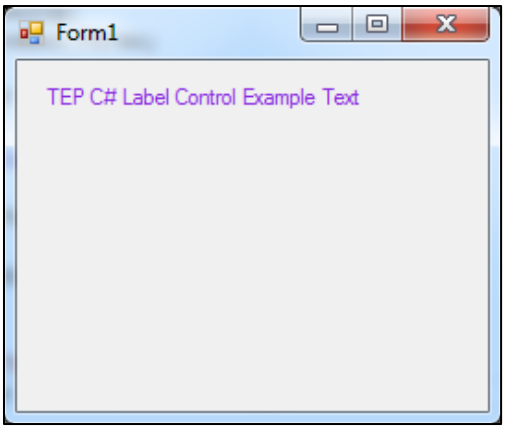


Рис. 2.22. Встановлення кольору тексту Label1

Для встановлення зображення як фону мітки потрібно використовувати такий код:

```
label1.Image = Image.FromFile("C:\\Pictures\\brownImage.jpg");
```

Метод Image.FormFile використовується для вказання шляху зображення, яке ви хочете встановити. У подвійні лапки пишуть шлях і використовують «\\» для розділення каталогів. Фрагмент коду, що змінює властивості Label1, та результат його виконання наведений на рис. 2.23.

```
private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "TEP C# Label Control Example Text";
    //label1.BackColor = Color.Aqua;
    label1.ForeColor = Color.BlueViolet;
    //label1.Image = Image.FromFile("C:\\Users\\Public\\Pictures\\img016.jpg");
}
```

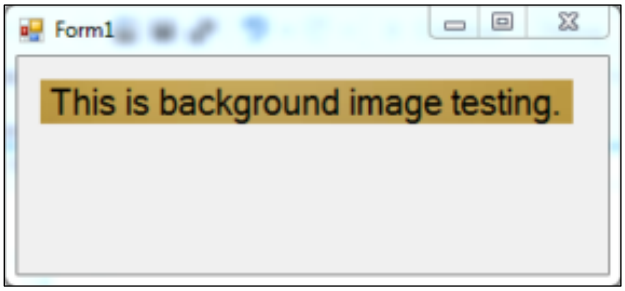


Рис. 2.23. Результат виконання фрагмента коду, що змінює властивості Label1

Є кілька подій, які можна використовувати для міток (рис. 2.24):

– Click Event;
– Double Click Event;
– Text Changed Event;
– MouseHover Event;
– MouseLeave Event.

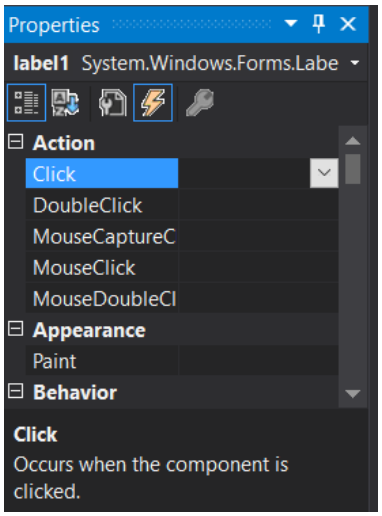


Рис. 2.24. Події для елементу Label1

Подія Click виконається, коли користувач натисне на мітку:

```
Private void label1_Click(object sender, EventArgs e)
        {
            label1.Text = "Text Changed";
        }
```

Подія Double Click відбудеться, коли користувач двічі натисне на label:

```
private void label1_Click(object sender, EventArgs e)
{
    label1.Text = "Text Changed";
}

private void label1_DoubleClick(object sender, EventArgs e)
{
    label1.Text = "Text New Changed";
}
```

Подія Text Changed (рис. 2.25) відбудеться під час зміни тексту мітки. Розглянемо це на прикладі. Використовуються дві мітки. Тепер логіка полягає в тому, що текст однієї мітки змінюється, тоді як інший текст іншої мітки повинен змінюватися автоматично.

Для першої мітки також використовується **_ClickEvent** та **_TextChanged Event.**

Отже, тепер, коли користувач натисне на мітку, текст мітки зміниться через _Click Event. Тепер, коли текст буде змінено, відбудеться **_TextChanged Event,** і текст другої мітки також зміниться.

Ось код, який варто спробувати:

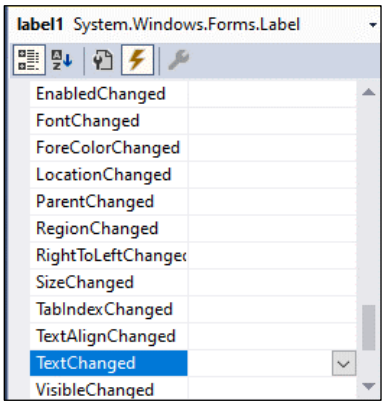


Рис. 2.25. Вибір події Text Changed для Label1

```
private void Form1_Load(object sender, EventArgs e)
{
    //label1.Text = "TEP C# Label Control Example Text";
    //label1.BackColor = Color.Aqua;
    label1.ForeColor = Color.BlueViolet;
    //label1.Image = Image.FromFile("C:\\Users\\Public\\Pictures\\img016.jpg");
}

private void label1_Click(object sender, EventArgs e)
{
    label1.Text = "Text Changed";
}

private void label1_DoubleClick(object sender, EventArgs e)
{
    label1.Text = "Text New Changed";
}

private void label1_TextChanged(object sender, EventArgs e)
{
    label2.Text = "2nd Text Changed";
}

        }
```

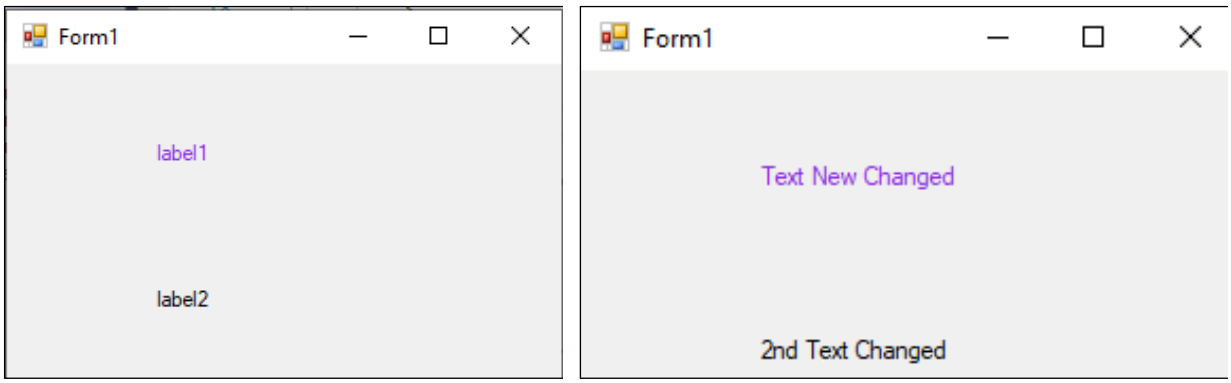


Рис. 2.26. Результат події Click та Text Changed для Label1 та Label2

Подія MouseHover відбудеться, коли користувач наведе курсор миші на мітку. Припустимо, ви хочете змінити текст, коли користувач наведе курсор миші на мітку. Ось код для цього:

```
private void label1_MouseHover(object sender, EventArgs e)
        {
        label1.ForeColor = Color.Red;
        label1.Text = "Text Too Changed";
        }
```

Подія MouseLeave відбудеться, коли курсор покине мітку. Припустимо, ви хочете змінити текст мітки, коли курсор миші покине мітку. Ось код, який буде виконувати це:

```
private void label1_MouseLeave(object sender, EventArgs e)
        {
            label1.Text = "again Text Changed";
        }
```

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для визначення віку за даними навчання на відповідному курсі університету.

1. На форму додайте дві кнопки з текстом «ClickHere» та «ОК», поле вводу – TexBox, Label1, Label2 (рис. 2.27).

2. Задайте властивості, щоб під час запуску форми елементи Label2, TexBox, Button2 були приховані. За замовчуванням значення TexBox дорівнює 0.

3. Задайте колір тексту Label1 – DarkBlue, Label2 – DarkOrange; розмір шрифту 14. Label1 повинна виводити текст «На якому курсі ви навчаєтесь? Введіть число після натиснення кнопки» у 2 рядки.

4. Після натиснення на кнопку «ClickHere» з'являється поле TexBox та кнопка «ОК».

5. Після введення числа у поле TexBox та натиснення кнопки «ОК» виводиться вікно з повідомленням про ваш вік. Для цього попередньо потрібно зчитати значення числа, що введено у поле TexBox, та додати до нього 18.

6. Після закриття вікна повідомлення приховайте Label1 та виведіть у Label2 повідомлення про ваш вік (рис. 2.28).

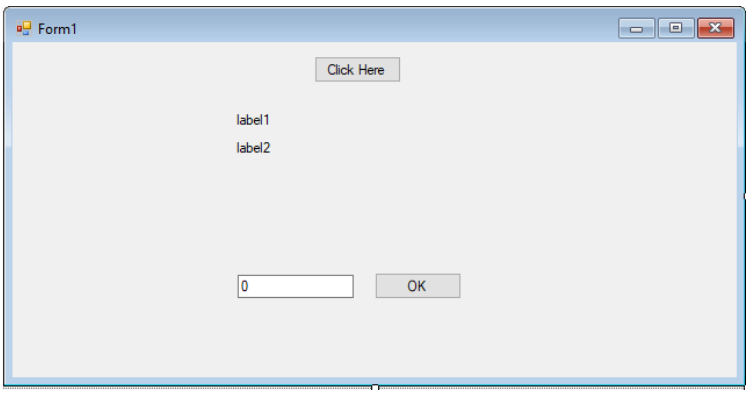


Рис. 2.27. Вигляд форми із доданими кнопками «ClickHere» та «ОК»

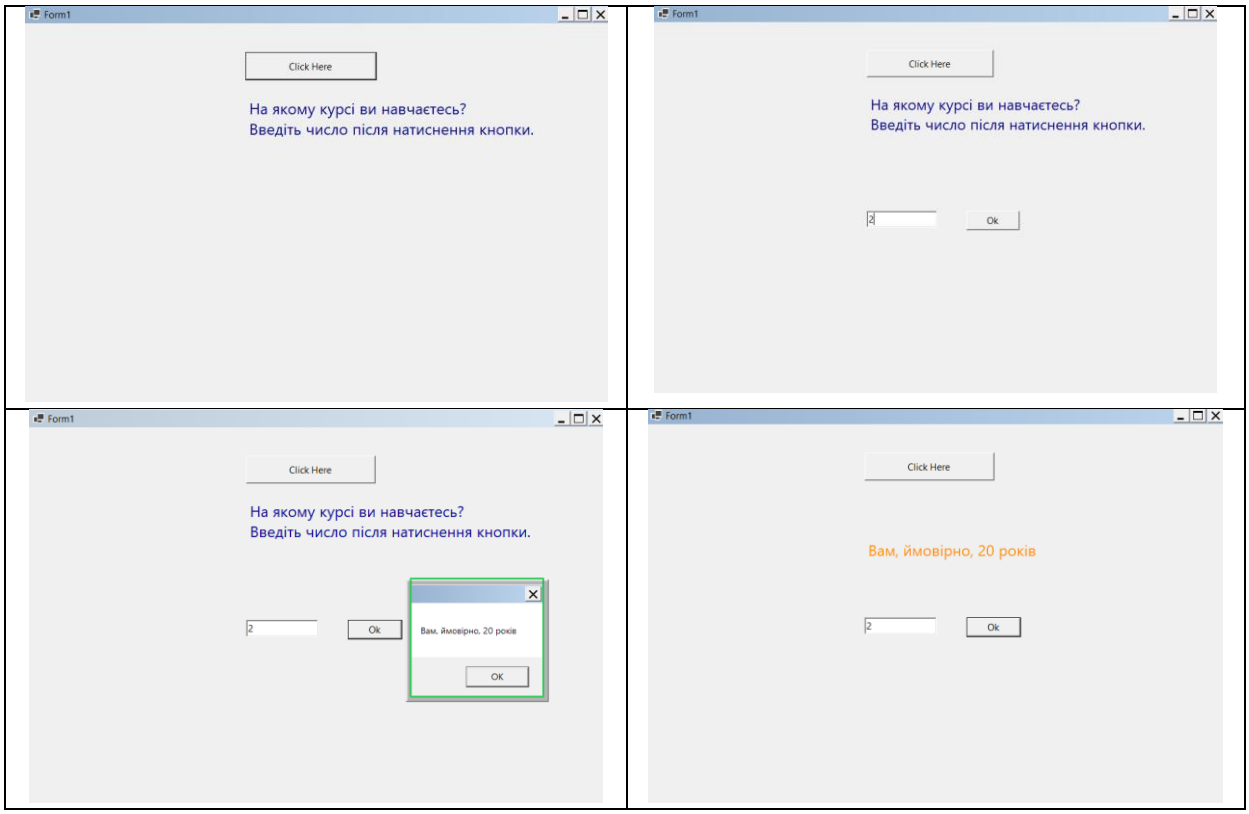


Рис. 2.28. Вигляд форми під час виконання завдання 1

```
namespace Lab2
{
    public partial class Form1 : Form
    {
        int c;
        public Form1()
        {
            InitializeComponent();
            textBox1.Visible = false;
            label2.Visible = false;
            button2.Visible = false;
            label2.Font = new Font(label1.Font.FontFamily, 14);
            label2.ForeColor = Color.DarkOrange;
            label1.Font= new Font(label1.Font.FontFamily,14);
            label1.ForeColor = Color.DarkBlue;
            label1.Text = "На якому курсі Ви навчаєтесь?\n Введіть число після натиснення кн
        }
        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Visible = true;
            button2.Visible = true;

        }
        private void button2_Click(object sender, EventArgs e)
        {
            c = Convert.ToInt16(textBox1.Text);
            MessageBox.Show("Вам ймовірно "+Convert.ToString(18+c)+" років");
            label2.Text = "Вам ймовірно " + Convert.ToString(18 + c) + " років";
            label1.Visible = false;
            label2.Visible = true;
        }
    }
}
```

Завдання 2. Скласти програму «Стрибаюча кнопка».

1. Створіть новий проєкт. Помістіть на форму компонент *Label* і кнопку *«Спіймай мене!»*.

2. Створіть для кнопки обробник події Click. Опишіть змінну *k* для збереження кількості клацань кнопки:

```
int k;
```

У код процедури запишіть оператори:

```
k= k + 1;
Label1.Text = Convert.ToString(k);
```

3. Додайте на форму компонент *Timer* (Таймер). Властивість таймера *Interval* встановіть 1 000 (1 000 мс = 1 с). Властивість *Enable* переведіть у положення *True*.

4. Створіть обробник події *«Tick»,* у якому запрограмуйте випадкову зміну властивостей *«Left»* і *«Top»* – кнопки:

```
Private void timer1_Tick(object sender, EventArgs e)
   {
```

```
    Random rand = new Random();
    button1.Left = rand.Next(Width-button1.Width);
    button1.Top = rand.Next(Height-button1.Height);
   }
```

Кнопка не має «вистрибнути» за межі форми, тому потрібно підключити розміри робочої поверхні форми *Width* і *Height.*

5. Запустіть програму на виконання. Чи вдається спіймати кнопку? Якщо ні, зупиніть виконання і збільшіть значення властивості таймера *Interval.*

6. Додайте на форму кнопку *«Button2»,* в програмному коді якої запрограмуйте збільшення значення *Interval*:

```
Timer1.Interval = Timer1.Interval + 100;
```

7. Додайте на форму кнопку *«Button3»,* призначену для зменшення значення *Interval*. Перевірте дію кнопок.

3. Контрольні запитання

1. У чому полягає принцип швидкої розробки застосувань?
2. На які частини умовно розділяють вікно середовища розробки MS Visual Studio?
3. Що таке проєкт в MS Visual Studio?
4. Як створити проєкт з користувацькими вікнами (формами)?
5. Які основні файли проєкту?
6. Як створити і переглянути процедури обробки подій форми і окремого елементу керування?
7. У якому файлі міститься головна програма у віконному проєкті?

ЛАБОРАТОРНА РОБОТА № 3
WINDOWS FORMS. ПЕРЕТВОРЕННЯ ТИПІВ ДАНИХ

Мета заняття: навчитися працювати з елементом TexBox у середовищі Microsoft Visual Studio; набути практику перетворення типів даних для введення даних користувачем, їх обробки та виводу результатів за допомогою Windows Forms мовою C#; навчитися працювати з типами даних string, Array, ArrayList.

1. Теоретичні відомості

1.1. Елемент TextBox Windows Forms

TextBox у Windows Forms використовується для отримання даних від користувача, а також може слугувати для відображення певних значень. Текстове поле є контейнером для тексту, де можна як вводити, так і відображати текст у вигляді абзаців.

Щоб додати TextBox на форму, потрібно перетягнути його з панелі інструментів і розташувати у потрібному місці на формі. За замовчуванням назва текстового поля – «textBox1», її можна змінити у вкладці властивостей. Також у властивостях можна змінити текст, що відображається у полі. Якщо ж потрібно оновлювати текст динамічно під час виконання програми, використовують код:

```
textBox1.Text = "TheEngineeringProjects.com";
```

Під час запуску програми текстове поле автоматично відобразить заданий текст, як показано на рис. 3.1.

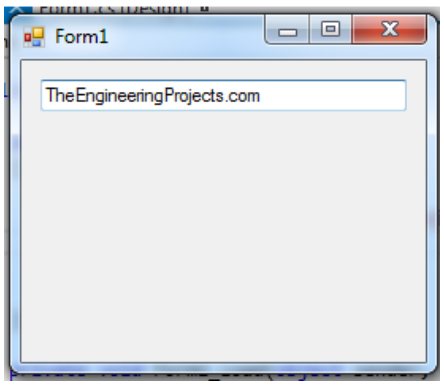


Рис. 3.1. Елемент TextBox

Значення текстового поля можна задавати за допомогою змінних. Наприклад, рядок можна зберегти у змінній, а потім присвоїти його текстовому полю:

```
string var = "TheEngineeringProjects.com";
textBox1.Text = var;
```

Щоб зчитати текст із текстового поля, використовують такий код:

```
string var;
var = textBox1.Text;
```

Для зміни властивостей текстового поля через панель властивостей використовується комбінація **F4.** Розмір поля (висоту та ширину) можна змінити у дизайнері, перетягнувши межі текстового поля, або динамічно за допомогою коду:

```
textBox1.Width = 250;
textBox1.Height = 50;
```

Змінити колір фону та тексту можна так (рис. 3.2):

```
textBox1.BackColor = Color.Blue;
textBox1.ForeColor = Color.White;
```

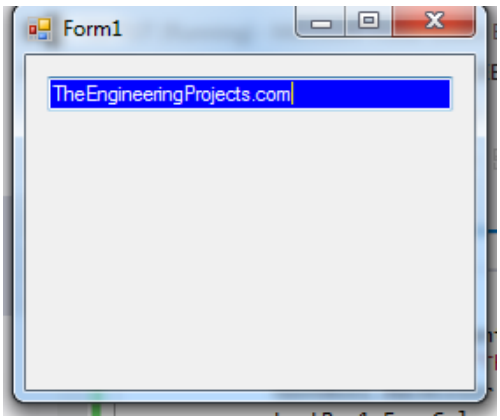


Рис. 3.2. Результат встановлення властивостей BackColor, ForeColor для елементу textBox1

Текстове поле підтримує три типи меж: FixedSingle, Fixed3D та None. Змінюють стиль межі так:

```
textBox1.BorderStyle = BorderStyle.Fixed3D;
textBox1.BorderStyle = BorderStyle.FixedSingle;
textBox1.BorderStyle = BorderStyle.None;
```

Для обмеження максимальної довжини введеного тексту використовується властивість:

```
textBox1.MaxLength = 40;
```

Щоб заборонити редагування тексту, застосовують властивість ReadOnly:

```
textBox1.ReadOnly = true;
```

Для багаторядкового текстового поля використовується властивість Multiline:

```
textBox1.Multiline = true;
```

Щоб текстове поле працювало як поле для введення пароля, застосовують:

```
textBox1.PasswordChar = '*';
```

Якщо потрібно, щоб користувач вводив текст із нового рядка, можна скористатися одним із двох способів:

```
// Перший метод
textBox1.Text + = "ваш текст" + "\ r \ n";
```

```
// Другий метод
textBox1.Text + = "ваш текст" + Environment.NewLine;
```

Число, що введене textBox1, є текстом, тому для роботи з числом його необхідно перетворити в ціле число або дійсне число:

```
int i;
i = int.Parse (textBox1.Text);
// String to Float перетворення
         float i;
         i = float.Parse (textBox1.Text);
//String to Double conversion
         double   i;
         i = float.Parse (textBox1.Text);
```

Усі властивості, які були розглянуті, додані до фрагмента коду (рис. 3.3):

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Width = 250;
            textBox1.Height = 50;
            textBox1.Multiline = true;
            textBox1.BackColor = Color.Blue;
            textBox1.ForeColor = Color.White;
            textBox1.BorderStyle = BorderStyle.Fixed3D;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string var;
            var = textBox1.Text;
            MessageBox.Show(var);
label1.Text = var;
        }
    }
}
```

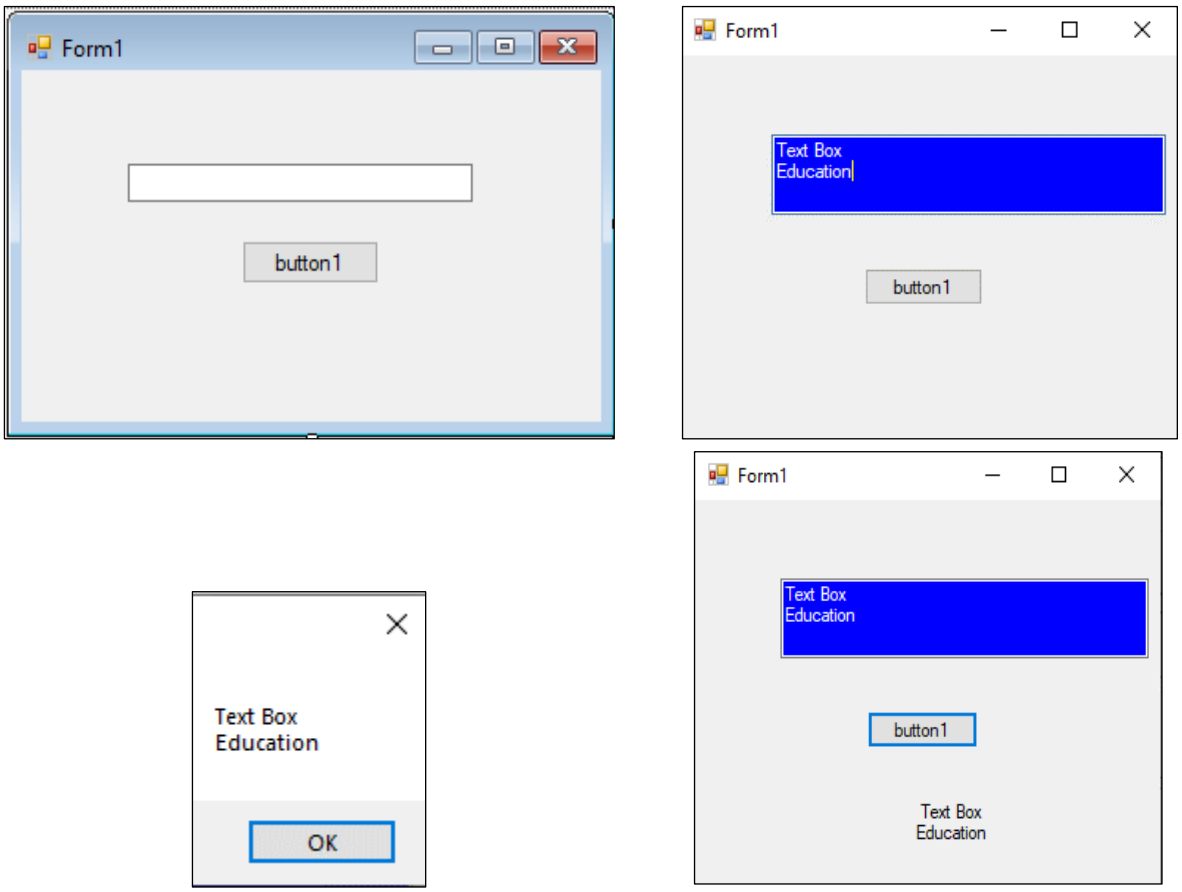


Рис. 3.3. Windows Forms, що показують особливості використання елементу textBox

1.2. Як використовувати у C# String змінні

Розробимо невеликий проєкт, щоб показати, як працює C# String.

Створіть простий проєкт C# з однією кнопку та одним текстовим полем, як показано на рис. 3.4.

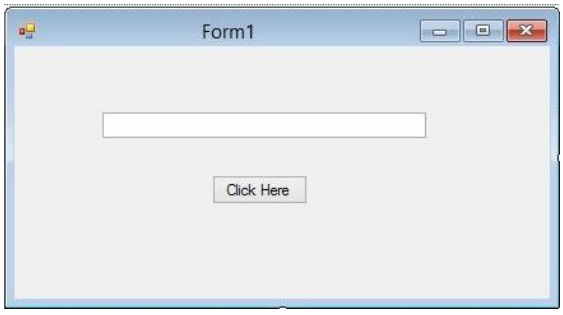


Рис. 3.4. Проєкт C# для вивчення особливостей роботи C# String

Змініть текст кнопки на «Click Here» та властивість «Name» на «ClickHere».

Аналогічно змініть «Name» текстового поля на txtClick.

До проєкту додайте код, що представлений на рис. 3.5.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Variables
{
    public partial class Form1 : Form
    {
        string webBlog;


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void ClickHere_Click(object sender, EventArgs e)
        {
            webBlog = "www.TheEngineeringProjects.com";
            txtClick.Text = webBlog;


        }
    }
}
```

Рис. 3.5. Фрагмент коду проєкту

У коді створена рядкова змінна **webBlog,** якій присвоєне значення. Це значення відображено у текстовому полі (рис. 3.6).

Рис. 3.6. Результат роботи програми, наведеної на рис. 3.5

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Variables
{
    public partial class Form1 : Form
    {
        string webBlog1, webBlog2;


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void ClickHere_Click(object sender, EventArgs e)
        {
            webBlog1 = "www.TheEngineeringProjects.com";
            webBlog2 = "www.TheEngineeringProjects.com";
            if (String.Compare(webBlog1, webBlog2) == 0)
            {
                txtClick.Text = "Both are equal";
            }
            else
            {
                txtClick.Text = "Both are not equal";
            }


        }
    }
}
```

Рис. 3.7. Фрагмент коду для порівняння двох рядків

Фрагмент коду, де порівнюються два рядки, наведений на рис. 3.7. На рис. 3.8 наводиться фрагмент програми, яка показує, як здійснювати пошук фрагмента тексту у рядку.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Variables
{
    public partial class Form1 : Form
    {
        string webBlog1, webBlog2;


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void ClickHere_Click(object sender, EventArgs e)
        {
            webBlog1 = "www.TheEngineeringProjects.com";
            if (webBlog1.Contains("Projects"))
            {
                txtClick.Text = "Its present in string";
            }
            else
            {
                txtClick.Text = "Its not present in string.";
            }


        }
    }
}
```

Рис. 3.8. Фрагмент коду для пошуку фрагменту тексту у рядку

Припустимо, в якомусь проєкті у вас дуже довгий рядок C#, і потрібна частина цього рядка C#. Тоді потрібно використовувати рядкову команду (рис. 3.9):

```
webBlog1.Substring (4).
```

У наведеному коді створені рядкові змінні C# String та задані їх значення. Друга змінна пропускає 4 символи першої змінної (рис. 3.10).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Variables
{
    public partial class Form1 : Form
    {
        string webBlog1, webBlog2;


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void ClickHere_Click(object sender, EventArgs e)
        {
            webBlog1 = "www.TheEngineeringProjects.com";
            webBlog2 = webBlog1.Substring(4);
            txtClick.Text = webBlog2;



        }
    }
}
```

Рис. 3.9. Фрагмент коду для вивчення поділу рядка на частини

Рис. 3.10. Результат виконання програми, що наведена на рис. 3.9

1.3. Перетворення типів даних у C#

Під час роботи над якоюсь програмою, керованою даними, завжди потрібно перетворювати одну форму даних в іншу, і саме там потрібні перетворення даних.

Невелика кількість перетворень не потребує жодного методу чи класу, тобто якщо ми хочемо перетворити ціле число у float, то компілятор може це легко зробити. Такі перетворення називаються неявними (рис. 3.11).

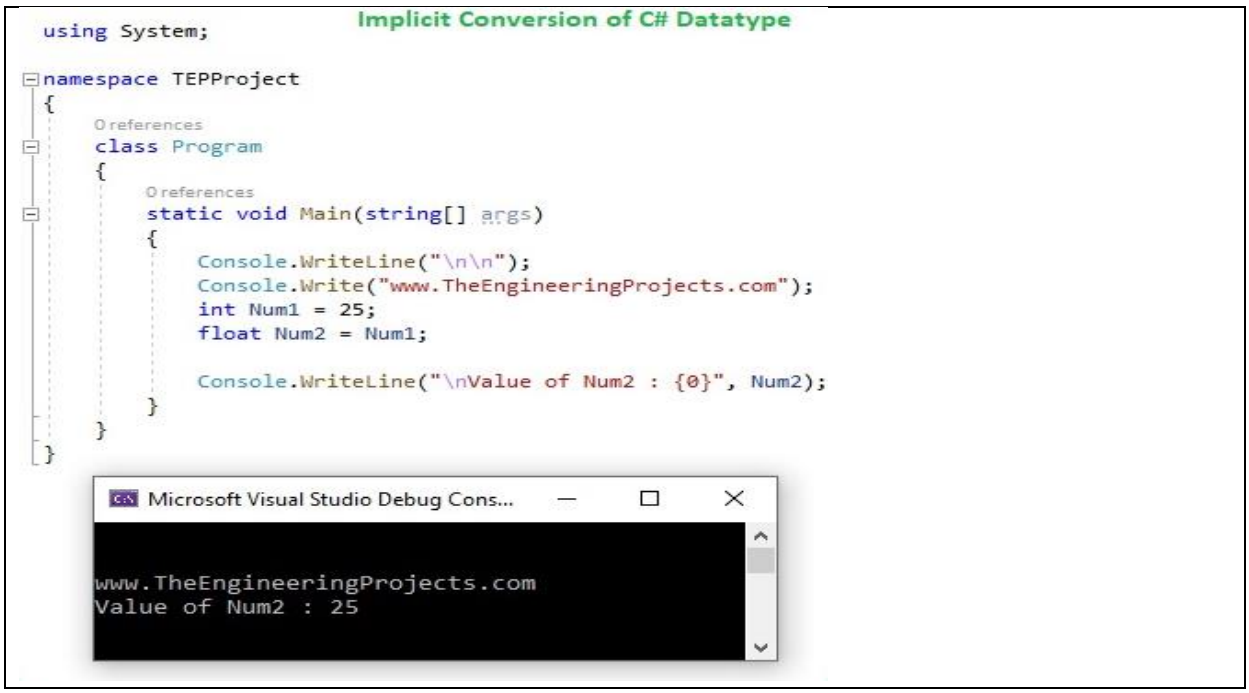


Рис. 3.11. Неявне перетворення даних типу Int у Float

Але якщо потрібно перетворити float (скажімо, 313.56) у ціле число, то виникає помилка компілятора, тобто виняток переповнення. Для здійснення таких перетворень необхідно використовувати метод явних перетворень, попередньо визначений у C#. Є два варіанти явних перетворень: Cast Operator та клас Convert у C#. Способи перетворення даних наведені на рис. 3.12. Оператор Cast (int) бере основне значення і ігнорує десяткову / дробову частину. Клас перетворення (Convert.ToInt32) округлює число і не ігнорує десяткової частини.

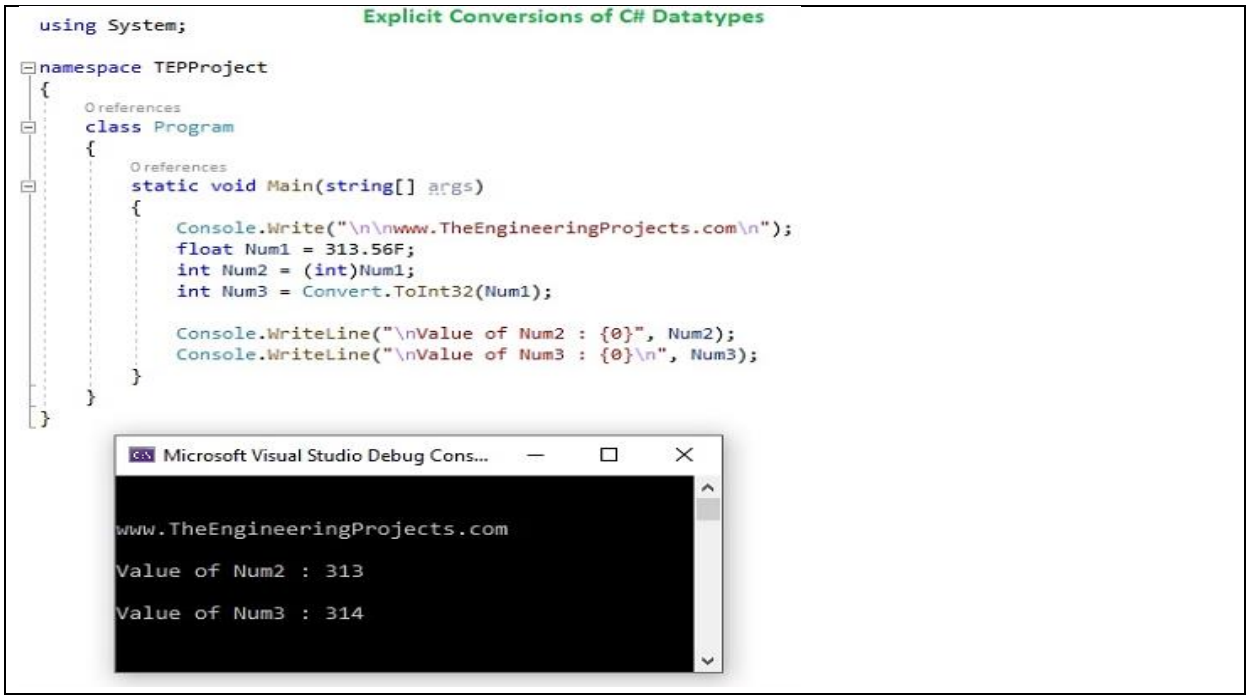


Рис. 3.12. Явне перетворення даних типу Float у Int

Для string-перетворень у C# є два варіанти: Parse, TryParse. Реалізація методу Parse для перетворення рядка у цілочисельне значення наведена на рис. 3.13.

Рис. 3.13. Застосування методу Parse для перетворення String в Int

Метод Parse має недолік: якщо рядок містить букви або спеціальні символи, то це створить помилку. Щоб уникнути помилок, необхідно використовувати метод TryParse (рис. 3.14).

Метод TryParse приймає два параметри: рядок, який перетворюється, та ціле число, в якому значення буде збережено. Метод TryParse також повертає булевий тип, який буде: True, якщо конверсія пройшла успішно, та False, якщо перетворення не вдалося. У прикладі рядок «323ABC» містить букви, тому ConversionCheck помилковий, і не отримуємо int значення у Num2.

TryParse Method for String Conversion

```
using System;

namespace TEPProject
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.Write("\n\nwww.TheEngineeringProjects.com\n");
            string Num1 = "323ABC";
            bool ConversionCheck = int.TryParse(Num1, out int Num2);


            Console.WriteLine("\nValue of Num2 : {0}", Num2);
            Console.WriteLine("\nValue of ConversionCheck : {0}", ConversionCheck);

            if(!ConversionCheck)
            {
                Console.WriteLine("\nString contains non-numeric Values.");
            }
            else
            {
                Console.WriteLine("\nOur Converted int : {0}", Num2);
            }
        }
    }
}
```

```
Microsoft Visual Studio Debug Cons...

www.TheEngineeringProjects.com

Value of Num2 : 0

Value of ConversionCheck : False

String contains non-numeric Values.
```

Рис. 3.14. Застосування методу TryParse для перетворення String в Int

1.4. Як користуватися масивом C#?

Масив має такий вигляд:

```
FirstArray [3] = {Елемент1, Елемент2, Елемент3};
```

Масив має ім'я FirstArray, і він може містити максимум три члени, що відображається в квадратних [] дужках.

Встановлені значення трьох його членів, які розділені комами. Для отримання доступу до окремих членів цього масиву C# необхідно викликати їх за їх індексами. Отже, перший елемент кожного масиву C# завжди є 0:

FirstArray [0] = Елемент1;

FirstArray [1] = Елемент2;

FirstArray [2] = Елемент3;

Створимо простий проєкт C#, у який додамо елементи керування: кнопку «ClickHere» та текстове поле txtClick (рис. 3.15).

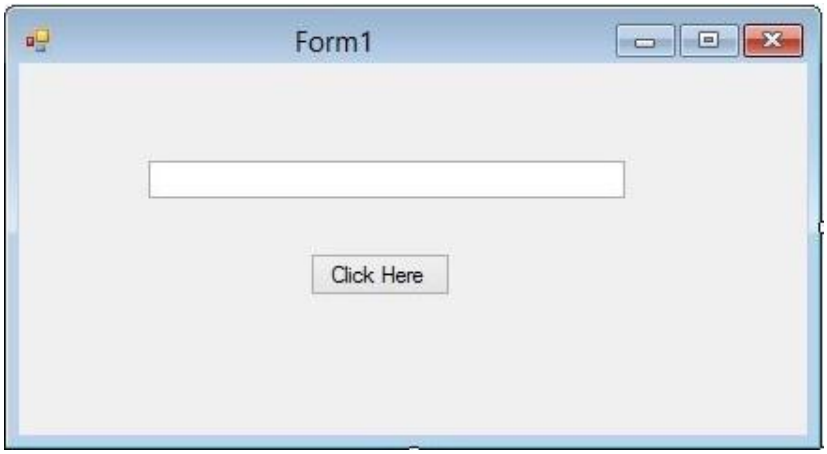


Рис. 3.15. Вікно форми з кнопкою та текстовим полем

Додайте масив рядків C# у проєкт. Спочатку потрібно оголосити масив String C#. Для цього використовується такий код:

```
// C# Array Initializing
String[] students = new String[5];
// Initializing Complete
```

Ім’я масиву C# **students** і використання оператора **new** для створення нового екземпляра масиву C#. Задана довжина масиву – 5 елементів. Можна не задавати довжину масиву, як показано нижче:

```
// C# Array Initializing.
String[] students = new String[];
// Initializing Complete
```

Один із способів додавання значень до масиву C# такий:

```
// Adding values to C# Array.
Students[0] = "Zain";
students[1] = "Nasir";
students[2] = "Kamraan";
students[3] = "John";
students[4] = "Jack";
// Values added.
```

Ще один зі способів призначення значень масиву C#:

```
// Adding values to C# Array.
String[] students = new String[5] {"Zain", "Nasir", "Kamraan", "John", "Jack"};
// Values added.
```

Тепер треба отримати елементи з цього масиву та відобразити їх у текстовому полі. Для цього використовуйте наведений нижче код:

```
txtClick.Text = students[0];
txtClick.Text += " , ";
txtClick.Text += students[1];
txtClick.Text += " , ";
```

```
txtClick.Text += students[2];
txtClick.Text += " , ";
txtClick.Text += students[3];
txtClick.Text += " , ";
txtClick.Text += students[4];
```

Повний код наведений на рис. 3.16.

