

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ**

**РОБОЧА ПРОГРАМА,  
методичні вказівки та індивідуальні завдання  
до вивчення дисципліни «Візуальне  
програмування» для студентів спеціальності  
051 – економіка**

**Дніпро НМетАУ 2019**

УДК 330.46(07)

Робоча програма, методичні вказівки та індивідуальні завдання до вивчення дисципліни «Візуальне програмування» для студентів спеціальності 051 – економіка / Укл.: Л. І. Лозовська. – Дніпро: НМетАУ, 2019. – 53 с.

Викладені робоча програма, методичні вказівки до виконання контрольної роботи та індивідуальні завдання з дисципліни «Візуальне програмування», наведені теоретичні відомості з основ візуального програмування мовою С# для виконання індивідуальних завдань, а також приклади їх виконання.

Призначені для студентів спеціальності 051 – економіка, спеціалізації «Інформаційні технології та моделювання в економіці» заочної форми навчання.

Укладач: Л.І. Лозовська, канд. фіз.-мат. наук, доц.

## ЗМІСТ

	Вступ.....	4
1	Загальні методичні вказівки.....	5
2	Рекомендована література.....	5
3	Робоча програма.....	5
4	Методичні вказівки до вивчення дисципліни та завдання.....	6
	Тема 1 Середовище Visual Studio. Консольні додатки.....	6
	Тема 2 Середовище Visual Studio. Windows-додаток.....	15
	Тема 3 Компоненти вікна. Структура класу Form1.....	18
	Тема 4 Події, пов'язані з вікном. Механізм обробки подій.....	23
	Тема 5 Обробка помилок введення за допомогою виключень. Механізми введення і контролю даних.....	29
	Тема 6 Типові алгоритми обробки масиву.....	35
	Тема 7 Робота с масивами випадкових чисел. Методи класів Random і Math.....	37
	Тема 8 Порядок та особливості розробки функцій.....	44

## ВСТУП

Візуальне програмування – це вид програмування, що передбачає створення програм за допомогою наглядних засобів, тобто шляхом оперування графічними об'єктами, а не написання програмного коду в текстовому вигляді. Основною суттю візуального програмування є побудова розв'язку поставленої задачі за допомогою візуальних заготовок, які вставляються у форму, присвоєння значень їхнім атрибутам і створення чи застосування потрібних для розв'язання даної задачі методів.

Візуальне програмування виникло на основі об'єктно-орієнтованого програмування, як засіб автоматизації його процесів. Тепер для складання програми користувачу необхідно маніпулювати наданими графічними засобами – компонентами. Компоненти мають певні атрибути (властивості). Властивості можуть набувати значення зі заздалегідь фіксованого значення чи набору, або можуть бути придумані користувачем. Для опрацювання числових та інших даних розробляються відповідні методи об'єктів.

В даному посібнику паралельно розглядаються можливості системи програмування і конструкції мови програмування, що дозволяє студентам з перших занять отримати уявлення про технології розробки Windows-додатків. Після вивчення дисципліни студент може набути всі необхідні навички і уміння для проектування програм достатньо високої складності.

В якості мови програмування використовується мова C#, що входить в комплект мов середовища програмування Visual Studio 2010. В додатках наведені відомості про компоненти, які використовуються під час викладення матеріалу, а також відомості про властивості та застосування стандартних вікон операційній системі Windows. Список літератури містить підручники, які дозволять студенту самостійно розширити об'єм знань в області програмування в операційній системі Windows.

У результаті вивчення дисципліни студент повинен знати прийоми візуального проектування автономних додатків мовою програмування C# з використанням інструментальних засобів Microsoft Visual Studio.NET, структуру і принципи роботи додатків Microsoft Windows, аспекти розробки додатків з візуальним інтерфейсом; вміти створювати проекти додатків і підлагоджувати додатки C# з візуальним графічним інтерфейсом, додавати в форми такі елементи управління, як текстові поля і кнопки, налагоджувати

зовнішній вигляд і поведінку форми, створювати оброблювачі подій, розміщених в вікні форми, використовувати елементи управління.

## **1 ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ**

Відповідно до навчального плану на вивчення дисципліни «Візуальне програмування» для студентів заочної форми навчання заплановано 180 годин, які розподілені за видами занять у такий спосіб:

аудиторні заняття - 28 годин;

з них:

лекції – 12 години;

лабораторні заняття – 16 годин;

самостійна робота – 152 години.

## **2 РЕКОМЕНДОВАНА ЛІТЕРАТУРА**

1. Лозовська Л.І. Візуальне програмування. Навч. посіб. / Л.І. Лозовська, Л.М. Бандоріна, Р.В. Савчук/ – Дніпро: НМетАУ, 2017.
2. Павловская, Т. А. С#. Программирование на языке высокого уровня / Т. А. Павловская. – СПб: Питер, 2009. – 432 с.
3. Фаронов, В. В. С#. Программирование на языке С# / В. В. Фаронов. – СПб: Питер, 2009. – 240 с.
4. Скит, Джон С#. Программирование для профессионалов / Джон Скит. – СПб: Питер, 2011. – 544 с.
5. Уотсон, К. Visual C# 2010. Полный курс / К. Уотсон., К. Нейгел. – СПб: Питер, 2011. – 955 с.
6. Кручинин, В. И. Программирование на языке С#. Часть 1. Основы языка: учеб. пособие / В. И. Кручинин. – Волгоград: ИУНЛ ВолгГТУ, 2011. – 92 с.
7. Кручинин, В. И. Программирование на языке С#. Часть 2. Объектно-ориентированное программирование: учеб. пособие / В. И. Кручинин. – Волгоград, ИУНЛ ВолгГТУ, 2012. – 72 с.

### 3 РОБОЧА ПРОГРАМА

Розподіл навчальних годин з дисципліни «Візуальне програмування» для студентів заочної форми навчання за темами та видами занять наведено в таблиці 3.1.

Таблиця 3.1 – Розподіл навчальних годин за темами та видами занять

№ теми	Назва теми	Кількість годин			
		Разом	Лекції	Лабораторні заняття	Самостійна робота
1	Середовище Visual Studio. Консольні додатки	8	1	1	6
2	Середовище Visual Studio. Windows-додаток	8	1	1	6
3	Компоненти вікна. Структура класу Form1	19	1	3	15
4	Події, пов'язані з вікном. Механізм обробки подій	18	1	2	15
5	Обробка помилок введення за допомогою виключень. Механізми введення і контролю даних	29	2	2	25
6	Типові алгоритми обробки масиву	30	2	3	25
7	Робота з масивами випадкових чисел. Методи класів Random і Math	34	2	2	30
8	Порядок та особливості розробки функцій	34	2	2	30
Разом:		180	12	16	152

## 4 МЕТОДИЧНІ ВКАЗІВКИ ДО ВИВЧЕННЯ ДИСЦИПЛІНИ

### ТЕМА 1 СЕРЕДОВИЩЕ VISUAL STUDIO. КОНСОЛЬНІ ДОДАТКИ

#### *Призначення середовища програмування*

Середовище програмування *Visual Studio* призначене для розробки програм мовами високого рівня. Використовуючи це середовище, можна розробляти програми такими мовами як *Basic, C#, C++, F#*.

#### *Запуск середовища*

Запустити середовище можна через кнопку «ПУСК» в меню «Програми». Порядок дій: *Пуск – Все програми – Microsoft Visual Studio*.