```
namespace Variables
{
    public partial class Form1 : Form
    {
        String[] students = new String[5] {"Zain", "Nasir", "Kamraan", "John", "Jack"};


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void ClickHere_Click(object sender, EventArgs e) // Button Click Function
        {
            txtClick.Text = students[0];
            txtClick.Text += " , ";
            txtClick.Text += students[1];
            txtClick.Text += " , ";
            txtClick.Text += students[2];
            txtClick.Text += " , ";
            txtClick.Text += students[3];
            txtClick.Text += " , ";
            txtClick.Text += students[4];


        }
    }
}
```

Рис. 3.16. Код програми з використанням елементів масиву

Рис. 3.17. Результати роботи програми, що наведена на рис. 3.16

1.5. Як користуватися елементом ArrayList?

Елемент Arraylist використовується для зберігання в ньому даних так само, як C#-масиви, але між ними є незначна різниця. У C# ArrayList можна будь-коли додавати або видаляти дані, ArrayList налаштовується автоматично.

Додавання або видалення даних у C# ArrayList здійснюється за допомогою індексів цих даних. Отже, коли додаються дані, то ArratList автоматично розтягується і створює вхідні дані в новому індексі. Аналогічно, коли видаляються дані з ArrayList, вони зменшуються і відповідно коригують дані. Для ініціалізації ArrayList використовується код:

```
// …. Ініціалізація C # ArrayList …..
     ArrayList TEP = new ArrayList ();
// …. Закінчується тут …....
```

Створений ArrayList має ім’я TEP. Важливим є те, що необхідно додати до простору імен **using System.Collections.**

Для додавання даних використовується команда **TEP.Add (),** де TEP – це ім’я ArrayList, а дані подають у дужках. Додайте дані до ArrayList, використовуючи такий код:

```
// …. Adding Data in ArrayList ….
     TEP.Add("The");
     TEP.Add("Engineering");
     TEP.Add("Projects");
// …. Data added in ArrayList ….
```

Додано три значення у TEP ArrayList. Виведіть ці значення у поле txtClick за подією Click кнопки «ClickHere». До функції Click додайте код:

```
// … Displaying values …..
     txtClick.Text = TEP[0].ToString();
     txtClick.Text += " , ";
     txtClick.Text += TEP[1].ToString();
     txtClick.Text += " , ";
     txtClick.Text += TEP[2].ToString();
// … Values Displayed …..
```

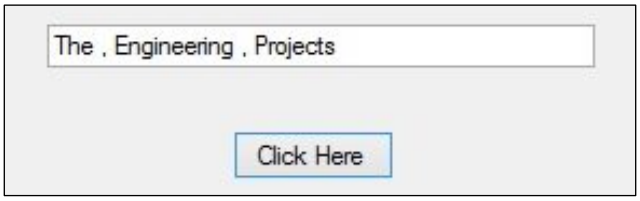
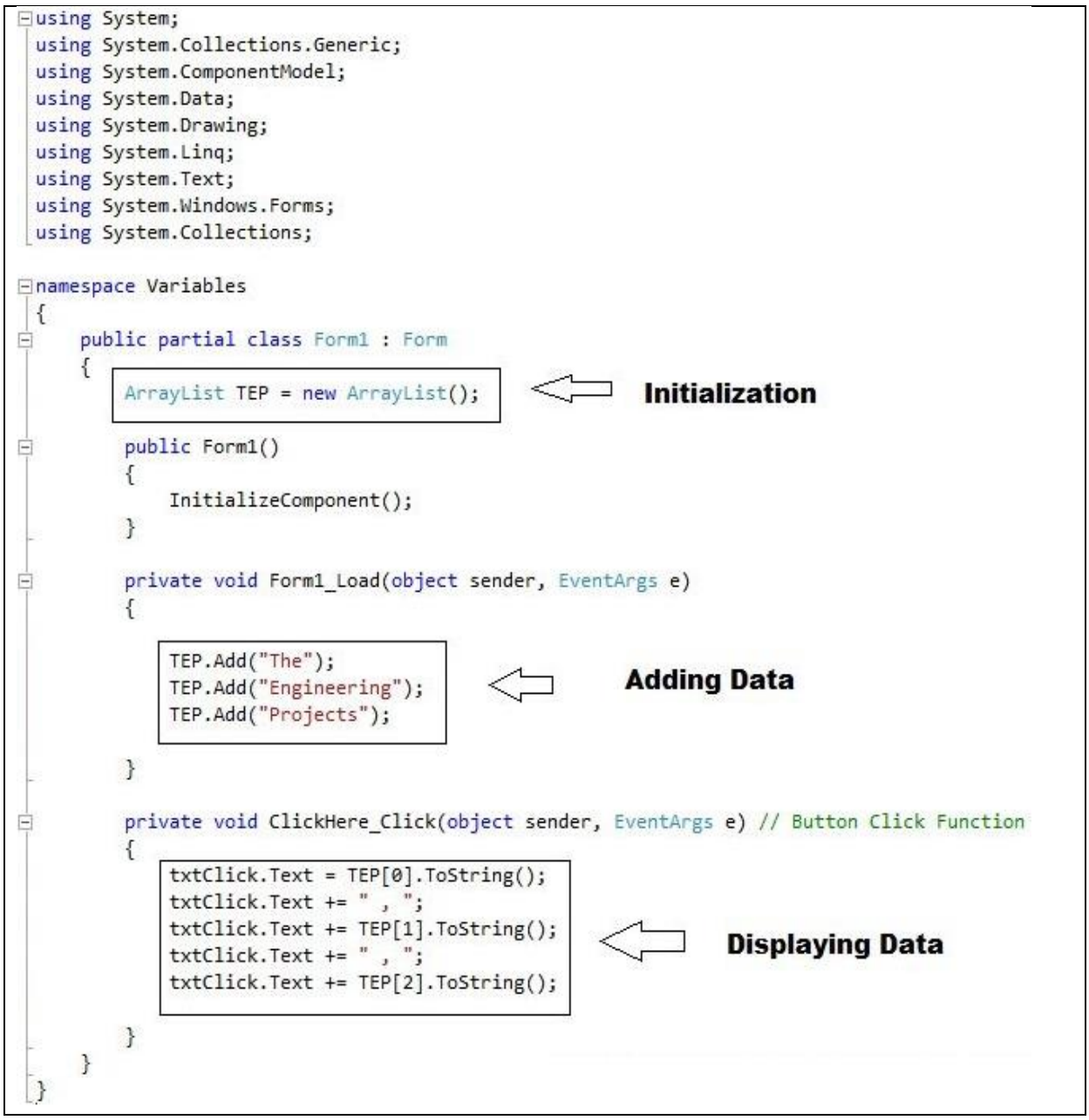


Рис. 3.18. Результат виконання коду програми для демонстрації роботи з ArrayList




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;

namespace Variables
{
    public partial class Form1 : Form
    {
        ArrayList TEP = new ArrayList();

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            TEP.Add("The");
            TEP.Add("Engineering");
            TEP.Add("Projects");
        }

        private void ClickHere_Click(object sender, EventArgs e) // Button Click Function
        {
            txtClick.Text = TEP[0].ToString();
            txtClick.Text += " , ";
            txtClick.Text += TEP[1].ToString();
            txtClick.Text += " , ";
            txtClick.Text += TEP[2].ToString();
        }
    }
}
```

Рис. 3.19. Код програми для демонстрації роботи з ArrayList

Розгляньте, як підрахувати загальну кількість елементів у ArrayList. Для цього вам потрібно додати TEP.Count до події Click кнопки «ClickHere», як показано нижче:

```
// …. Display Values …..
      txtClick.Text = TEP[0].ToString();
      txtClick.Text += " , ";
      txtClick.Text += TEP[1].ToString();
      txtClick.Text += " , ";
      txtClick.Text += TEP[2].ToString();
      txtClick.Text += " , ";
      txtClick.Text += TEP.Count;
// ….. Values Displayed …..
```

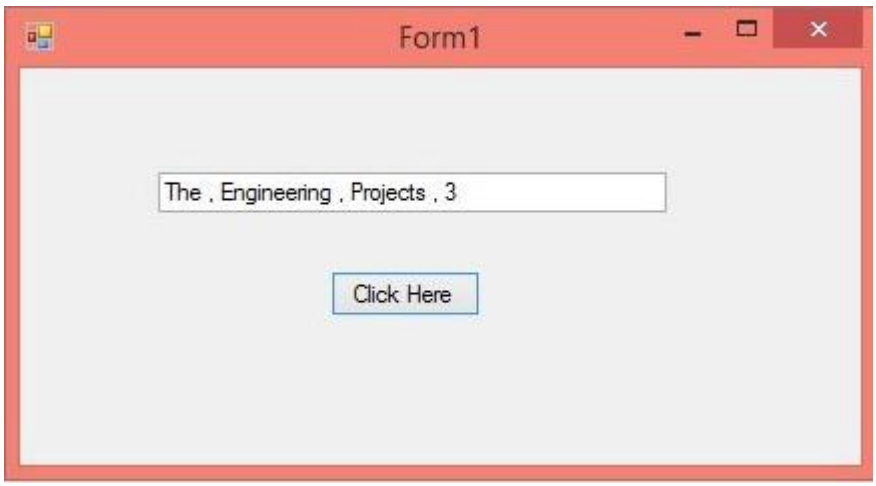


Рис. 3.20. Результат виконання коду програми з підрахунком загальної кількості елементів у ArrayList

Для очищення ArrayList використовується команда TEP.Clear(). У наведеному нижче коді відображені значення ArrayList та виводиться загальна кількість елементів ArrayList; далі очищається ArrayList командою TEP.Clear() та знову виводиться загальна кількість елементів. Оскільки ArrayList очищений, то в ньому немає елементів і буде виведено значення 0 (рис. 3.21):

```
txtClick.Text = TEP[0].ToString();
      txtClick.Text += " , ";
      txtClick.Text += TEP[1].ToString();
      txtClick.Text += " , ";
      txtClick.Text += TEP[2].ToString();
      txtClick.Text += " , ";
      txtClick.Text += TEP.Count;
      TEP.Clear();
      txtClick.Text += " , ";
      txtClick.Text += TEP.Count;
```

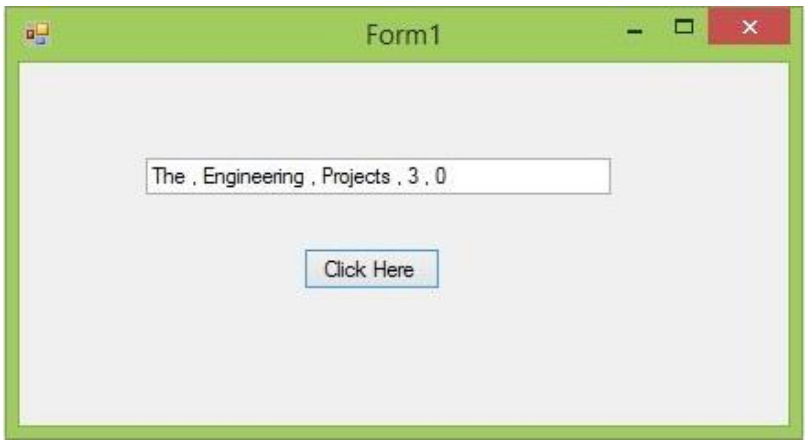


Рис. 3.21. Демонстрація роботи команди TEP.Count

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв'язання задачі.

Два потяги виїхали одночасно назустріч один одному. Знайдіть, через який час вони зустрінуться, якщо задано значення відстані між ними в момент початку руху та швидкості руху кожного потяга (рис. 3.22).

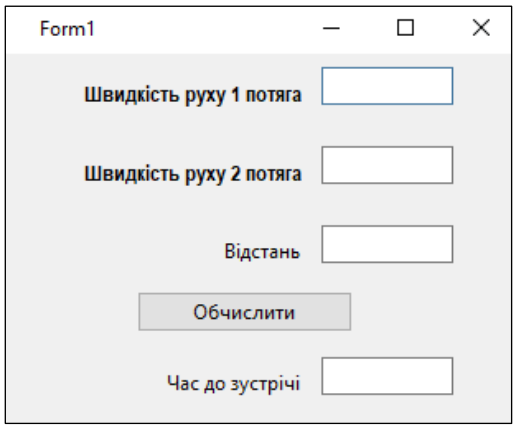


Рис. 3.22. Вікно форми для завдання 1

Завдання 2. Скласти програму для знаходження значення функції за різних значень x.

1. $y = \sqrt{|x-1| + \sin x}$;
2. $y = 1 + \frac{1}{x} + \frac{1}{x^2}$.

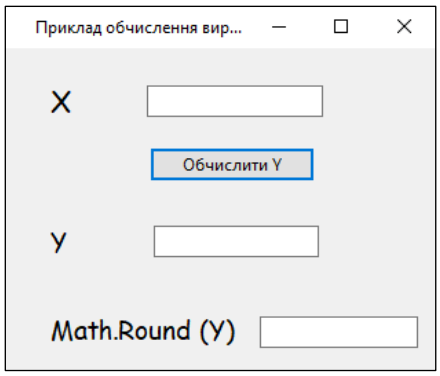


Рис. 3.23. Вікно форми для завдання 2

Завдання 3. Скласти програму для визначення найбільшого з трьох чисел A, B, C. Значення змінних A, B і C мають бути впорядковані за зростанням (поміняти місцями A, B і C так, щоб виявилося $A \le B \le C$).

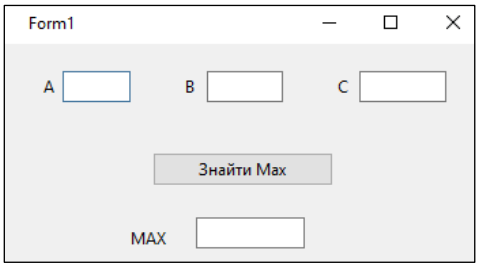


Рис. 3.24. Вікно форми для завдання 3

Завдання 4. Відповідно свого варіанта вибрати індивідуальне завдання.

Встановити необхідну кількість елементів TextBox, тексти заголовків на формі, розміри шрифтів, а також типи змінних та функції перетворення під час введення і виведення результатів. Для перевірки правильності роботи програми навести контрольний приклад.

Індивідуальні завдання

1. t = (2*cos(x – π/6) / (0.5 + sin^2(y))) * (1 + z^2 / (3 – z^2/5))
При x = 14.26, y = –1.22, z = 3.5 × 10 ^ – 2 t = 0.564849

2. u = (((8 + |x – y|^2)^(1/3) + 1) / (x^2 + y^2 + 2)) – e^|x – y| * ((tg^2(z) + + 1)^x)
При x = –4.5, y = 0.75 × 10 ^ – 4, z = 0.845 × 10 ^ 2 u = –55.6848

3. v = ((1 + sin^2(x + y)) / (x – (2y / (1 + x^2*y^2)))) * x^|y| + cos^2 (arctg(1/z))
При x = 3.74 × 10 ^ – 2, y = –0.825, z = 0.16 × 10^2 v = 1.0553

4. w = |cos(x) – cos(y)|^(1 + 2*sin^2(y)) * (1 + z + z^2/2 + z^3/3 + z^4/4)
При x = 0.4 × 10^4, y = –0.875, z = –0.475 × 10^ – 3 w = 1.9873

5. α = ln(y^√|x|) * (x – y/2) + sin^2(arctg(z))
При x = –15.246, y = 4.642 × 10^ – 2, z = 20.001 × 10^2 α = –182.036

6. β = √(10*((∛x + x^(y^2)))) * (arcsin^2(z – |x – y|))
При x = 16.55 × 10^ – 3, y = –2.75, z = 0.15 β = –38.902

7. γ = 5*arctg(x) – (1/4)*arccos(x) * ((x + 3 |x – y| + x^2) / (|x – y |z + x^2))
При x = 0.1722, y = 6.33, z = 3.25 × 10^ – 4 γ = –172.025

8. φ = (e^(|x – y|) * |x – y| ^ (x + y) / (arctg(x) + arctg(z))) + ∛(x^6 + ln^2(y))
При x = –2.235 × 10 ^ – 2, y = 2.23, z = 15.221 φ = 39.374

9. ψ = (y / (x ^ x) – 3√(y / x)) + (y–x) * (cos(y) – z/(y – x)) / (1 + (y – x)^2)
При x = 1.825 × 10^ 2, y = 18.225, z = –3.298 × 10 ^ – 2 ψ = 1.2131

10. a = 2^(–x) * (x + ⁴√|y|) * ∛(e^(x – 1/sin(z)))
При x = 3.981 × 10 ^ – 2, y = –1.625 × 10 ^ 3, z = 0.512 a = 1.26185

11. b = y^(√h) + cos^3(y) * ((|x – y| * (1 + sin^2(z)/√(x + y))) / (e^(|x – y|) + x/2))
При x = 6.251, y = 0.827, z = 25.001 b = 0.7121

12. c = (2^(y^x) + (3^x)^y) * (y * arctg(z – π/6)) / (|x| + (1/(y ^ 2 + 1)))
При x = 3.251, y = 0.325, z = 0.466 × 10^ – 4 c = 4.025

13. f = (⁴√y + ∛(x – 1)) / (|x – y|(sin^2(z) + tg(z)))
При x = 17.421, y = 10.365 × 10^ – 3, z = 0.828 × 10^5 f = 0.33056

14. g = (y^(x + 1)/∛|y – 2|+3) + (x + y/2)/(2|x + y|) * (x + 1)^(–1/sin(z))
При x = 12.3 × 10^ – 1, y = 15.4, z = 0.252 × 10^3 g = 82.8257

15. h = (x^(y + 1) + e^(y – 1))/(1 + x) |y – tg(z)| * (1 + |y – x|) + (|y – x|^2)/2 –
– (|y – x|^3)/3
При x = 2.444, y = 0.869 × 10^ – 2, z = –0.13 × 10^3 h = –0.49871

16. y = √(cx – 2.7|c| + |x|)/(c^2 x^2) * e^x + cos((a + b)^2 / (cx – b))
a = 3.7; b = 0.07; c = 1.5; x = 5.75

3. Контрольні запитання

1. Для чого використовується елемент TextBox?

2. Як ввести та вивести дані через TextBox?

3. Наведіть властивості елемента TextBox. Як їх можна попередньо встановити та змінити під час роботи програми?

4. Як задати максимальну довжини тексту та задати багаторядкове текстове поле?

5. Які існують методи перетворення типів числових даних між собою?

6. Які існують методи перетворення рядкових даних у числові та навпаки?

7. Як порівняти між собою рядкові дані, як здійснювати пошук фрагмента тексту у рядку?

8. Як оголосити масив рядків та задати йому значення?

9. Які переваги та особливості застосування елементу Arraylist?

ЛАБОРАТОРНА РОБОТА № 4
WINDOWS FORMS. ЕЛЕМЕНТ LISTBOX

Мета заняття: навчитися працювати з елементом ListBox у середовищі Microsoft Visual Studio; набути практику додавання, видалення даних, зміни властивостей, їх обробки та виводу результатів за допомогою Windows Forms мовою C#; навчитись використовувати події (events) під час роботи з елементом ListBox.

1. Теоретичні відомості

1.1. Додавання елементів до ListBox

Елемент ListBox використовується у формах реєстрації та додатках із продажу товару. Для роботи з елементом ListBox потрібно з панелі інструментів перетягнути його на форму Form1. Для встановлення значень всередині Listbox потрібно написати назву поля списку, а після цього написати **Items.Add («YourData»),** наприклад (рис. 4.1):

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("Pakistan");
            listBox1.Items.Add("United States");
            listBox1.Items.Add("United Kingdom");
            listBox1.Items.Add("India");
        }
   }
}
```

Рис. 4.1. Додавання елементів до ListBox

Можна додати значення у ListBox за допомогою масивів та змінних типу string. Приклад коду, який демонструє використання для вставки значень елементу ListBox у C# рядкових змінних:

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            String var1 = "Lahore";
            String var2 = "Multan";
            String var3 = "Islamabad";
            String var4 = "Karachi";
            listBox1.Items.Add(var1);
            listBox1.Items.Add(var2);
            listBox1.Items.Add(var3);
            listBox1.Items.Add(var4);
        }

   }
}
```

Приклад коду, який демонструє використання для вставки значень елементу ListBox у C# масиву:

```
using System;
using System.Windows.Forms;

namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            String[] city = new String[] {"Lahore","Multan", "Islamabad",
"Karachi"};

            for (int x = 0; x < city.Length; x++)
            {
                listBox1.Items.Add(city[x]);
            }
        }
   }
}
```

Приклад коду, який демонструє читання даних із TextBox та їх копіювання до елементу ListBox у C# (рис. 4.2):

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            listBox1.Items.Add(textBox1.Text);
        }

   }
}
```

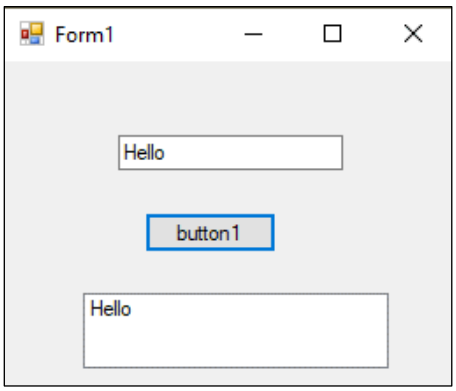


Рис. 4.2. Читання даних із TextBox та їх копіювання до елементу ListBox

Розглянемо, як вибрати будь-який елемент із ListBox. Для отримання вибраного значення елементу використовується властивість SelectedItem.

У коді, що наведений нижче, після вибору значення з ListBox і натисненням кнопки, виникає подія, що створює спливаюче повідомлення про вибране значення елементу:

```
Using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("A");
            listBox1.Items.Add("B");
```

```
            listBox1.Items.Add("C");
            listBox1.Items.Add("D");
        }
        private void button1_Click(object sender, EventArgs e)
        {

            MessageBox.Show(listBox1.SelectedItem.ToString());
        }
    }
}
```

1.2. Вибір кількох елементів ListBox

Для вибору кількох варіантів одночасно необхідно у властивості **Selection Mode** встановити значення **SelectionMode.MultiSimple.**

У прикладі коду (рис. 4.3) використовується властивість SelectionMode. MultiSimple та цикл foreach, щоб показати спливаюче повідомлення зі значенням вибраних елементів.

```
namespace LIST_BOX
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            listBox1.Items.Add("Sunday");
            listBox1.Items.Add("Monday");
            listBox1.Items.Add("Tuesday");
            listBox1.Items.Add("Wednesday");
            listBox1.Items.Add("Thursday");
            listBox1.Items.Add("Friday");
            listBox1.Items.Add("Saturday");
            listBox1.SelectionMode = SelectionMode.MultiSimple;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            //listBox1.Items.Add(textBox1.Text);
            //MessageBox.Show(listBox1.SelectedItem.ToString());
            foreach (Object obj in listBox1.SelectedItems)
            {
                MessageBox.Show(obj.ToString());
            }
        }
    }
}
```

Рис. 4.3. Демонстраційний код використання властивості SelectionMode

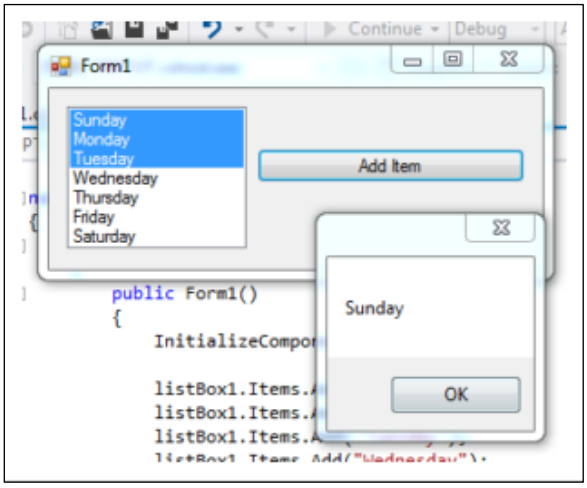


Рис. 4.4. Демонстрація роботи форми з вибору кількох елементів ListBox

З рис. 4.4 видно, що вибрані перші три індекси, і під час натискання кнопки генерується спливаюче повідомлення лише з Sunday.

Оскільки у програмі (рис. 4.3) використаний цикл foreach, то кожна ітерація циклу містить лише один вивід. Коли буде закрито спливаюче вікно, у полі повідомлень відображатиметься наступне вибране значення індексу. Цикл працює, поки не залишиться вибраних індексів.

1.3. Очищення та видалення значень ListBox

Для видалення всіх значень зі списку використовується метод Items.Clear.

У коді, що наведений нижче, встановлюються значення у Listbox, і коли користувач натисне на кнопку, очищується весь список:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("A");
            listBox1.Items.Add("B");
            listBox1.Items.Add("C");
            listBox1.Items.Add("D");
        }
        private void button1_Click(object sender, EventArgs e)
        {
            listBox1.Items.Clear();
        }
    }
}
```

Для видалення певного індексу або значення елементу використовують методи Remove або RemoveAt. Метод RemoveAt використовується для видалення значення за індексом. Демонстраційний код для методу RemoveAt:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("A");
            listBox1.Items.Add("B");
            listBox1.Items.Add("C");
            listBox1.Items.Add("D");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            listBox1.Items.RemoveAt(0);
        }
   }
}
```

У коді користувач натискає кнопку, нульовий індекс видаляється зі списку. Коли нульовий індекс видаляється, наступний індекс автоматично стає нульовим. Наступний код демонструє реалізацією методу Remove, який використовується для видалення значення за точним формулюванням:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("A");
            listBox1.Items.Add("B");
            listBox1.Items.Add("C");
            listBox1.Items.Add("D");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            listBox1.Items.Remove("A");
        }
   }
}
```

У наведеному коді видаляється лише точне відповідне значення «**A**» за натисненням кнопки. Під час наступного натискання кнопки не видалиться жоден індекс.

1.4. Зміна властивостей тексту ListBox

Для зміни розміру тексту у ListBox використовується властивість **Font.** У демонстраційному коді після натиснення кнопки користувачем збільшиться розмір тексту C# ListBox (рис. 4.5).

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("A");
            listBox1.Items.Add("B");
            listBox1.Items.Add("C");
            listBox1.Items.Add("D");
        }
        private void button1_Click(object sender, EventArgs e)
        {
    listBox1.Font = new Font(Font.FontFamily, 12);
        }
   }
}
```

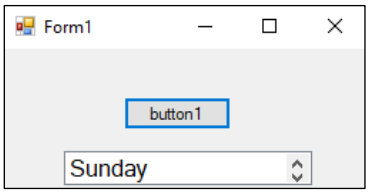


Рис. 4.5. Демонстрація використання властивості Font елементу ListBox

Для зміни кольору тексту або кольору фону тексту використовується властивість BackColor. У демонстраційному коді змінюється колір тексту списку поля:

```
Using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
            InitializeComponent();
            listBox1.Items.Add("A");
            listBox1.Items.Add("B");
            listBox1.Items.Add("C");
            listBox1.Items.Add("D");
            listBox1.BackColor = Color.LightCyan;
        }
    }
}
```

Для зміни кольору тексту або кольору переднього плану використовують властивість ForeColor. У демонстраційному коді змінюється колір тексту на білий, а backColor – на чорний (рис. 4.6):

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("The");
            listBox1.Items.Add("Engineering");
            listBox1.Items.Add("Projects");
            listBox1.Font = new Font(Font.FontFamily, 15);
            listBox1.BackColor = Color.Black;
            listBox1.ForeColor = Color.White;
        }
    }
}
```

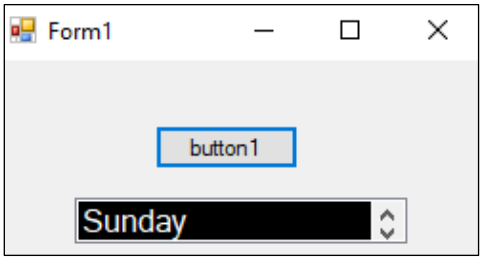


Рис. 4.6. Демонстрація роботи властивостей ForeColor та BackColor

1.5. Події ListBox у C#

Елемент ListBox може використовувати такі події (Events): Click, Double Click, ForeColorChanged, BackColorChanged, SelectedIndexChanged, MouseHover, MouseLeave.

Подія ListBox Click використовується, коли потрібно виконати будь-яку дію у відповідь одним клацанням на «Listbox», наприклад, показати будь-яке повідомлення – попередження у вигляді спливаючого вікна (рис. 4.7):

```
   private void listBox1_Click(object sender, EventArgs e)
     {
        MessageBox.Show("Warn ! Click is disable..");
     }
```

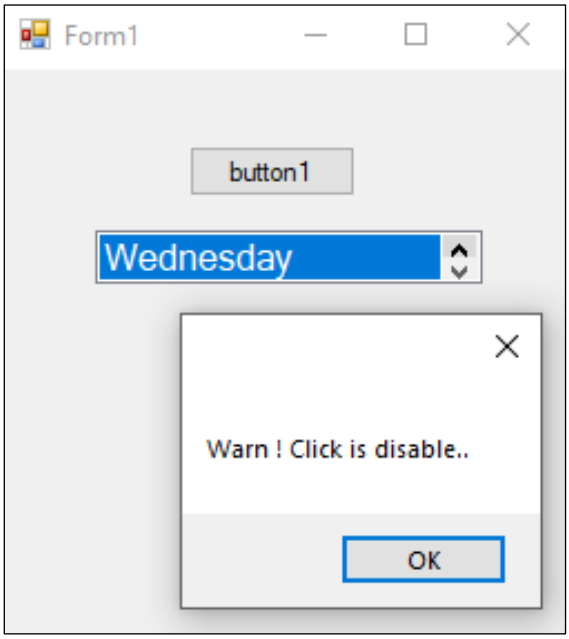


Рис. 4.7. Демонстрація реакції на подію ListBox Click

Ця подія DoubleClick відбувається, коли користувач два рази натисне на «Listbox». У демонстраційному коді змінюється колір списку «ListBox», коли користувач натисне двічі на елемент ListBox:

```
     Private void listBox1_DoubleClick(object sender, EventArgs e)
        {
            listBox1.BackColor = Color.Red;
        }
```

Подія MouseHover відбувається, коли користувач наведе курсор миші на «Listbox». У демонстраційному коді змінюється властивість foreColor під час наведення курсору миші:

```
private void listBox1_MouseHover(object sender, EventArgs e)
  {
     listBox1.ForeColor = Color.Bisque;
  }
```

Подія MouseLeave відбувається, коли користувач відведе курсор від списку. У демонстраційному коді змінюється фоновий колір на чорний, коли користувач відведе курсор миші:

```
   Private void listBox1_MouseHover(object sender, EventArgs e)
        {
            listBox1.ForeColor = Color.Bisque;
        }
   private void listBox1_MouseLeave(object sender, EventArgs e)
```

```
        {
            listBox1.ForeColor = Color.Black;
        }
```

Подія BackColorChanged відбувається, коли користувач змінить колір фону списку. У демонстраційному коді буде змінюватись колір фону, коли користувач наведе мишу на елемент «ListBox». Коли колір зміниться, відбудеться подія BackColorChanged, у якій буде показано спливаюче повідомлення:

```
Private void listBox1_MouseHover(object sender, EventArgs e)
        {
            listBox1.BackColor = Color.Bisque;
        }
private void listBox1_BackColorChanged(object sender, EventArgs e)
        {
            MessageBox.Show("BackGround Color is changed !!");
        }
```

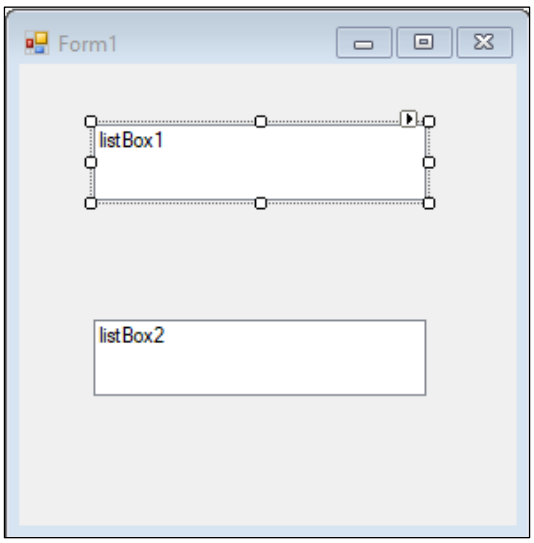
Подія ForeColorChanged відбувається, коли користувач змінить колір переднього плану списку. У демонстраційному коді використовується подія наведення миші, щоб змінити foreColor та викликати подію ForeColorChanged. Подія ForeColorChanged викликає спливаюче повідомлення в якому позначено, що колір переднього плану змінений:

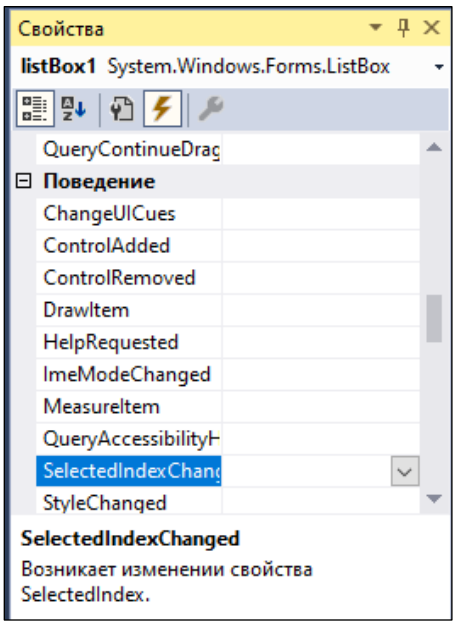
```
Private void listBox1_MouseHover(object sender, EventArgs e)
        {
            listBox1.ForeColor = Color.Bisque;
        }

private void listBox1_ForeColorChanged(object sender, EventArgs e)
        {
            MessageBox.Show("ForeGround Color is changed !!");
        }
```

Подія SelectedIndexChanged відбувається, коли користувач вибере будь-який елемент у списку. У демонстраційному коді використовуються два списки (рис. 4.8). Коли користувач вибере будь-яке значення з першого списку, відповідно це значення відобразиться у другому списку (рис. 4.9).

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("The Engineering Projects");
            listBox1.Items.Add("C# Tutorials");
        }
        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
```

```
            if (listBox1.SelectedIndex == 0)
            {
                listBox2.Items.Clear();
                listBox2.Items.Add("Arduino Projects");
                listBox2.Items.Add("Proteus Projects");
                listBox2.Items.Add("Matlab Projects");
            }
            else
                if (listBox1.SelectedIndex == 1)
                {
                    listBox2.Items.Clear();
                    listBox2.Items.Add("C# Label");
                    listBox2.Items.Add("C# TextBox");
                    listBox2.Items.Add("C# ComboBox");
                }
        }
       }
}
```





```
namespace LIST_BOX2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add("The Engineering Projects");
            listBox1.Items.Add("C# Tutorials");

        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (listBox1.SelectedIndex == 0)
            {
                listBox2.Items.Clear();
                listBox2.Items.Add("Arduino Projects");
                listBox2.Items.Add("Proteus Projects");
                listBox2.Items.Add("Matlab Projects");
            }
            else
                if (listBox1.SelectedIndex == 1)
            {
                listBox2.Items.Clear();
                listBox2.Items.Add("C# Label");
                listBox2.Items.Add("C# TextBox");
                listBox2.Items.Add("C# ComboBox");
            }

        }
    }
}
```

Рис. 4.8. Форма та код програми для демонстрації події SelectedIndexChanged

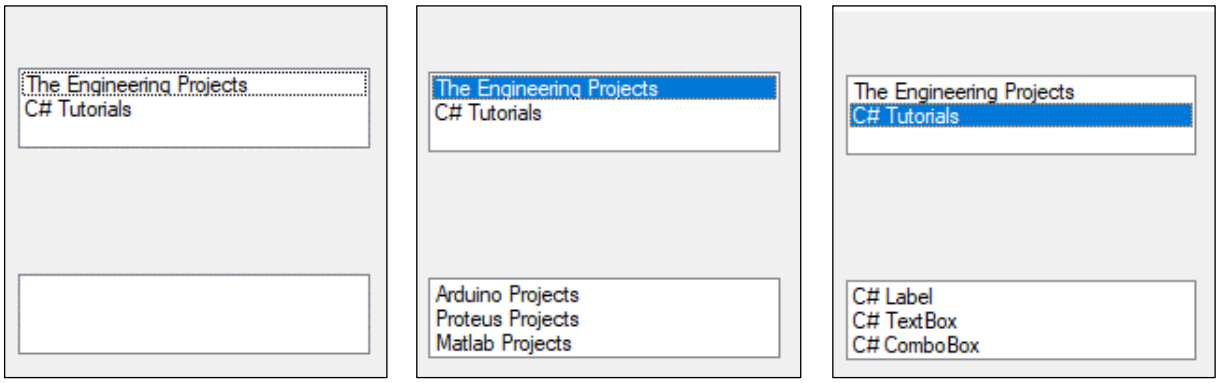


Рис. 4.9. Демонстрація роботи програми, що використовує подію SelectedIndexChanged

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв'язання задачі.

Є послідовність із N цілих чисел (де N ≤ 100). Потрібно визначити, скільки серед них є парних елементів із парними індексами.

1. Створіть новий проєкт. Розмістіть на формі елементи Label1, Label2, Button1 (Name – bT1), Button2 (Name – bT2), ListBox (Name – listB1), TexBox1 (Name – texB1) TexBox2 (Name – texB2) і налаштуйте їх властивості згідно з рис. 4.10.

2. Для зберігання числових даних використовується **масив.** Оскільки максимальна кількість елементів дорівнює 100, треба описати масив відповідного розміру: int[] Num_list = new int[100]; опишіть глобальні змінні: Random rand = = new Random(); N – кількість елементів послідовності; K – кількість парних чисел, що мають парні індекси; i – індекс поточного елементу масиву.

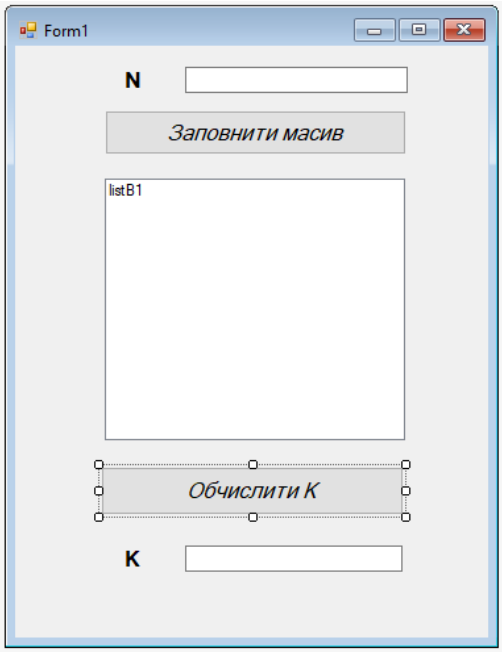


Рис. 4.10. Зовнішній вигляд форми для реалізації завдання 1

3. Створіть процедуру обробки події Click для кнопки «Заповнити масив». У ній необхідно: ввести значення N, згенерувати випадкові числа та додати їх у список ListBox.

```
int N = Convert.ToInt32(texB1.Text);
for (int i=0; i < N; i++)
{
Num_list[i] = rand.Next(100);
listB1.Items.Add(Convert.ToString(Num_list[i]));
}
```

4. Для кнопки «Обчислити K» потрібно створити процедуру обробки події OnClick. У циклі послідовно перевіряються всі елементи ListBox: якщо listB1. Items[i] відповідає умові (число парне та індекс парний), то змінна K збільшується на 1:

```
int k = 0;
            for (int i = 0; i < listB1.Items.Count; i++)
            {
                int temp = Convert.ToInt32(listB1.Items[i]);
                if ((temp % 2 == 0) && (i % 2 == 0))
                {
                       k = k + 1;
                }
```

Перевірте роботу програми для різних значень N. Поясніть результати роботи програми, наведені на рис. 4.11.

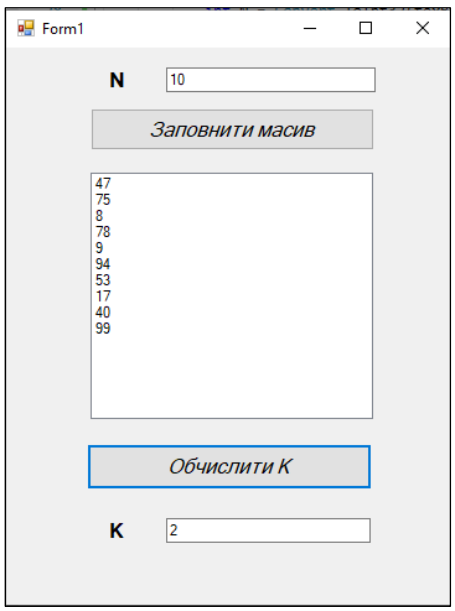


Рис. 4.11. Результат роботи програми до завдання 1

Завдання 2. Створити проєкт для розв'язання задачі.

Дано послідовність із N цілих чисел ($N \leq 100$). Визначити кількість непарних чисел, які мають непарні індекси.

```
namespace Lab4
{
    public partial class Form1 : Form
    {
        int[] Num_list = new int[100];
        Random rand = new Random();

        public Form1()
        {
            InitializeComponent();
            listB1.Items.Clear();
        }

        private void bT1_Click(object sender, EventArgs e)
        {

            int N = Convert.ToInt32(texB1.Text);
            for (int i=0; i < N; i++)
            {
                Num_list[i] = rand.Next(100);
                listB1.Items.Add(Convert.ToString(Num_list[i]));
            }
        }

        private void bT2_Click(object sender, EventArgs e)
        {
            int k = 0;
            for (int i = 0; i < listB1.Items.Count; i++)
            {
                int temp = Convert.ToInt32(listB1.Items[i]);
                if ((temp % 2 == 0) && (i % 2 == 0))
                {
                      k = k + 1;
                }
             }

            textB2.Text = Convert.ToString(k);

        }
    }
}
```

Рис. 4.12. Код програми для реалізації завдання 1

Завдання 3. Створити проєкт для розв’язання задачі.

Дано послідовність із N цілих чисел ($N \leq 100$). Визначити кількість непарних чисел, які мають парні індекси.

3. Контрольні запитання

1. Для чого використовується елемент ListBox?
2. Як ввести та вивести дані через ListBox?
3. Наведіть властивості елемента ListBox. Як їх можна попередньо встановити та змінити під час роботи програми?
4. Як додавати, видаляти, очищати елементи ListBox?
5. Як визначити кількість елементів в ListBox?
6. Які існують події під час роботи з ListBox? Як їх налаштувати?

ЛАБОРАТОРНА РОБОТА № 5
WINDOWS FORMS. ЕЛЕМЕНТ CHECKBOX

Мета заняття: навчитися працювати з елементом RadioButton, CheckBox у середовищі Microsoft Visual Studio; навчитися використовувати події (events) під час роботи з елементом RadioButton, CheckBox за допомогою Windows Forms мовою C#.

1. Теоретичні відомості

1.1. Елемент RadioButton

C# RadioButton використовують як перемикач для отримання конкретних вхідних значень. RadioButton дає змогу користувачу одночасно перевіряти лише одну опцію, а інші параметри залишаються неперевіреними або невибраними. Для додавання RadioButton необхідно знайти RadioButton у ToolBox і перетягнути його до своєї програми. На рис. 5.1 додано 8 RadioButtons та вибрано лише одну, тому що не можна вибрати більше однієї RadioButton одночасно.

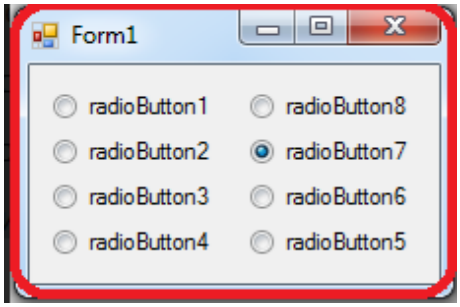


Рис. 5.1. Робота з RadioButtons у Windows Forms

Назва перемикачів – це імена за замовчуванням. Для зміни їх назви є два варіанти. Перший варіант – змінити властивість Text у розділі властивості (рис. 5.2).

Рис. 5.2. Налаштування властивості RadioButton.Text

Другий спосіб зміни назви радіокнопки – це програмний. У вихідному коді потрібно встановити ім’я перемикача за допомогою властивості Text перемикача. У коді, що наведений нижче, показано, як встановити значення властивості тексту перемикача.

```
Using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEP
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            radioButton5.Text = "TEP Code RB5";
            radioButton6.Text = "TEP Code RB6";
            radioButton7.Text = "TEP Code RB7";
            radioButton8.Text = "TEP Code RB8";
        }
    }
}
```

Якщо потрібно змінити стан RadioButton, то це дозволяє властивість Checked. За замовчуванням кожна RadioButton не встановлена, тому можна змінити значення властивості Checked на true, як показано у коді нижче:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEP
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            radioButton1.Checked = true;
        }
    }
}
```

Властивість BackColor дає змогу встановити колір фону. Існують різновиди кольорів, які можна використовувати для встановлення кольору фону.

У коді, що наведений нижче, використовуються різні кольори для кожної кнопки (рис. 5.3).

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace strnull
{
    public partial class Form1 : Form
    {
```

```
        public Form1()
        {
            InitializeComponent();
            radioButton1.BackColor = Color.AliceBlue;
            radioButton2.BackColor = Color.AntiqueWhite;
            radioButton3.BackColor = Color.Aqua;
            radioButton4.BackColor = Color.Aquamarine;
            radioButton5.BackColor = Color.Azure;
            radioButton6.BackColor = Color.Beige;
            radioButton7.BackColor = Color.Bisque;
            radioButton8.BackColor = Color.BlueViolet;
        }
    }
}
```

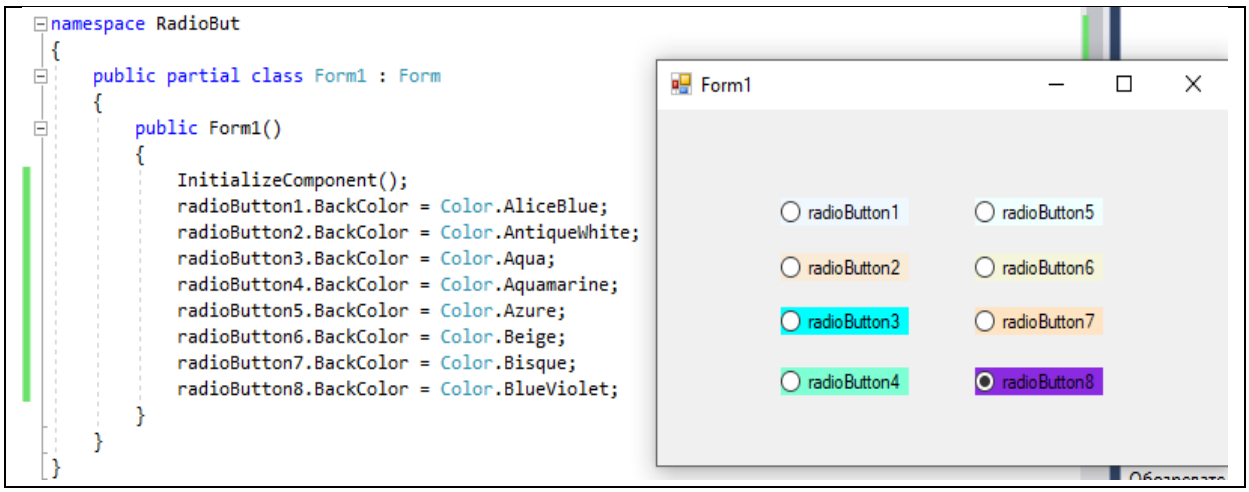


Рис. 5.3. Демонстраційний приклад роботи з властивістю BackColor елементу RadioButton

Властивість ForeColor дає змогу змінити колір тексту (рис. 5.4). Властивість Font дає змогу змінити розмір шрифту RadioButton (рис. 5.5). Щоб встановити розмір необхідно передати прототип сімейства шрифтів та розмір шрифту як параметр у конструкторі шрифтів, як показано нижче:

```
Using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEP
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            radioButton1.Font = new Font(Font.FontFamily,10);
            radioButton2.Font = new Font(Font.FontFamily, 11);
            radioButton3.Font = new Font(Font.FontFamily, 12);
            radioButton4.Font = new Font(Font.FontFamily, 13);
            radioButton5.Font = new Font(Font.FontFamily, 12);
```

```
            radioButton6.Font = new Font(Font.FontFamily, 11);
            radioButton7.Font = new Font(Font.FontFamily, 10);
            radioButton8.Font = new Font(Font.FontFamily, 9);
        }
    }
}
```

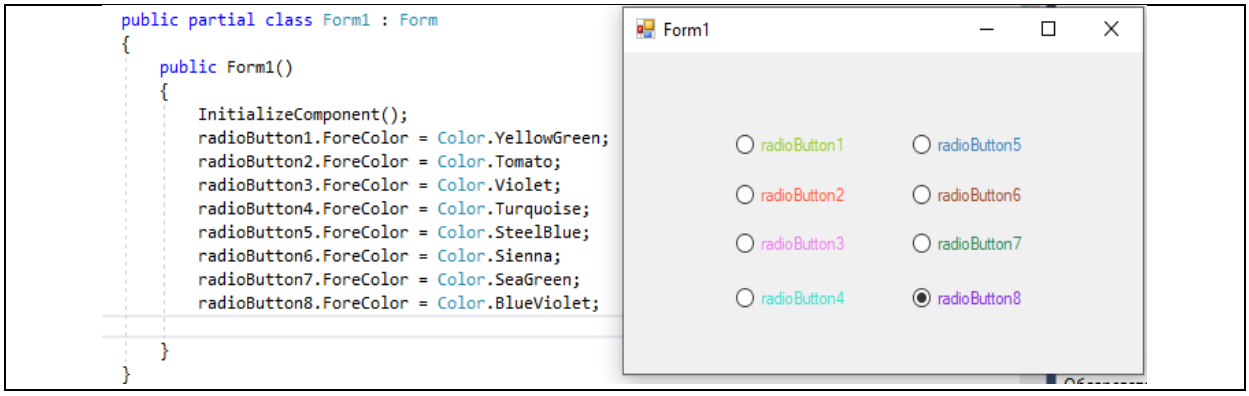


Рис. 5.4. Демонстраційний приклад роботи з властивістю ForeColor елемента RadioButton

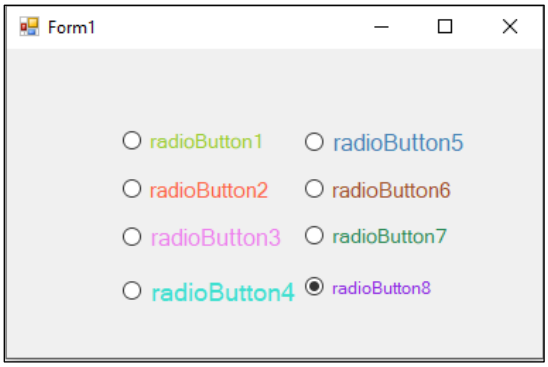


Рис. 5.5. Демонстрація зміни розміру шрифту RadioButton (властивість Font)

Для зміни типу шрифту використовують властивість Font, у якій потрібно задати ім’я шрифту та розмір тексту (рис. 5.6).

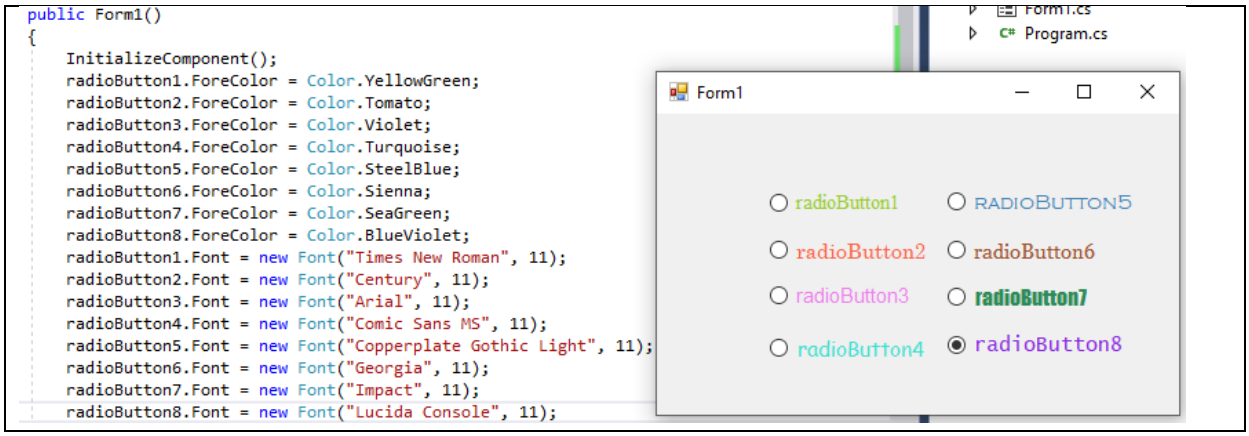


Рис. 5.6. Демонстрація зміни імені шрифту та розміру тексту RadioButton (властивість Font)

Властивість Checked використовується для визначення стану radioButton. Умовний оператор *if* використовується, щоб створити логіку роботи перемикача. У демонстраційному коді кожна властивість RadioButton містить логічне твердження, яке повертає значення у вікні повідомлення, якщо воно встановлено (рис. 5.7).

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace TEP
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (radioButton1.Checked == true)
            {
                MessageBox.Show(radioButton1.Text);
            }
            else if (radioButton2.Checked == true)
            {
                MessageBox.Show(radioButton2.Text);
            }
            else if (radioButton3.Checked == true)
            {
                MessageBox.Show(radioButton3.Text);
            }
            else if (radioButton4.Checked == true)
            {
                MessageBox.Show(radioButton4.Text);
            }
            else if (radioButton5.Checked == true)
            {
                MessageBox.Show(radioButton5.Text);
            }
            else if (radioButton6.Checked == true)
            {
                MessageBox.Show(radioButton6.Text);
            }
            else if (radioButton7.Checked == true)
            {
                MessageBox.Show(radioButton7.Text);
            }
            else if (radioButton8.Checked == true)
            {
                MessageBox.Show(radioButton8.Text);
            }
        }
    }
}
```

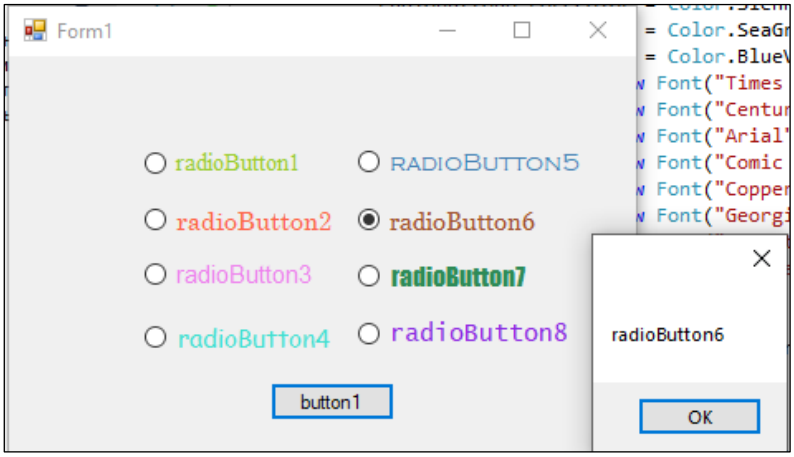


Рис. 5.7. Демонстрація застосування властивості Checked

Властивість Image використовується для встановлення зображення як фону для перемикача. У наступному коді показаний приклад, як встановити зображення як фон:

```
radioButton1.Image = Image.FromFile("C:\\Pictures\\brownImage.jpg");
```

Для роботи з елементом RadioButton використовують такі події користувача:

– RadioButton BackColorChanged;
– RadioButton CheckedChanged;
– RadioButton Click;
– RadioButton ForeColorChanged;
– RadioButton MouseHover;
– RadioButton MouseLeave;
– RadioButton TextChanged.

Подія RadioButton BackColorChanged відбувається лише під час зміни кольору фону перемикача. Припустимо, що потрібно показати повідомлення або виконати будь-яку функціональність, коли колір фону перемикача змінився, тоді використовується BackColorChanged Event. Є два способи, за допомогою яких можна встановити цю подію для RadioButton. Перший спосіб – активувати цю подію з розділу властивостей перемикача через події. Другий метод полягає у створенні визначеної користувачем функції та встановленні як події BackColorChanged у властивостях події (рис. 5.8).

Подія CheckedChanged відбувається, коли користувач змінить стан Radio Button (checked / unchecked). Наприклад, користувач вибирає RadioButton A, а потім вибирає RadioButton B. Тоді буде викликана подія CheckedChanged.

Подія Click відбудеться, коли користувач натисне на RadioButton. Подія ForeColorChange виконується, коли користувач змінить колір RadioButton.

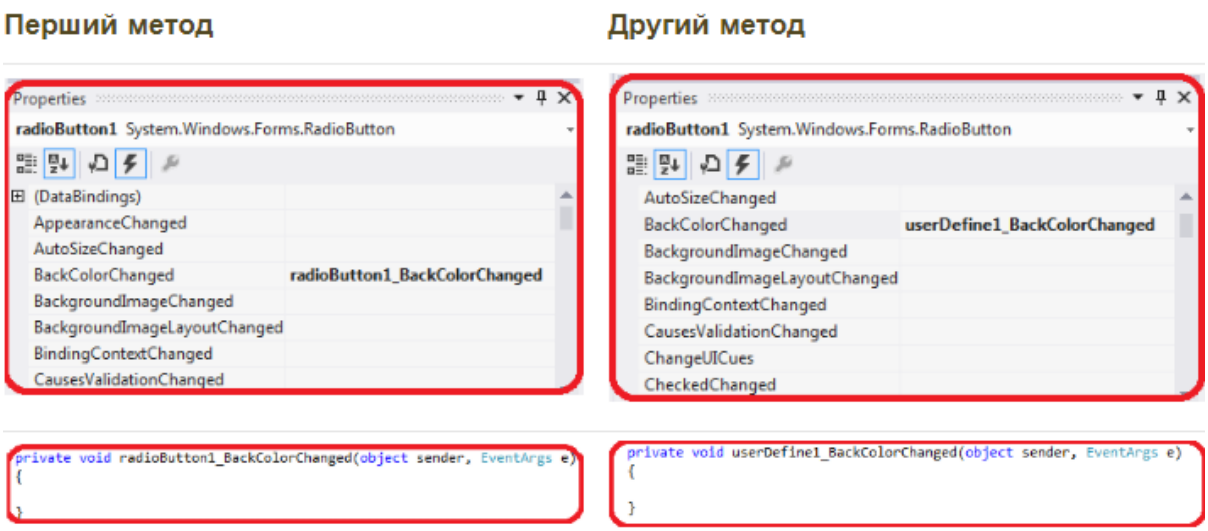


Рис. 5.8. Встановлення події BackColorChanged для RadioButton

Подія MouseHover виконується щоразу, коли користувач наводить курсор миші на перемикач. Якщо навести курсор миші на текст перемикача, то подія також виконується, тому що текст також входить до меж RadioButton. Продемонструємо вищевказаний сценарій.

```
Private void radioButton1_MouseHover(object sender, EventArgs e)
{
    MessageBox.Show("Mouse Hover to " + radioButton1.Text.ToString());
 }
```

Подія MouseLeave відбувається, коли курсор миші залишить межі перемикача, що протилежне події mouseHover. Продемонструємо цей сценарій у наступному коді разом із подією MouseHover. Можна спостерігати обидві події одночасно:

```
Private void radioButton1_MouseHover(object sender, EventArgs e)
{
    MessageBox.Show("Mouse Hover to " + radioButton1.Text.ToString());
}
private void radioButton1_MouseLeave(object sender, EventArgs e)
{
    MessageBox.Show("Mouse Left " + radioButton1.Text.ToString());
}
```

Подія TextChanged буде виконана, коли користувач змінить текст. Щоб спрацювала така подія, створіть подію натискання кнопки, в якій змінюється текст RadioButton. У наступному коді демонструється запропонований сценарій.

```
Private void button1_Click(object sender, EventArgs e)
{
    radioButton1.Text = "New Text";
}
private void radioButton1_TextChanged(object sender, EventArgs e)
{
```

```
            MessageBox.Show("Radio Button Text changed to " +
radioButton1.Text.ToString());
        }
```

1.2. Елемент CheckBox

CheckBox – це прапорець разом із текстом, який є назвою прапорця. Він дає змогу користувачу зробити множинний вибір. Якщо користувач перший раз натисне на прапорець, то на позначці прапорець буде позначено галочку. Коли користувач знову натисне прапорець, він знімає позначку.

Для додавання CheckBox у свій додаток необхідно перетягнути його з панелі інструментів на робочий стіл програми. На рис. 5.9 показана форма з 15 CheckBox-елементами.

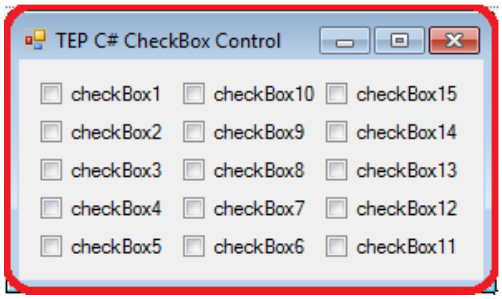


Рис. 5.9. Вікно проєкту з елементами CheckBox

Є два варіанти змінити назву CheckBox. Перший спосіб полягає в тому, щоб змінити властивість «Text» на вкладці «Properties» Visual Studio (рис. 5.10).

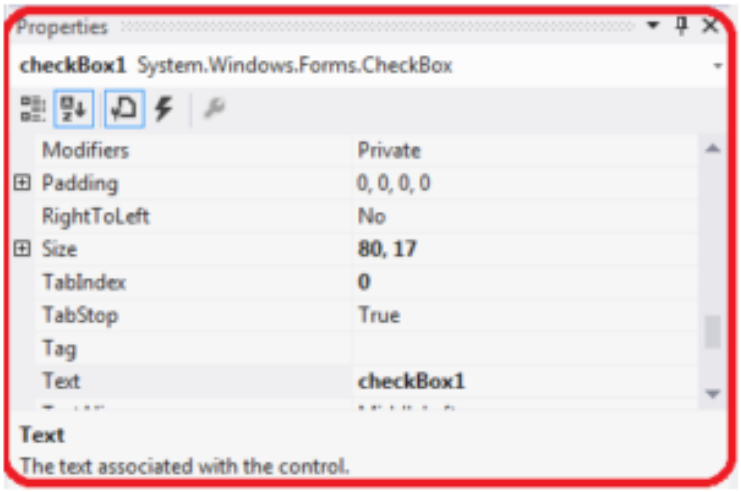


Рис. 5.10. Властивість Text вкладки «Properties»

Другий метод – це зміна назви програмно, як показано нижче:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        checkBox1.Text = "TEP CheckBox1";
        checkBox2.Text = "TEP CheckBox2";
    }
}
```

Колір тексту, розмір тексту, сімейство текстових шрифтів можемо використовувати для налаштування CheckBox. Так, можемо використовувати властивість Checked, щоб задати початковий стан CheckBox:

```
public Form1()
        {
            InitializeComponent();
            checkBox1.Checked = true;
            checkBox2.Checked = true;
            checkBox3.Checked = true;
                checkBox4.BackColor = Color.BurlyWood;
            checkBox5.BackColor = Color.DarkBlue;
            checkBox6.BackColor = Color.DarkOrange;
               checkBox7.ForeColor = Color.DeepSkyBlue;
               checkBox8.ForeColor = Color.Gainsboro;
               checkBox9.ForeColor = Color.LawnGreen;
                checkBox6.Font = new Font("Georgia", 11);
            checkBox7.Font = new Font("Impact", 11);
            checkBox8.Font = new Font("Lucida Console", 11);
        }
```

Для перевірки стану CheckBox застосовують умовні оператори, як у наступному коді (рис. 5.11):

```
        private void button1_Click(object sender, EventArgs e)
        {
            if (checkBox1.Checked == true)
            {
                MessageBox.Show(checkBox1.Text);
            }
            if (checkBox2.Checked == true)
            {
                MessageBox.Show(checkBox2.Text);
            }
            if (checkBox3.Checked == true)
            {
                MessageBox.Show(checkBox3.Text);
            }
```

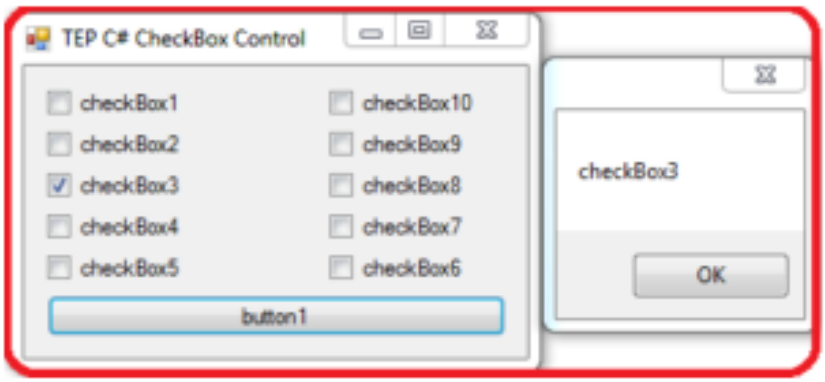


Рис. 5.11. Демонстрація фрагмента коду з перевіркою стану CheckBox

Для роботи з елементом CheckBox використовують такі події користувача:

– RadioButton BackColorChanged;
– RadioButton CheckedChanged;
– RadioButton Click;
– RadioButton ForeColorChanged;
– RadioButton MouseHover;
– RadioButton MouseLeave;
– RadioButton TextChanged.

Особливості роботи з подіями CheckBox аналогічні подіям елемента RadioButton, що були вже розглянуті.

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв’язання задачі.

Скласти програму, в якій реалізовано головоломку Лойда: із заданого набору чисел вибрати ті, сума яких дорівнює 50:

1. Додайте на форму елемент CheckedListBox. Властивість Item змініть, щоб додати назви CheckBox (рис. 5.12). Властивість CheckOnClick встановіть True (щоб переключалось одним кліком миши, а не двома).

2. Додайте на форму елементи Label1, Label2, Button1, Button2 (рис. 5.13). Елементу Label2 задайте значення label2.Text = "" та оголосіть глобальну змінну int sum.

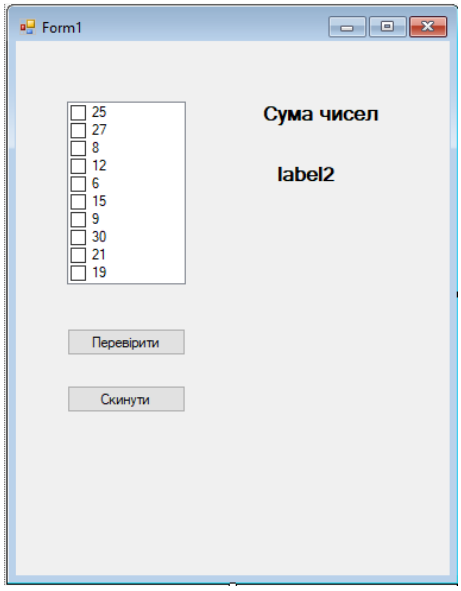


Рис. 5.12. Зовнішній вигляд форми для реалізації головоломки Лойда

3. Створіть подію Click для Button1, у якій виконується додавання значень ChexBox з CheckedListBox, що встановлені. Результат виводиться до Label2.

```
Private void button1_Click(object sender, EventArgs e)
        {
            sum = 0;
            for (int i = 0; i < checkedListBox1.Items.Count; i++)
                if (checkedListBox1.GetItemChecked(i))
                {
                    sum += int.Parse(checkedListBox1.Items[i].ToString());
                }
            label2.Text = Convert.ToString(sum);
        }
```

4. Перевірте роботу програми та переконайтеся, що виконується обробка значень ChexBox.

5. Додайте обробник події Click для Button2, у якому скидаються встановлені ChexBox елементи та результат.

```
Private void button2_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < checkedListBox1.Items.Count; i++)
                if (checkedListBox1.GetItemChecked(i))
                {
                    checkedListBox1.SetItemChecked(i, false);
                }
            label2.Text = "";
        }
```

6. Запустіть проєкт. Намагайтеся розв’язати головоломку.

```
using System;
using System.Windows.Forms;

namespace ChekBox
{
    public partial class Form1 : Form
    {
        int sum;

        public Form1()
        {
            InitializeComponent();
            label2.Text = "";

        }

        private void button1_Click(object sender, EventArgs e)
        {
            sum = 0;
            for (int i = 0; i < checkedListBox1.Items.Count; i++)
                if (checkedListBox1.GetItemChecked(i))
                {
                    sum += int.Parse(checkedListBox1.Items[i].ToString());
                }
            label2.Text = Convert.ToString(sum);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < checkedListBox1.Items.Count; i++)
                if (checkedListBox1.GetItemChecked(i))
                {
                    checkedListBox1.SetItemChecked(i, false);
                }
            label2.Text = "";
        }

    }
}
```

Рис. 5.13. Код програми, що реалізовує головоломку Лойда

Завдання 2. Створити проєкт для розв'язання задачі.

Скласти програму, в якій виконується додавання значень елементів ChexBox.

1. Додайте на форму три елементи CheckBox. Властивість Text елементів checkBox встановіть відповідно до рис. 5.14. Додайте на форму елемент Label.

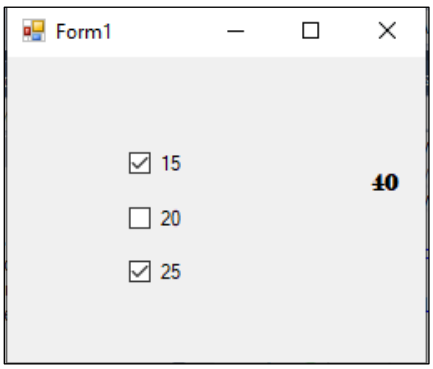


Рис. 5.14. Зовнішній вигляд форми для реалізації завдання 2

2. Додайте для кожного елементу обробник події CheckedChanged, за яким додаються значення вибраних значень CheckBox. Лістинг програми, що реалізовує запропонований сценарій, наведений нижче.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Test_Check
{
    public partial class Form1 : Form
    {
        int sum=0;
        int ch1, ch2, ch3;

        public Form1()
        {
            InitializeComponent();
            label1.Text = "";

        }
        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox1.Checked) {
                ch1 = int.Parse(checkBox1.Text);
             }
            else
            {
                ch1=0;
            }

            sum = ch1+ch2+ch3;
            label1.Text = sum.ToString();
        }

        private void checkBox2_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox2.Checked)
            {
                ch2 = int.Parse(checkBox2.Text);
            }
            else
            {
                ch2 = 0;
            }

            sum = ch1+ch2+ch3;
            label1.Text = sum.ToString();
        }

        private void checkBox3_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox3.Checked)
            {
```

```
                ch3 = int.Parse(checkBox3.Text);
            }
            else
            {
                ch3 = 0;
            }
            sum = ch1 + ch2+ch3;
            label1.Text = sum.ToString();
        }
    }
}
```

3. Внесіть зміни до форми та програми, щоб реалізувати головоломку Лойда.

3. Контрольні запитання

1. Для чого використовуються елементи RadioButton, ChexBox, Checked ListBox?

2. Як перевірити стан RadioButton, ChexBox, CheckedListBox?

3. Наведіть властивості елементів RadioButton, ChexBox, CheckedListBox. Як їх можна попередньо встановити та змінити під час роботи програми?

4. Які існують події під час роботи з RadioButton, ChexBox, CheckedListBox? Як їх налаштувати?

5. Як визначити кількість встановлених елементів в CheckedListBox?

ЛАБОРАТОРНА РОБОТА № 6
WINDOWS FORMS. ЕЛЕМЕНТ PICTUREBOX

Мета заняття: навчитися працювати з елементом PictureBox, закріпити навички роботи з елементами TextBox, ListBox у середовищі Microsoft Visual Studio

1. Теоретичні відомості

Елемент PictureBox використовується для відображення зображень. Він підтримує формати bmp, jpg, gif, а також метафайли та іконки. Для встановлення зображення у PictureBox можна скористатися кількома властивостями:

Image – задає об'єкт типу Image.

ImageLocation – встановлює шлях до зображення на диску або в інтернеті.

InitialImage – початкове зображення, яке відображається під час завантаження основного зображення, що зберігається у властивості Image.

ErrorImage – зображення, яке показується у випадку, якщо основне зображення не вдалося завантажити.

Щоб додати зображення через Visual Studio, у панелі властивостей Picture Box треба вибрати властивість **Image.** Після цього відкриється вікно імпорту зображення в проєкт, де можна вибрати потрібний файл на комп'ютері та встановити його для PictureBox (рис. 6.1).

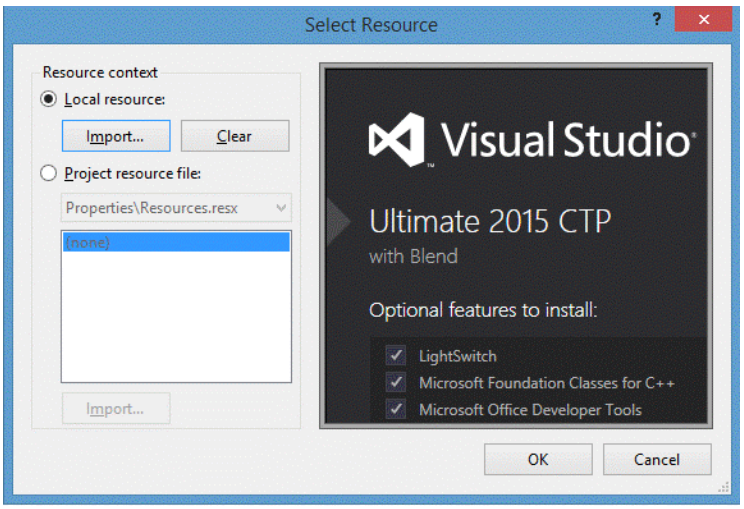


Рис. 6.1. Вікно імпорту зображення для PictureBox

Або можна завантажити зображення в коді:

```
pictureBox1.Image = Image.FromFile("C:\Users\Eugene\Pictures\12.jpg");
```

Для вибору способу відображення зображення в PictureBox використовується властивість **SizeMode,** яка може набувати таких значень (рис. 6.2):

Normal – зображення розташовується у лівому верхньому кутку PictureBox без зміни свого розміру. Якщо PictureBox більший за зображення, праворуч і знизу залишається порожнє місце; якщо менший – частини зображення обрізаються;

StretchImage – зображення розтягується або стискається так, щоб повністю заповнити ширину та висоту PictureBox;

AutoSize – PictureBox автоматично підлаштовується під розміри зображення;

CenterImage – якщо PictureBox менший за зображення, обрізається лише зовнішня частина, а центр зображення залишається видимим; якщо PictureBox більший – зображення розташовується по центру;

Zoom – зображення масштабується під розміри PictureBox, водночас зберігаються пропорції.

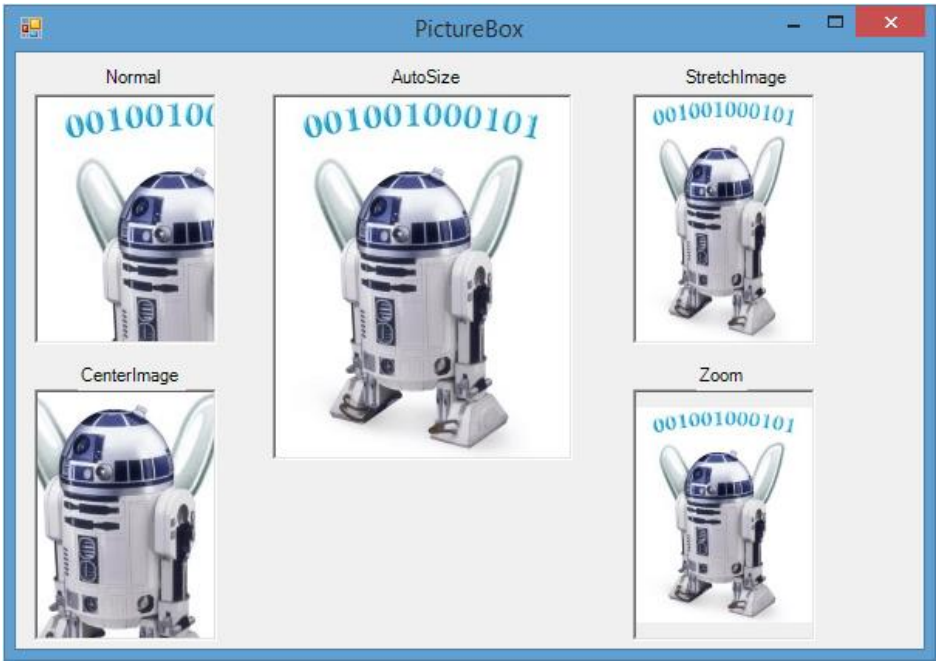


Рис. 6.2. Властивість SizeMode елементу PictureBox

Для роботи з елементом PictureBox використовують такі події:

– C# PictureBox Click Events;
– C# PictureBox DoubleClick Events;
– C# PictureBox MouseEnter Events;
– C# PictureBox MouseHover Events;
– C# PictureBox MouseLeave Events.