#### *Початок роботи*

Після запуску на екрані з'явиться стартове вікно середовища Visual Studio, яке має наступний вигляд (рис.1.1):

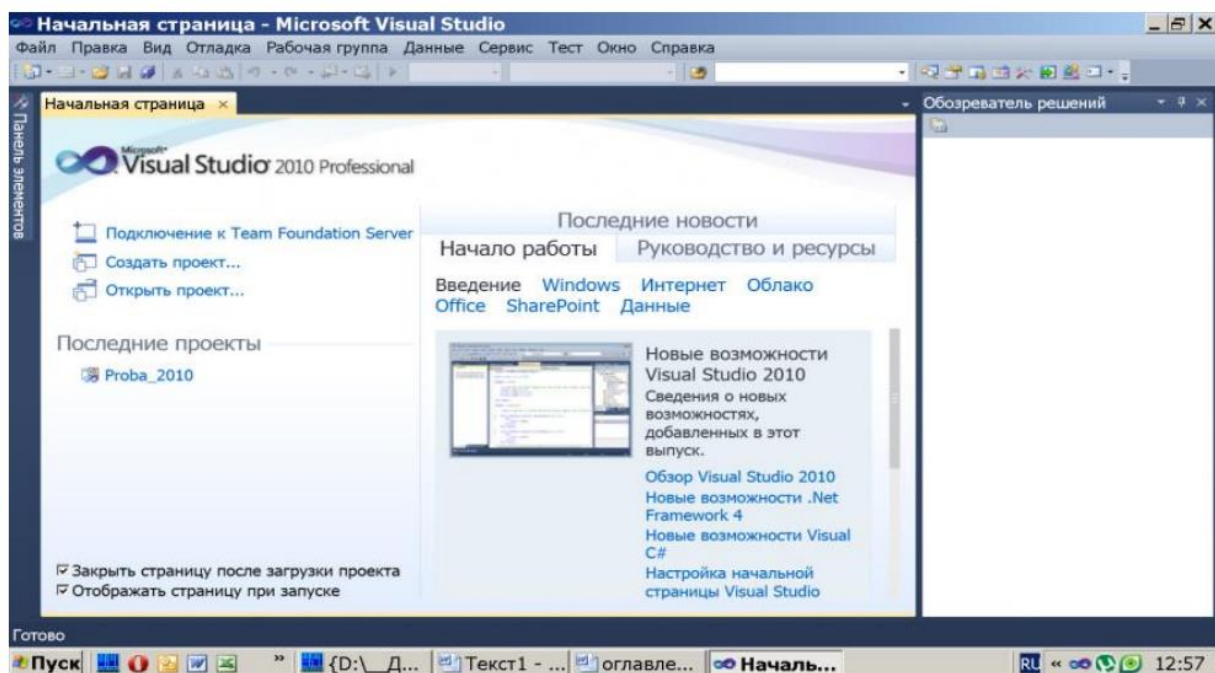


Рисунок 1.1 – Стартове вікно середовища

Стартове вікно містить дві частини: стартова сторінка і оглядач рішень. На стартовій сторінці є корисні можливості. Використовуючи посилання *Открыть проект*, програміст може продовжити роботу з одним із додатків, який він розробив раніше. Якщо необхідно створити новий додаток, то можна скористатися посиланням *Создать проект*. Закрийте стартову сторінку.

Зверніть увагу на головне меню програми. Лінійка меню містить можливості, з якими будемо познайомимося пізніше.

Вікно справа називається **Обозреватель решений**. Тут буде відображатися структура додатка. Поки вікно пусте. Воно буде наповнюватися під час включення в додаток різноманітних компонентів. Це вікно являється службовим. Оберемо в головному меню закладку **Вид**. Випадаючий список містить різні назви службових вікон, серед яких є і вікно стартової сторінки. Відобразимо вікно **Вывод**. Закріпимо його в нижній частині екрана. Тут будуть відображатися результати роботи програміста: протокол створення програми, повідомлення про синтаксичні помилки, попередження та інше.

### ***Вибір варіанта власного проекту***

Кожний додаток розробляється в середовищі **Visual Studio** як проект. В одному додатку може бути кілька проектів. Створимо проект, в якому будуть міститися всі компоненти, необхідні для роботи програми. Це можна зробити, використовуючи стартову сторінку, а можна скористатися головним меню. Виконаємо дії: **Файл – Создать – Проект**. На екрані з'явиться вікно. Ліва секція в цьому вікні (рис. 1.2) містить список мов програмування, для яких можна будувати проекти. Обираємо мову C#, зазначаючи при цьому, що розробка буде проводитися для роботи під Windows.

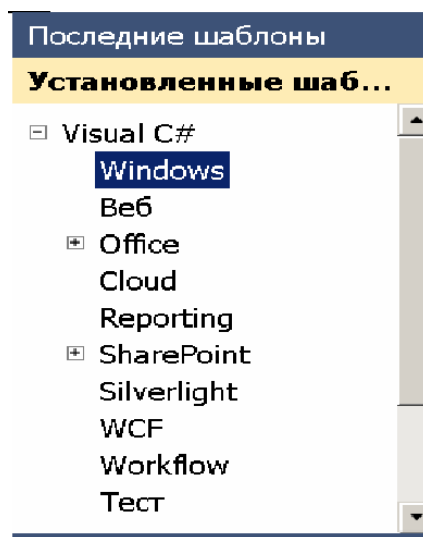


Рисунок 1.2 – Список мов

В центральній секції вікна (рис. 1.3) перераховані типи шаблонів проектів.

Мовою C# можна розробляти додатки різних призначень. Наприклад:



- **Додаток Windows Forms.** Це програма, яка використовує всі можливості ОС Windows, такі як: багатозадачність, графічний багатовіконий режим, управління подіями.
- **Консольний додаток.** Це додаток, який працює в середовищі Windows в консольному режимі, коли для введення і виведення даних використовується текстовий режим одного вікна. Робота такої програми схожа на роботу в середовищі DOS.
- **Бібліотека класів.** Це не додаток в звичайному сенсі, а динамічна бібліотека (dll).
- **Служба Windows.** Це проект для створення служб Windows.

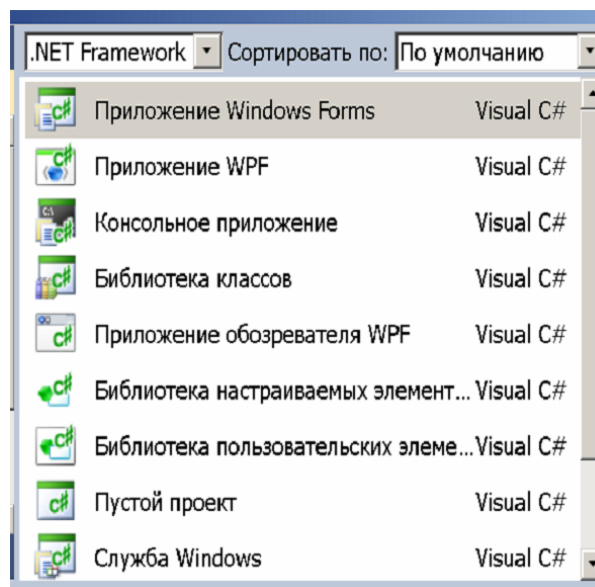


Рисунок 1.3 – Призначення додатка

### ***Ім'я та місце розташування додатка***

В нижній секції вікна є три поля для введення даних:

**Имя.** Це поле імені проекту. Щоб задати ім'я проекту, можна використовувати літери (кирилицю і латиницю) і цифри.

**Имя решения.** Це ім'я додатка, який створюється. Якщо в додатку один проект, то імена співпадають.

**Расположение.** В цьому полі указується місце розміщення додатка. Указується повний шлях. Замість набору можна використати кнопку «**ОБЗОР**» і обрати потрібне місце. Можна рекомендувати вибір місця розміщення з точністю до деякої папки, а потім врахувати, що для додатка в обраній папці

буде створена нова папка (галочка біля прапорця «*Создать каталог для решения*»).