Подія Click відбудеться, коли користувач натисне на PictureBox. Наприклад:

```
private void pictureBox1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("You Just Click On the PictureBox");
        }
```

Подія DoubleClick відбудеться, коли користувач зробить подвійне натискання на PictureBox. Обробник події MouseEnter використовується для виконання будь-яких дій, коли курсор миші находиться на елементі PictureBox.

Подія MouseHover відбувається кожного разу, коли користувач на деякий час наводить курсор миші на PictureBox. Існує дуже мала різниця між обробниками MouseEnter та MouseHover Event. Подія MouseLeave виконується щоразу, коли курсор миші залишає межі PictureBox або видиму область:

```
private void pictureBox1_MouseHover(object sender, EventArgs e)
        {
            pictureBox1.Image = Image.FromFile("C: \\Users\\Pictures\\pic1.jpg");
        }

        private void pictureBox1_MouseLeave(object sender, EventArgs e)
        {
            pictureBox1.Image = Image.FromFile("C: \\Users\\Pictures\\pic2.jpg");

        }
```

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв'язання задачі.

Скласти програму для обчислення N-го числа Фібоначчі.

Числами Фібоначчі називають числа, які знаходять за таким правилом:

$$F1 = F2 = 1; \qquad Fn = Fn - 1 + Fn - 2.$$

Для обчислення N-го числа виконується така послідовність дій:

1) виділяються змінні A і B для зберігання двох чисел $Fn - 1$, $Fn - 2$;
2) сума чисел $A + B$ заноситься в змінну C;
3) на наступній ітерації циклу:
   $B = Fn - 1$ стає $(Fn - 2)$-м членом ряду, тому $A = B$;
   $C = Fn$ стає $(Fn - 1)$-м членом, тому $B = C$;
   $F1 = F2 = 1$, тому обчислення починається з $N = 3$.

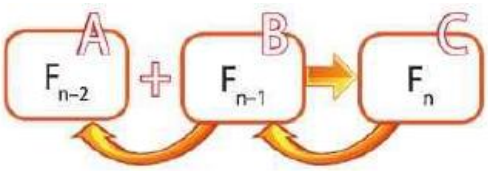


Рис. 6.3. Алгоритм обчислення N-го числа Фібоначчі

1. Створіть новий проєкт. Розробіть інтерфейс програми згідно з рис. 6.4. Додайте на форму елементи TextBox1, Label1, Button1, TextBox2, Label2.

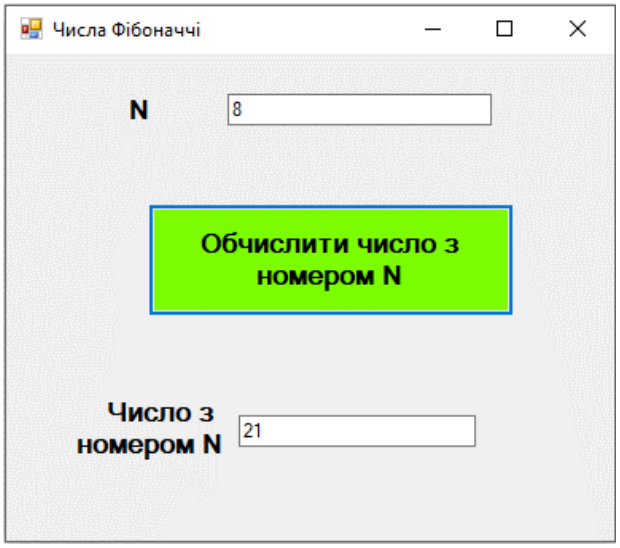


Рис. 6.4. Зовнішній вигляд форми для обчислення N-го числа Фібоначчі

2. Опишіть змінні, необхідні для реалізації завдання:

n – змінна цілого типу для збереження номера елементу послідовності, який потрібно знайти;

i – змінна цілого типу для збереження номера поточного елементу послідовності;

a, b, c – змінні цілого типу для збереження $(i-1)$-го, $(i-2)$-го, i-го елементів послідовності під час чергової ітерації циклу.

3. Створіть обробник події Click для кнопки «Button1». Запишіть програмний код для обчислення N перших чисел Фібоначчі.

```
private void button1_Click(object sender, EventArgs e)
{
    n = Convert.ToInt16(textBox1.Text);
    a = 1; b = 1; i = 2;
    while (i < n)
    {
        i++;
        c = a + b;
        a = b; b = c;
    }
    textBox2.Text = Convert.ToString(c);
}
```

4. Випробуйте проєкт. Додайте на форму елемент TextBox3. Доповніть програмний код операторами для обчислення суми N перших чисел Фібоначчі і виведення результату до текстового поля TextBox3. Перевірте дію кнопки.

5. Додайте на форму елемент PictureBox з зображенням чисел Фіббоначі.

Завдання 2. Скласти програму для розв'язання старовинної задачі.

Відомо, що плата за одного бика становить 20 карбованців, за корову – 10 карбованців, а за теля – 1 карбованець. Потрібно визначити, скільки биків, корів і телят можна купити на 200 карбованців, якщо загальна кількість худоби має бути 100 голів.

1. Створіть новий проєкт. Змініть заголовок форми на «Старовинна задача». Додайте на форму такі елементи: Label1, TextBox1, ListBox, Button1 (рис. 6.5).

2. Позначте **b** – кількість биків, **k** – кількість корів, **t** – кількість телят. Загальна кількість голів визначається рівнянням:

$$b + k + t = 100.$$

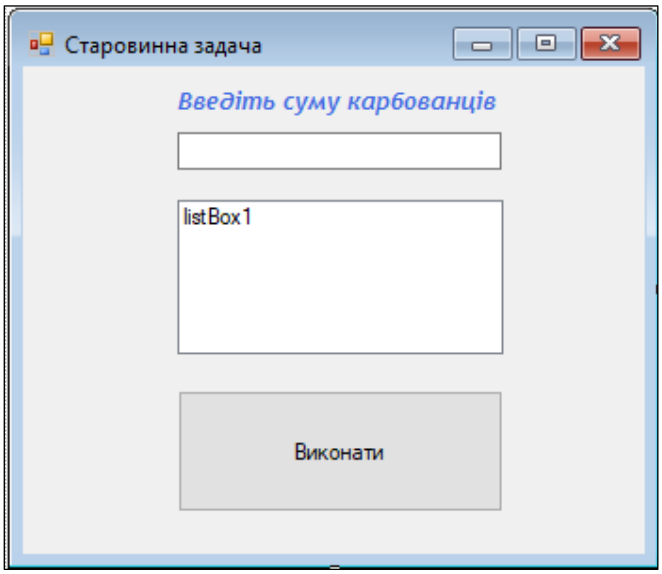


Рис. 6.5. Зовнішній вигляд форми для завдання 2

За биків заплатили **20b** карбованців, за корів – **10k,** а за телят – **t** карбованців. Тобто маємо рівняння:

$$20b + 10k + t = 200.$$

На 200 карбованців можна купити:

– не більше 10 биків, тобто $0 <= b <= 10$;

– не більше 20 корів, тобто $0 <= k <= 20$;

– не більше 200 телят, тобто $0 <= t <= 200$.

Отже, необхідно перебрати всі можливі значення змінних $b,\ k,\ t$ і вивести в поле *ListBox* той набір значень, для яких виконується умова:

$$(20 * b + 10 * k + 1 = 200) \quad \text{and} \quad (b + k + t = 100).$$

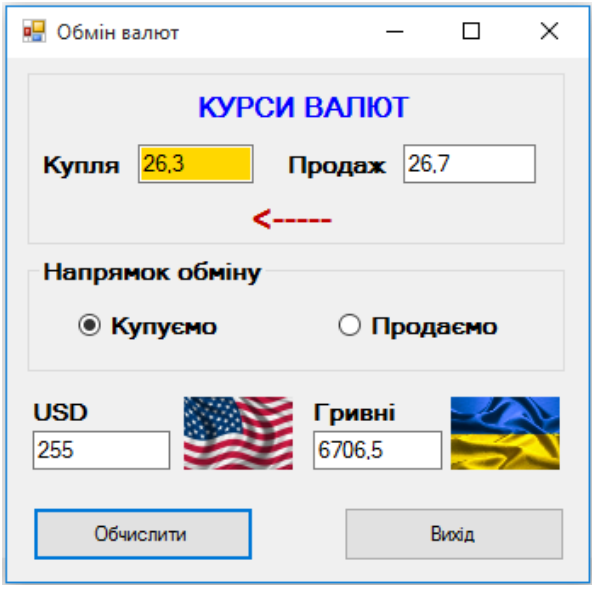
3. Створіть процедуру обробки події Click для кнопки. Запишіть програмний код.

4. Розв'яжіть задачу: якщо у вас є в наявності 300 карбованців та 400 карбованців.

```
private void button1_Click(object sender, EventArgs e)
{
    int S = Convert.ToInt32(textBox1.Text);
    int N = 100;
    listBox1.Items.Clear();
    for (int b=0; b<10; b++)
        for (int k=0; k<20; k++)
            for (int t = 0; t < 200; t++)
            {
                if ((20 * b + 10 * k + t == S) && (b + k + t == N))
                {
                    listBox1.Items.Add("Биків " + Convert.ToString(b));
                    listBox1.Items.Add("Корів " + Convert.ToString(k));
                    listBox1.Items.Add("Телят " + Convert.ToString(t));
                }
            }
}
```

Завдання 3. Створити проєкт для розв'язання задачі.

Розробити програму «Обмін валюти» для моделювання відповідних операцій обмінного пункту. Під час перемикання напряму обміну стрілка повинна вказувати напрям та підсвічувати жовтим кольором відповідний курс.



3. Контрольні запитання

1. Для чого використовуються елементи PictureBox?
2. Як вибрати зображення для PictureBox?
3. Які є атрибути властивості SizeMode ?
4. Які існують події під час роботи з PictureBox? Як їх налаштувати?
5. Наведіть перші 10 чисел Фібоначчі. Поясніть, як вони обраховуються.

ЛАБОРАТОРНА РОБОТА № 7
WINDOWS FORMS. КОНТЕЙНЕР PANEL. СТВОРЕННЯ ДОДАТКА LABYRYNT

Мета заняття: навчитися працювати з контейнерами GroupBox, Panel і FlowLayoutPanel, навчитися задавати розміри елементів та їх позиціонування у контейнері, закріпити навички роботи з елементом Label та обробниками подій MouseEnter, MouseLeave у середовищі Microsoft Visual Studio.

1. Теоретичні відомості

1.1. Елементи GroupBox, Panel і FlowLayoutPanel

GroupBox – це спеціальний контейнер, відокремлений від іншої частини форми рамкою. Він має заголовок, який задається через властивість Text. Якщо потрібно створити GroupBox без заголовка, достатньо залишити цю властивість порожньою. Найчастіше елемент використовується для об’єднання перемикачів (RadioButton), оскільки дає змогу чітко розділити їх на окремі групи (рис. 7.1).

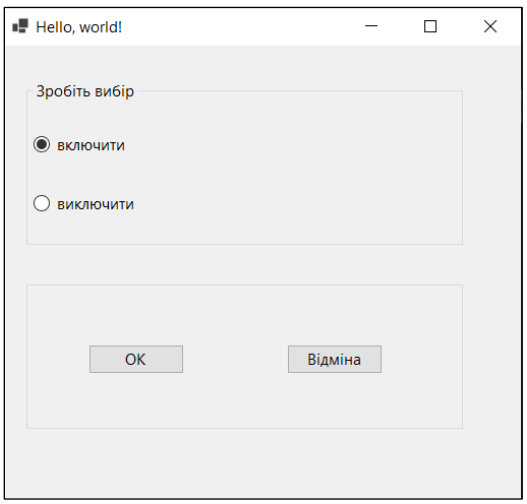


Рис. 7.1. Приклад застосування контейнера GroupBox

Елемент Panel є контейнером, що, подібно до GroupBox, використовується для групування елементів. Якщо для властивості BackColor панелі встановити такий самий колір, як у форми, вона може зливатися з нею візуально. Щоб зробити панель помітною, можна використати властивість BorderStyle, яка за замовчуванням має значення None (без межі).

Якщо в панелі розміщено багато елементів і вони виходять за її межі, можна увімкнути прокрутку, встановивши властивість AutoScroll = True (рис. 7.2).

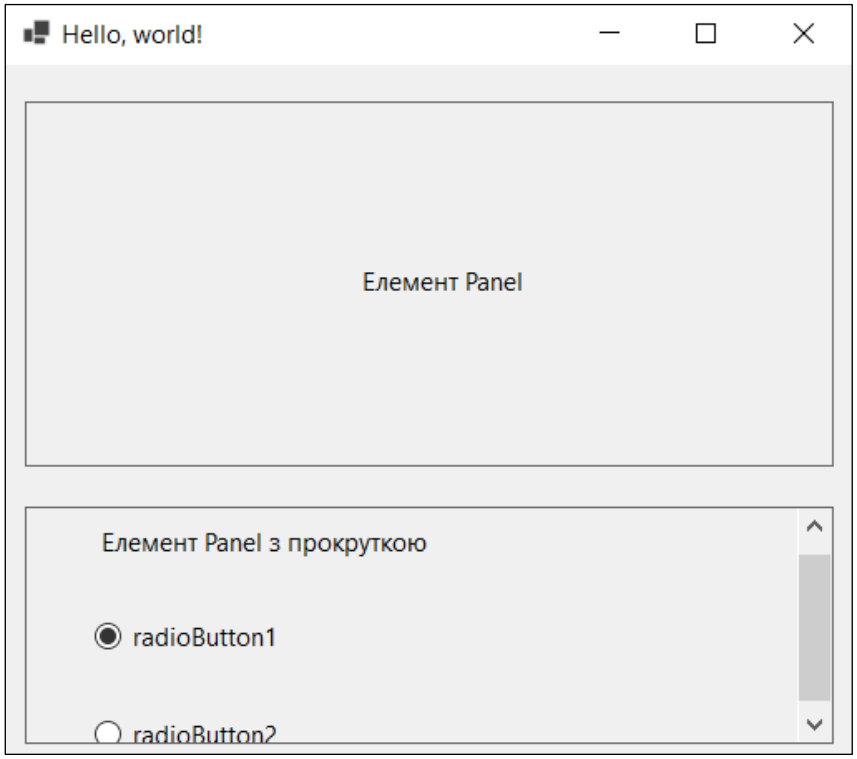


Рис. 7.2. Приклад застосування контейнера Panel

Подібно до форми та GroupBox, Panel містить колекцію елементів, які можна додавати динамічно. Наприклад, на формі є елемент GroupBox з ім'ям groupBox1:

```
private void Form1_Load(object sender, EventArgs e)
{
    Button helloButton = new Button();
    helloButton.BackColor = Color.LightGray;
    helloButton.ForeColor = Color.Red;
    helloButton.Location = new Point(30, 30);
    helloButton.Text = "Привіт";
    groupBox1.Controls.Add(helloButton);

}
```

Для визначення позиції елементу всередині контейнера використовується структура Point. Наприклад: new Point(30, 30), де координати задають розташування елементу відносно верхнього лівого кута контейнера (у цьому випадку – GroupBox).

Варто враховувати, що контейнером верхнього рівня завжди є форма, а groupBox1 знаходиться в її колекції елементів. За потреби його можна навіть видалити.

```
1 this.Controls.Remove(groupBox1);
```

FlowLayoutPanel успадковує властивості від Panel, але має додаткові можливості. Він дає змогу автоматично змінювати розташування дочірніх елементів у разі зміни розміру форми під час виконання програми.

Напрям розташування задається властивістю FlowDirection, яка може набувати таких значень:

LeftToRight (за замовчуванням) – елементи розташовуються зліва направо, починаючи з верхнього кута;

RightToLeft – елементи розміщуються справа наліво;

TopDown – елементи йдуть зверху вниз;

BottomUp – елементи розташовуються знизу вгору (рис. 7.3).

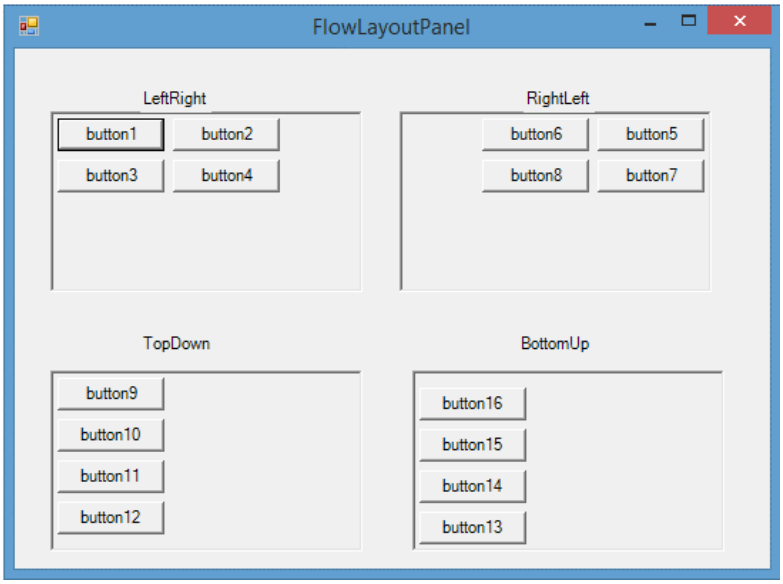


Рис. 7.3. Властивість FlowDirection контейнера FlowLayoutPanel

Ще одна важлива властивість – WrapContents. Якщо вона дорівнює True (за замовчуванням), елементи, які не вміщуються, переносяться на новий рядок або колонку. Якщо встановлено False, елементи не переносяться, і у випадку ввімкненої прокрутки (AutoScroll = True) додаються смуги прокрутки.

Фрагмент коду для динамічного створення елементу Panel, а в ньому textbox:

```
Panel p = new Panel();
p.Controls.Add(new TextBox());
```

1.2. Розміри та позиціювання елементів

Для кожного елементу керування можна задати властивість Location, яка визначає координати його верхнього лівого кута відносно контейнера. Після перетягування елементу з панелі інструментів ця властивість встановлюється автоматично, але її можна змінити вручну у вікні властивостей або програмно (рис. 7.4):

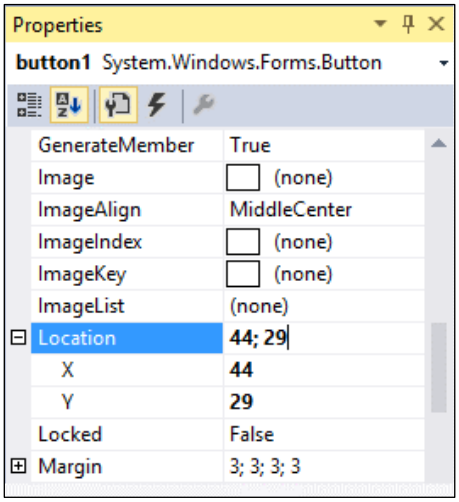


Рис. 7.4. Властивість Location елементу керування

Також ми можемо встановити позицію елементу в коді:

```
private void Form1_Load(object sender, EventArgs e)
{
    button1.Location = new Point(50, 50);
}
```

За допомогою властивості Size можна задати розміри елементу (рис. 7.5):

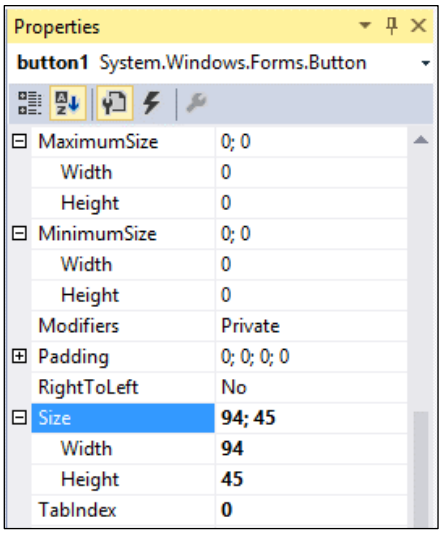


Рис. 7.5.Властивість Size елементу керування

Додаткові властивості MaximumSize та MinimumSize дають змогу обмежити мінімальний та максимальний розміри. Установка властивостей у коді:

```
button1.Size = new Size { Width = 50, Height = 25 };
```

```
button1.Width = 100;
button1.Height = 35;
```

Для динамічного вирівнювання використовується властивість Anchor. Вона визначає відстань між певними сторонами елементу і сторонами контейнера. Якщо контейнер змінює розмір, то й елемент автоматично масштабується відповідно. За замовчуванням увімкнені Top і Left (рис. 7.6):

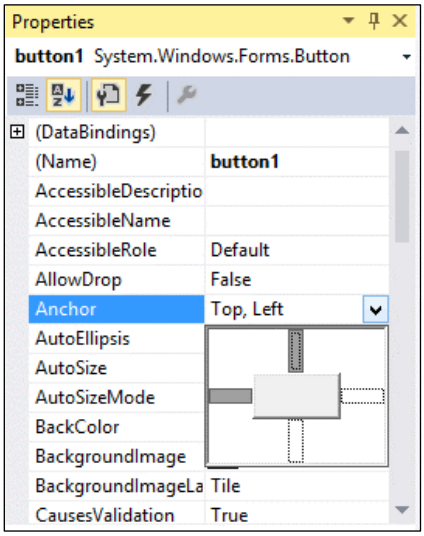


Рис. 7.6. Позиціювання елементу керування властивістю Anchor

Це означає, що якщо будемо розтягувати форму вліво або вгору, то елемент збереже відстань від лівої і верхньої межі елементу до меж контейнера, в якості якого виступає форма. Можна задати чотири можливі значення для цієї властивості або їх комбінацію: Top, Bottom, Left, Right. Наприклад, якщо змінимо значення цієї властивості на протилежне – Bottom, Right – відстань буде незмінною між правою і нижньою стороною елементу та формою. Також треба зазначити, що ця властивість враховує відстань до межі контейнера, а не форми. Тобто якщо на формі є елемент Panel, а на Panel розташована кнопка, то на кнопку впливатиме зміна меж Panel, а не форми. Розтягування форми буде в цьому випадку впливати тільки якщо вона впливає на контейнер Panel. Щоб задати цю властивість у коді, треба використовувати перерахування AnchorStyles:

```
button1.Anchor = AnchorStyles.Left;
```

```
button1.Anchor = AnchorStyles.Left;   |   AnchorStyles.Top;
```

Властивість Dock – «прикріплює» елемент до певної сторони контейнера. За замовчуванням має значення None (рис. 7.7). Доступні варіанти: Top – прикріплення до верхньої межі; Bottom – до нижньої межі; Left – до лівої межі; Right – до правої межі; Fill – елемент займає весь простір контейнера.

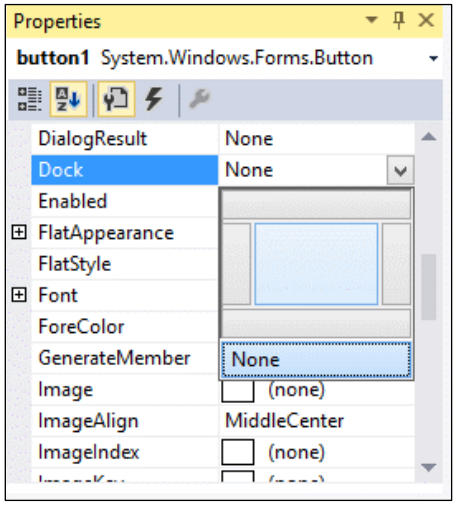


Рис. 7.7. Властивість Dock елементу керування

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв'язання задачі.

Під час запуску програми курсор повинен бути на старті. Якщо курсор торкається стінки лабіринту (внутрішньої або зовнішньої), то його перекидає на початок.

1. Створіть проєкт ***winForm*** із назвою ***Labyrinth.*** Розмір форми довільний.

2. Властивість ***Text*** форми змінити на «Лабіринт».

3. Додайте на форму елемент ***Panel,*** розтягніть по краях так, щоб був невеликий проміжок.

4. Властивості ***FormBorderStyle*** виставте значення ***Fixed3D*** (забороняє зміну розмірів форми користувачем).

5. Відключіть кнопку «Розгорнути» заголовка вікна. Для цього встановіть властивість форми ***MaximizeBox*** у значення ***False.***

6. Виділіть панель і встановіть властивості ***BorderStyle*** значення ***Fixed3D.*** Це необхідно, щоб добре була видима межа поля гри.

7. Додайте на форму елемент ***Label.*** Властивості ***AutoSize*** встановіть значення ***False.*** Властивості ***BackColor*** задайте колір ***SkyBlue.*** Видаліть значення властивості ***Text.***

8. Виділіть елемент ***Label*** та зробіть множину копій (Ctrl + C – Ctrl + V). Із цих елементів побудуйте лабіринт (рис. 7.8).

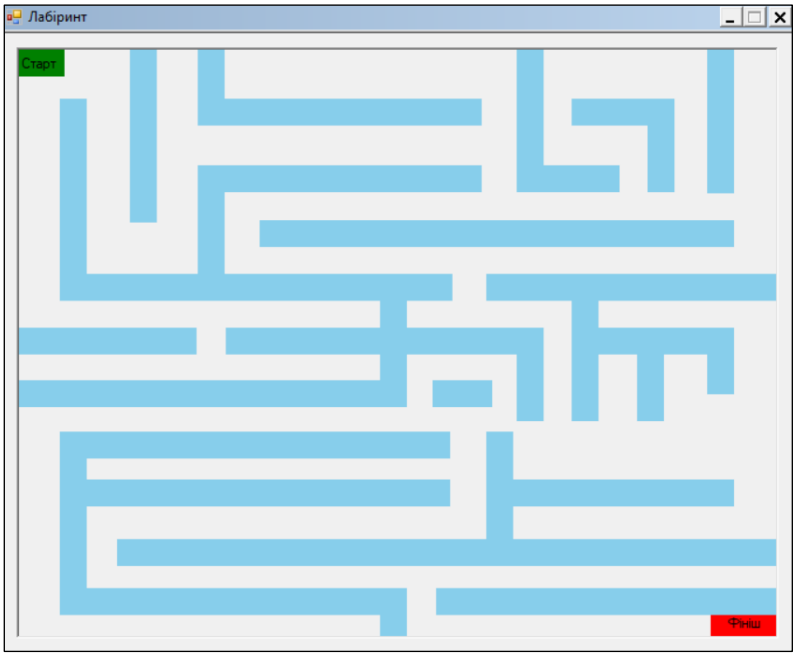


Рис. 7.8. Зовнішній вигляд додатка, що проєктується

9. Додайте ***Label*** фініш: властивість ***Name*** – ***labelFinish, Text*** – «Фініш», колір червоний. Додайте ***Label*** старт: властивість ***Name*** – ***labelStart, Text*** – «Старт», колір – зелений.

10. Виділіть усі сині ***label*** та створіть обробник події ***MouseEnter*** подвійним натисненням (рис. 7.9).

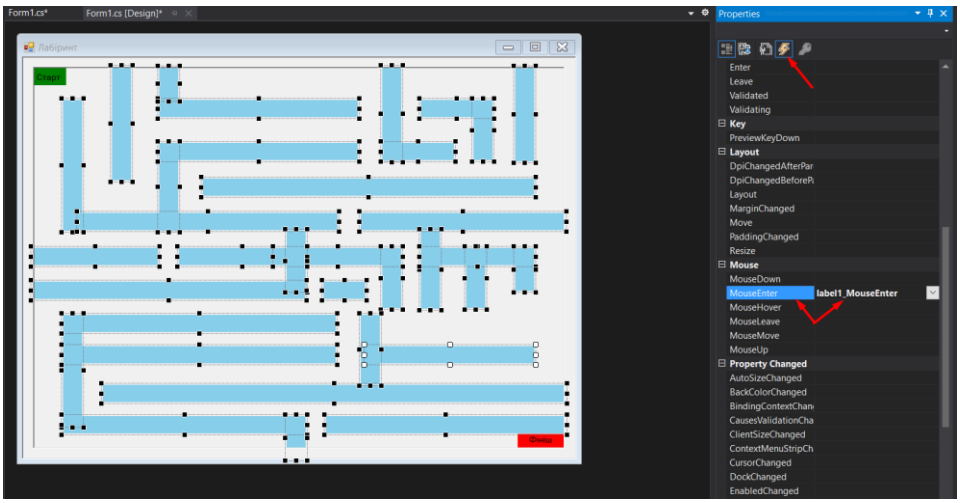


Рис. 7.9. Створення обробника події MouseEnter для елементів label

11. На формі виділіть тільки елемент ***Panel*** та створіть обробник події ***MouseLeave*** і вкажіть подію ***Label1_MouseEnter*** (рис. 7.10).

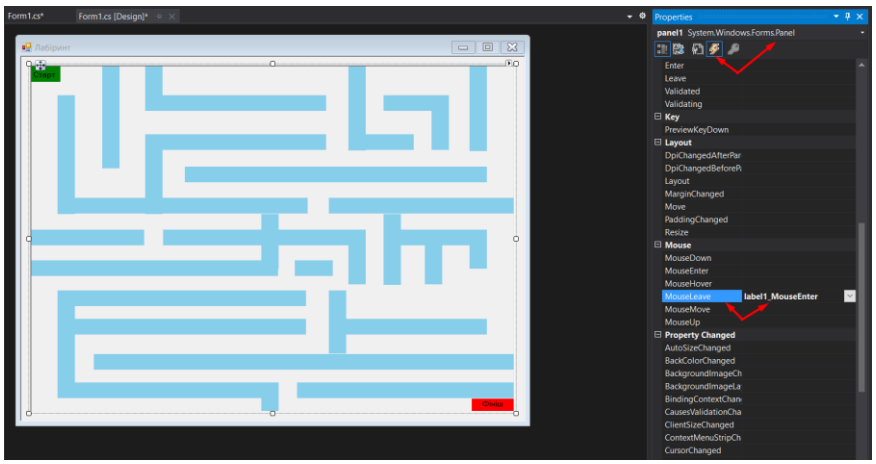


Рис. 7.10. Створення обробника події MouseLeave для елементу Panel

12. Створіть обробник для форми ***Load,*** який необхідний, щоб у момент запуску форми задати позицію курсора (рис. 7.11).

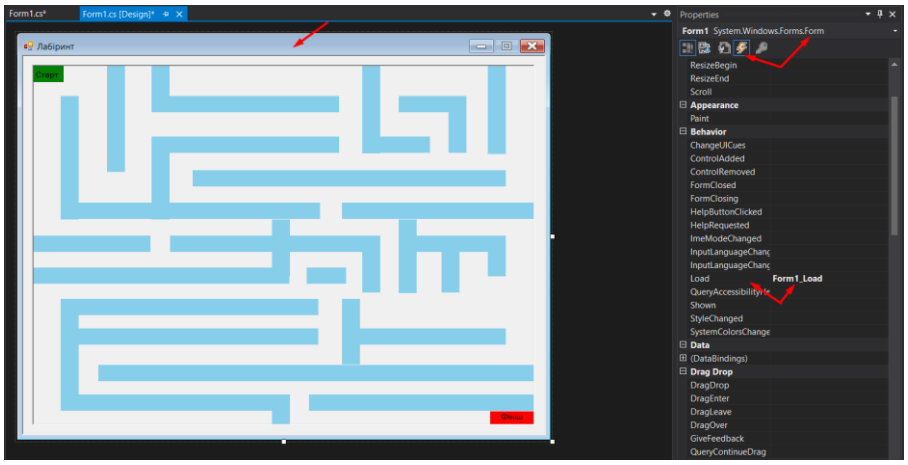


Рис. 7.11. Створення обробника події Load для елементу Form

13. Виділіть елемент ***labelFinish*** та створіть обробник події ***MouseEnter*** для нього (рис. 7.12).

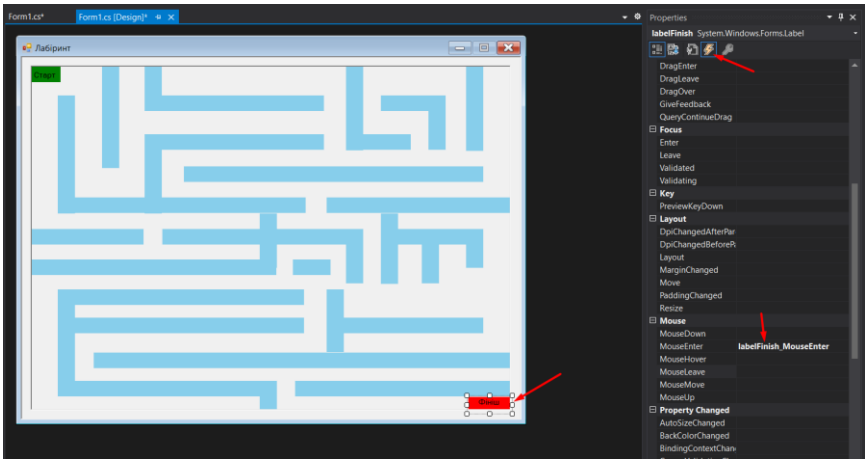


Рис. 7.12. Обробник події MouseEnter елементу labelFinish

Лістинг програмного коду проєкту

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Labyrinth
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        //Під час доторкання до стінки лабіринту позиція буде оновлюватись до
стартової.
        private void label71_MouseEnter(object sender, EventArgs e)
        {
            Cursor.Position = new Point(this.Location.X + 55, this.Location.Y +
60);
        }
       //Виставляємо стартову позицію курсора.
        // +55 і +60 – це позиція labelStart відносно позиції форми на робочому
столі.
        private void Form1_Load(object sender, EventArgs e)
        {
            Cursor.Position = new Point(this.Location.X + 55, this.Location.Y +
60);
        }
        //Під час наведення на Finish з’явиться повідомлення
        private void labelFinish_MouseEnter(object sender, EventArgs e)
        {
MessageBox.Show("Перемога!!!","Ура!!!",MessageBoxButtons.OK,MessageBoxIcon.Informa
tion);
        }
    }
}
```

3. Контрольні запитання

1. Призначення контейнерів GroupBox, Panel, FlowLayoutPanel. У чому різниця їх застосування?

2. Як задати розміри елементів та їх позиціонування у контейнері?

3. Яке призначення властивостей: Location, Size, MaximumSize, MinimumSize, Anchor, Dock?

4. Для чого використовується властивість FormBorderStyle елементу Panel?

5. Для чого використовується властивість BorderStyle елементу Panel?

6. Для чого використовується властивість MaximizeBox елементу Form?

7. Для чого використовується атрибут Fixed3D властивостей FormBorder Style, BorderStyle елементу Panel?

8. Як вибрати декілька елементів Label?

9. Для чого використовується обробник події MouseEnter? Як додати таку подію в проєкт?

10. Для чого використовується обробник події MouseLeave? Як додати таку подію в проєкт?

11. Які існують події під час роботи з PictureBox? Як їх налаштувати?

12. Для чого використовується обробник події форми Load?

ЛАБОРАТОРНА РОБОТА № 8
ЕЛЕМЕНТ TIMER. ДОДАТОК «ДРУКАРСЬКА МАШИНКА»

Мета заняття: навчитися працювати з елементом Timer та його обробником події Tick, навчитися працювати з елементом StatusStrip та обробником подій форми Key_Down за допомогою Windows Forms мовою C# у середовищі Microsoft Visual Studio.

1. Теоретичні відомості

1.1. Елемент Timer

Timer – це компонент, який дає змогу виконувати дії через певні проміжки часу. Хоча він не є візуальним елементом, його можна додати на форму, перетягнувши з Панелі інструментів (рис. 8.1):

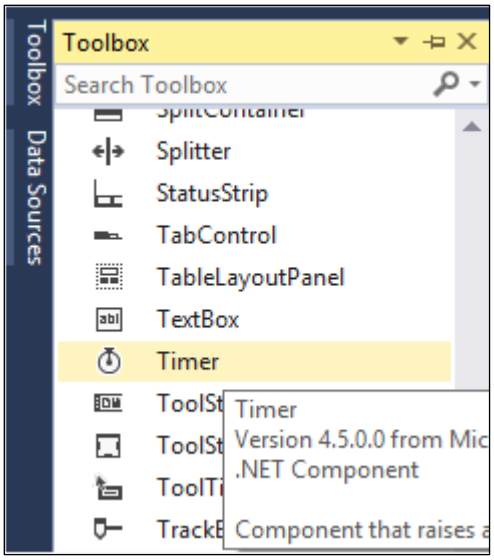


Рис. 8.1. Елемент Таймер на панелі інструментів у середовищі Microsoft Visual Studio

Основні властивості та методи Timer:

Enabled – якщо має значення *true*, таймер запускається разом із формою;

Interval – задає інтервал у мілісекундах, через який виконується обробник події Tick;

Start() – запускає таймер;

Stop() – зупиняє таймер.

Для прикладу визначимо просту форму, на яку додамо кнопку і таймер (рис. 8.2). У файлі коду форми визначимо такий код:

```
public partial class Form1 : Form
{
    int koef = 1;
    public Form1()
```

```
    {
        InitializeComponent();
         this.Width = 400;
        button1.Width = 40;
        button1.Left = 40;
        button1.Text = "";
        button1.BackColor = Color.Aqua;
         timer1.Interval = 500; // 500 мс
        timer1.Enabled = true;
        button1.Click += button1_Click;
        timer1.Tick += timer1_Tick;
    }
    // обробник події Tick таймера
    void timer1_Tick(object sender, EventArgs e)
    {
        if (button1.Left == (this.Width-button1.Width-10))
        {
            koef=-1;
        }
        else if (button1.Left == 0)
        {
            koef = 1;
        }
        button1.Left += 10 *koef;
    }
    // обробник натиснення на кнопку
    void button1_Click(object sender, EventArgs e)
    {
        if(timer1.Enabled==true)
        {
            timer1.Stop();
        }
        else
        {
            timer1.Start();
        }
    }
}
```

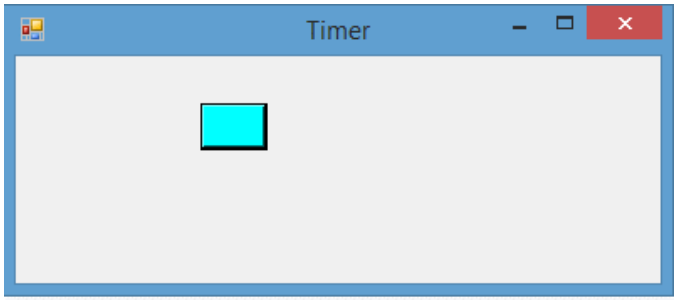


Рис. 8.2. Вікно форми для роботи з елементом Timer

У конструкторі форми задаються початкові параметри таймера, кнопки та самої форми.

Через кожен інтервал спрацьовує обробник timer1_Tick, у якому горизонтальне положення кнопки змінюється за допомогою властивості button1.Left. Напрямок руху контролюється змінною koef.

Подія button1_Click дає змогу зупинити роботу таймера (а отже, і рух кнопки) або знову його запустити.

1.2. Рядок стану StatusStrip

StatusStrip – це рядок стану, який використовується для відображення актуальної інформації про роботу програми. Під час додавання на форму він за замовчуванням розташовується внизу вікна, але його позицію можна змінити за допомогою властивості Dock, яка підтримує такі значення:

Bottom – нижня частина форми (за замовчуванням);

Top – верхня частина;

Fill – заповнює всю форму;

Left – ліва частина;

Right – права частина;

None – довільне розташування.

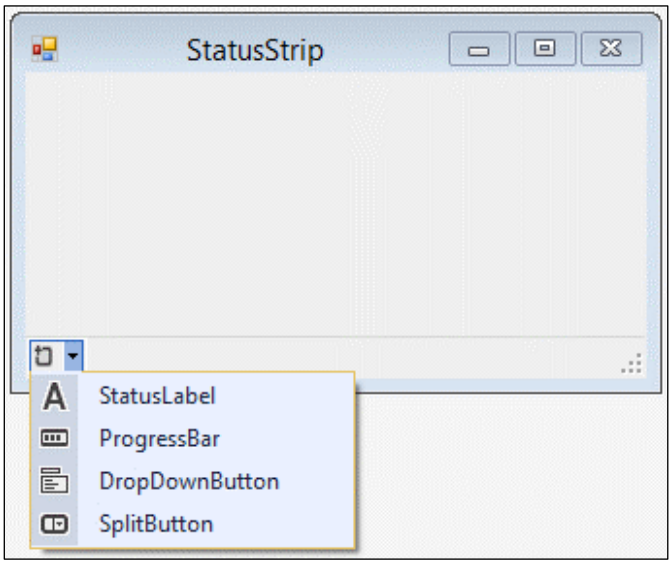


Рис. 8.3. Елементи StatusStrip у режимі дизайнера

У режимі дизайнера до StatusStrip можна додати різні елементи:

StatusLabel – текстова мітка *(ToolStripLabel);*

ProgressBar – індикатор виконання *(ToolStripProgressBar);*

DropDownButton – кнопка зі спадним списком *(ToolStripDropDown Button);*

SplitButton – кнопка зі схожою логікою *(ToolStripSplitButton).*

Можна звернутися до властивості Items компонента StatusStrip і у вікні додати та налаштувати всі елементи (рис. 8.4):

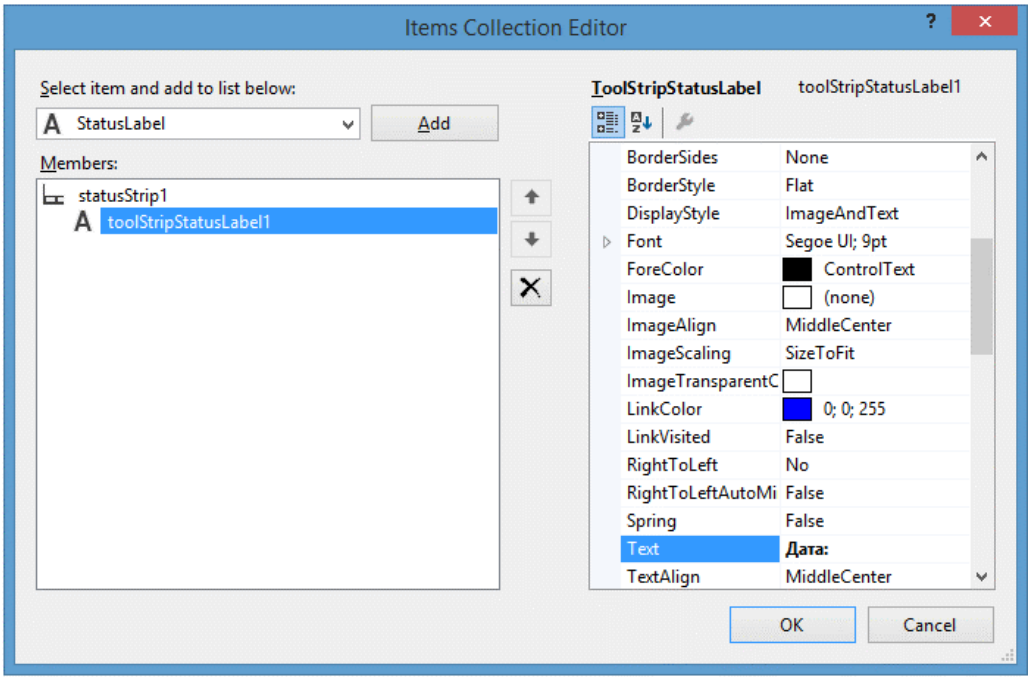


Рис. 8.4. Вікно властивостей Items компонента StatusStrip

Також ми можемо додати елементи програмно, як у коді форми, що наведений нижче:

```
public partial class Form1 : Form
{
    ToolStripLabel dateLabel;
    ToolStripLabel timeLabel;
    ToolStripLabel infoLabel;
    Timer timer;
    public Form1()
    {
        InitializeComponent();

        infoLabel = new ToolStripLabel();
        infoLabel.Text = "Текущие дата и время:";
        dateLabel = new ToolStripLabel();
        timeLabel = new ToolStripLabel();
        statusStrip1.Items.Add(infoLabel);
        statusStrip1.Items.Add(dateLabel);
        statusStrip1.Items.Add(timeLabel);
        timer = new Timer() { Interval = 1000 };
        timer.Tick += timer_Tick;
        timer.Start();
    }
     void timer_Tick(object sender, EventArgs e)
    {
        dateLabel.Text = DateTime.Now.ToLongDateString();
        timeLabel.Text = DateTime.Now.ToLongTimeString();
    }
}
```

У наведеному прикладі створюються три мітки **(StatusLabel)** і таймер. Після запуску форми таймер активується, і під час кожного спрацювання події **Tick** в обробнику оновлюється текст міток.

2. Практичне завдання


У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв'язання задачі.

У формі будуть з'являтися випадкові літери. Якщо гравець вводить букви правильно, вони зникають, рівень зростає і букви з'являються все частіше. Якщо вся форма заповнена літерами, гра закінчена.

1. Створіть проєкт ***winForm*** із назвою ***Type.*** Для властивостей форми виставте такі значення:

```
FormBorderStyle = Fixed3D
Size = 876; 174
Text = Hit the keys!
KeyPreview = true;
```

2. Додайте на форму елемент ***ListBox.*** Для цього елементу виставте такі значення:

```
Dock = Fill;
MultiColumn = True;
Font Size = 72;
```

3. Додайте на форму елемент ***Timer.***

4. Додайте на форму елемент ***StatusStrip,*** для якого встановіть властивість:

```
SizingGrip = False.
```

5. Додайте в елемент ***StatusStrip*** 4 мітки ***StatusLabel*** (рис. 8.5).

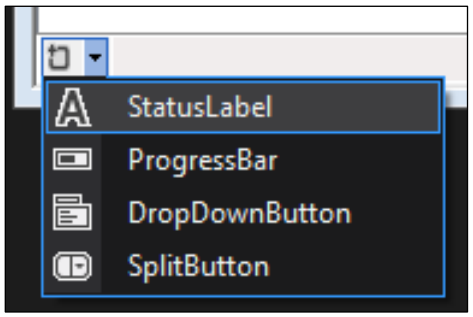


Рис. 8.5. Елементи StatusStrip

6. Для міток виставте такі властивості:

Мітка 1	Мітка 2	Мітка 3	Мітка 4
Name = correctLabel; Text = Correct: 0;	Name = missedLabel; Text = Missed: 0;	Name = totalLabel; Text = Total: 0;	Name = accuracyLabel; Text = Accuracy: 0%;

7. Додайте в ***StatusStrip*** ще один ***StatusLabel*** з властивостями:

```
Spring: True;
TextAlign: MiddleRight;
Text: Difficulty.
```

8. Додайте в ***StatusStrip*** елемент ***ProgressBar*** (рис. 8.5), властивості Name присвойте значення difficultyProgressBar, а властивості Maximum = 701. Форма повинна мати вигляд, як наведено на рис. 8.6.

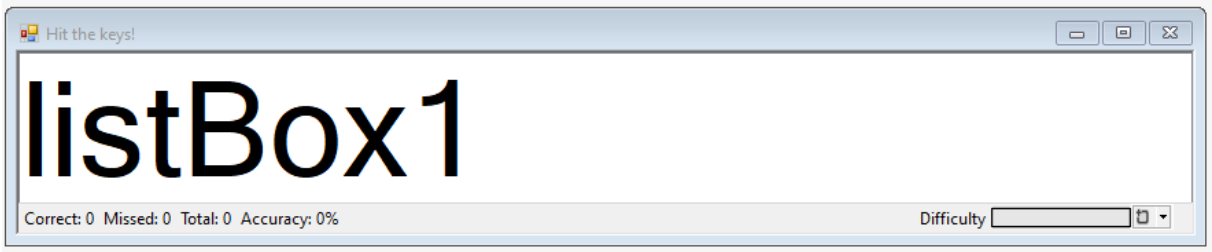


Рис. 8.6. Вигляд форми для реалізації завдання 1

9. Виділіть елемент ***Timer*** та встановіть властивість Interval: 800.

10. Створіть обробник події ***Tick*** для елементу ***Timer.*** Додайте такий код:

```
namespace Type
{
    public partial class Form1 : Form
    {
        //Додаємо екземпляр класу Random, який необхідний для отримання випадкової
літери
        Random random = new Random();

        public Form1()
        {
            InitializeComponent();
            timer1.Enabled = true; //Робимо таймер активним
            timer1.Start(); //Запускаємо таймер
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            listBox1.Items.Add((Keys)random.Next(65, 90)); //Додаємо в listBox
літеру
            if (listBox1.Items.Count > 7) //Якщо літер більше 7, гра завершена.
            {
                listBox1.Items.Clear(); //Очищаємо listBox
                listBox1.Items.Add("Гра завершена!");
                timer1.Stop(); // Зупиняємо таймер.
            }
```

```
        }
    }
}
```

11. Додайте до проєкту новий клас ***Stats*** для отримання статистики (рис. 8.7).

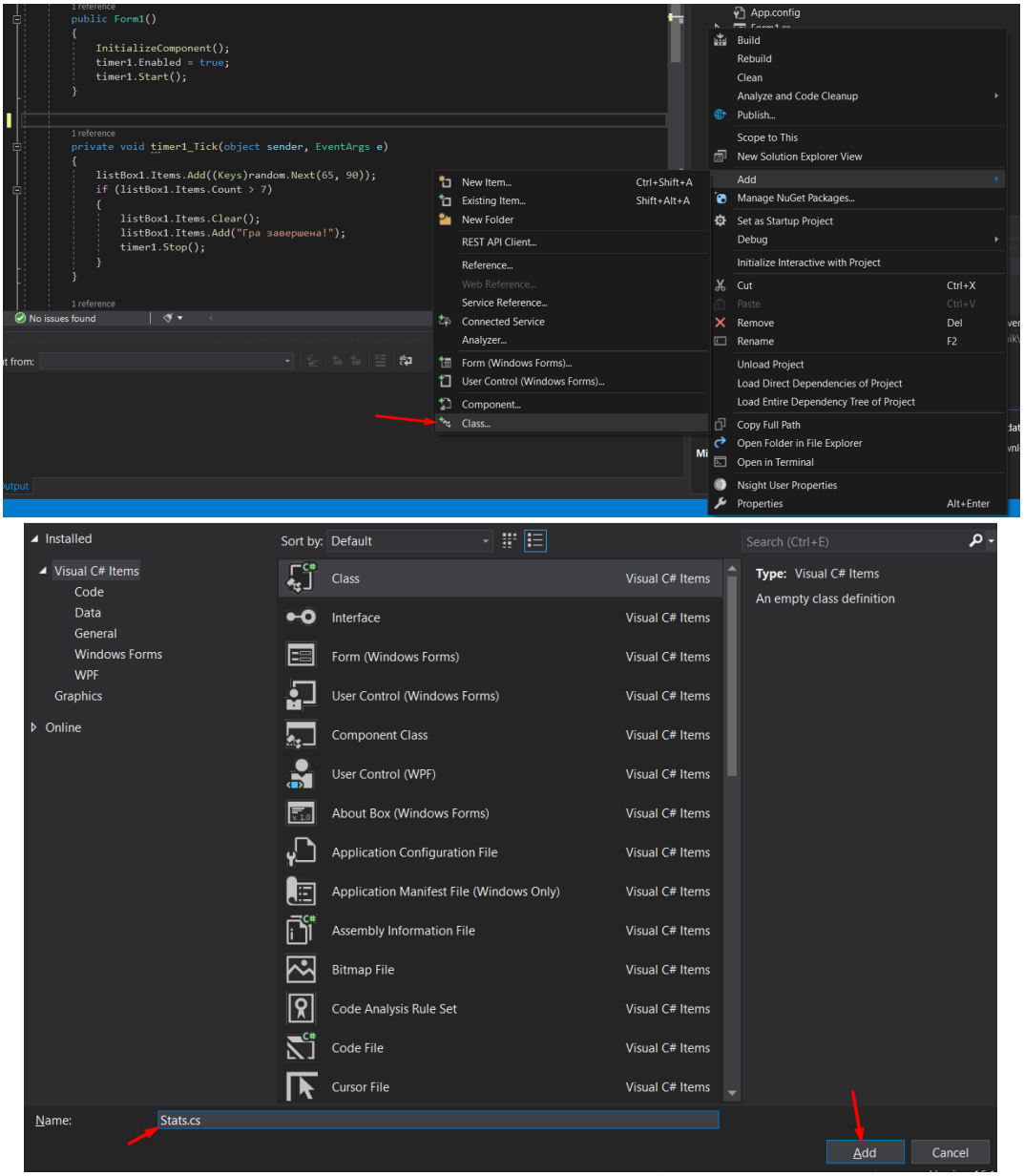


Рис. 8.7. Додаємо до проєкту новий клас Stats

12. Додайте в нього такий код:

```
namespace Type
{
    class Stats
    {
        public int Total = 0;
        public int Missed = 0;
        public int Correct = 0;
```

```
        public int Accuracy = 0;
        public void Update(bool correctKey)
        {
            Total++;
            if (!correctKey) Missed++;
            else Correct++;

            Accuracy = 100 * Correct / (Missed + Correct);
        }
    }

}
```

13. Поверніться до класу ***Form1.cs*** і створіть у ньому екземпляр класу Stats (рис. 8.8).

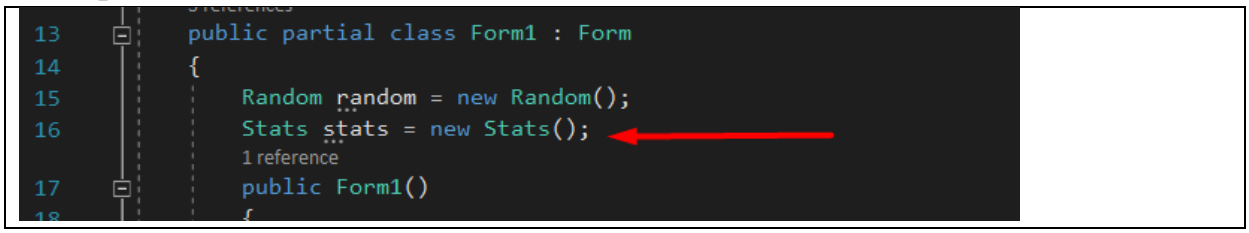


Рис. 8.8. Фрагмент коду для створення екземпляру класу Stats

14. У конструкторі форми виберіть форму та перейдіть у події. Створіть для форми подію Key_Down (натисніть 2 рази) (рис. 8.9).

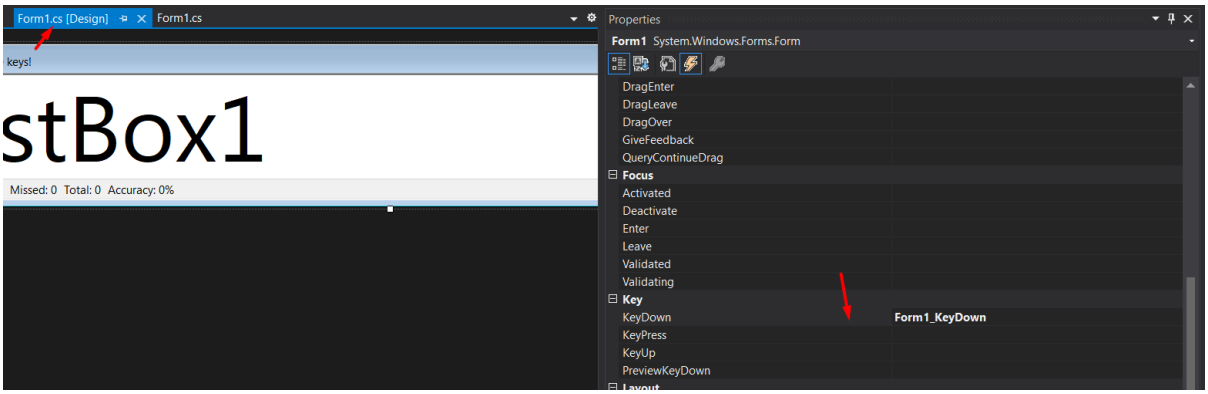


Рис. 8.9. Створення для форми події Key_Down

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            //Якщо користувач натискає клавіші, то літери видаляються,
            //а швидкість їх появи збільшується.
            if (listBox1.Items.Contains(e.KeyCode)) //Перевірка на правильність
натиснення.
            {
                listBox1.Items.Remove(e.KeyCode); //видалення літери з listBox
                listBox1.Refresh();
```

```
            if (timer1.Interval > 400) timer1.Interval -= 10;
            if (timer1.Interval > 250) timer1.Interval -= 7;
            if (timer1.Interval > 100) timer1.Interval -= 2;

            difficultyProgressBar.Value = 800 - timer1.Interval;
            //Заповнення ProgressBar
            stats.Update(true); //Оновлюємо статистику.
        }
        else
        {
            stats.Update(false);
        }

        correctLabel.Text = "Correct: " + stats.Correct;
        missedLabel.Text = "Missed " + stats.Missed;
        totalLabel.Text = "Total " + stats.Total;
        accuracyLabel.Text = "Accuracy: " + stats.Accuracy + "%";
    }
```

3. Контрольні запитання

1. Яке призначення елемента Timer?
2. Як задати час спрацьовування таймера?
3. Як увімкнути / вимкнути таймер?
4. Як налаштувати обробник події Tick елементу Timer?
5. Яке призначення елементу StatusStrip?
6. Яке призначення властивості Dock?
7. Які елементи можна додати в режимі дизайнера елементу StatusStrip?

ЛАБОРАТОРНА РОБОТА № 9
ЕЛЕМЕНТ TABLELAYOUTPANEL.
ДОДАТОК ХРЕСТИКИ-НУЛИКИ

Мета заняття: навчитися працювати з елементом TableLayoutPanel за допомогою Windows Forms мовою C# у середовищі Microsoft Visual Studio та реалізовувати логіку й інтерфейс гри «Хрестики-нулики».

1. Теоретичні відомості

Елемент TableLayoutPanel перевизначає панель і має у своєму розпорядженні дочірні елементи управління у вигляді таблиці, де для кожного елементу є своя комірка. Кількість рядків та стовпців таблиці задається властивості Rows і Columns. Під час вибору одного з цих пунктів у вікні Properties (Властивості) відобразиться вікно для налаштування стовпців і рядків (рис. 9.1).

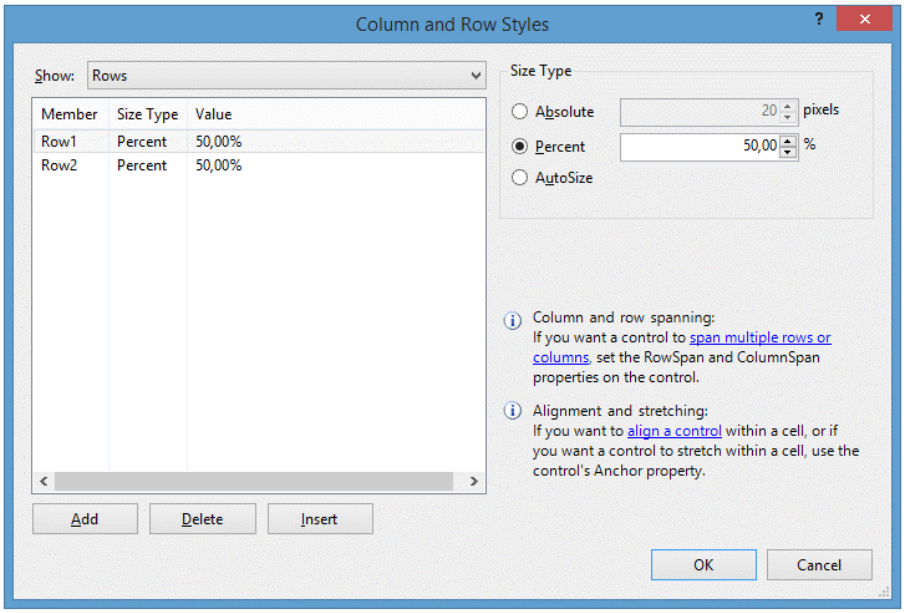


Рис. 9.1. Вікно властивостей Rows та Columns елементу TableLayoutPanel

Поле Size Type використовується для вказання розміру стовпців / рядків:

Absolute – задається абсолютна величина для рядків або стовпців у пікселях;

Percent – задається відносний розмір у відсотках (елементи управління в комірках таблиці автоматично масштабуються під час зміни розмірів форми);

AutoSize – висота рядків і ширина стовпців задається автоматично залежно від розміру найбільшої в рядку або стовпці комірки.

У вікні (рис. 9.1) можемо додати або видалити рядки і стовпці. У коді динамічно можемо змінювати значення стовпців і рядків. Причому всі стовпці представлені типом ColumnStyle, а рядки – типом RowStyle:

```
tableLayoutPanel1.RowStyles[0].SizeType = SizeType.Percent;
tableLayoutPanel1.RowStyles[0].Height = 40;
tableLayoutPanel1.ColumnStyles[0].SizeType = SizeType.Absolute;
tableLayoutPanel1.ColumnStyles[0].Width = 50;
```

Додавання елементу в контейнер TableLayoutPanel має свої особливості. Можна додати його або в наступну вільну комірку, або явно вказати елемент таблиці:

```
Button saveButton = new Button();
// додаємо кнопку в наступну вільну комірку
tableLayoutPanel1.Controls.Add(saveButton);
// додаємо кнопку в комірку (2,2)
tableLayoutPanel1.Controls.Add(saveButton, 2, 2);
```

2. Практичне завдання

 У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв'язання задачі.

«Хрестики-нулики» є відомою грою логічного типу на полі 3 × 3. Гравці по черзі ставлять свій знак «X» або «O» так, щоб по прямій або діагоналі виникла максимальна кількість його знаків. У такому випадку гравець є переможцем.

1. Створіть проєкт ***Windows Forms*** (.NET Framework) з назвою ***NoughtsCrosses.*** Для властивостей форми виставте такі значення:

```
FormBorderStyle = Fixed3D;  Size = 594, 494.
```

2. Додайте на форму елемент ***tableLayoutPanel.*** Для цього елементу виставте такі значення:

```
Dock = Fill;
BackColor = White;
Columns = 3 (Percent =33,33%);
Rows = 3 (Percent =33,33%);
```

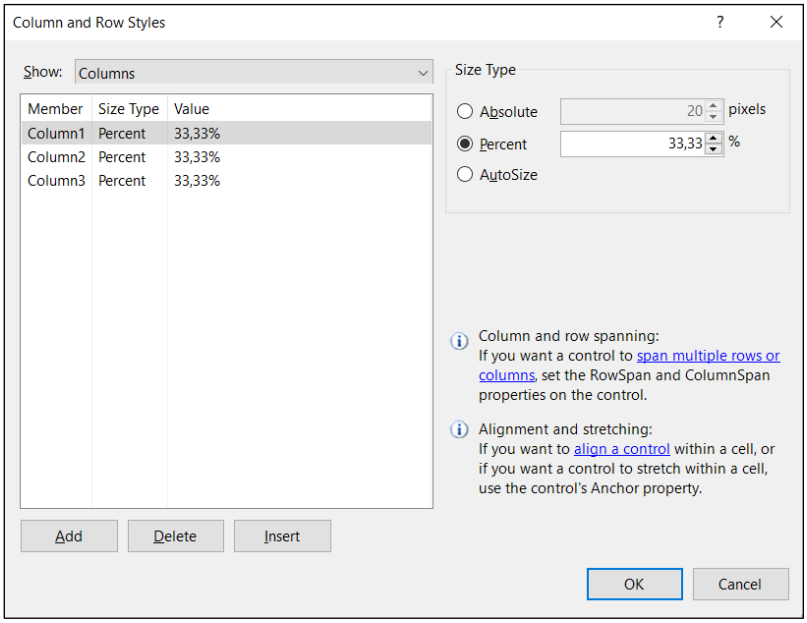


Рис. 9.2. Налаштування елементів tableLayoutPanel

3. Додайте на форму елементи ***button1 - button9*** (рис. 9.3).

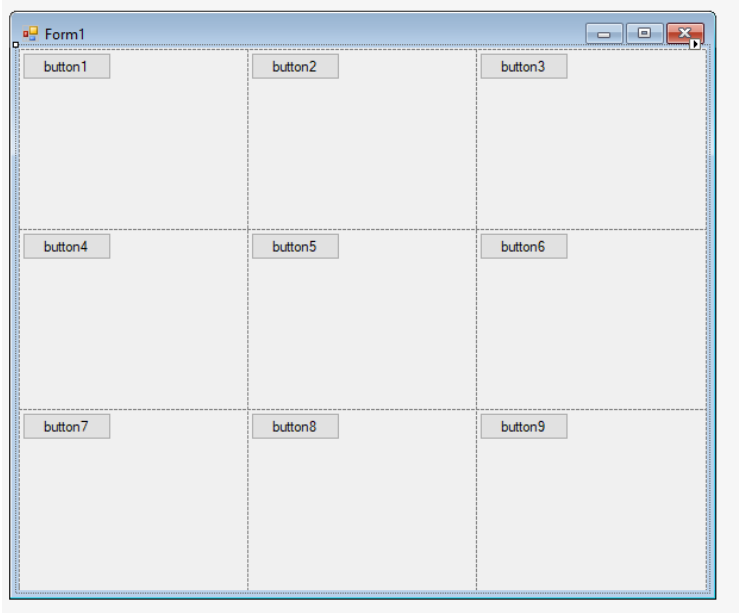


Рис. 9.3. Додавання елементів Button до tableLayoutPanel

4. Налаштуйте властивості ***button1 – button9*** (рис. 9.4):

```
BackgroundImageLayout - Zoom;
FlatStyle - Flat;
FlatAppearance.BorderSize = 0;
Font - Algerian 70pt;
ForeColor - Silver;
Dock - Fill;
Text - _.
```

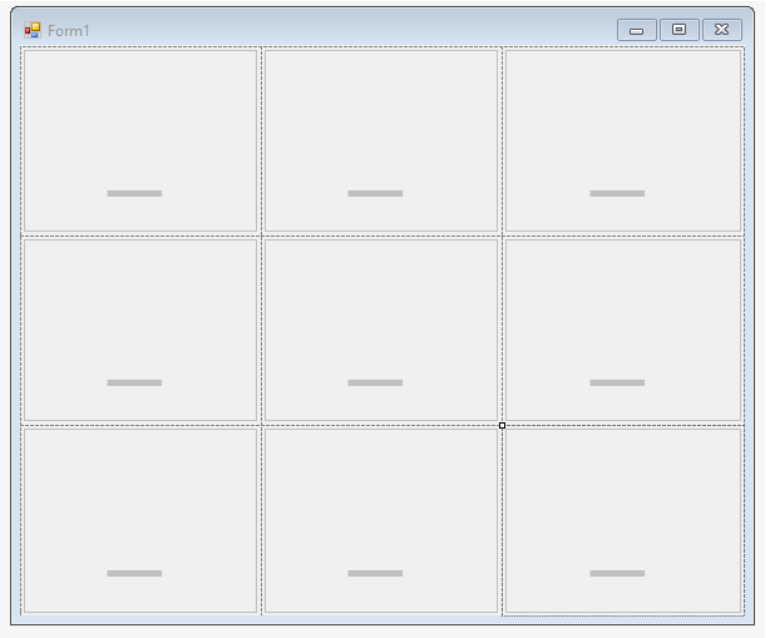


Рис. 9.4. Налаштування властивостей елементів Button

5. Створіть обробник події натисненням ***button1*.** Відредагуйте код:

```
bool turnX0 = true;
string symbolFirstPlayer = "X";
string symbolSecondPlayer = "0";
private void button1_Click(object sender, EventArgs e)
{
    Button currentButton = (Button)sender;
    if (currentButton.Text == "_")
    {
        currentButton.ForeColor = Color.Teal;
        currentButton.Text = UpdateTurn();
        IsVinner(symbolFirstPlayer);
        IsVinner(symbolSecondPlayer);
    }
}
         string UpdateTurn()
{
    string turnSymbol;

    if (turnX0)
        turnSymbol = symbolFirstPlayer;
    else
        turnSymbol = symbolSecondPlayer;
    turnX0 = !turnX0;

    return turnSymbol;
}
void IsVinner(string symbol)
{
    bool result = CheckWinner(symbol);
    if (result)
    {
```

```
                MessageBox.Show("Player " + symbol + " win!");
                ClearField();
            }
        }
                void ClearField()
        {
            int elemCount = tableLayoutPanel1.Controls.Count;
            for (int i = 0; i < elemCount; i++)
            {
                tableLayoutPanel1.Controls[i].Text = "_";
                tableLayoutPanel1.Controls[i].ForeColor = Color.Silver;
            }
        }
        bool CheckWinner(string symbol)
        {
            if (button1.Text == symbol && button2.Text == symbol && button3.Text
== symbol ||
                button4.Text == symbol && button5.Text == symbol && button6.Text
== symbol ||
                button7.Text == symbol && button8.Text == symbol && button9.Text
== symbol ||
                button1.Text == symbol && button4.Text == symbol && button7.Text
== symbol ||
                button2.Text == symbol && button5.Text == symbol && button8.Text
== symbol ||
                button3.Text == symbol && button6.Text == symbol && button9.Text
== symbol ||
                button1.Text == symbol && button5.Text == symbol && button9.Text
== symbol ||
                button3.Text == symbol && button5.Text == symbol && button7.Text
== symbol)
            {
                return true;
            }
            return false;
 }
```

6. Прив’яжіть обробник події ***button1_Click*** до кнопок «***button2–button9»***. Протестуйте проєкт.

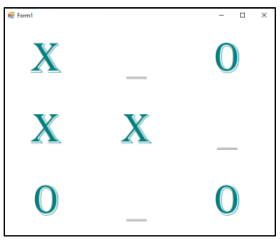


Рис. 9.5. Тестування роботи додатка «Хрестики-нулики»

3. Контрольні запитання

1. Яке призначення елементу TableLayoutPanel?
2. Як задати кількість рядків та стовпців?
3. Як зробити розмір комірок гнучким? Як це реалізувати програмно?
4. Як зробити розмір комірок фіксованим? Як це реалізувати програмно?
5. Як автоматично додавати елементи керування до комірок таблиці?
6. Яке призначення властивості Dock елементу TableLayoutPanel?
7. Які елементи можна додати в режимі дизайнера до елементу TableLayout Panel?

ЛАБОРАТОРНА РОБОТА № 10
ПРОЄКТ MYUTILITES. РОБОТА З ЕЛЕМЕНТОМ MENUSTRIP, TABCONTROL, NUMERICUPDOWN, RICHTEXTBOX

Мета заняття: навчитися працювати з елементом MenuStrip для створення меню та підменю додатка, TabControl для створення вкладок додатка, Numeric UpDown для створення діапазону значень генератора випадкових чисел, Rich TextBox для створення текстового редактора; навчитися працювати з обробником подій Click елементів MenuStrip, Button, checkBox та виводом інформації у елемент TexBox за допомогою Windows Forms мовою C# у середовищі Microsoft Visual Studio.

1. Теоретичні відомості

1.1. MenuStrip

Для створення меню у Windows Forms використовується елемент MenuStrip. Основні властивості MenuStrip:

Dock – дає змогу закріпити меню до певної сторони форми;

LayoutStyle – визначає спосіб розташування елементів меню на формі. Можливі варіанти:

– ***HorizontalStackWithOverflow*** – горизонтальне розташування з переповненням; елементи, що не вміщуються, не відображаються;

– ***StackWithOverflow*** – автоматичне розташування елементів із переповненням;

– ***VerticalStackWithOverflow*** – вертикальне розташування з переповненням;

– ***Flow*** – автоматичне розміщення без переповнення; якщо панель менша за контейнер, елементи переносяться;

– ***Table*** – розміщення елементів у вигляді таблиці;

– ***ShowItemToolTips*** – вмикає чи вимикає відображення підказок для пунктів меню;

– ***Stretch*** – дає змогу розтягнути меню на всю ширину контейнера;

– ***TextDirection*** – визначає напрямок відображення тексту у пунктах меню.

MenuStrip є контейнером для пунктів меню, що представлені об’єктами ToolStripMenuItem. Нові елементи можна додавати як у режимі конструктора (рис. 10.1), так і програмно.

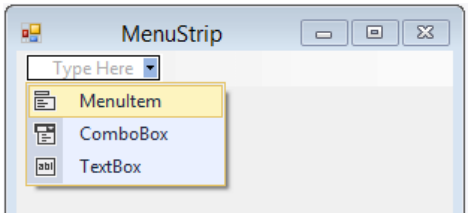


Рис. 10.1. MenuStrip. Режим дизайнера

У меню можна додавати три типи елементів:

– MenuItem (об’єкт ToolStripMenuItem),
– ComboBox,
– TextBox.

Хоча спадні списки й текстові поля частіше застосовуються у панелях інструментів, їх можна використовувати й у меню (рис. 10.2).

```
1 reference
public MainForm()
{
    InitializeComponent();

    ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");

    fileItem.DropDownItems.Add("Створити");
    fileItem.DropDownItems.Add(new ToolStripMenuItem("Зберегти"));

    menuStrip1.Items.Add(fileItem);

    ToolStripMenuItem aboutItem = new ToolStripMenuItem("Про програму");
    aboutItem.Click += tsmiAbout_Click;
    menuStrip1.Items.Add(aboutItem);
}

0 references
private void aboutItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("Про програму");
}
```

Рис. 10.2. Додавання пунктів меню у коді програми

Об’єкт ToolStripMenuItem створюється через конструктор із текстовою міткою, яка використовується як підпис пункту меню. Кожен такий об’єкт містить колекцію DropDownItems, де зберігаються вкладені пункти меню. Отже, меню може мати ієрархічну структуру у вигляді дерева.

Якщо під час додавання передати рядок тексту, об’єкт ToolStripMenuItem створюється автоматично, наприклад: fileItem.DropDownItems.Add («Створити») (рис. 10.3).

Рис. 10.3. Створення елементу «Створити» MenuItem «Файл»

Призначивши обробники для події Click, можна реалізувати реакцію на натискання пунктів меню, наприклад: aboutItem.Click += aboutItem_Click.

Щоб швидко викликати потрібний пункт меню, використовуються клавіші швидкого доступу, які задаються через властивість ShortcutKeys (рис. 10.4). Для цього застосовується перелік Keys. Наприклад, під час використання комбінації Ctrl + P автоматично виконується дія пункту меню «Зберегти».

Меню можна зробити більш наочним, додавши зображення. Для цього передбачені такі властивості:

– ***DisplayStyle*** – визначає, чи відображається текст, зображення, або і те, і інше;

– ***Image*** – задає використане зображення;

– ***ImageAlign*** – вирівнювання зображення відносно пункту меню;

– ***ImageScaling*** – визначає, чи буде масштабуватися картинка;

– ***ImageTransparentColor*** – задає прозорий колір зображення.

```
13      public partial class MainForm : Form
14      {
15
16
            1 reference
17          public MainForm()
18          {
19              InitializeComponent();
20
21              ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
22
23              ToolStripMenuItem saveItem = new ToolStripMenuItem("Зберегти") { Checked = true, CheckOnClick = true };
24              saveItem.Click += saveItem_Click;
25              saveItem.ShortcutKeys = Keys.Control | Keys.P;
26
27              fileItem.DropDownItems.Add(saveItem);
28              menuStrip1.Items.Add(fileItem);
29
30          }
31
            1 reference
32          private void saveItem_Click(object sender, EventArgs e)
33          {
34              MessageBox.Show("Збереігання");
35          }
```

Рис. 10.4. Створення клавіш швидкого доступу у коді програми

У середовищі дизайнера вибір картинки виконується через властивість Image, після чого відкривається вікно імпорту зображення в проєкт (рис. 10.5).

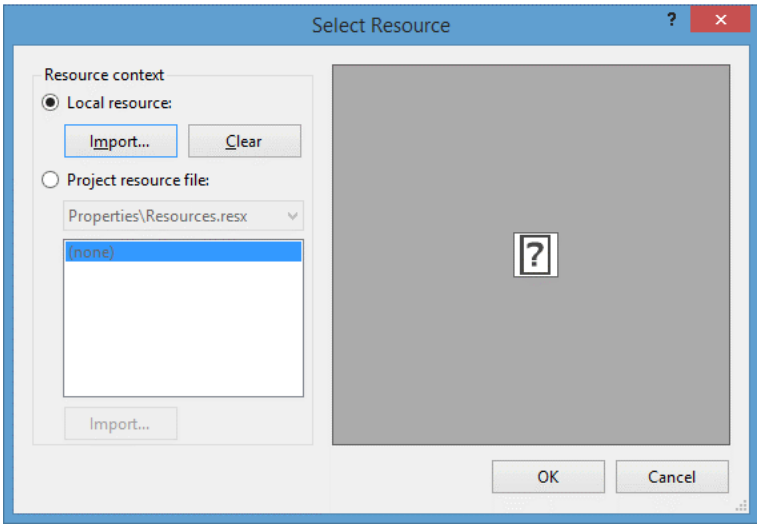


Рис. 10.5. Встановлення зображення пункту меню у режимі дизайнера

Щоб вказати, як розмістити зображення, у властивості DisplayStyle треба встановити значення Image. Якщо хочемо, щоб кнопка відображала тільки текст, то треба вказати значення Text, або можна комбінувати два значення за допомогою іншого значення ImageAndText. За замовчуванням зображення розміщується зліва від тексту (рис. 10.6):

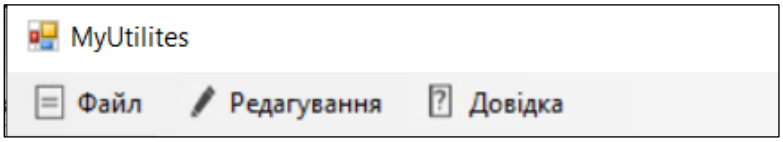


Рис. 10.6. Приклад відображення зображення та тексту пункту меню

Також можна встановити зображення пункту меню в коді програми:

```
fileToolStripMenuItem.Image = Image.FromFile(@"D:\Icons\block32.png");
```

1.2. TabControl

Елемент TabControl використовується для створення багатовкладкового інтерфейсу. Кожна вкладка (об’єкт класу TabPage) може містити різні елементи керування – кнопки, поля введення тощо.