### ***Створення консольного додатка***

Оберемо тип проекту **Консольний додаток**. Залишимо те ім'я, яке запропонує програмне середовище: **ConsoleApplication1**. В якості місця розміщення обираємо загальну папку студентів **D:\USERS\** або особисту папку студента. Натискаємо кнопку «**ОК**». Програмне середовище відкриє вікно з заготовкою консольного додатка. Тепер у вікні **Обозреватель решений** видно склад нашого проекту. Майстер проекту розмістив в них такі компоненти, яких достатньо для організації консольного додатка. Кожне рішення в С# може містити кілька проектів. В нашому випадку – це один проект під назвою **ConsoleApplication1**. В складі проекту вже є один програмний файл – **Program.cs**. Тип файлу (**cs**, сі-шарп) визначає його як файл, що містить код мовою С#. Вміст цього файлу відображено в лівій секції вікна.

Закриємо програмне середовище. Знайдемо, де саме на жорсткому диску розмістилося наше рішення. Через провідник або **Мой компьютер** відкриємо папку **USERS** і знайдемо папку **ConsoleApplication1**. Розкриємо цю папку. В ній побачимо ще одну папку **ConsoleApplication1** і один файл з розширенням **sln**. Саме цей файл містить інформацію про рішення. Його можна використовувати як файл, що запускає рішення. Двічі клацнемо лівою кнопкою мишки: рішення знову розкриється.

Знову закриємо вікно рішення.

Звернемо увагу на інший файл: **Program.cs**. Це файл, який містить код мовою С#. Спробуємо використати його в якості стартового – двічі клацнемо його мишкою. Відкриється тільки сам файл, але не рішення. Закрийте файл і знову відкрийте рішення. Перейдемо до розгляду заготовки в файлі **Program.cs**.

### ***Вміст заготовки файлу***

Зверніть увагу на структуру інформації в програмному вікні. Виділено кілька блоків. Кожен блок можна відобразити в розгорнутому (–) або зжатому (+) вигляді. Це зручно, коли маємо великий програмний текст. Частина блоків можна згорнути, щоб не мішали, а розкрити тільки той блок, що потрібний. Розгорнемо всі блоки і розглянемо кожний з них докладніше.

В першому блоці описані чотири рядки:

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

Слово *using* використовується для опису системних можливостей, які надані програмісту. Всі мови програмування мають системні можливості, що оформлені в вигляді системних бібліотек. В мові C# використовуються особлива термінологія для опису системних можливостей. В цьому випадку мова йде про простір імен. Цей термін має своє інтуїтивне пояснення. Кожна вбудована можливість має деяке ім'я, а ряд близьких за сенсом можливостей об'єднуються в групи. Ряд імен можливостей існують в деякому просторі (*namespace*). Прикладом такого простору є простір *System*. Це основний простір імен. Поставте в першому рядку після слова *System* крапку і подивіться, що буде. У вікні, що випадає, є ще багато уточнень, які, в свою чергу, можуть бути також іншими просторами імен або іменами класів, які також містять у собі вбудовані засоби мови.

Три інші рядки цього блока використовують підпростори імен простору *System*. Описання просторів дозволяють викликати для виконання вбудованих можливостей за їх іменами, тому що самі простори вже описані. Наприклад, всередині простору *System* є простори імен *Linq*, *Text* і *Collection*, а в останньому – простір імен *Generic*. Які саме можливості будуть потрібні, визначає сам програміст. Він підключає в своїй програмі потрібні йому простори імен. Наша заготовка створена середовищем програмування, яке у відповідності до обраного типу проекту (*консольний додаток*) підключило саме ці простори імен.

Перейдемо до розгляду другого блока. Він називається так, як і наш проект – *ConsoleApplication1*. Перед ним розміщено слово *namespace*. Це значить, що середовище автоматично створило в нашій програмі новий простір імен. Тепер всі імена (ідентифікатори), які ми будемо використовувати в тексті, будуть автоматично включені в простір імен *ConsoleApplication1*. Таким чином, в процесі роботи будуть використовуватися імена і системних просторів, і простір імен *ConsoleApplication1*. Середовище програмування буде «підказувати» нам імена з усіх просторів імен, що використовуються. Перевіримо це експериментально. В просторі імен *ConsoleApplication1* є опис класу з іменем *Program*. Будь-який клас – це структурна одиниця програми, її окремий модуль, де описуються алгоритми. Опис алгоритмів оформлено у вигляді функцій. Кожна функція має ім'я, яке використовується під час виклику

функції для виконання. Власне, саме імена класів і функцій і визначають функціональність простору імен. Зазначимо, що замість терміна «ім'я функції», зазвичай використовується термін «метод». Наприклад, в складі класу *Program* є метод *Main*. Це особливий метод. Він є в будь-якій програмі, причому зустрічається один раз. Це головний метод програми. Виконання будь-якої програми починається саме з нього. Крім нього в програмі ми можемо описати інші методи, в різних класах. Переконаємося, що система «бачить» простір імен *ConsoleApplication1*. Для цього установимо курсор всередині методу *Main* (між фігурними дужками) і наберемо фразу *ConsoleA*. Зверніть увагу, що в процесі набору система намагається підказувати різні продовження набору. Коли ми наберемо фразу *ConsoleA*, то побачимо підказку. Оберемо її мишкою і натиснемо кнопку «*ENTER*», щоб слово *ConsoleApplication1* з'явилося в тексті програми. Затим натиснемо клавішу «*КРАПКА*». Система підказує, що в просторі *ConsoleApplication1* є клас *Program*. Повторимо ще раз: мишкою оберемо ім'я класу, натиснемо кнопку «*ENTER*», а потім знову клавішу «*КРАПКА*». І знову побачимо підказку: можна обрати метод *Main* і ще пару методів. Висновок очевидний: при розробці програмного коду програміст постійно отримує дружню підтримку системи.

### *Набір тексту програми*

Текст програмного коду в консольному додатку повинен розміщуватися всередині методу *Main* між фігурними дужками. Наберемо у вікні файлу *Program.cs* текст. Не рекомендується копіювати наведений текст – економія часу може зробити ведмежу послугу, тому що при копіюванні ми не побачимо, як середовище розуміє текст. При наборі різні слова будуть зафарбовуватися різним кольором, а щось середовище буде нам підказувати. В середині методу *Main* набираємо наступний текст:

```
int a,b,c1; // оголошення змінних цілого типу  
a=Console.ReadLine(); // введення значення з клавіатури  
b=3; // привласнення  
c1=a+b; // обчислення  
// виведення результатів на екран  
Console.WriteLine("Сума={0}",c);
```

Різні фрагменти тексту будуть відображатися різним кольором. Зелений колір використовується середовищем програмування для відображення коментарів, синій – для службових слів. Наприклад, слово *int* вказує, що три

імені (*a, b, c1*) – це імена змінних, які при виконанні програми будуть містити значення цілого типу. Голубим кольором відображаються імена класів. Червоний колір – це колір для відображення рядків.

Суть програми, яку зараз набрали, достатньо проста. Оголошується, що буде використано три змінних цілого типу. Дві з них отримують числові значення: значення першої вводиться з клавіатури, а друга отримує значення під час виконання програми. Значення третьої змінної обчислюється як сума двох інших. Отриманий результат виводиться на екран, після чого програма завершає роботу.

### ***Підготовка програми до виконання***

Написати текст програми – ще не означає, що програма уже створена. Важливо написати її так, щоб в ній не було помилок. Середовище програмування частину програмного коду підкреслює хвилястою лінією – це свідчить про наявність помилок. В другому рядку підкреслено майже весь оператор, в п'ятому – змінна *c*. Найпростіший спосіб перевірити нормальність написаного коду – це спробувати виконати програму. Середовище програмування перед запуском програми обов'язково перевірить програму на наявність помилок і повідомить про о них. Але можна спробувати розібратися з помилками до запуску. Для цього розглянемо підкреслені елементи.