Вкладки керуються колекцією TabPages. За замовчуванням під час додавання TabControl на форму створюються дві вкладки (tabPage1 і tabPage2), їх можна перейменувати через властивість Text (рис. 10.7–10.8).

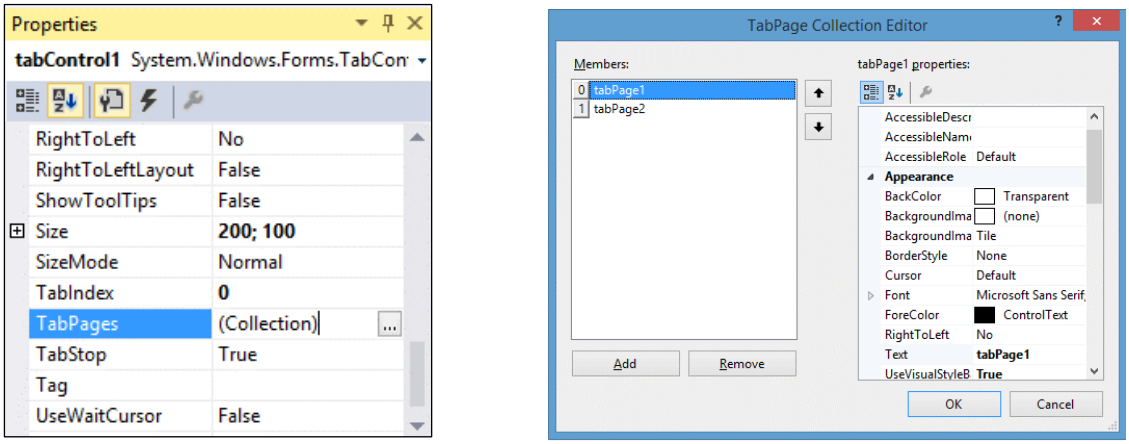


Рис. 10.7. Властивість TabPages елементу TabControl

Рис. 10.8. Зовнішній вигляд вкладок на формі

Для додавання нової вкладки можна її програмно створити і додати в колекцію **tabControl1.TabPages** за допомогою методу **Add:**

```
//додавання вкладки
TabPage newTabPage = new TabPage();
newTabPage.Text = "Розрахунок";
tabControl1.TabPages.Add(newTabPage);
```

Видалення вкладки:

```
// за індексом
tabControl1.TabPages.RemoveAt(0);
// за об'єктом
tabControl1.TabPages.Remove(newTabPage);
```

Отримавши в колекції tabControl1.TabPages потрібну вкладку за індексом, можемо їй задавати властивості:

```
// налаштування властивостей
tabControl1.TabPages[0].Text = "Перша вкладка";
```

1.3. NumericUpDown

Елемент **NumericUpDown** дає змогу вводити числа в заданому діапазоні.

– Мінімальне й максимальне значення визначаються властивостями **Minimum** та **Maximum.**

– Поточне значення зберігається у властивості **Value** (рис. 10.9).

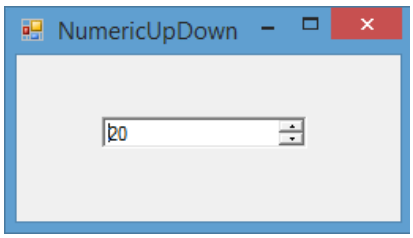


Рис. 10.9. Властивість Value елемента NumericUpDown

За замовчуванням елемент відображає десяткові числа. Однак якщо встановимо його властивість Hexadecimal рівного true, то елемент буде відображати всі числа в шістнадцятковій системі. Навіть якщо в коді встановимо десяткове значення, то елемент все одно відобразить його в шістнадцятковій системі:

```
numericUpDown1.Value = 66;
```

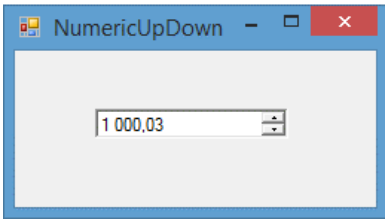


Рис. 10.10. Властивості DecimalPlaces та ThousandsSeparator елементу NumericUpDown

Якщо необхідно відображати в полі дробові числа, то можна використовувати властивість **DecimalPlaces,** яка вказує, скільки знаків після коми повинно відображатися. За замовчуванням ця властивість дорівнює нулю. Також можна задати відображення роздільника тисяч. Для цього властивість **Thousands Separator** необхідно встановити у значення true. Наприклад, із кодом елемент NumericUpDown буде мати вигляд (рис. 1.10):

```
numericUpDown при Value=1000,03,
DecimalPlaces=2
ThousandsSeparator=true:
```

За замовчуванням під час натискання на стрілочки вгору-вниз на елементі значення буде збільшуватися або зменшуватися на одиницю. Але за допомогою властивості Increment можна задати інший крок збільшення, зокрема і дробовий.

Під час роботи з NumericUpDown треба враховувати, що його властивість Value (як і властивості Minimum і Maximum) зберігає значення decimal. Тому в коді також повинні з ним працювати як з decimal, а не як з типом int або double.

Елемент **DomainUpDown** призначений для введення текстової інформації. Він має текстове поле для введення рядка і дві стрілки для переміщення за списком рядків (рис. 10.11).

Рис. 10.11. Елемент DomainUpDown

Список для DomainUpDown задається за допомогою властивості **Items.** Список можна відразу впорядкувати за алфавітом. Для цього треба властивості **Sorted** привласнити значення true.

Щоб можна було циклічно переміщатися по списку, тобто за досягнення кінця або початку списку його перегляд починався з першого або останнього елементу, треба встановити для властивості **Wrap** значення true.

У коді вибране значення в DomainUpDown доступне через властивість Text. Наприклад, додамо програмно список рядків у **DomainUpDown** і опрацюємо зміну вибору у списку:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        List<string> states = new List<string>
        {
            "Аргентина", "Бразилія", "Венесуела", "Колумбія", "Чилі"
        };
        // додаємо список елементів
        domainUpDown1.Items.AddRange(states);
        domainUpDown1.TextChanged += domainUpDown1_TextChanged;
    }
    // обробка зміни тексту в елементі
    void domainUpDown1_TextChanged(object sender, EventArgs e)
    {
        MessageBox.Show(domainUpDown1.Text);
    }
}
```

Для обробки зміни тексту можна використовувати подію TextChanged, в обробнику якого виводимо обраний текст у повідомлення.

1.4. RichTextBox

Елемент **RichTextBox** використовується для введення та редагування великих обсягів тексту (понад 64 КБ) із форматуванням. Він підтримує:

– зміну кольору та шрифтів,
– вставку зображень,
– форматування подібне до Microsoft Word.

Текст задається й отримується через властивість **Text.** Загальний вигляд показано на рис. 10.12.

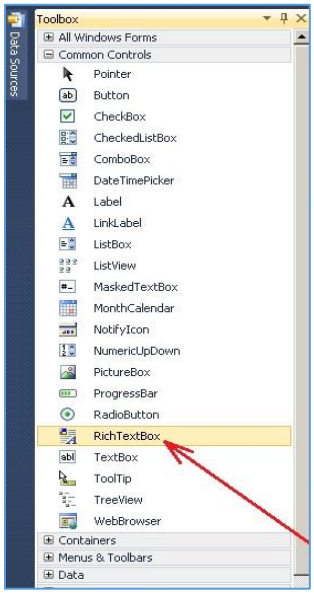


Рис. 10.12. Елемент управління типу RichTextBox

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт для розв’язання задачі. Створити меню File, Help додатка MyUtilites.

1. Запустіть Visual Studio. Виберіть «створити додаток Windows». Назва проєкту – «MyUtilites».

2. Додайте Menu. Для цього на панелі інструментів знайдіть MenuStrip (рис. 10.13).

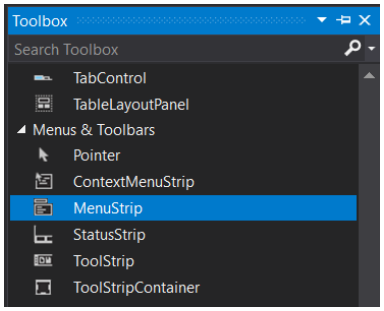


Рис. 10.13. Вибір елементу MenuStrip на панелі інструментів

3. Перейменуйте назву Form1 на MainForm, Text на «MyUtilites» (рис. 10.14).

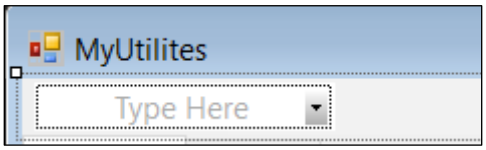


Рис. 10.14. Форма «MyUtilites» з елементом MenuStrip

5. Створіть елементи меню: File, Help (рис. 10.15).

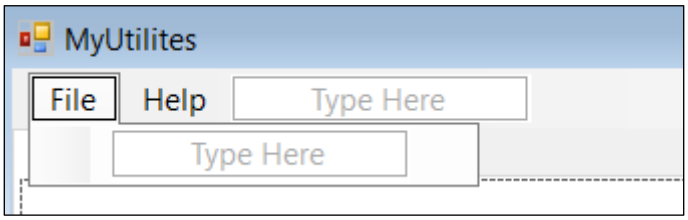


Рис. 10.15. Елемент MenuStrip. Режим дизайнера

5. Створіть підменю File → Exit, Help → About (рис. 10.16).

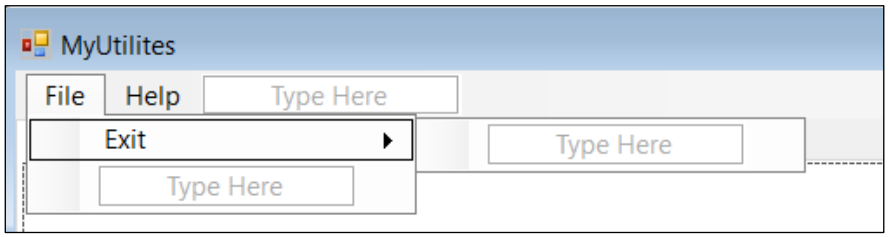


Рис. 10.16. Елемент MenuStrip. Режим дизайнера. Створення підменю

6. Перейменуйте поле Name підменю Exit на tsmiExit та створіть для нього подію Click.

```
namespace MyUtilites
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void tsmiExit_Click(object sender, EventArgs e)
        {

        }
    }
}
```

7. У конструктор події додайте метод this.Close();

```
namespace MyUtilites
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void tsmiExit_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

8. Перейменуйте поле Name підменю About на tsmiAbout. Створіть для елементу подію Click, у якій додайте метод MessageBox.Show(); в дужках у лапках напишіть текст «Програма MyUtilites містить набір невеликих програм, які можуть стати у нагоді. А головне – навчить мене основам програмування на C#. Автор: Іванов Іван» (рис. 10.17). Можна змінити прізвище та ім'я автора на власне. Можна додати перенесення тексту на новий рядок через \n.

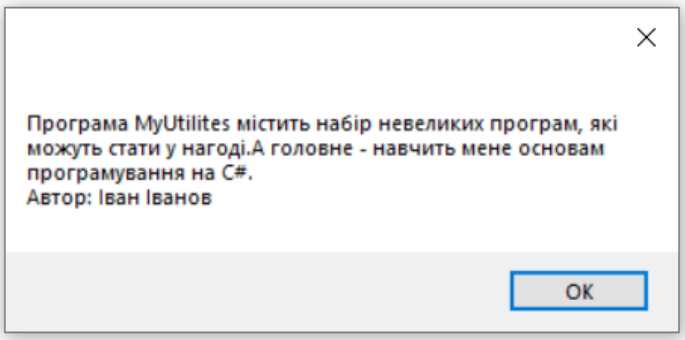


Рис. 10.17. Метод MessageBox.Show події Click підменю About

9. Додайте заголовок вікна (рис. 2.6). Для цього в MessageBox.Show() у дужках після тексту напишіть через кому в лапках заголовок («…….», «Про програму»).

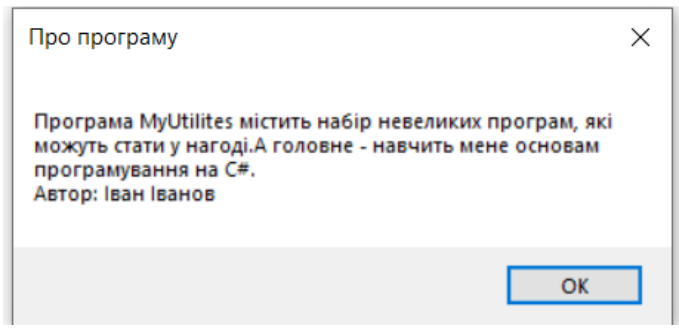


Рис. 10.18. Вікно MessageBox, яке викликається після натиснення About

10. Перевірте роботу елементів меню додатка MyUtilites (Ctrl + F5).

Завдання 2. Створити TabPages та додаток-лічильник.

1. На панелі елементів виберіть елемент TabControl і перетягніть його на форму проєкту (рис. 10.19). Властивості Dock встановіть значення Fill (рис. 10.20а).

2. Виберіть властивість TabPages «Collection» (рис. 10.20б). У вікні, що відкрилось, змінити властивість Text у TabPage на «Лічильник» (рис. 10.21).

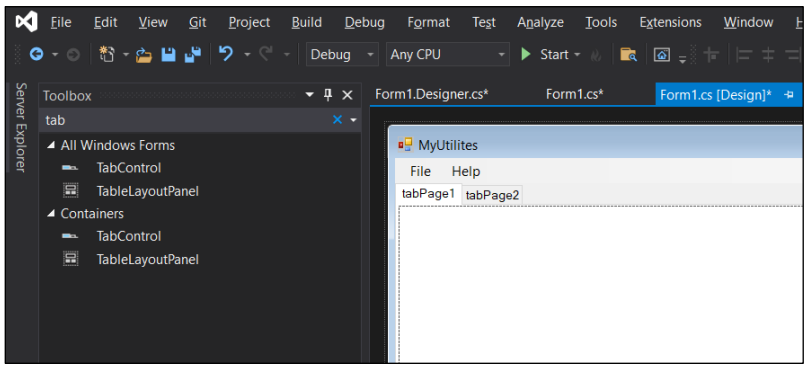
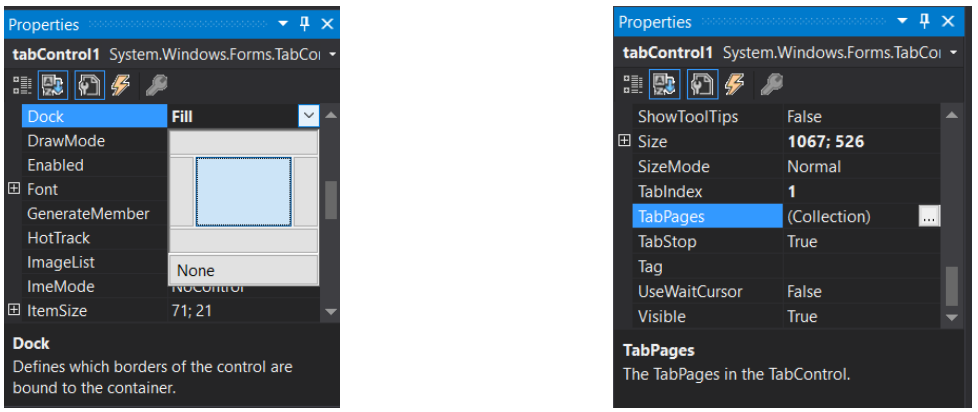


Рис. 10.19. Панель керування. Вибір елементу TabControl



а) б)

Рис. 10.20. Налаштування властивості Dock та Collection TabControl

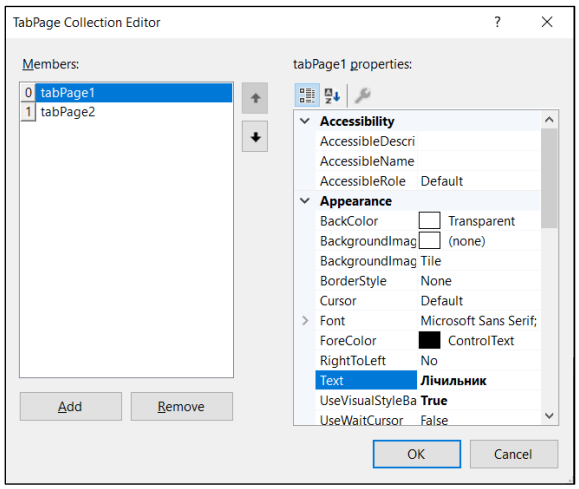


Рис. 10.21. Встановлення значення властивості Text елементу TabPage

3. До проєкту додайте три кнопки з назвами («+», «-», «Reset») та мітку Label1 (рис. 10.22).

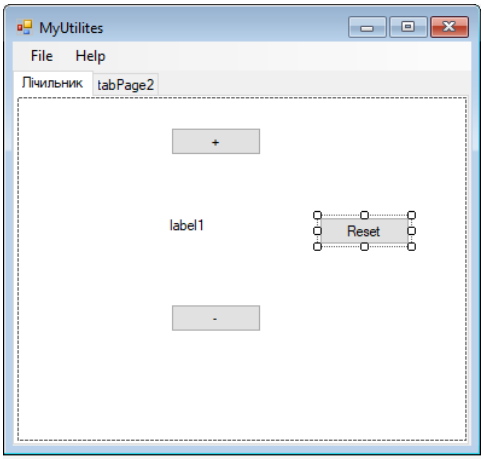


Рис. 10.22. Додавання до проєкту елементів Button, Label

4. Властивість кнопок Name змініть на btnPlus, btnMinus, btnReset. Властивість Name мітки змініть на lblCount, а властивість Text на 0. Для кнопок та мітки встановіть властивість Font 14.

5. У конструкторі коду додайте int count = 0 (рис. 10.23).

```
namespace MyUtilites
{
    public partial class MainForm : Form
    {
        int count=0;

        public MainForm()
        {
```

Рис. 10.23. Додавання змінної count типу int

6. Додайте обробник подій для кнопки «+» (рис. 10.24).

```
private void button1_Click(object sender, EventArgs e)
{
    count++;
    lblCount.Text = count.ToString();
}
```

Рис. 10.24. Код обробника події Click кнопки btnPlus

7. Додайте обробник подій для кнопки «-» (рис. 10.25).

```
private void button2_Click(object sender, EventArgs e)
{
    count--;
    lblCount.Text = count.ToString();
}
```

Рис. 10.25. Код обробника події Click кнопки btnMinus

8. Додайте обробник подій для кнопки «Reset» (рис. 10.26).

```
private void button3_Click(object sender, EventArgs e)
{
    count=0;
    lblCount.Text = Convert.ToString(count);
}
```

Рис. 10.26 – Код обробника події Click кнопки btnReset

9. Перевірте роботу елементів меню додатка MyUtilites (Ctrl + F5).

Завдання 3. Створити додаток-генератор.

1. Виберіть властивість TabPages «Collection». У вікні, що відкрилось, змініть властивість Text у TabPage2 на «Генератор» (рис. 10.27).

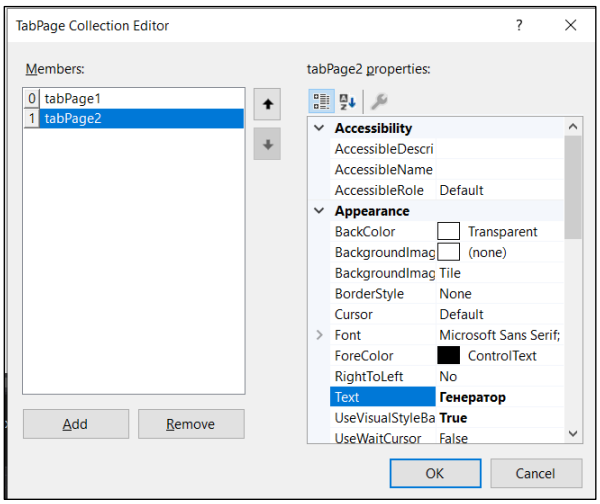


Рис. 10.27. Перейменування властивості Text у TabPage2

2. До проєкту додайте кнопку, 3 мітки та 2 елементи NumericUpDown (рис. 10.28).

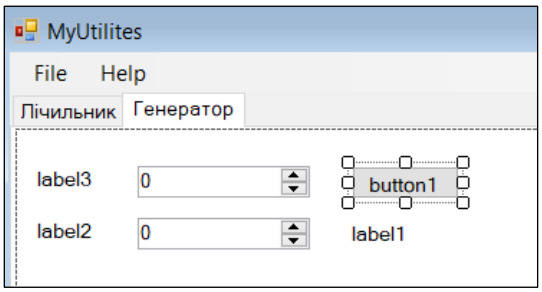


Рис. 10.28. Проєктування дизайну MyUtilites. Вкладка «Генератор»

3. Перейменуйте елементи (властивість «Text») відповідно до рис. 10.29. Властивість Name кнопки змініть на btnRandom, Name мітки Label1 на lblRandom.

Задайте величину за замовчуванням для елементів NumericUpDown значення Value – 1 та 6.

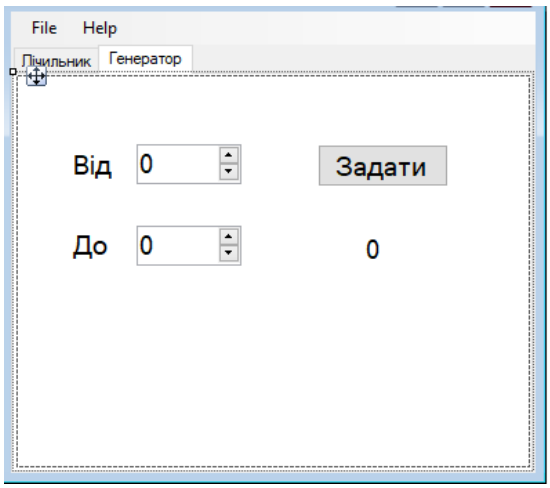


Рис. 10.29. Дизайн додатка MyUtilites. Вкладка «Генератор»

4. Створіть елемент rnd класу Random (рис. 10.30).

```
public partial class MainForm : Form
{
    int count=0;
    Random rnd;

    public MainForm()
    {
        InitializeComponent();
        rnd = new Random();
    }
```

Рис. 10.30. Створення елементу rnd класу Random

5. Додайте обробник подій для кнопки «Задати» (рис. 10.31).

```
private void btnRandom_Click(object sender, EventArgs e)
{
    int n;
    n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value)+1);
    lblRandom.Text = n.ToString();
}
```

Рис. 10.31. Код обробника події Click кнопки «Задати»

6. Додайте на форму елемент TextBox та задайте властивість Name tbRandom. Встановіть режим MultiLine та властивість ScrollBars - Vertical (рис. 10.32).

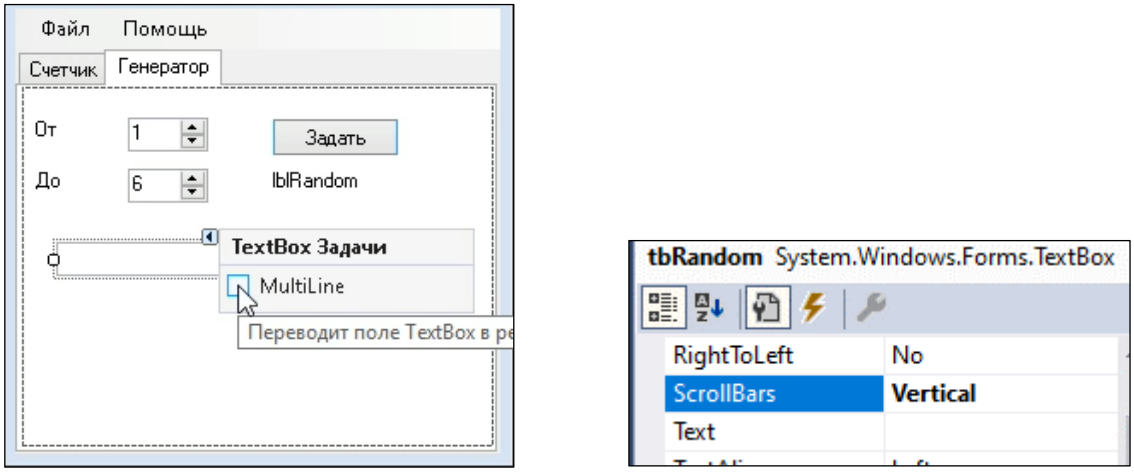


Рис. 10.32. Налаштування властивостей елементу TextBox

7. Додайте код виводу в «TextBox» (рис. 10.33).

```
private void btnRandom_Click(object sender, EventArgs e)
{
    int n;
    n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value
    lblRandom.Text = n.ToString();
    tbRandom.AppendText(n + "\r\n");
}
```

Рис. 10.33. Код обробника події Click кнопки btnRandom

8. Додайте кнопку «Очистити» (рис. 10.34). Name – btnRandomClear, Text – «Очистити». Створіть обробник події (рис. 10.35).

```
private void btnRandomClear_Click(object sender, EventArgs e)
{
    tbRandom.Clear();
}
```

Рис. 10.34. Код обробника події Click кнопки «Очистити»

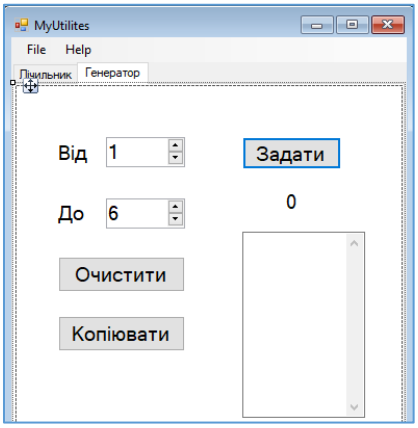


Рис. 10.35. Оновлений дизайн додатка MyUtilites. Вкладка «Генератор»

9. Додайте кнопку копіювання в буфер обміну (рис. 10.36). Name – btnRandomCopy, Text – «Копіювати». Створіть обробник події (рис. 10.36).

```
private void btnRandomCopy_Click(object sender, EventArgs e)
{
    Clipboard.SetText(tbRandom.Text);
}
```

Рис. 10.36. Код обробника події Click кнопки «Копіювати»

10. Організуйте додавання чисел до textBox, які раніше не зустрічались. Для цього в обробник події кнопки btnRandom допишіть код (рис. 10.37).

```
private void btnRandom_Click(object sender, EventArgs e)
{
    int n;
    n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value)+1
    lblRandom.Text = n.ToString();
    if (tbRandom.Text.IndexOf(n.ToString())==-1)
        tbRandom.AppendText(n + "\r\n");
}
```

Рис. 10.37. Оновлений код обробника події Click кнопки btnRandom

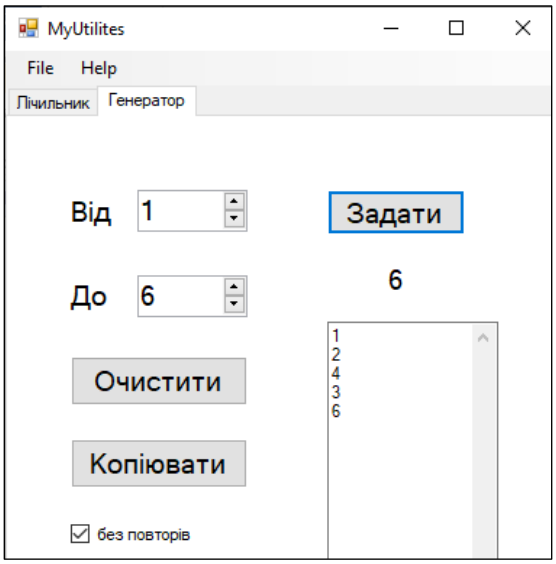


Рис. 10.38. Додавання елементу checkBox до дизайну додатка MyUtilites.

11. Додайте елемент checkBox, який буде визначати, додавати з повторами значення чи ні до textBox. Властивість Text перейменуйте «Без повторів», Name в chRandom. Змініть код обробника події Click кнопки btnRandom (рис. 10.38).

```
private void btnRandom_Click(object sender, EventArgs e)
{
    int n;
    n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value)+1);
    lblRandom.Text = n.ToString();
    if (cbRandom.Checked)
    {
        while (tbRandom.Text.IndexOf(n.ToString()) != -1)
            n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value) + 1);
    tbRandom.AppendText(n + "\r\n");
    }

    else tbRandom.AppendText(n + "\r\n");
}
```

Рис. 10.38. Код обробника події Click кнопки btnRandom, що враховує стан елементу checkBox

12. Змініть код обробника події Click кнопки btnRandom. Якщо числа усі вибрані, то більше нічого не додавайте. Для цього додайте змінну *i*, яка підраховує кількість натиснень кнопки. Якщо більше 1 000, то виходимо з циклу while, якщо ні, то пробуємо додати число (рис. 10.39).

```
private void btnRandom_Click(object sender, EventArgs e)
{
    int n;
    n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value)+1);
    lblRandom.Text = n.ToString();
    if (cbRandom.Checked)
    {
        int i = 0;
        while (tbRandom.Text.IndexOf(n.ToString()) != -1)
        {
            n = rnd.Next(Convert.ToInt32(numericUpDown1.Value), Convert.ToInt32(numericUpDown2.Value) + 1);
            i++;
            if (i > 1000) break;
        }
        if (i<1000) tbRandom.AppendText(n + "\r\n");
    }

    else tbRandom.AppendText(n + "\r\n");
}
```

Рис. 10.39. Оптимізований код обробника події Click кнопки btnRandom

13. Перевірте роботу елементів меню додатка MyUtilites (Ctrl + F5).

Завдання 4. Створити додаток Блокнот.

1. Додайте ще одну сторінку – «Блокнот». Виберіть елемент tabControl1 та властивість TabPages (рис. 10.40).

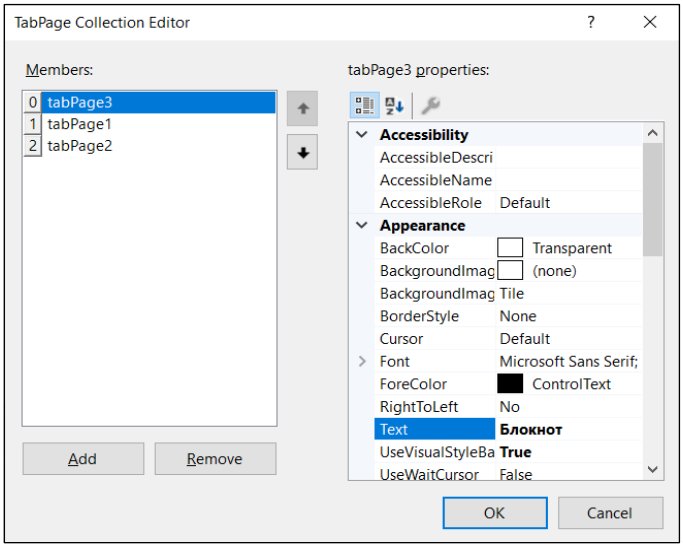


Рис. 10.40. Додавання TabPage3 (сторінка Блокнот) до елементу tabControl1

2. Додайте на форму елемент RichTextBox. Встановіть властивість Dock Fill, щоб розтягнути елемент на весь екран (рис. 10.41). Властивості Name задайте значення rtbNotePad.

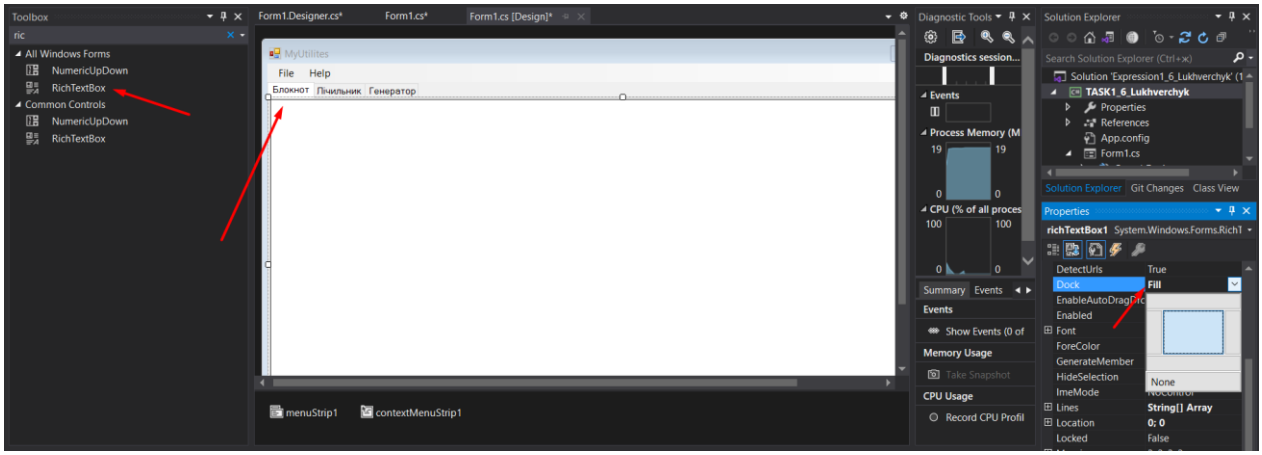


Рис. 10.41. Елемент RichTextBox та встановлення властивості Dock

3. Створіть новий пункт меню «Блокнот» та пункти підменю «Вставити дату», «Вставити час» (рис. 10.42).

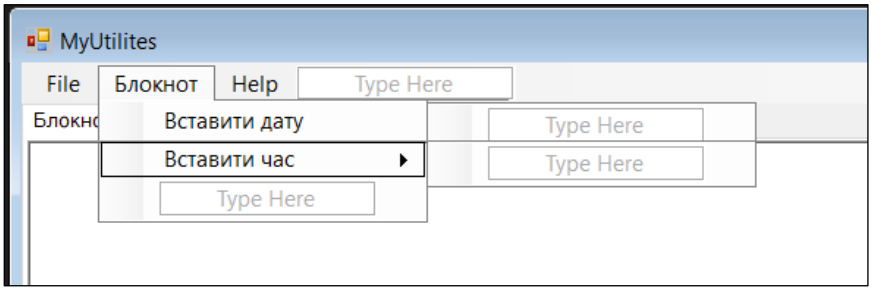


Рис. 10.42. MenuStrip. Додавання меню «Блокнот» та елементів підменю

4. Задайте властивість Name елементів підменю на tsmInsertDate, tsmInsert Time. Створіть обробники подій Click для «Вставити дату», «Вставити час» (рис. 10.43).

```
private void tsmInsertDate_Click(object sender, EventArgs e)
{
    rtbNotepad.AppendText(DateTime.Now.ToShortDateString() + "\n");
}

private void tsmInserttTime_Click(object sender, EventArgs e)
{
    rtbNotepad.AppendText(DateTime.Now.ToShortTimeString() + "\n");
}
```

Рис. 10.43. Код обробника подій Click для елементів підменю

5. Додайте до елементів підменю гарячі клавіші Ctrl + Shift + D / Ctrl + + Shift + T у властивості ShortcuKeys. Додайте новий пункт підменю «–» (створює лінію) (рис. 10.44) та елементи підменю «Save» і «Load» (рис. 10.45).

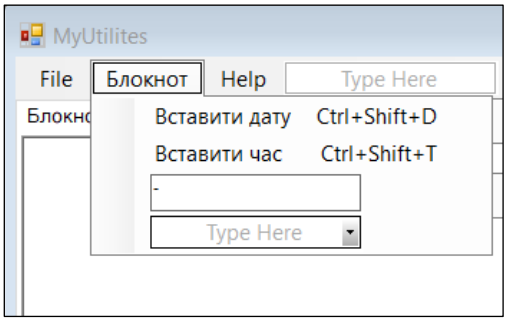


Рис. 10.44. Створення розділової лінії в підменю

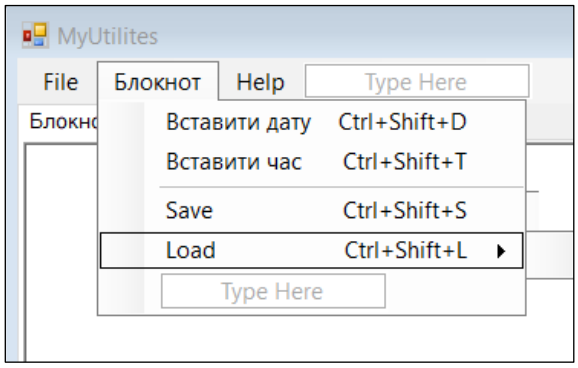


Рис. 10.45. Елементи меню «Блокнот»

6. Перевірте, як працюють елементи Ctrl + F5. Додайте до елементів гарячі клавіші Ctrl + Shift + S / Ctrl + Shift + L.

7. Для усунення помилок з файлами додайте елементи try та catch в обробник подій Click елементів підменю Save / Load (рис. 10.46).

```
private void tsmiSave_Click(object sender, EventArgs e)
{
    try
    {
        rtbNotepad.SaveFile("notepad.rtf");
    }
    catch
    {
        MessageBox.Show("Помилка при збережені");
    }
}

private void tsmiLoad_Click(object sender, EventArgs e)
{
    try
    {
        rtbNotepad.LoadFile("notepad.rtf");
    }
    catch
    {
        MessageBox.Show("Помилка при завантажені");
    }
}
```

Рис. 10.46. Код обробника події Click для елементів підменю Save, Load

8. Створіть метод, щоб файл завантажувався автоматично під час запуску блокнота (рис. 10.47).

```
private void tsmiSave_Click(object sender, EventArgs e)
{
    try
    {
        rtbNotepad.SaveFile("notepad.rtf");
    }
    catch
    {
        MessageBox.Show("Помилка при збережені");
    }
}
 void LoadNotepad ()
{
    try
    {
        rtbNotepad.LoadFile("notepad.rtf");
    }
    catch
    {
        MessageBox.Show("Помилка при завантажені");
    }
}
private void tsmiLoad_Click(object sender, EventArgs e)
{
    LoadNotepad();

}
```

Рис. 10.47. Код обробника події Click для елементів підменю Save, Load та методу LoadNotepad ()

9. Додайте для форми подію Load (рис. 10.48), у яку треба додати виклик методу LoadNotepad () (рис. 10.49).

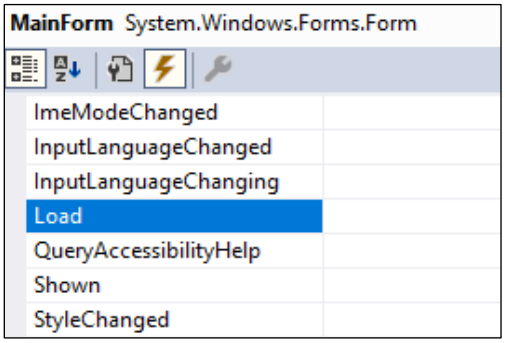


Рис. 10.48. Створення події Load для форми

```
private void MainForm_Load(object sender, EventArgs e)
{
    LoadNotepad();
}
```

Рис. 10.49. Код обробника події Load для форми

10. Додайте новий пункт підменю «Clear». Налаштуйте гарячі клавіші Ctrl + Shift + E. Значення поля Name встановіть в tsmiClear. Створіть обробник події Click, у середину якого додайте рядок:

```
rtbNotepad.Text="";
```

11. Перевірте роботу програми «Блокнот» додатка MyUtilites (Ctrl + F5). Оформіть звіт.

3. Контрольні запитання

1. Як створити меню, підменю для форми?
2. Як налаштувати дії під час вибору елементу підменю?
3. Які види елементів можна додати в конструкторі для MenuStrip?
4. Яке призначення властивостей Dock, LayoutStyle, Stretch?
5. Як додати пункти меню у програмному коді?
6. Як задати клавіші швидкого доступу пунктів меню?
7. Як додати зображення (піктограми) до елементів меню?
8. Яке призначення елементу TabControl?
9. Як задати власну назву вкладки TabPage1?
10. Яке призначення властивості Dock та його значення Fill елементу TabControl?
11. Як додати декілька вкладок елементу TabControl?
12. Як додати вкладку елементу TabControl у коді програми та задати її назву?
13. Як користуватись класом Convert для перетворення типу даних?
14. Які існують способи перетворення числових даних у рядкові, та навпаки?
15. Яке призначення елементів NumericUpDown та DomainUpDown?

16. Як задати мінімальне та максимальне число елементу NumericUpDown?

17. Яке призначення властивостей Hexadecimal, DecimalPlaces та Thousands Separator?

18. Яке призначення властивості Increment елементу NumericUpDown?

19. Як задати діапазон значень елементу DomainUpDown?

20. Як задати циклічне перебирання значень елементу DomainUpDown?

21. Як створити випадкове число в діапазоні значень?

22. Як задати максимальну довжину тексту та багаторядкове текстове поле?

23. Для чого використовується елемент RichTextBox? У чому різниця у застосуванні з елементом TextBox ?

24. Які властивості елементу RichTextBox задає параметр Dock? Які значення він може приймати?

ЛАБОРАТОРНА РОБОТА № 11
ПРОЄКТ MYUTILITES. ГЕНЕРАТОР ПАРОЛІВ. КОНВЕРТЕР ВЕЛИЧИН. VIEW PICTURE

Мета заняття: напрацювати навички роботи з елементом CheckedListBox, NumericUpDown для створення генератора паролів, із словником (Dictionary) для створення списку значень елементу ComboBox, з класом OpenFileDialog для створення діалогу відкриття графічних файлів та роботи з обробником подій Click елементів меню MenuStrip та виводом інформації у елемент TexBox, PictureBox за допомогою Windows Forms мовою C# у середовищі Microsoft Visual Studio.

1. Теоретичні відомості

Словник використовується для збереження пар «ключ – значення». Кожен елемент подається у вигляді структури **KeyValuePair<TKey, TValue>,** де доступ до ключа та значення здійснюється через властивості **Key** та **Value.**

Розглянемо на прикладі використання словників:

```
1   using System;
2   using System.Collections.Generic;
3
4   class MainClass {
5     public static void Main (string[] args) {
6       Console.WriteLine ("Hello World");
7       Dictionary<int, string> countries = new Dictionary<int, string>(5);
8       countries.Add(1, "Ukraine");
9       countries.Add(3, "Great Britain");
10      countries.Add(2, "USA");
11      countries.Add(4, "France");
12      countries.Add(5, "China");
13
14      foreach (KeyValuePair<int, string> keyValue in countries)
15    {
16      Console.WriteLine(keyValue.Key + " - " + keyValue.Value);
17    }
```

```
19    // отримання елементу по ключу
20    string country = countries[4];
21    // зміна об'єкту
22    countries[4] = "Spain";
23    // видалення по ключу
24    countries.Remove(2);
25
26    }
27  }
```

Для роботи зі словниками передбачені методи:

Add – додає елемент за вказаним ключем і значенням;

Remove – видаляє елемент за ключем (а не за індексом, як у списках).

Наприклад, у словнику ключі можуть бути типу **int,** а значення – типу **string,** тоді фактично він зберігатиме об'єкти **KeyValuePair<int, string>.** Для перебору елементів можна застосовувати цикл **foreach,** отримуючи як ключ, так і значення. До того ж існує можливість окремо звертатися до колекції ключів або значень словника. Можемо отримати окремо колекції ключів і значень словника:

```
using System;
using System.Collections.Generic;

class MainClass {
  public static void Main (string[] args) {
  Dictionary<char, string> people = new Dictionary<char, string>();
  people.Add('b', "Bill");
  people.Add('t', "Tom");
  people.Add('j', "John");

  foreach (KeyValuePair<char, string> keyValue in people)
  {
      //Перебір елементів словника (Ключ-значення)
      Console.WriteLine(keyValue.Key + " - " + keyValue.Value);
  }

  // перебор ключів
  foreach (char c in people.Keys)
  {
    Console.WriteLine(c);
  }

  // перебор по значенню
  foreach (string p in people.Values)
  {
    Console.WriteLine(p);
  }

  }
}
```

Словники можна ініціалізувати кількома способами:

```
Dictionary<string, string> countries = new Dictionary<string, string>
  {
    {"Франція", "Париж"},
    {"Німеччина", "Берлін"},
    {"Велика Британія", "Лондон"}
  };
   foreach(var pair in countries)
      Console.WriteLine("{0} - {1}", pair.Key, pair.Value);
Dictionary<string, string> countries = new Dictionary<string, string>
  {
    ["Франція"]= "Париж",
    ["Німеччина"]= "Берлін",
    ["Велика Британія"]= "Лондон"
  };
```

```
    foreach(var pair in countries)
      Console.WriteLine("{0} - {1}", pair.Key, pair.Value);
```

Для роботи з файлами у Windows Forms передбачені діалогові вікна **Open FileDialog** та **SaveFileDialog.** Вони мають спільні властивості, серед яких:

DefaultExt – встановлює розширення файла за замовчуванням, якщо користувач не вказав його;

AddExtension – у разі значення *true* автоматично додає розширення (береться з DefaultExt або Filter);

CheckFileExists – перевіряє наявність файла;

CheckPathExists – перевіряє існування шляху;

FileName – повертає повне ім'я обраного файла;

Filter – дає змогу відображати лише файли певного типу, наприклад, *Текстові файли (.txt)|.txt*;

InitialDirectory – визначає початкову папку під час відкриття вікна;

Title – задає заголовок діалогового вікна.

Клас SaveFileDialog додатково містить властивості:

– ***CreatePrompt*** – показує повідомлення, якщо вказаний файл ще не існує;

– ***OverwritePrompt*** – попереджає під час спроби перезаписати файл.

Для відображення діалогового вікна використовується метод Show Dialog(). Наприклад, на формі можна розмістити текстове поле textBox1 та дві кнопки (button1 і button2), а також додати компоненти OpenFileDialog та SaveFile Dialog.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
         button1.Click += button1_Click;
        button2.Click += button2_Click;
        openFileDialog1.Filter = "Text files(*.txt)|*.txt|All
   files(*.*)|*.*";
        saveFileDialog1.Filter = "Text files(*.txt)|*.txt|All
        files(*.*)|*.*";
    }
    // збереження файла
    void button2_Click(object sender, EventArgs e)
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // отримаємо вибраний файл
        string filename = saveFileDialog1.FileName;
        // зберігаємо текст в файл
        System.IO.File.WriteAllText(filename, textBox1.Text);
        MessageBox.Show("Файл збережений");
    }
    // відкриття файла
    void button1_Click(object sender, EventArgs e)
```

```
    {
        if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // отримаємо вибраний файл
        string filename = openFileDialog1.FileName;
        // читаємо файл в рядок
        string fileText = System.IO.File.ReadAllText(filename);
        textBox1.Text = fileText;
        MessageBox.Show("Файл відкритий");
    }
}
```

Натискання на першу кнопку відкриє діалог вибору файла, його вміст буде зчитано та показано в текстовому полі. Друга кнопка викликає вікно збереження файла, де користувач вказує ім'я, після чого текст із поля записується у файл.

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити проєкт «Генератор паролів».

1. Запустіть Visual Studio. Відкрийте проєкт «MyUtilites» (продовження проєкту, створеного у лабораторній роботі 10).

2. Додайте ще одну сторінку «Пароль». Виберіть елемент tabControl1 та властивість TabPages (рис. 11.1).

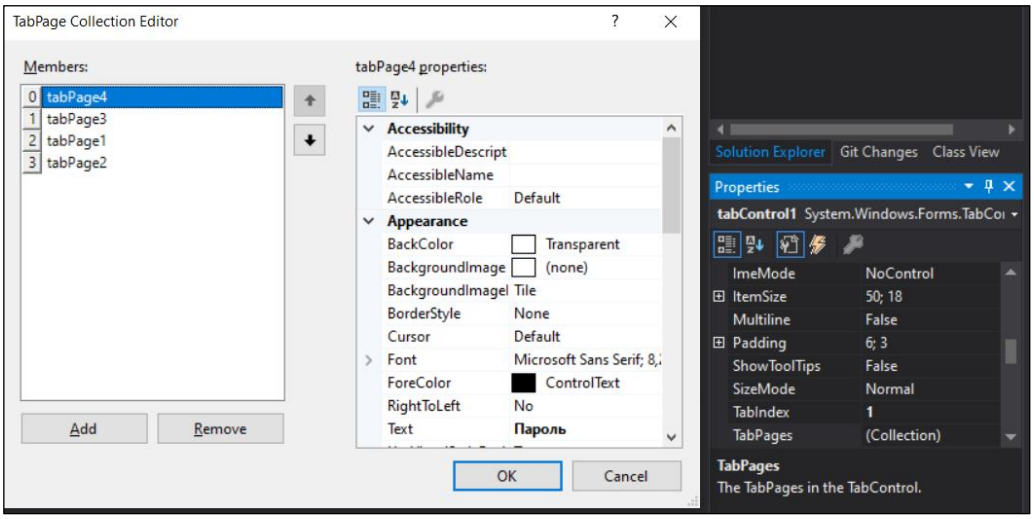


Рис. 11.1. Додавання TabPage4 (сторінка «Пароль») до елементу tabControl1

3. Додайте на форму елемент CheckedListBox. Властивість Item змініть, щоб додати назви CheckBox (рис. 11.2).

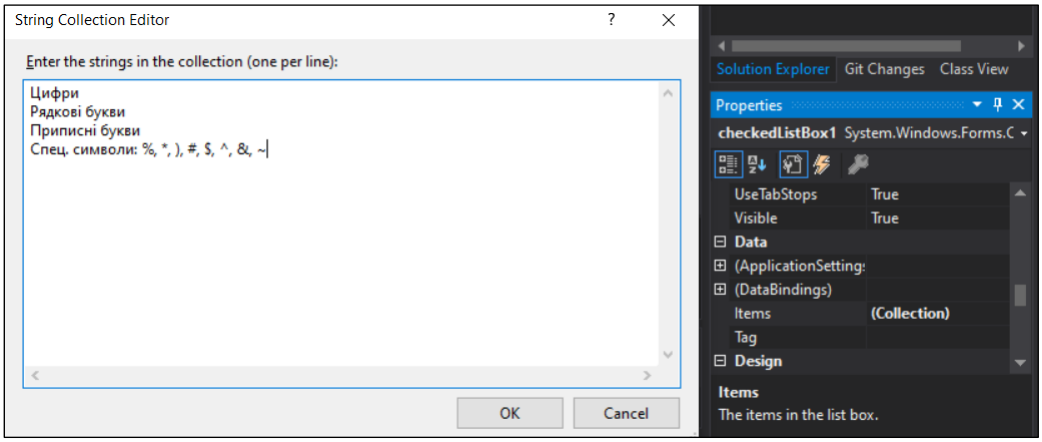


Рис. 11.2. Елемент CheckedListBox та встановлення назв CheckBox (властивість Item)

4. Додайте на форму елемент NumericUpDown (задає довжину пароля), Label, Button, TexBox (рис. 11.3).

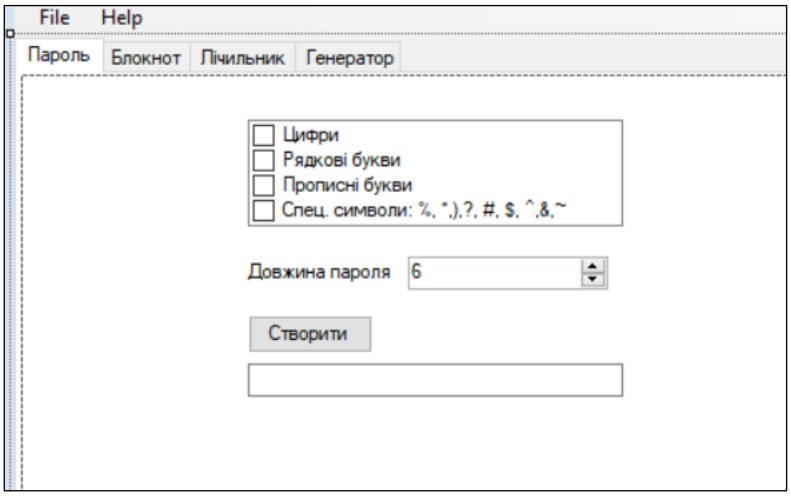


Рис. 11.3. Елементи форми. Сторінка «Пароль»

5. Встановіть властивість елементів за таблицею 11.1.

Таблиця 11.1 – Властивості елементів форми. Сторінка «Пароль»

Елемент	Поле	Значення
NumericUpDown	Name	nudPasswordLengh
	Value	6,
	Max	20
	Min	1
CheckedListBox	Name	clbPassword
	CheckOnClick	True
Button	Name	btnCreatePassword
TexBox	Name	tbPassword

У елементу CheckedListBox властивість CheckOnClick = True дає змогу переключати стан CheckBox одним кліком миші, а не двома.

6. Створіть обробник події Click для кнопки. Допишіть код (рис. 11.4).

```
if (clbPassword.CheckedItems.Count == 0) return;
            string password = "";
            for (int i=0; i<nudPasswordLengh.Value; i++)
            {
                int n = rnd.Next(0, clbPassword.CheckedItems.Count);
           string s = clbPassword.CheckedItems[n].ToString();
```

```
private void btnCreatePassword_Click(object sender, EventArgs e)
{
    if (clbPassword.CheckedItems.Count == 0) return;
    string password = "";
    for (int i = 0; i < nudPasswordLength.Value; i++)

    {
        int n = rnd.Next(0, clbPassword.CheckedItems.Count);
        string s = clbPassword.CheckedItems[n].ToString();
    }
```

Рис. 11.4. Код обробника подій Click для кнопки «Створити пароль»

7. Внесіть зміни коду обробника подій Click для кнопки «Створити пароль», щоб врахувати вибір елементів CheckedListBox (рис. 11.5).

```
switch (s)
   {
    case "Цифри": password += rnd.Next(10).ToString();
           break;
    case "Рядкові букви": password += Convert.ToChar(rnd.Next(65,88));
      break;
    case "Прописні букви": password += Convert.ToChar(rnd.Next(97, 122));
      break;
```

```
private void btnCreatePassword_Click(object sender, EventArgs e)
{
    if (clbPassword.CheckedItems.Count == 0) return;
    string password = "";
    for (int i = 0; i < nudPasswordLength.Value; i++)

    {
        int n = rnd.Next(0, clbPassword.CheckedItems.Count);
        string s = clbPassword.CheckedItems[n].ToString();
        switch (s)
        {
            case "Цифри":
                password += rnd.Next(10).ToString();
                break;
            case "Рядкові букви":
                password += Convert.ToChar(rnd.Next(65, 88));
                break;
            case "Прописні букви":
                password += Convert.ToChar(rnd.Next(97, 122));
                break;
        }
    }
}
```

Рис. 11.5. Код обробника подій Click для кнопки «Створити пароль» з оператором switch ()

8. На початку програми створіть масив спецсимволів (рис. 11.6) для випадку default (рис. 11.7).

```
char[] spec_char = new char[] { '%', '*', ')', '?', '#', '$', '^', '&', '~' };
```

```
public partial class MainForm : Form
{
    int count = 0;
    Random rnd;
    char[] spec_char = new char[] { '%', '*', ')', '?', '#', '$', '^', '&', '~' };
```

Рис. 11.6. Оголошення масиву спецсимволів

```
switch (s)
{
    case "Цифри":
        password += rnd.Next(10).ToString();
        break;
    case "Рядкові букви":
        password += Convert.ToChar(rnd.Next(65, 88));
        break;
    case "Прописні букви":
        password += Convert.ToChar(rnd.Next(97, 122));
        break;
    default:
        password += spec_char[rnd.Next(spec_char.Length)];
        break;
}
```

Рис. 11.7. Код обробника подій Click з оператором switch (). Секція default

9. Додайте вивід пароля в елемент TexBox (рис. 11.8).

```
tbPassword.Text = password;
private void btnCreatePassword_Click(object sender, EventArgs e)
{
    if (clbPassword.CheckedItems.Count == 0) return;
    string password = "";
    for (int i = 0; i < nudPasswordLength.Value; i++)

    {
        int n = rnd.Next(0, clbPassword.CheckedItems.Count);
        string s = clbPassword.CheckedItems[n].ToString();
        switch (s)
        {
            case "Цифри":
                password += rnd.Next(10).ToString();
                break;
            case "Рядкові букви":
                password += Convert.ToChar(rnd.Next(65, 88));
                break;
            case "Прописні букви":
                password += Convert.ToChar(rnd.Next(97, 122));
                break;
            default:
                password += spec_char[rnd.Next(spec_char.Length)];
                break;
        }
        tbPassword.Text = password;
    }
```

Рис. 11.8. Код обробника подій Click для кнопки «Створити пароль»

10. Додайте ще 1 рядок у коді обробника подій Click, щоб скопіювати пароль до буферу обміну:

```
Clipboard.SetText(password);
tbPassword.Text = password;
Clipboard.SetText(password);
```

11. У розділ завантаження додайте значення, що будуть встановлені під час запуску програми (рис. 11.9).

```
private void MainForm_Load(object sender, EventArgs e)
{
    LoadNotepad();
    clbPassword.SetItemChecked(0, true);
    clbPassword.SetItemChecked(1, true);
}
```

Рис. 11.9. Секція завантаження форми

12. Перевірте роботу програми «Пароль» додатку MyUtilites (Ctrl + F5).

Завдання 2. Створити додаток «Конвертер величин».

1. Додайте ще одну сторінку «Конвертер». Виберіть елемент tabControl1 та властивість TabPages (рис. 11.10).

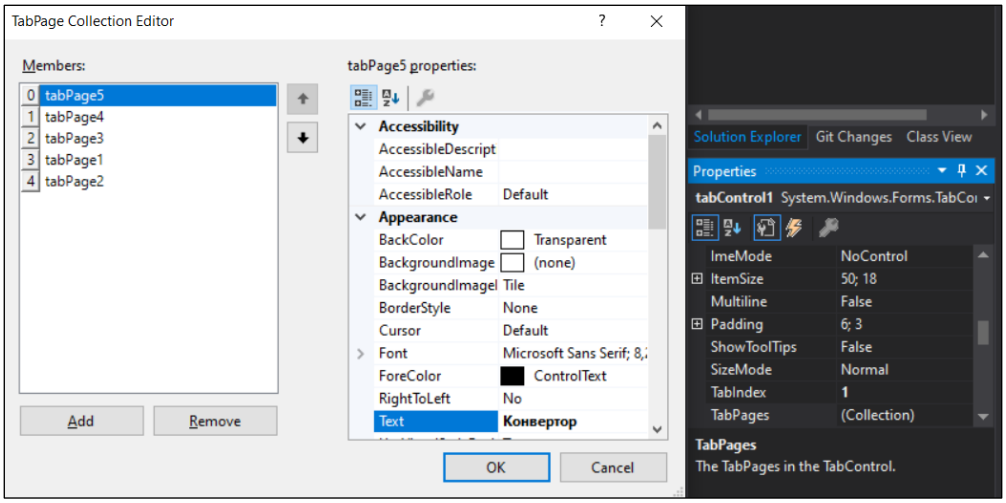


Рис. 11.10. Додавання TabPage5 (сторінка «Конвертер») до елементу tabControl1

2. Додайте на форму 2 елементи ComboBox, кнопку Button, 2 елементи TexBox (рис. 11.11).

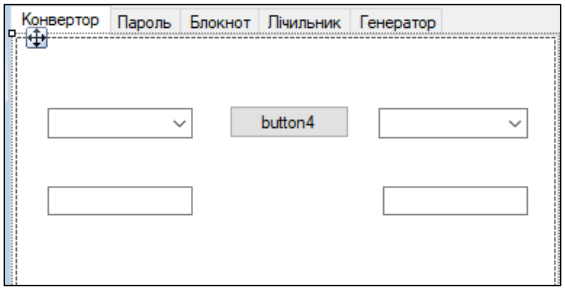


Рис. 11.11. Елементи форми. Додаток «Конвертер величин»

3. Встановіть властивість елементів за таблицею 11.2.

Таблиця 11.2 – Властивості елементів форми. Сторінка «Конвертер»

Елемент	**Поле**	**Значення**
ComboBox1	Name Items Text	cbFrom mm cm dm m kl mile mm
ComboBox2	Name Items Text	cbTo mm cm dm m kl mile mm
Button	Name Text	btnConvert Конертувати
TextBox1	Name Text	tbFrom 1
TextBox2	Name ReadOnly	tbTo True

4. Створіть обробник події Click для кнопки. Створіть елемент словник metrica (рис. 11.12).

```
Dictionary<string, double> metrica;
namespace MyUtilites
{
    public partial class MainForm : Form
    {
        int count=0;
        Random rnd;
        char[] spec_char = new char[] { '%', '*', ')', '?', '#', '$', '^', '&', '~' };
        Dictionary<string, double> metrica;
```

Рис. 11.12. Оголошення словника metrica

5. Створіть об’єкт metrica в конструкторі форм та занесіть дані у словник (рис. 11.13).

```
metrica = new Dictionary<string, double>();
            metrica.Add("mm", 1);
            metrica.Add("cm", 10);
            metrica.Add("dm", 100);
            metrica.Add("m", 1000);
            metrica.Add("kl", 1000000);
            metrica.Add("mile", 1609344);
```

```
public MainForm()
{
    InitializeComponent();
    rnd = new Random();
    metrica = new Dictionary<string, double>();
    metrica.Add("mm", 1);
    metrica.Add("cm", 10);
    metrica.Add("dm", 100);
    metrica.Add("m", 1000);
    metrica.Add("kl", 1000000);
    metrica.Add("mile", 1609344);
}
```

Рис. 11.13. Конструктор форм. Створення об’єкт metrica

6. Додайте код в обробник кнопки «Конвертувати» (рис. 11.14).

```
double m1 = metrica[cbFrom.Text];
double m2 = metrica[cbTo.Text];
double n = Convert.ToDouble(tbFrom.Text);
tbTo.Text = (n * m1 / m2).ToString();
```

```
private void btnConvert_Click(object sender, EventArgs e)
{
    double m1 = metrica[cbFrom.Text];
    double m2 = metrica[cbTo.Text];
    double n = Convert.ToDouble(tbFrom.Text);
    tbTo.Text = (n * m1 / m2).ToString();

}
```

Рис. 11.14. Код обробника подій Click для кнопки «Конвертувати»

7. Додайте на форму кнопку зміни напрямку конвертації «<- ->» (рис. 11.15). Властивості Name присвойте значення btnSwap та створіть для неї подію Click.

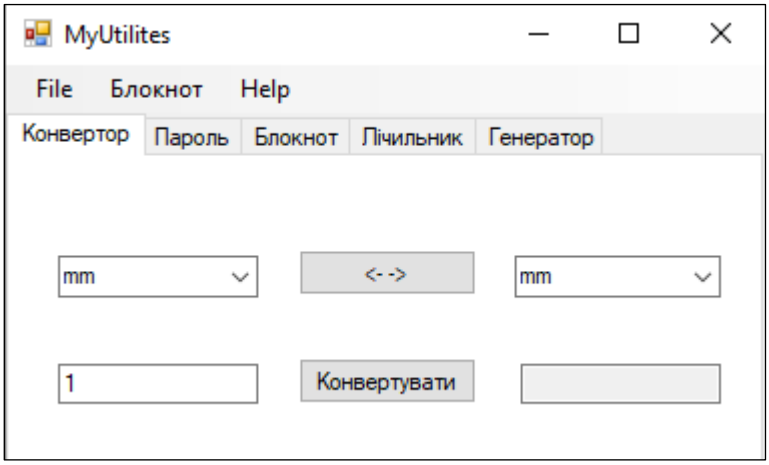


Рис. 11.15. Оновлений дизайн додатка «Конвертер величин»

```
private void btnSwap_Click(object sender, EventArgs e)
        {
            string t = cbFrom.Text;
            cbFrom.Text = cbTo.Text;
            cbTo.Text = t;
  }
```

8. Перевірте роботу програми «Конвертер величин» додатку MyUtilites (Ctrl + F5).

Завдання 3. Створити проєкт «View Picture».

1. Додайте ще одну сторінку ViewPicture. Виберіть елемент tabControl1 та властивість TabPages (рис. 11.16).

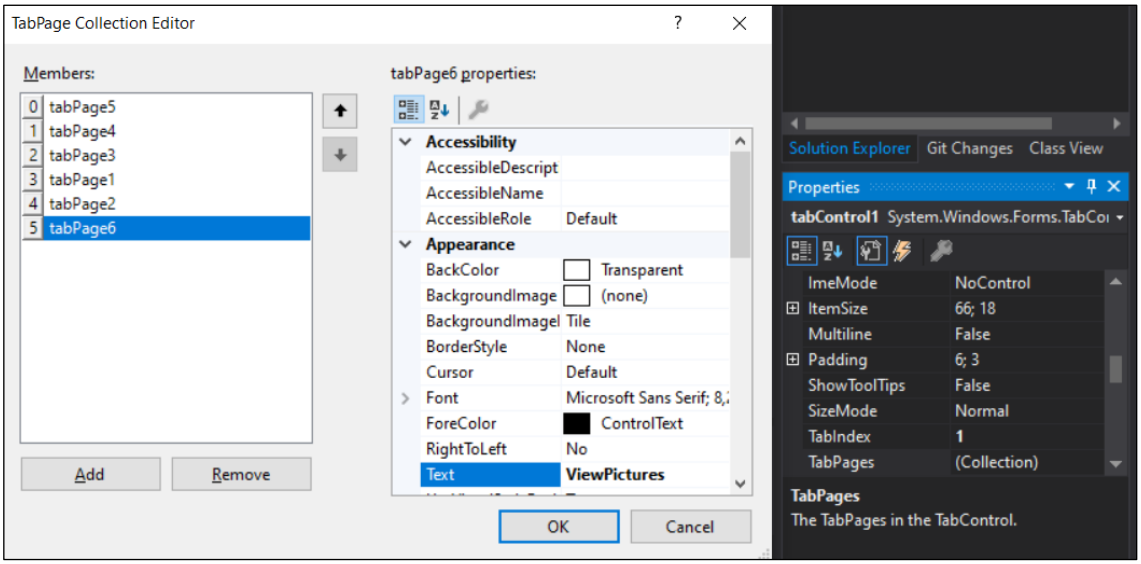


Рис. 11.16. Додавання TabPage 6 (сторінка ViewPicture) до елементу tabControl1

2. Додайте на форму 2 елемент OpenFileDialog. Після додавання він буде відображатися внизу дизайнера форми (рис. 11.17).

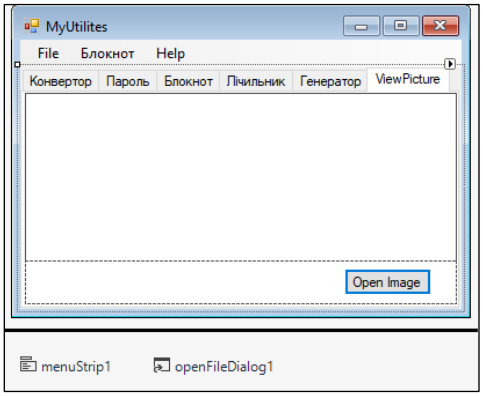


Рис. 11.17. Дизайн форми. Додаток «ViewPicture»

3. Додайте елемент PictureBox та розтягніть його по краях, залишивши місце для кнопки внизу. Додайте до форми кнопку. Встановіть властивість елементів за таблицею 11.3.

Таблиця 11.3 – Властивості елементів форми. Додаток «ViewPicture»

Елемент	Поле	Значення
PictureBox	SizeMode	Zoom
	Anckor	Top, Bottom, Left, Right
Button	Name	btOI
	Text	Open Image
	Anckor	Top, Right

Значення Zoom властивості SizeMode використовується для того, щоб зображення, яке вибирається, змінювало розміри під розмір елементу PictureBox.

Значення «Top, Bottom, Left, Right» властивості Anckor робить круговую прив’язку до форми по колу.

Значення «Top, Right» властивості Anckor елементу Bottom прив’язує кнопку до правого нижнього краю.

4. Створіть обробник події Click для кнопки. У конструкторі форми після методу InitializeComponent() додайте такий код (рис. 11.18):

```
openFileDialog1.FileName="";
openFileDialog1.Filter = "Image Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.*";
public MainForm()
{
    InitializeComponent();
    rnd = new Random();
    metrica = new Dictionary<string, double>();
    metrica.Add("mm", 1);
    metrica.Add("cm", 10);
    metrica.Add("dm", 100);
    metrica.Add("m", 1000);
    metrica.Add("kl", 1000000);
    metrica.Add("mile", 1609344);
    openFileDialog1.FileName = "";
    openFileDialog1.Filter = "Image Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.*";
```

Рис. 11.18. Конструктор форми. Додаємо openFileDialog1

5. В обробнику події Click для кнопки зробіть перевірку, що файл дійсно вибраний, і якщо вибраний, то передайте шлях до файла елементу PictureBox; якщо файл не є зображенням, то видає помилку. Для цього використайте конструкцію try…catch (рис. 11.19):

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
 {
   try
    {
     pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);
    }
    catch (Exception)
     {
      MessageBox.Show("Не коректний файл. Виберіть файл типу.png|.jpeg|
       .bmp|.gif","Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
     }
```

6. Перевірте роботу програми «View Picture» додатка MyUtilites (Ctrl + F5). Оформіть звіт.

```
private void btOI_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        try
        {
            pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);
        }
        catch (Exception)
        {
            MessageBox.Show("Не коректний файл. Виберіть файл типу .png|.jpeg|.bmp|.gif",
                "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

Рис. 11.19. Код обробника подій Click для кнопки «Open Image»

3. Контрольні запитання

1. Для чого використовується елемент NumericUpDown? Як задати значення за замовчуванням та діапазон значень Min – Max?

2. Як згенерувати випадкові цифри, рядкові та прописні символи, спецсимволи у програмі?

3. Як вивести значення пароля у TexBox та скопіювати пароль до буфера обміну?

4. Як виконати ініціалізацію словника?

5. Як отримати доступ до елементів словника за ключем, значенням?

6. Як додати, видалити елемент словника? Як змінити значення словника?

7. Яка назва словника, що використовується у програмі? Який тип даних Key, Value у програмі?

8. Які дані знаходяться у словнику, що використовується у програмі лабораторної роботи?

9. Як виконана ініціалізація елементів словника у програмі? Яким іншим методом можна змінити код програми для ініціалізації елементів словника?

10. Які зміни потрібно зробити у коді програми, щоб зробити додаток «Конвертер валют»?

11. Для чого призначений елемент OpenFileDialog? Як додати його до проєкту?

12. Для чого використовується метод ShowDialog() у коді програми?

13. Яке призначення властивостей FileName та Filter у програмі?

14. Як задати типи графічних файлів, що будуть відкриватись у програмі?

15. Що буде робити програма, якщо вибраний не графічний файл?

16. Як програма перевіряє тип файла, що вибраний для відкриття?

17. Як забезпечити, щоб розмір зображення, яке відкривається, відповідав розмірам вікна програми?

ЛАБОРАТОРНА РОБОТА № 12
СТВОРЕННЯ ДОДАТКА TODO APPLICATION ЗА ТЕХНОЛОГІЄЮ WPF .NET

Мета заняття: навчитися створювати додатки за допомогою Windows Presentation Foundation (WPF) мовою C# у середовищі Microsoft Visual Studio.

1. Теоретичні відомості

Windows Presentation Foundation (WPF) надає широкий набір можливостей для створення додатків. Серед них – використання мови XAML, елементи керування, механізми прив'язки даних, системи розташування (layout), підтримка 2D- та 3D-графіки, анімація, стилі, шаблони, робота з документами, мультимедіа та текстом із розширеними типографічними функціями. WPF є складником середовища .NET і реалізована у вигляді набору типів, розташованих у просторі імен System.Windows.

Ця технологія дає змогу будувати додатки, комбінуючи XAML-розмітку та програмний код, що подібно до підходу в ASP.NET. Зазвичай XAML використовується для опису інтерфейсу, тоді як керований код – для реалізації логіки та поведінки програми.

XAML (eXtensible Application Markup Language) – це мова розмітки, заснована на XML, яка застосовується для декларативного опису інтерфейсу. За її допомогою визначаються вікна, сторінки, елементи управління та їх наповнення графічними елементами, формами й іншими контролами. Наприклад, у фрагменті коду XAML можна задати вікно з однією кнопкою:

```
<Window
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.AWindow"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>
</Window>
```

У цьому випадку елемент Window відповідає за створення вікна, а Button – за відображення кнопки. Кожен елемент має набір атрибутів для налаштування, наприклад, атрибут Title елемента Window визначає заголовок вікна. Під час виконання WPF перетворює вказані в XAML елементи та їх атрибути на відповідні об'єкти та властивості класів WPF. Так, тег Window стає екземпляром класу Window, а його властивість Title отримує значення з розмітки.

На рис. 12.1 показаний інтерфейс користувача, який визначається кодом XAML з попереднього прикладу. Оскільки мова XAML заснована на XML, створю-

ваний за його допомогою інтерфейс користувача утворює ієрархію вкладених елементів.

Рис. 12.1. Інтерфейс користувача, який визначається кодом XAML

Основна поведінка програми полягає у реалізації функції, що реагує на взаємодію з користувачем. Наприклад, натискання меню або кнопки та виклик бізнес-логіки і логіки доступу до даних у відповідь. У WPF така поведінка реалізується у коді, пов'язаному з розміткою.

Розмітка визначає простір імен xmlns:x та зіставляє її зі схемою, яка додає підтримку для типів коду програмної частини. Атрибут x:Class використовується для зв'язування класу програмного коду з цією конкретною розміткою XAML. З огляду на те, що цей атрибут оголошено в елементі <Window>, клас коду програмної частини повинен успадковувати від класу Window.