Встановимо курсор мишки на змінну *c* в п'ятому рядку тексту. На екрані з'явиться підказка: ***«Елемент 'с' не існує в поточному контексті»***. Як це розуміти? Справа в тому, що п'ятий оператор видає на екран обчислене значення змінної *c*. Але, якщо уважно проаналізувати текст програми, то можна побачити, що в першому рядку оголошені для використання змінні *a, b* і *c1*. Змінної *c* серед них немає – її взагалі ніде ніхто не оголошував. Це порушення відзначене середовищем. Після аналізу можна зрозуміти як виправити помилку. В програмі є змінна *c1*, яка оголошена і використовується в четвертому рядку для підрахунку суми. Очевидно, що цю змінну і потрібно використовувати в п'ятому рядку. Замінімо *c* на *c1*.

Помилка в другому рядку ідентифікується більш складно. При наведенні курсору на цей рядок ми отримаємо значно більший об'єм інформації, ніж в першому випадку. Система прокоментує нам, що собою представляє указана мовна конструкція і перерахує деякі особливості її використання. Але в кінці цієї інформації також буде описана суть помилки: ***«неявне перетворення типу string в int неможливе»***. В чому тут справа? Вираз *Console.ReadLine()* викликає для виконання метод *ReadLine* з класу *Console*. Цей метод вводиться з

клавіатури всі символи, які набере користувач до того, як буде натиснута клавіша «**ENTER**». Всі набрані символи будуть сформовані в один рядок, тобто значення типу *string*. Це значення в другому операторі привласнюється змінній *a*. Але змінна *a* оголошена в програмі як змінна типу *int*, тобто як цілочислова. Таке привласнення заборонено: неможна цілочисловій змінній привласнити рядкове значення. Перед привласненням необхідно виконати перетворення. Значить, цю помилку потрібно також виправляти програмісту. Це зробити нескладно, якщо знати, що в мові є відповідні можливості. В просторі імен *System* є клас *Convert*, в якому зберігаються різноманітні методи перетворення рядкових значень в інші типи даних. Існує метод *ToInt32*, який як раз і виконує перетворення рядка в ціле число. Скористаємося цим і замінімо другий рядок тексту на наступний:

```
a = Convert.ToInt32(Console.ReadLine());
```

Зазначимо, що текст більше не підкреслено. Схоже, що помилок немає. Можна спробувати виконати програму.

### ***Виконання програми***

Для виконання програми скористаємося комбінацією клавіш «**CTRL-F5**». Отримаємо чорне пусте вікно. Це вікно, в якому відображаються всі результати виконання нашої програми. Так як наша програма розроблена як консольний додаток, то ми бачимо на екрані саме таке вікно. Але, воно ще пусте: програма тільки почала виконання і ще не завершилась. Вона чекає. Згадаємо, що другий рядок програми – це введення значення з клавіатури. Значить, потрібно це значення ввести. Введемо, наприклад, число – 8 і натиснемо клавішу «**ENTER**». Відобразиться наступне вікно, в якому буде видно результат виконання оператора введення, виведення і рядок, який пропонує для завершення роботи програми натиснути будь-яку клавішу.

Якщо натиснути яку-небудь клавішу, то чорне вікно закриється. Програмне середовище по завершенні виконання програми штучно затримала вікно, щоб користувач встиг побачити результати своєї роботи. Рядок завершення видається тому, що ми запустили програму з середовища програмування. Якщо б ми запустили готовий програмний файл, то цього рядка не було б. Перевіримо це.

Відкриємо папку *D:\USERS\ ConsoleApplication1\ ConsoleApplication1 \bin\debug*. В цій теці знаходиться готова програма *ConsoleApplication1* (exe-файл). Спробуємо її виконати. Після запуску ми зможемо ввести дані, але зразу

після цього вікно закривається – так швидко вона завершує свою роботу (в тексті програми немає ніяких затримок). А ось при запуску з середовища вікно на екрані затримується до натискання якої-небудь клавіші. Натисніть будь-яку клавішу. Робота з програмою завершена.

Завершіть роботу з програмним середовищем.

### ***Завдання для самостійної роботи***

1. При запуску розглянутого прикладу робота програми починається з відображення пустого вікна. Користувач повинен сам здогадатися, що можна вводить ціле значення. Доопрацюйте текст програми. Видайте на екран пояснюючі повідомлення. Наприклад, запропонуйте користувачу ввести значення.

2. Розробіть власний простий проект, збережіть його і продемонструйте роботу з ним.

## **ТЕМА 2 СЕРЕДОВИЩЕ VISUAL STUDIO. WINDOWS-ДОДАТОК**

### ***2.1 Створення Windows-додатка***

Запускаємо середовище *Visual Studio* і на стартовій сторінці обираємо посилання **Создать проект**. В розділі шаблонів обираємо **Приложение Windows Forms**. **Имя проекта і приложения – Проба**. Місце розміщення: **D:\USERS\**. Буде створена заготовка (рис. 2.1).

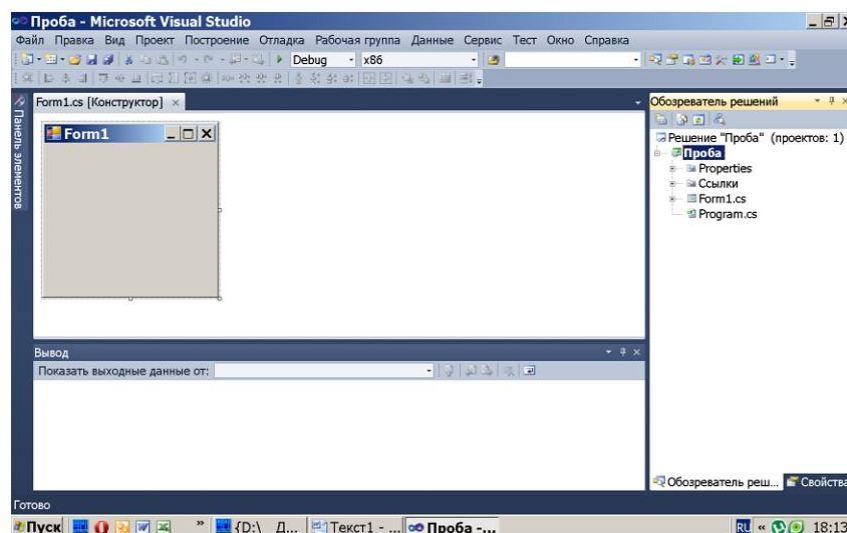


Рисунок 2.1 – Заготовка Windows-додатка

В вікні *Обозреватель решений* видно склад нашого проекту. Він не пустий – в папках є компоненти, необхідні для організації *Windows*-додатка. Кожне рішення в C# може містити кілька проектів. Зараз існує один проект під назвою *Проба*, у складі якого є два програмних файли – *Program.cs* і *Form1.cs*.

## 2.2 Склад заготовки

На екрані зліва ми бачимо відображення складу файлу *Form1.cs*.

Зверніть увагу на закладку, що відповідає цьому відображенню. Закладка позначена словом *[Конструктор]*. Файл відображається візуально. Показано, які об'єкти в ньому використовуються. Зараз в ньому немає ніяких об'єктів, крім самого вікна. Установимо мишку на вікно і натиснемо праву кнопку. Відобразиться відповідне контекстне меню.

Оберемо режим *Перейти к коду*. З'явиться нова закладка з іменем *Form1*, але без позначки *Конструктор* (рис. 2.2). Відображено склад файлу *Form1.cs*, але уже не у вигляді об'єктів, а у вигляді програмного коду мовою C#. Операторів, що підключають до програми системні простори імен, тут значно більше, ніж в консольному додатку. Є вже відомий нам простір імен *System*. Інші рядки підключають підпростори імен простору *System*. Є серед них і простір *System.Windows.Forms*, який містить все необхідне для розробки вікон *Windows*-додатка.

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

Рисунок 2.2 – Програмний код Form1



Є простір імен нашої програми – **PROBA**. В ньому є опис класу з іменем **Form1**. Описи алгоритмів оформлені у вигляді функцій (методів).

Наприклад, у складі класу **Form1** уже є метод **Form1()**. Це особливий метод: його називають конструктором об'єктів класу. В даному випадку – це конструктор вікна **Form1**. Що він містить і як саме він будує об'єкти – буде вивчено пізніше.