```
using System.Windows;
namespace SDKSample
{
   public partial class AWindow : Window
   {
      public AWindow()
      {
         // InitializeComponent call is required to merge the UI
         // that is defined in markup with this class, including
         // setting properties and registering event handlers
         InitializeComponent();
      }

      void button_Click(object sender, RoutedEventArgs e)
      {
          // Show message box when button is clicked.
          MessageBox.Show("Hello, Windows Presentation Foundation!");
       }
    }
    }
```

Метод InitializeComponent викликається з конструктора класу коду програмної частини для злиття інтерфейсу користувача, визначеного в розмітці, з класом коду програмної частини. (InitializeComponent створюється під час побудови програми, тому реалізовувати його вручну не потрібно). Поєднання x:Class та InitializeComponent гарантує правильну ініціалізацію реалізації під час створення.

Завдяки тому, що розмітка та код програмної частини ініціалізовані та працюють разом, подія Click для кнопки автоматично зіставляється з методом button_click. Після натискання кнопки викликається обробник подій, а вікно повідомлення відображається під час виклику методу System.Windows.Message Box.Show. На рис. 12.2 показано результат натискання кнопки.

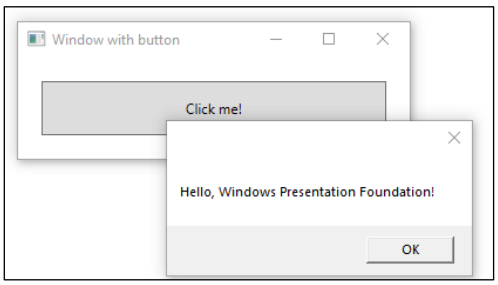


Рис. 12.2. Результат натискання кнопки

2. Практичне завдання

У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

Завдання 1. Створити додаток ToDo Application (рис. 12.3).

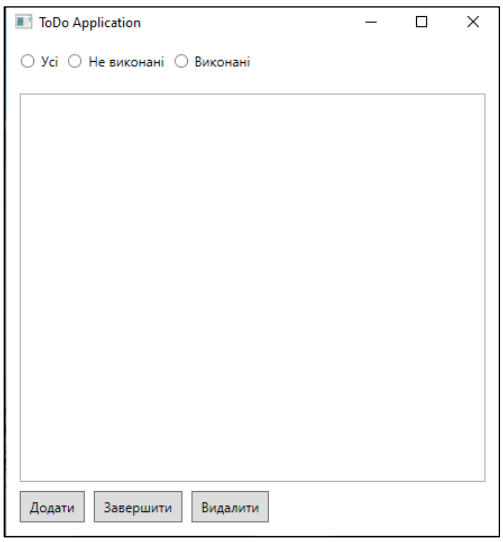


Рис. 12.3. Інтерфейс додатка ToDo

1. У Visual Studio 2019 створіть WPF-проєкт із назвою ToDoApp (рис. 12.4).

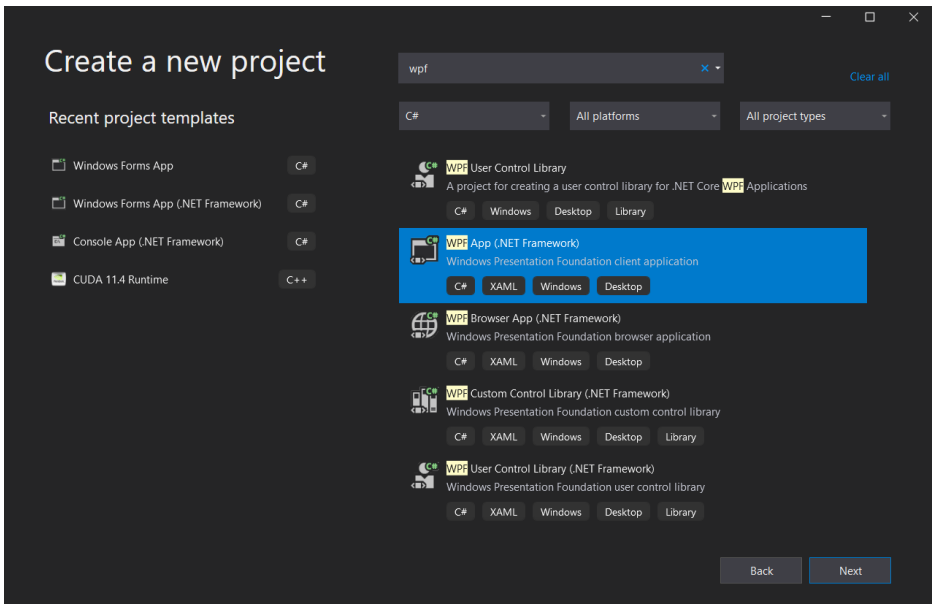


Рис. 12.4. Вікно створення проєктів у VS2019

2. У властивості Title головного вікна задайте заголовок ToDoApplication замість mainWindow (рис. 12.5).

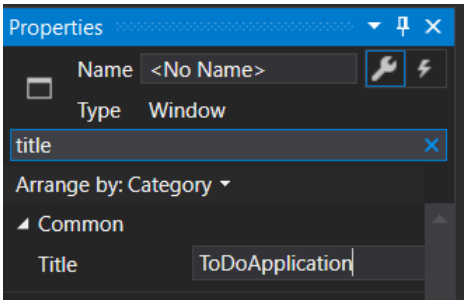


Рис. 12.5. Налаштування властивості Title вікна додатка ToDoApp

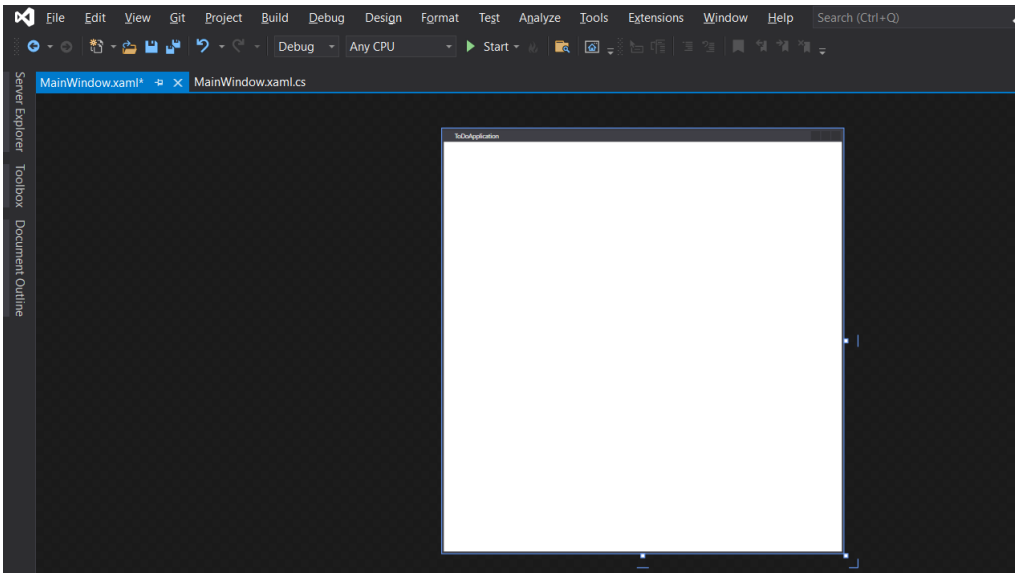


Рис. 12.6. Налаштування розмірів вікна додатка

3. Щоб вікно могло змінювати розміри разом з елементами, використайте контейнер **Grid** у XAML (рядки 9–11). Задайте параметри: Height="480" і Width="455" (рис. 12.6).

4. Використовуючи дизайнер, додайте два рядки висотою по 40 пікселів (угорі та внизу). Внаслідок цього макет складатиметься з трьох рядків (рис. 12.7).

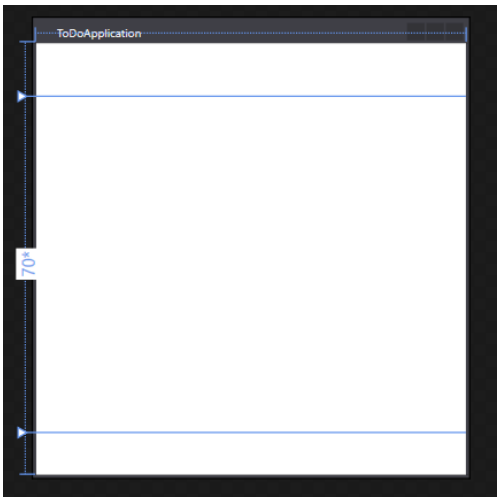


Рис. 12.7. Додавання 2 рядків у конструкторі додатка ToDoApp

5. У центральний рядок вставте ListBox, розтягніть його на все вікно, встановіть відступи Margin="12,12,8,8" і перейменуйте на ToDoListBox. Протестуйте масштабування (Ctrl+F5).

6. У нижній рядок додайте StackPanel з орієнтацією Horizontal та розмістіть три кнопки:

– AddButton – «Додати»;
– CompleteButton – «Завершити»;
– DeleteButton – «Видалити».

Задайте зовнішні відступи (Margin 12 зліва / справа, 8 знизу) і внутрішні (Padding 8;8;1;1) (рис. 12.8).

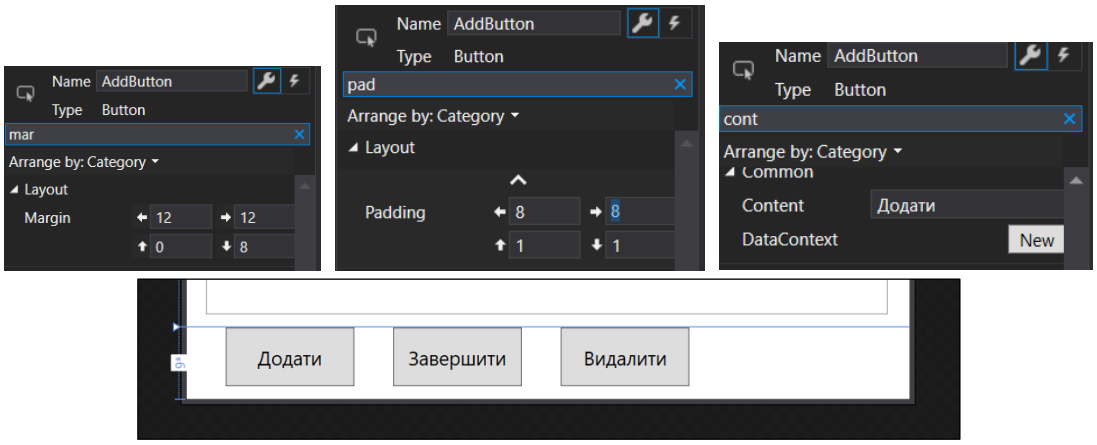


Рис. 12.8. Налаштування властивостей елементів Button

7. Протестуйте. Якщо висота мала, кнопки звужуються та мають негарний вигляд. Щоб кнопки не спотворювалися під час зменшення висоти, нижньому рядку задайте фіксовану висоту замість пропорційної (рис. 12.9).

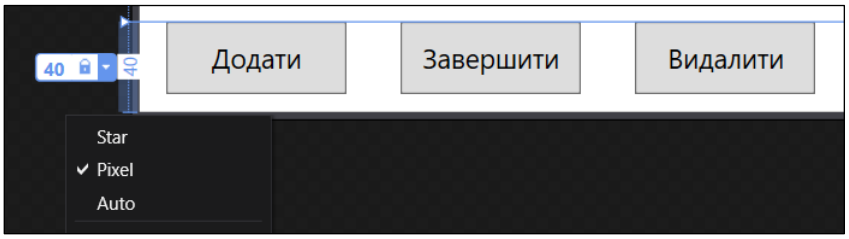


Рис. 12.9. Налаштування прив'язки до нижнього рядка елементу Grid

8. У верхньому рядку розмістіть StackPanel (Horizontal) з трьома RadioButton:

– AllRadioButton – «Усі»;

– NotCompletedRadioButton – «Не виконані»;

– CompletedRadioButton – «Виконані».

Встановіть відступи Margin 12; 0; 12; 0. (рис. 12.10).

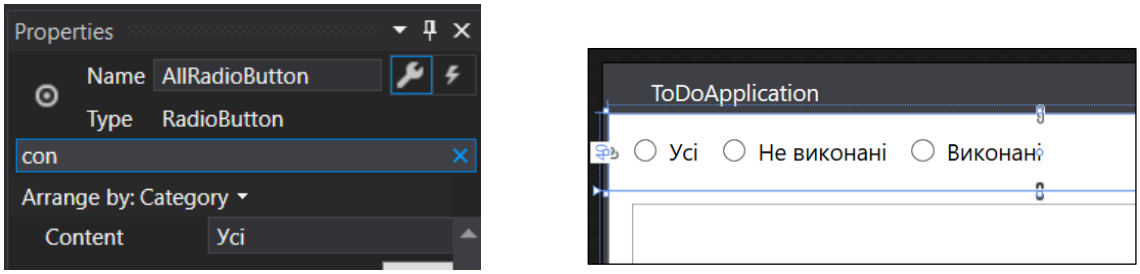


Рис. 12.10. Налаштування властивостей RadioButton

9. У файлі MainWindow.xaml.cs створіть масив рядків із трьома задачами. Згодом його замініть на колекцію.

```
public partial class MainWindow : Window
{
        string[] tasks = new string[3];
        public MainWindow()
        {
            InitializeComponent();
            tasks[0] = "Купити молока";
            tasks[1] = "Вивчити C#";
            tasks[2] = "Створити резюме";

            ToDoListBox.ItemsSource = tasks;
        }
    }
```

Властивості ItemsSource елементу ToDoListBox присвойте значення масиву tasks. Протестуйте додаток (рис. 12.11).

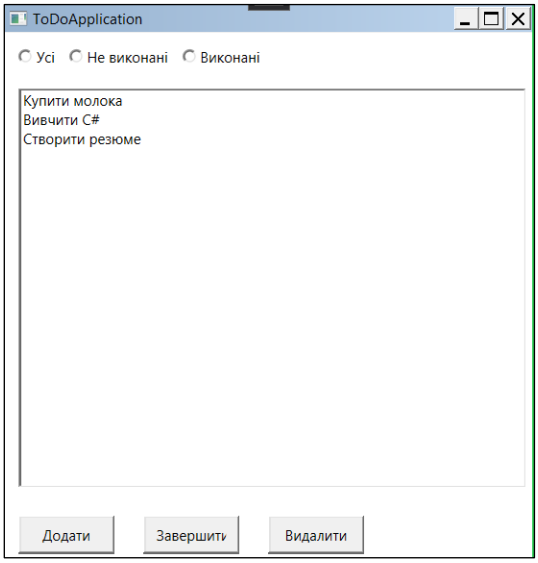


Рис. 12.11. Вікно додатка ToDoApp з RadioButton, ListBox, Button

10. Створіть клас Task. Натисніть правою кнопкою миші на ToDoApp в Solution Explorer. Виберіть Add – Create Element (Ctrl + Shift + A). Виберіть Clas та задайте ім'я класу – Task (рис. 12.12).

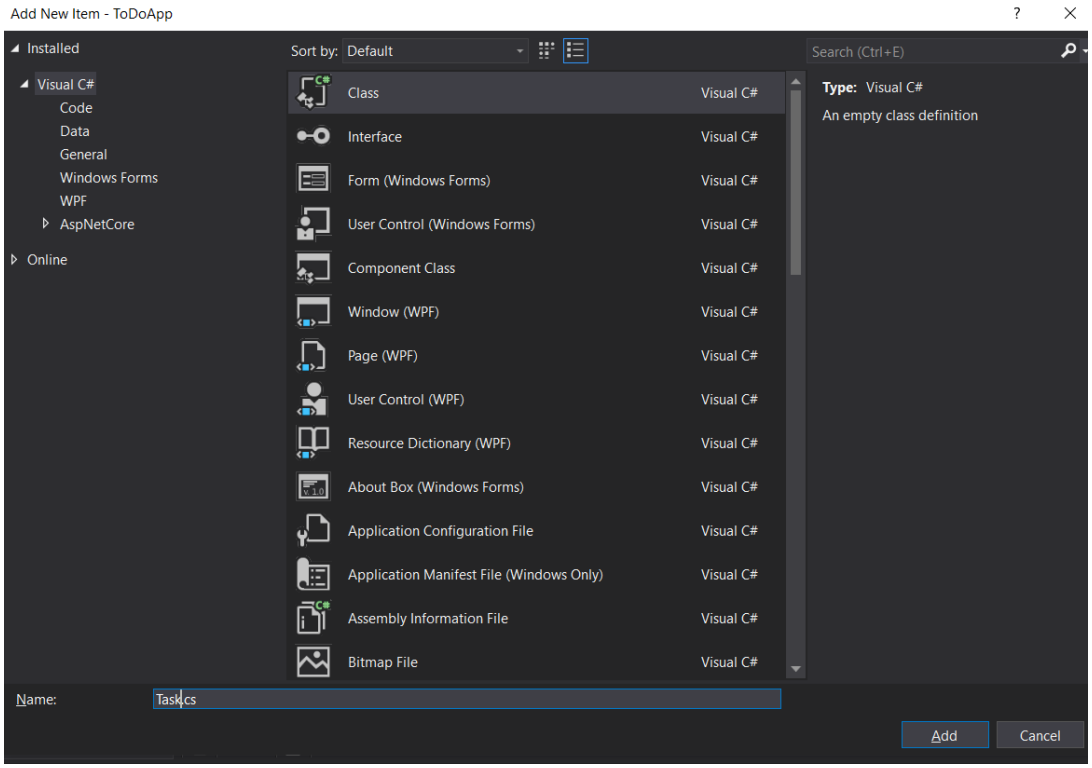


Рис. 12.12. Додавання класу Task до проєкту

11. У класі Task створіть 3 властивості:

```
public string Name { get; set; }
public bool IsCompleted { get; set; }
public string Description { get; set; }
```

12. Перейдіть до файла MainWindows.xaml.cs. Видаліть масив та створіть колекцію.

```
List<Task> tasksList = new List<Task>();
```

Внесіть необхідні зміни у код: створіть три об'єкти класу Task, задайте їх властивості та додайте у колекцію. Колекцію призначте як джерело даних для властивості ToDoListBox.ItemsSource.

```
public MainWindow()
{
    InitializeComponent();

    Task t1 = new Task();
    t1.Name = "Купити молока";
    t1.IsCompleted = false;
    t1.Description = "Піти до магазину Метро. Придбати молока українського
виробника";

    Task t2 = new Task();
    t2.Name = "Вивчити C#";
    t2.IsCompleted = false;
    t2.Description = "Виконати домашні завдання. Здати на перевірку. Знайти
додатковий навчальний матеріал. Опанувати його";

    Task t3 = new Task();
    t3.Name = "Створити резюме";
    t3.IsCompleted = false;
    t3.Description = "Створити CV на LinkeIn";

    tasksList.Add(t1);
    tasksList.Add(t2);
    tasksList.Add(t3);

    ToDoListBox.ItemsSource = tasksList;
}
```

Після запуску проєкту помічаємо проблему (рис. 12.13) – елементи у списку відображаються некоректно. Це відбувається тому, що клас Task має кілька властивостей, і потрібно вказати, яку саме треба показувати у ToDo ListBox. Для цього додайте відповідний рядок:

```
ToDoListBox.DisplayMemberPath = "Name";
```

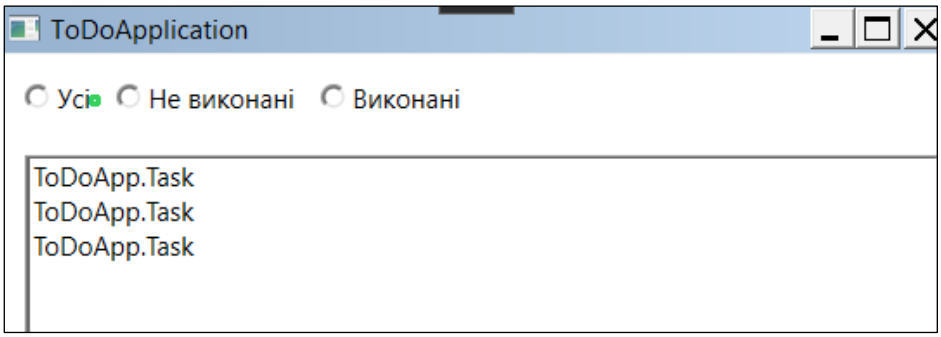


Рис. 12.13. Некоректне виведення об'єктів у ToDoListBox

13. Реалізуйте таке: під час подвійного кліку на елемент списку має відображатися значення властивості Description у вікні MessageBox. Для цього створіть подію MouseDoubleClick, додайте обробник і визначте, яка задача була вибрана. Для неї виведіть повідомлення:

```
private void ToDoListBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
  Task selected = ToDoListBox.SelectedItem as Task;
  if (selected != null)
  {
      MessageBox.Show(selected.Description);
  }
}
```

Протестуйте роботу (рис. 12.14).

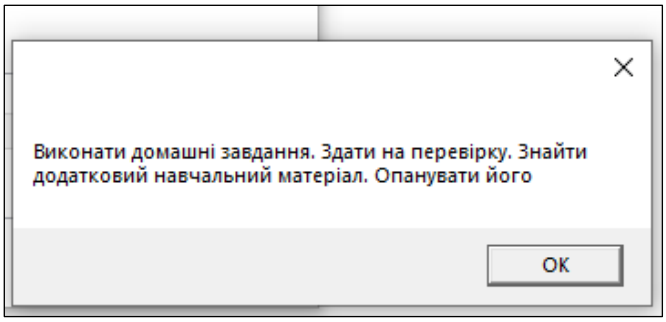


Рис. 12.14. Тестування роботи елементу MessageBox у додатку ToDoApp

14. Налаштуйте кнопку «Додати». Створити обробник події та додати у проєкт нове вікно NewTaskWindow (Add → Window (WPF)). Встановити назву «Нова задача» і розміри 450 × 300 (рис. 12.15).

```
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:ToDoApp"
    mc:Ignorable="d"
    Title="Нова задача" Height="450" Width="300">
<Grid>
```

Рис. 12.15. Налаштування заголовка та розмірів нового вікна NewTaskWindow

Запустіть проєкт (Ctrl+F5) і перевірте роботу кнопки. Вікно відкривається, але розташоване незручно – зробимо його відцентрованим відносно головного вікна, додавши потрібний код у метод AddButton_Click:

```
private void AddButton_Click(object sender, RoutedEventArgs e)
{
   NewTaskWindow window = new NewTaskWindow();
   window.Owner = this;
   window.WindowStartupLocation = WindowStartupLocation.CenterOwner;
   window.ShowDialog();
}
```

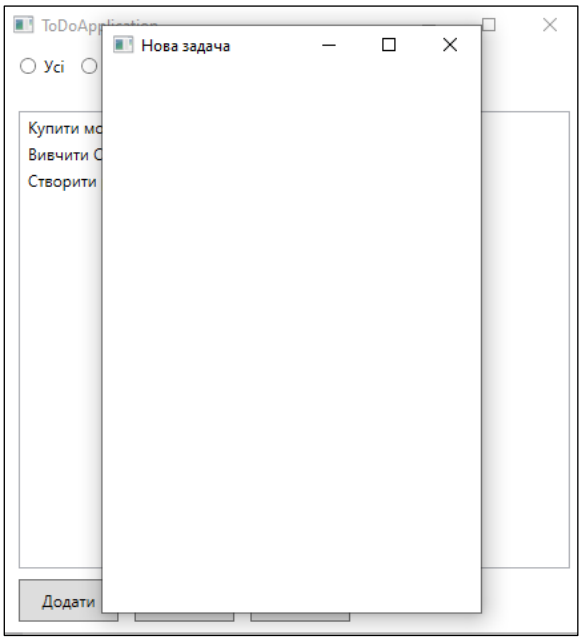


Рис. 12.16. Тестування роботи кнопки «Додати» додатка ToDoApp

15. Створіть інтерфейс у новому вікні. Додайте поля: Name, IsCompleted, Description і дві кнопки (рис. 12.17). У нижній частині зробіть рядок із висотою 60. Усередині розмістіть StackPanel із двома кнопками, вирівняними вертикально, з відступами (12; 12; 0; 8). Кнопки мають назви: CancelButton – «Скасувати», SaveButton – «Зберегти».

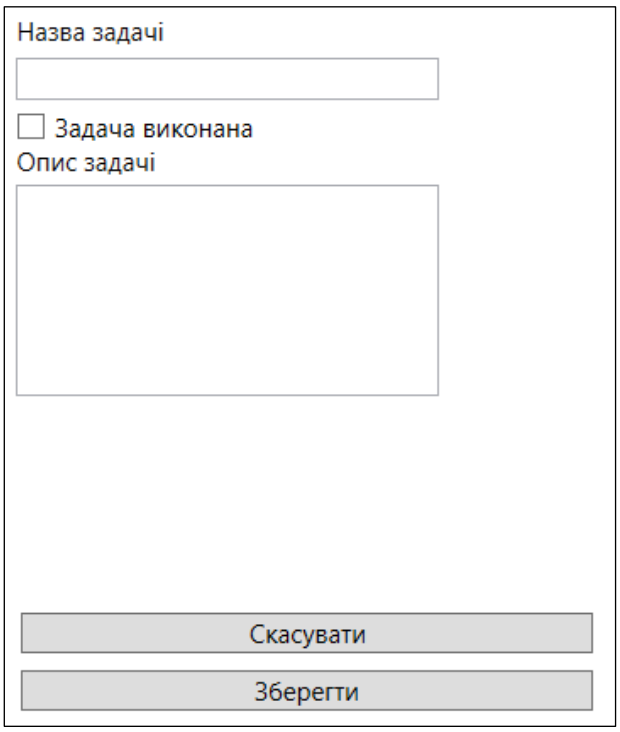


Рис. 12.17. Інтерфейс вікна Нова задача додатка ToDoApp

У центральній частині також розмістіть StackPanel з відступами 12. Додайте TextBlock «Назва задачі», під ним TextBox (ширина 200, вирівнювання зліва, нижній відступ 8). Заберіть властивість Text, задайте ім’я NameTextBox.

Додайте CheckBox із текстом «Задача виконана» (нижній відступ 8), ім’я IsCompletedCheckBox.

Додайте TextBlock «Опис задачі» (нижній відступ 4), під ним TextBox (ширина 200, висота 100, вирівнювання зліва). Заберіть властивість Text, задайте ім’я DescriptionTextBox.

Реалізуйте обробники натискання на кнопки «Скасувати» та «Зберегти».

16. Клас Task робимо public:

```
public class Task
```

Відредагуйте файл NewTaskWindow.xaml.cs. Створіть властивість Result об’єкта класу Task:

```
public Task Result {get; set;}
```

Додайте код в обробник подій кнопки «Скасувати», «Зберегти»:

```
private void SaveButton_Click(object sender, RoutedEventArgs e)
{
    Task t = new Task();
    t.Name = NameTextBox.Text;
    t.IsCompleted = IsComplitedCheckBox.IsChecked.Value;
    t.Description = DescriptionTextBox.Text;
    Result = t;

   DialogResult = true;
}

private void CancelButton_Click(object sender, RoutedEventArgs e)
{
    DialogResult = false;
}
```

17. Додайте логіку обміну даними між вікнами. Якщо метод window.Show Dialog() повертає true, це означає, що створена нова задача, яку потрібно додати у список.

```
private void AddButton_Click(object sender, RoutedEventArgs e)
{
   NewTaskWindow window = new NewTaskWindow();
   window.Owner = this;
   window.WindowStartupLocation = WindowStartupLocation.CenterOwner;
   if (window.ShowDialog() == true)
   {
       Task newTask = new Task();
       tasksList.Add(newTask);
    }
}
```

18. Нові задачі зберігаються у tasksList, але не відображаються у вікні додатка. Щоб виправити це, необхідно змінити тип колекції:

```
List<Task> tasksList = new List<Task>();
```

на:

```
ObservableCollection<Task> tasksList = new ObservableCollection<Task>();
```

Для виправлення помилок додайте using System.Collections.ObjectModel (рис. 18).

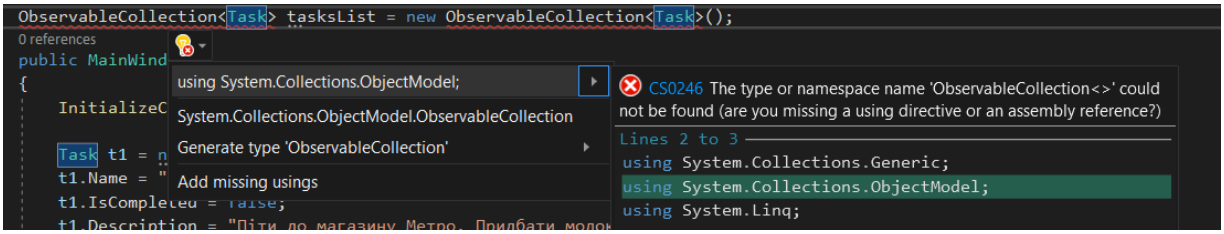


Рис. 12.18. Додавання using System.Collections.ObjectModel у проєкт

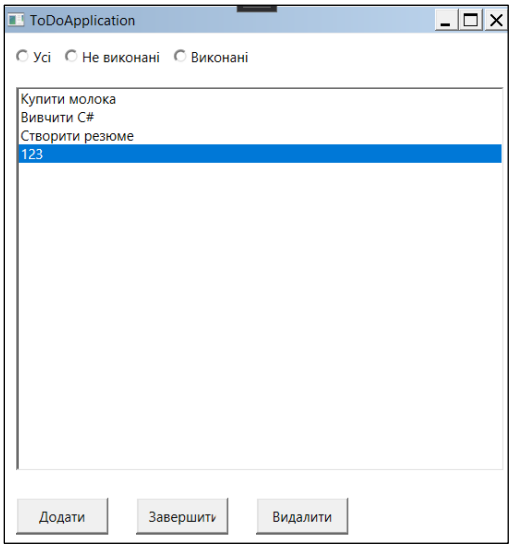


Рис. 12.19. Тестування додавання нової задачі до списку задач

19. Протестуйте проєкт. Створіть нову задачу, яка має з'явитися на головному вікні додатка (рис. 12.19).

20. Створіть обробник події на кнопку «Видалити» та «Завершити».

```
private void DeleteButton_Click(object sender, RoutedEventArgs e)
{
   int index = ToDoListBox.SelectedIndex;
   if (index != -1) //index=-1, якщо немає вибраних об'єктів
   {
        tasksList.RemoveAt(index);
    }
}
private void CompleteButton_Click(object sender, RoutedEventArgs e)
{
    int index = ToDoListBox.SelectedIndex;
    if (index != -1) //index=-1, якщо немає вибраних об'єктів
```

```
    {
       tasksList[index].IsCompleted = true;
    }
}
```

Протестуйте роботу кнопок.

21. Додайте функціональність фільтрації за дією елементів CheckBox. Створіть обробники події Checked для них.

```
private void AllRadioButton_Checked(object sender, RoutedEventArgs e)
{
      ToDoListBox.ItemsSource = tasksList;
}

private void NotCompletedRadioButton_Checked(object sender, RoutedEventArgs e)
{
     ObservableCollection<Task> filtered = new ObservableCollection<Task>();
     for (int i = 0; i < tasksList.Count; i++)
        {
            Task current = tasksList[i];
            if (current.IsCompleted == false)
            {
                    filtered.Add(current);
            }
         }
         ToDoListBox.ItemsSource = filtered;
    }

    private void ComplettedRadioButton_Checked(object sender, RoutedEventArgs e)
    {
          ObservableCollection<Task> filtered = new ObservableCollection<Task>();
          for (int i = 0; i < tasksList.Count; i++)
          {
             Task current = tasksList[i];
             if (current.IsCompleted == true)
             {
                    filtered.Add(current);
             }
          }
          ToDoListBox.ItemsSource = filtered;
    }
```

Протестуйте роботу проєкту з CheckBox.

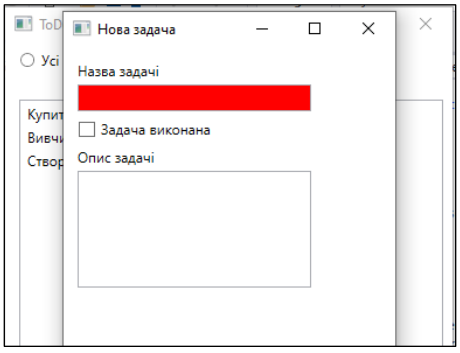


Рис. 12.20. Тестування роботи додатка ToDoApp під час створення нової задачі

22. Внесіть додаткову логіку: якщо під час створення нової задачі користувач не ввів назву, то дані не зберігаються, а поле з назвою підсвічується червоним (рис. 12.20).

```
private void SaveButton_Click(object sender, RoutedEventArgs e)
{
   if (NameTextBox.Text != "")
   {
      Task t = new Task();
      t.Name = NameTextBox.Text;
      t.IsCompleted = IsComplitedCheckBox.IsChecked.Value;
      t.Description = DescriptionTextBox.Text;
      Result = t;

      DialogResult = true;
   }
   else
   {
      NameTextBox.Background = Brushes.Red;
   }
 }
```

23. Перейдіть у конструктор головного вікна. У властивостях створіть події Closed і Loaded (рис. 12.21).

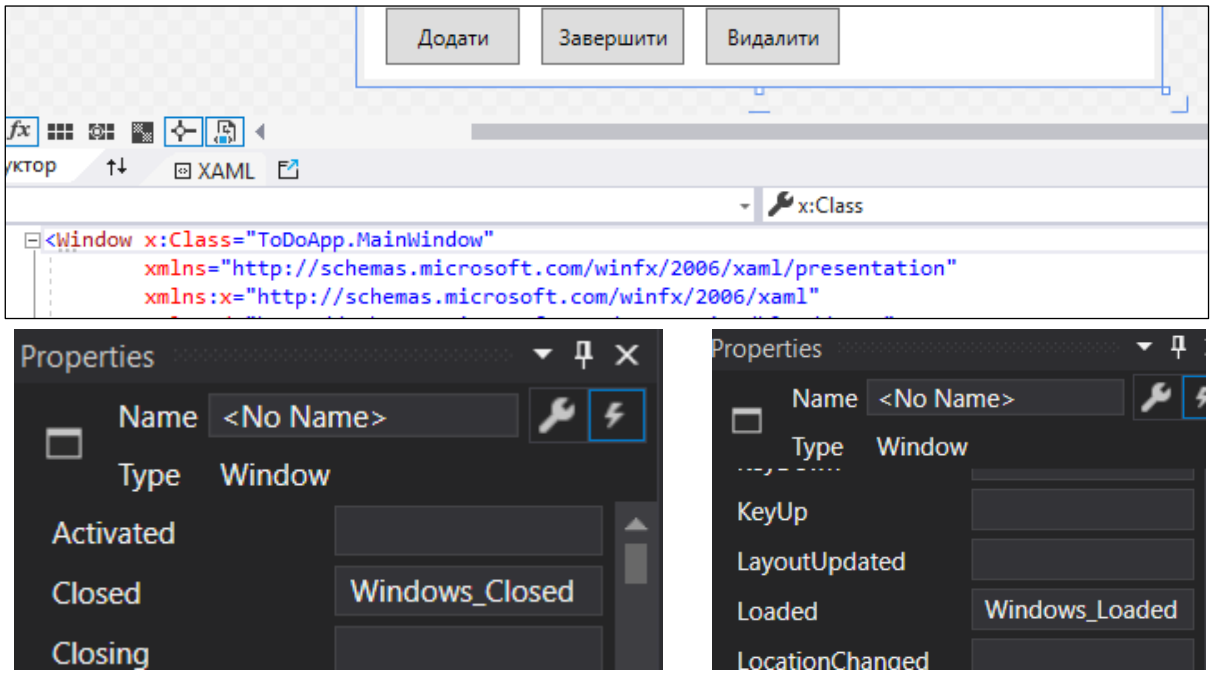


Рис. 12.21. Створення події Closed та Loaded для додатка ToDoApp

У події Loaded пропишіть завантаження задач із файла.

У події Closed реалізуйте збереження задач у файл. Перед обробниками подій оголосьте змінну:

```
string fileName = "data.bin";
```

У методі Window_Loaded перевірте, чи існує файл, і тільки тоді виконайте читання. Для цього додайте простір імен using System.IO.

```
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            //MessageBox.Show("Loaded");
            if (File.Exists(fileName))
            {
                //read
            }
        }
```

```
private void Windows_Closed(object sender, EventArgs e)
{
    BinaryFormatter
}
```

using System.Runtime.Serialization.Formatters.Binary;
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
Generate variable 'BinaryFormatter'
Use expression body for methods

CS0103 The name 'BinaryFormatter' does not exist in the current context
Lines 5 to 6
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;

Рис. 12.22. Додавання using System.Runtime.Serialization.Formatter.Binary

У методі Window_Closed створіть тип даних BinaryFormatter. Для цього додайте using System.Runtime.Serialization.Formatter.Binary (рис. 12.22).

```
private void Window_Closed(object sender, EventArgs e)
{
   BinaryFormatter formatter = new BinaryFormatter();
   Stream file = File.OpenWrite(fileName);
   formatter.Serialize(file, tasksList);
   file.Close();
}
```

24. У файлі MainWindow.xaml.cs прокоментуйте рядки 31–48, де вручну створювалися задачі (t1, t2, t3). Запустіть проєкт, створіть нову задачу – вона з'явиться у списку. Але після перезапуску додатка список знову порожній. Щоб це виправити, реалізуйте зчитування даних із файла. Для цього у класі Task (файл Task.cs) додайте [Serializable] (рис. 12.23).

```
namespace ToDoApp
{
    [Serializable]
    Ссылок: 14
    public class Task
    {
        ссылка: 1
        public string Name { get; set; }
        Ссылок: 4
        public bool IsCompleted { get; set; }
        Ссылок: 2
        public string Description { get; set; }

    }
}
```

Рис. 12.23. Додавання інтерфейсу Serializable до файлу Task.cs

25. Додайте код в обробник події Window_Loaded для читання збереженого списку задач.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
  if (File.Exists(fileName))
  {
     BinaryFormatter formatter = new BinaryFormatter();
     Stream file = File.OpenRead(fileName);
     tasksList = formatter.Deserialize(file) as ObservableCollection<Task>;
     file.Close();
     ToDoListBox.ItemsSource = tasksList;
   }
}
```

Протестуйте додаток.

3. Контрольні запитання

1. Що таке WPF і які його основні переваги над Windows Forms?
2. Які основні компоненти (елементи керування) WPF використовуються для створення ToDo-додатка?
3. Яку роль у WPF-розробці відіграє мова XAML?
4. Що таке патерн MVVM і як він застосовується у створенні ToDo-додатка?
5. Як реалізується двостороннє зв’язування даних (Data Binding) у WPF?
6. Що таке ObservableCollection і чому її доцільно використовувати для списку завдань?
7. Як у WPF реалізується команда (ICommand) та обробка подій?
8. Які властивості має клас «Завдання» (Task / ToDoItem) у типовому ToDo-додатку?
9. Як забезпечити збереження списку завдань між запусками програми (серіалізація, база даних, файл)?
10. Як можна реалізувати функціонал «додати завдання», «видалити завдання» та «позначити виконаним»?
11. Які стилі та ресурси можна застосувати у WPF для покращення зовнішнього вигляду ToDo-додатка?
12. Що таке DataTemplate і як його можна використати у списку завдань (ListBox / ListView)?
13. Які можливості надає Grid та StackPanel при компонуванні інтерфейсу користувача?
14. Як у WPF реалізується робота з подіями клавіатури або кнопок миші?
15. Які підходи існують для тестування функціоналу WPF-додатків?

СПИСОК ЛІТЕРАТУРИ

1. Глинчук Л. Я., Гришанович Т. О. Програмування: підручник. Луцьк: ВНУ ім. Лесі Українки, 2022. 160 с.

2. Дегтярьова Л. М., Гроза П. М., Сомов С. В. Навчальний посібник з дисципліни «Технології розробки програмного забезпечення» для студентів спеціальності 123 «Комп'ютерна інженерія». Полтава: ПолтНТУ, 2017. 218 с.

3. Івохін Є. В., Махно М. Ф., Піскунов О. Г. Розробка додатків засобами мови програмування C#: навч.-метод. посіб. для проведення лабораторних робіт для студентів вищих навчальних закладів спеціальності «системний аналіз». Київ: Вид.-полігр. центр «Київський університет», 2021. 100 с.

4. Коноваленко І. В., Марущак П. О. Платформа .NET та мова програмування C# 8.0: навч. посіб. Тернопіль: ФОП Паляниця В. А., 2020. 320 с.

5. Настенко Д. В., Нестерко А. Б. Об'єктно-орієнтоване програмування. Частина 1. Основи об'єктно-орієнтованого програмування на мові C#. Київ: НТУУ «КПІ», 2016. 76 с.

6. Нестерко А. Б., Настенко Д. В., Труніна Г. О. Обчислювальна техніка та програмування: Лабораторні роботи (Частина 1). Київ: КПІ ім. Ігоря Сікорського, 2020. 83 с.

7. Бендюг В. І., Комариста Б. М. Сучасні технології програмування: частина І. Практичні роботи: навч. посіб. для студ. Спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології. Київ: КПІ ім. Ігоря Сікорського, 2019. 269 с.

8. Коноваленко І. В., Марущак П. О., Савків В. Б. Програмування мовою C# 7.0: навч. посіб. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2017. 300 с.

9. Технічна документація Microsoft. URL: https://learn.microsoft.com/uk-ua/docs/

10. C#. Теорія та практика. URL: https://www.bestprog.net/uk/sitemap_ua/ c-3

11. C# .Net посібник. URL: https://programm.top/uk/c-sharp/tutorial

12. Офіційний сайт компанії Microsoft щодо технологій WPF та Windows Forms. URL: https://learn.microsoft.com/uk-ua/dotnet/desktop/

13. C# Tutorial. URL: https://www.theengineeringprojects.com

14. Уроки C#. URL: https://itproger.com/course/csharp

15. Desktop Guide (WPF .NET). URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0

16. C#. WPF. Приклад створення нового додатку, що підтримує платформу WPF. Створення додатку-вітання «HELLO WORLD!». URL: https://lnnk.in/aznR

17. C# Notes for professionals book. URL: https://goalkicker.com/CSharpBook

Навчальне видання

Цирульник Сергій Михайлович
Хмелівський Юрій Сергійович
Потапова Надія Анатоліївна
Зелінська Оксана Владиславівна

ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

Практикум до виконання лабораторних робіт:
навчальний посібник для студентів
галузі знань 12 Інформаційні технології

Редактор О. А. Солдатова
Технічний редактор Т. О. Важеніна-Гопрак

Підписано до друку 26.11.2025.
Формат 60 × 84/16. Папір офсетний.
Друк – цифровий. Умовн. друк. арк. 9,99.
Тираж 30. Зам. 51.

Донецький національний університет імені Василя Стуса
21021, м. Вінниця, 600-річчя, 21.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК № 5945 від 15.01.2018