Крім конструктора в складі класу будуть у інші методи, які напише програміст. Всі ці методи будуть мати відношення саме до вікна **Form1**.

Розглянемо модуль **Program.cs**. Оберемо його мишкою у вікні **Обозреватель решения**. В цьому модулі є клас – **Program**. Він містить метод **Main()**. Але, на відмінність від консольного додатка тут цей метод не пустий – в ньому вже є три оператори. Якщо послідовно навести курсор мишки на кожний оператор, то можна отримати відомості про призначення кожного оператора.

В цілому сенс операторів такий. Два перших оператори дозволяють використовувати візуальні стилі в процесі виконання програми і задають деякі стандартні режими управління програмою. Третій оператор забезпечує запуск процесу, який пов'язаний з вікном **Form1**. Для цього використовується метод **Run()** з класу **Application**. В круглих дужках указано ім'я того об'єкта, з яким пов'язаний процес. А тепер за допомогою закладки **Form1.cs [Конструктор]** переключимося в вікно візуального відображення форми **Form1**.

Установимо курсор мишки на вікні, викличемо контекстне меню і оберемо **Свойства**. Відобразиться додаткове вікно властивостей форми. Властивостей багато, вони різні, знайомитися с ними треба поступово, по мірі необхідності.

Розглянемо властивість **Name**, яка визначає ідентифікатор вікна. Фактично це ім'я змінної: **Form1**. Саме це ім'я буде використовуватися в програмному коді при зверненні до об'єкту вікна **Form1**.

Властивість **Text** містить заголовок вікна. Її можна змінити, це приведе до зміни назви вікна.

Властивість **MaximizeBox** дозволяє заборонити розгорнути вікно на повний екран, якщо установити значення **False**.

### **Завдання для самостійної роботи**

*Проекспериментуйте з установкою різних значень у властивостях форми. Спробуйте експериментально зрозуміти сенс різних значень. Щось можна побачити зразу при зміні значення властивості, а щось – тільки після*

запуску програми. Спробуйте змінити значення деяких властивостей, запустивши програму (**Ctrl-F5**), подивіться, як різні значення властивостей проявляються при виконанні програми.

Дослідіть:

1. **AutoSizeMode** (переконайтеся, що при одному із значень вікно неможливо розтягнути).
2. **BackColor** (установіть фоновий колір).
3. **BackgroundImage** (встановлює фонову картинку).
4. **ControlBox** (при установці значення *false* у вікна відсутні кнопки управління; завершити програму тепер можна тільки через диспетчер задач Windows).
5. **Cursor** (змінюється зовнішній вид курсору мишки).
6. **Enabled** (забороняє доступ до елементів вікна – якщо вони там є).
7. **FormBorderStyle** (різноманітні види оформлення границь вікна).
8. **Icon** (зміна значка-іконки в заготовочному рядку вікна).
9. **ShowIcon** (показати або приховати іконки).
10. **MaximizeBox** (активізація або блокування кнопки розгортання вікна на повний екран).
11. **MinimizeBox** (активізація або блокування кнопки згортання вікна).
12. **Opacity** (ступінь прозорості вікна).
13. **Size** (установка розмірів вікна).
14. **Text** (установка надпису в заголовок вікна).

## ТЕМА 3 КОМПОНЕНТИ ВІКНА. СТРУКТУРА КЛАСУ **Form1**

### 3.1 Організація класу **Form1**

Створимо нове рішення з іменем **Задача1**. Заготовка має вікно **Form1**. Відобразимо програмний код, що відповідає цьому вікну. Зазначимо, що у відповідності з назвою задачі простір імен також називається **Задача1**, а клас, що відповідає вікну, називається **Form1**. Клас містить лише один метод з іменем **Form1()** – це конструктор об'єктів класу. Всередині методу в фігурних дужках написано вираз:

**InitializeComponent();**

Це вираз являється оператором мови C#. Ім'я **InitializeComponent()** – це ім'я методу. Даний оператор викликає метод для виконання. Те, що це саме метод, можна встановити за круглими дужками – для виклику методу

використовується саме така форма: ім'я з круглими дужками. В круглих дужках при виклику методу можуть бути записані якісь значення або імена змінних, але в нашому випадку не вказано нічого – тому що просто не потрібно. Метод, що викликається, повинен бути десь описаний – адже якщо метод викликається для виконання, то повинно бути відомо, що буде зроблено, який саме алгоритм або алгоритми будуть виконані. Оголошення може бути вбудованим в систему або наперед описаним програмістом в його програмі. В нашому випадку алгоритм методу *InitializeComponent()* не являється системним, його оголошення знаходиться в нашій програмі. Питання: де саме? Звернемо увагу на заголовок оголошення класу:

**public partial class Form1: Form**

В заголовку присутнє слово *partial*, воно означає «частковий». Це значить, що опис класу в цьому модулі наведено не повністю. Наведена тільки частина опису класу, а десь є ще одна частина. Спробуємо відшукати другу частину опису класу. В вікні *Обозреватель решений* розкриємо плюс біля імені файлу *Form1.cs*. Ми побачимо, що є ще один модуль, який має відношення до вікна *Form1: Form1.Designer.cs*. Подвійним кліком мишки розкриємо програмний модуль *Form1.Designer.cs*. Саме тут і міститься друга частина опису класу *Form1*. Зверніть увагу на рядок:

**partial class Form1**

Це означає, що ми бачимо ще одну частину класу *Form1*. Давайте ще раз перерахуємо, що ми встигли зробити:

- замовили програмному середовищу створення проекту *Задача1*;
- вказали, що проект повинен бути *Windows-додатком*;
- вказали, що місцем його розташування буде папка *USERS*.

Після цього отримали проект з одним вікном. Вікно, правда, доки є пустим. Але в склад нашої програми середовище програмування уже включило два модулі: *Form1.cs* і *Form1.Designer.cs*. Обидва імені ми бачимо в вікні *Обозреватель решений*. При автоматичній побудові нашої програми середовище створило опис об'єкта *Form1* у вигляді класу з таким же ім'ям. При цьому опис класу розділено на два модулі. Чому? Необхідно це зрозуміти і запам'ятати. Середовище програмування «прийняло мудре рішення», розділивши клас на два модулі.

1. Перший модуль *Form1.Designer.cs* використовується самим середовищем для автоматичного розміщення програмного коду, який пов'язаний з об'єктом. Зокрема, при створенні додатка середовище створило

вікно, включивши в цей модуль всі оператори, що необхідні для відображення вікна на екрані.

2. Другий модуль *Form1.cs* призначений для програміста, тобто для нас. Тут ми будемо розміщувати програмний код, який напишемо самі.

А тепер поглянемо, що саме добавила система. Відкриємо закладку *Form1.Designer.cs*. Знайдемо сірий рядок:

***Код, що автоматично створений конструктором форм Windows.***

Розкриємо плюс проти нього. Відкриється секція програми, обмежена фразами *#region* і *#endregion*. Фрази, що починаються знаком *#*, називаються директивами. Ці директиви обмежують область програмного коду, який створено автоматично самим середовищем програмування. Зверніть увагу на наступний фрагмент коду:

```
private void InitializeComponent()  
{ this.components= new System.ComponentModel.Container();  
  this.AutoScaleMode= System.Windows.Forms.AutoScaleMode.Font;  
  this.Text = "Form1";  
}
```

Це і є опис методу, ім'я якого *InitializeComponent()*. Всередині опису методу між фігурними дужками записані оператори, які «рисують» вікно на екрані. Спробуємо зрозуміти сенс цих операторів. Перший оператор оголошує поточне вікно контейнером. Це означає, що вікно може містити в собі інші об'єкти. Другий оператор забезпечує автоматичне масштабування. Наприклад, зміна розмірів тексту при зміні розмірів вікна. Третій оператор задає назву вікна, яка відображається в заголовку. Назва задається через властивість *Text*. Цю властивість можна установити і вручну через вікно властивостей.

Таким чином, в модулі *Form1.cs* в методі-конструкторі викликається метод *InitializeComponent()*. При його виклику виконуються ті самі три оператори, які знаходяться в оголошенні методу в модулі *Form1.Designer.cs*. При виконанні операторів на екрані з'являється програмне вікно.

Тепер спробуємо наповнити наше вікно іншими об'єктами. Для цього скористаємося панеллю елементів.

### **3.2 Панель елементів**

Відкриємо закладку з візуальним відображенням вікна **Form1.cs [Design]**. Оберемо мишкою вікно, а потім використаємо головне меню програмного середовища: **Вид – Панель елементов**. На екрані з'явиться ще одне вікно – **Панель елементов** або, як її називали в більш ранніх версіях середовища, **Панель інструментов**. Інструменти, представлені на цій панелі, являються вбудованими компонентами середовища програмування. За допомогою цих інструментів можна включити в вікно різні об'єкти: поля введення даних, поля надписів, кнопки управління, прапорці, перемикачі та інші компоненти.

Принцип використання будь-якого інструмента простий. Достатньо захватити його мишкою і перенести на вікно. На вікні з'явиться зображення об'єкта, зазвичай в тому вигляді, в якому його буде видно користувачу. Програміст переносить ті об'єкти, які, як він вважає, потрібні для розв'язання задачі. Він розміщує їх у вікні так, щоб користувачу було зручно з ними працювати, і щоб дизайн вікна був на рівні. Перенесемо на вікно компонент **Label**. Після переносу цей стандартний компонент відобразився як об'єкт з іменем **label1**. Це ім'я ми будемо використовувати для звернення до об'єкта в програмному коді. Об'єкт зазвичай використовується для зберігання різноманітних надписів, що пояснюють хід роботи. оберемо мишкою цей об'єкт. Потім викличемо контекстне меню і оберемо **Свойства**. Справа з'явиться вікно властивостей об'єкта **label1**. Властивість **Text** уже має значення **label1**. Саме цей текст і записано в полі об'єкта на вікні. Якщо значення цієї властивості змінити на інший текст, то в полі об'єкта буде відображатися інший текст. Це означає, що об'єкт можна використати для відображення вихідних даних і результатів обчислень.

#### **Завдання для самостійної роботи.**

1. Заповніть вікно **Form1** так, щоб в ньому розмістилися поля для завдання і відображення характеристик геометричного тіла – циліндра. В програмному середовищі це вікно повинне відображатися так, як показано на рис. 3.1.

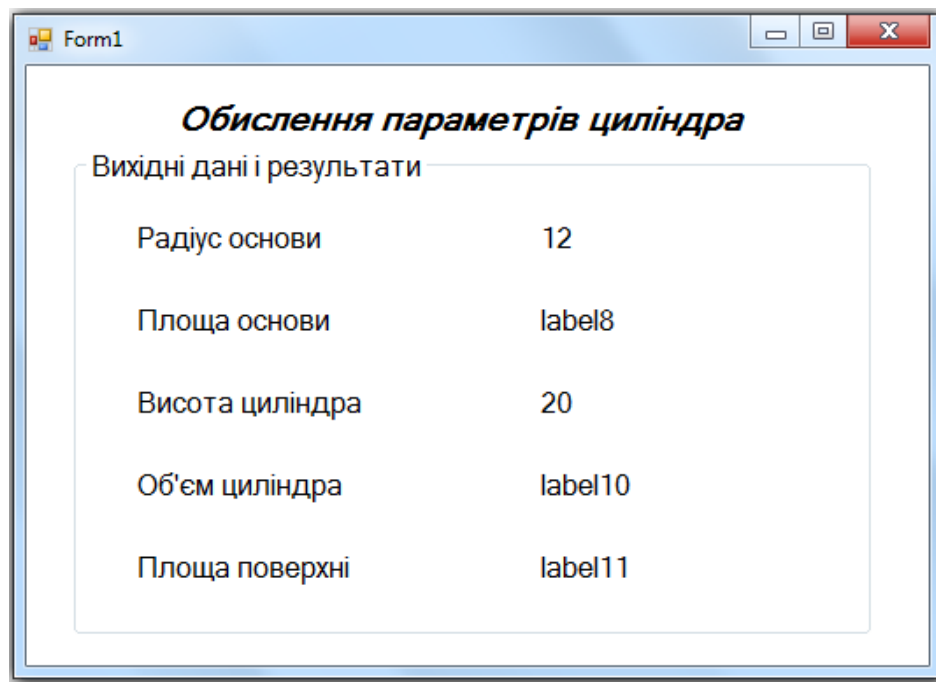


Рисунок 3.1 – Вікно задачі

Всі надписи повинні бути реалізовані як об'єкти **label**. Очевидно, що їх 11 штук. Вони мають імена от **label1** до **label11**. Що якому імені відповідають, легко встановити за рисунком:

- заголовок – це **label1**.
- заголовки зліва (радіус, площа і т.д.) – це **label2, ..., label6**.
- тексти справа – це **label7, ... label11**.

Властивість **Text** кожного об'єкта мають значення у відповідності з текстом на екрані. «**Вихідні дані і результати**» – це об'єкт **groupBox**. Він використовується для того, щоб об'єднати в один блок близькі за сенсом об'єкти заради зручності користувача.

2. Зверніть увагу, як змінився склад модуля **Form1.Designer.cs** після завершення оформлення вікна. В нього додалися секції створення і оформлення кожного об'єкта, який добавлено в вікно.

3. Проаналізуйте секцію коду, що відноситься, наприклад, до **label7**. Кожний з операторів починається зі слова **this** (цей). Це означає, що оператор відноситься до об'єкта поточного класу. Поточний клас – **Form1**. Значить, мова йде про склад класу **Form1**. Далі, через точку записано ім'я об'єкта, для якого записано оператор – це **label7**. Далі через точку записано ім'я властивості, якій привласнюється значення. Так програмним шляхом можна змінити властивість об'єкта. Таким чином, якщо властивість **AutoSize** приймає значення **true**, то розмір об'єкта буде змінюватися автоматично в залежності від довжини тексту в надписі. Далі, встановлюється шрифт і

колір. Властивість **Location** містить координати точки лівого верхнього кута об'єкта. Властивість **Name** визначає ім'я змінної, яка використовується при зверненні до об'єкта (не змінюйте її, якщо не бажаєте зайвих помилок в програмі!). Поточний розмір об'єкта визначає властивість **Size**. Властивість **Text** містить те значення, яке ми бачимо на екрані. З властивістю **TabIndex** познайомимося пізніше.

4. Запустіть програму і подивіться, як це виглядає з точки зору користувача. Зараз забезпечено тільки відображення вихідних даних (радіус і висота), які задані в ручному режимі через властивість **Text**. Результатів обчислень немає.

## ТЕМА 4 ПОДІЇ, ПОВ'ЯЗАНІ З ВІКНОМ. МЕХАНІЗМ ОБРОБКИ ПОДІЙ

### 4.1 Завантаження форми

Будемо використовувати рішення, що отримали під час вивчення попередньої теми. Зробіть копію папки з програмою (оригінал ще знадобиться під час вивчення Теми 5). Дайте їй ім'я **Задача1\_T4**. Відкрийте скопійований проект. Упевніться, що відкрилося таке ж саме вікно, як і фінальне вікно попереднього проекту.

Запустимо програму (**Ctrl-F5**). У вікні містяться тільки вихідні дані: радіус основи циліндра і його висота. А результатів немає. Це і зрозуміло: вихідні дані ми задали вручну, установивши властивість **Text** для об'єктів **label7** і **label9**. Тепер хотілося б, щоб обчислення по заданим даним були виконані програмно.

Ми запускаємо програму і хочемо бачити результати. Очевидно, що коли вікно з'являється на екрані, результати в ньому уже повинні бути. Постає питання: коли, в який момент часу повинні виконуватися розрахунки? Розв'язок може бути знайдено, якщо розуміти, як здійснюється виконання програми. Виконання програми – це процес. Під час цього процесу проходить багато подій. Наприклад, таких:

1. Операційна система (ОС) отримала замовлення на запуск програми.
2. ОС розмістила програму в оперативній пам'яті.
3. Програма отримала від ОС команду «**Працюй**».
4. Програма почала виконуватися.

5. Програма «зобразила» на екрані вікно разом з усіма об'єктами.

6. Програма чекає реакції користувача.

7. Користувач натиснув на кнопку закриття вікна, тому що невдоволений відсутністю результатів.

8. Програма завершила роботу і повідомила про це ОС. ОС вивільнила пам'ять, яку займала програма під час виконання.

Чого не вистачає серед цих подій? Очевидно, що відсутня подія «**обчислення результатів**». Коли ця подія повинна відбутися? Мабуть перед подією № 5. Адже «зображувати» вікно на екрані має сенс тільки тоді, коли вже є що зображувати! Постає питання: «куди і як» потрібно вставити в програму команди-оператори, що виконують обчислення? В нашій програмі є два модулі для програмного тексту. Один з них (**Form1.cs**) призначений для написаного програмістом програмного коду. А питання «як» легко вирішується за допомогою програмного середовища.

Виконаємо подвійний клік мишкою на вікні, але не де попало, а на вільному місці: не всередині об'єкта **GroupBox1** і не на будь-якому об'єкті **Label** – в цьому випадку система «вважатиме», що наш подвійний клік відноситься до цих об'єктів.

Наприклад, на сірому фоні трохи вище об'єкту **label1**. Після подвійного кліку відобразиться програмний текст (рис. 4.1).

```
namespace Задача1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender,
EventArgs e)
        {
        }
    }
}
```

Рисунок 4.1 – Вікно Form1.cs

Зверніть увагу: середовище саме розкрило модуль **Form1.cs** і додало в програму код:



```
private void Form1_Load(object sender, EventArgs e)
{
}
}
```

Цей код – заготовка методу під назвою **Form1\_Load()**. Чому така назва і для чого зроблена заготовка? Справа в тому, що з об'єктом **Form1** пов'язана вбудована подія, яка має ім'я **Load** («завантаження»), а це означає, що метод с іменем **Form1\_Load()** буде виконано в той момент виконання програми, коли вікно вже готове до відображення, але ще не відображене на екрані. Тобто, ті команди-оператори, які будуть записані в цьому методі (між фігурними дужками), виконуються до появи вікна на екрані.

Таким чином, ми отримали відповідь на питання «куди». Саме в цей метод потрібно записати команди-оператори, що обчислюють площу основи, об'єм і площу повної поверхні циліндра. Зверніть увагу: курсор установлено системою саме в те місце, куди потрібно записати оператори. Згадаємо формули.

Площа основи:  $S = \pi R^2$ .

Об'єм циліндра:  $V = S \cdot h$ .

Повна поверхня:  $P = 2\pi Rh + 2S$ .

Всі значення, що використовуються – дійсні числа. Оголошуємо п'ять змінних: для радіуса, висоти, площі, об'єму і поверхні:

**double R, h, S, V, P;**

Радіус і висота задані в об'єктах **label7** і **label9** через властивість **Text**, а це рядкове значення – тип **string**. Таким чином, ці значення потрібно перетворити в дійсне число. Вбудований клас **Convert** містить потрібні методи перетворення, для даного випадку метод **ToDouble()**:

**R=Convert.ToDouble(label7.Text);**

**h=Convert.ToDouble(label9.Text);**

Тепер можна записати формули. Використаємо число  $\pi$  – константу з вбудованого класу **Math**:

**S=Math.PI \* R \* R;**

**V=S \* h;**

**P=2 \* Math.PI \* R \* h + 2 \* S;**

Щоб результати відобразилися на екрані, обчисленні значення потрібно розмістити в відповідні об'єкти **Label**. Привласнимо обчислені значення властивості **Text** кожного об'єкта, скориставшись методом **Format()** з класу **string**:

```
label8.Text = string.Format("{0,10:###.###}",S);
label10.Text = string.Format("{0,10:###.###}",V);
label11.Text = string.Format("{0,10:###.###}",P);
```

Запустимо програму (рис. 4.2).

Вихідні дані і результати	
Радіус основи	15
Площа основи	706,86
Висота циліндра	30
Об'єм циліндра	21205,75
Площа поверхні	4241,15

Рисунок 4.2 – Виконання програми

#### 4.2 Обробка подій, пов'язаних з формою

Ми забезпечили розв'язання задачі, вставивши алгоритм обчислення результату в метод, який підключився точно в момент завантаження вікна. Але чому він підключився саме в цей момент? В його імені є слово **Load**, але це не означає, що із-за цього слова все і виконалося. І як система «узнала», що коли відбулася подія **Load**, цей метод потрібно виконати? Дійсно, слово **Load** тут ні до чого. Відкриємо модуль **Form1.Designer.cs**, знайдемо секцію **Form1**, а в ній оператор:

```
this.Load += new System.EventHandler(this.Form1_Load);
```

Фраза **this.Form1\_Load()** визначає ім'я методу, в який ми вставили алгоритм обчислення. Фраза **this.Load()** – це посилання на подію **Load**. Зарезервоване слово **this** означає посилання на об'єкт «**вікно Form1**» – бо слово **this**, що використовується всередині класу **Form1**. А через крапку указано ім'я події, пов'язаної з вікном. Будь-яка подія містить список, в якому зберігаються відомості про методи (будемо умовно вважати, що зберігаються імена методів).

Спеціалізована операція **+=** забезпечує запис імені методу в список подій. Таким чином, саме цей оператор записав наш метод в список подій. Інакше

кажучи, це і є оператор підключення нашого методу до події. А наш метод з повним правом можна назвати обробником подій. Він буде працювати кожного разу, коли настає подія **Load**.

І останнє питання: хто вставив даний оператор в текст секції **Form1**? Не програміст. В той момент, коли ми двічі клікнули по об'єкту **Form1**, середовище програмування зробило не одну, а дві вставки:

1) в модуль **Form1.cs** вставила заготовку обробника;

2) в модуль **Form1.Designer.cs** в секцію **Form1** було вставлено оператор підключення до події.

Ім'я методу також було автоматично сформоване середовищем програмування по імені події. Постає питання: чи завжди працює така автоматика? Чи завжди середовище формує і заготовку обробника, і оператор підключення? Ні, не завжди. Для події **Load** – так, автоматично формує і те, і інше. Цю подію можна вважати головною подією для вікна. Але є і інші події. Крім події **Load**, з об'єктом **Form1** пов'язано ще кілька подій. Наприклад:

- **FormClosing** – настає при закритті вікна;
- **FormClosed** – настає після закриття вікна;
- **Click** – настає, якщо виконується клік по вікну, тобто яке вікно обирається для роботи.

Обробку цих подій доводиться робити вручну: і заготовку обробника писати, і оператор підключення. Продемонструємо цю подію **FormClosing**.

Припустимо, що ми хочемо при закритті вікна (натискання користувачем кнопки з хрестиком) видати на екран результати обчислення в такому вигляді:

**Об'єм циліндра = ... (значення)**

**Поверхня = ... (значення)**

Потрібно написати обробник і підключити його до події. Краще за все це робити в зворотному порядку: спочатку виконати підключення, а потім описувати метод-обробник – в цьому випадку можна максимально використовувати підказку середовища програмування. Відкриємо модуль **Form1.Designer.cs**, знайдемо секцію **Form1**. В кінці цієї секції почнемо набирати: слово **this**, потім знак «.». В списку, що випаде, оберемо подію **FormClosing** (подія в списку позначена значком «блискавка»). Тепер наберемо операцію **+=** і побачимо підказку середовища. Середовище підказує, що воно рекомендує натиснути клавішу «**Tab**». Натиснемо, і запропонований текст буде включено в програму. Це і буде оператор підключення:

**this.FormClosing+=new  
System.Windows.Forms.FormClosingEventHandler (Form1\_FormClosing);**

Але це ще не все. З'являється друга підказка – і знову натискання клавіші «*Tab*». Знову натискаємо, і в текст модуля вставиться заготовка обробника. Всередині обробника уже буде один оператор, але он нам не потрібний: видалимо його. А потім перенесемо обробник в модуль **Form1.cs**. Після цього запустимо програму і упевнимся, що вона працює правильно. Але при закритті вікна нічого не відбувається – метод обробник є, він підключений, але в ньому немає жодного оператора. Запишемо всередину методу потрібний алгоритм. Нам потрібно видати результат на екран. Для цього скористаємося методом **Show()** з класу **MessageBox**:

**MessageBox.Show(string.Format( "Об'єм циліндра = {0} Площа поверхні = {1}",label10.Text, label11.Text));**

При закритті вікна на екрані з'явиться вікно повідомлення. Нас не влаштовує видача тексту в один рядок. Вставимо в шаблон форматування сполучення символів «*\n*» перед словом «**Площа поверхні**». Це сполучення забезпечує «переведення рядка» – наступний текст друкується з нового рядка. Додайте і перевірте, запустивши програму (рис. 4.3).

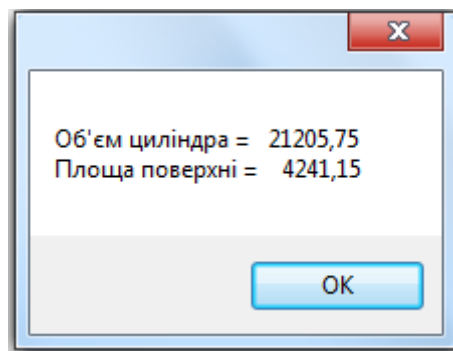


Рисунок 4.3 – Результати виконання

### ***Завдання для самостійної роботи***

1. Реалізуйте алгоритм: після закриття вікна на екран видається повідомлення «**Кінець роботи програми**».
2. Реалізуйте алгоритм: при виборі вікна програми на екран видається текст «**Ви знову повернулися в свою програму**».

## ТЕМА 5 ОБРОБКА ПОМИЛОК ВВЕДЕННЯ ЗА ДОПОМОГОЮ ВИКЛЮЧЕНЬ. МЕХАНІЗМИ ВВЕДЕННЯ І КОНТРОЛЮ ДАНИХ

Ми розробили програму для розв'язання задачі. Обчислення характеристик циліндра користувач повинен здійснювати в наступному порядку:

- використовуючи середовище програмування, візуально задати значення радіуса і висоти через властивість *Text* об'єктів *label7* і *label9*;
- запустити програму і отримати результат.

Такий порядок роботи має суттєві недоліки:

- користувач повинен уміти працювати в візуальному середовищі, щоб змінювати вихідні дані, а він не повинен це уміти – він же не програміст;
- після зміни даних за допомогою середовища програмування програму кожний раз потрібно запускати заново.

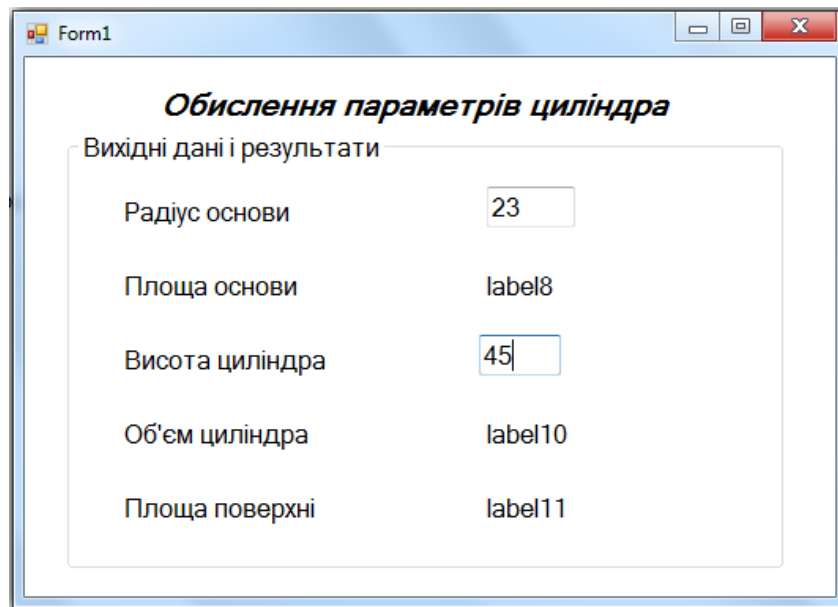
Очевидно, було б краще, якщо б кожний займався своїми справами. Програміст нехай розробляє програму, а користувач за її допомогою розв'язує свою задачу. Програміст повинен розробити програму так, щоб користувач, запустивши її один раз, міг розв'язувати свою задачу багато разів: сам задавати вихідні дані і отримувати результати. Необхідно переробити програму так, щоб користувач міг запустити її один раз, а вихідні дані зміг вводити уже під час її виконання. І не один раз, а стільки, скільки потрібно.

### ***5.1 Розв'язання проблеми введення даних***

Візьмемо за основу розв'язання задачі з теми 3. Там є заготовлене вікно, але алгоритми ще не вставлені. Скопіюємо його в доступну робочу область і перейменуємо скопійоване рішення в *Задача1\_T5*. Відкриємо проект. Перед нами поставлена задача: надати можливість користувачу набирати вихідні дані уже після запуску програми. Але в нашому проекті вихідні дані – це об'єкти *Label*. Їх значення неможна змінювати вручну під час роботи програми. Значить, необхідно підібрати для збереження вихідних даних інший об'єкт.

Відкрийте панель інструментів. Знайдіть там компонент *textBox*. Цей компонент створює об'єкт, який можна використовувати для введення значень під час виконання програми. Видаліть з вікна *Form1* об'єкти *label7* і *label9*.

Замість кожного з них установіть компонент *TextBox* – в вікні з'являться об'єкти *textBox1* і *textBox2*. Зверніть увагу: ці об'єкти відображаються не як надписи, а як поля введення. Імена об'єктів: *textBox1* і *textBox2*. Запустіть програму і спробуйте в цих полях набрати які-небудь числові значення (наприклад, радіус 10 і висота 28). Введення виконується, але ніякі розрахунки не виконуються (рис. 5.1).



Вихідні дані і результати	
Радіус основи	23
Площа основи	label8
Висота циліндра	45
Об'єм циліндра	label10
Площа поверхні	label11

Рисунок 5.1 – Розрахунки відсутні

Перед нами знову проблема: куди вставити алгоритм обчислення результатів? Минулого разу ми вставили алгоритм в метод-обробник події *Load*. Цим самим ми забезпечили обчислення безпосередньо перед відображенням вікна. Але зараз це не проходить, тому що при завантаженні вікна ще немає вихідних даних – вони з'являться після відображення вікна, коли користувач набере їх в полях введення. Це означає, обчислення повинні виконуватися зразу після того, як користувач закінчить набір значення. Таким чином, подією, яка повинна запускати алгоритм обчислення, являється завершення введення значення.

В процесі введення значення може вводитися багатозначне число. Більш того, в процесі введення можуть виправлятися помилки введення – користувач може помилково ввести не ту цифру. Значить, потрібно чітко визначити момент, коли саме введення завершено.

З об'єктом *textBox* пов'язано ряд подій. Одна з них називається «*втрата фокусу введення*». Сутність події така. Доки дані набираються користувачем, на об'єкті встановлено фокус введення, тобто об'єкт в стані введення даних. Але як тільки користувач за допомогою мишки (або клавіші «*Tab*») переведе

фокус введення на інший об'єкт, то зразу наступає подія *Leave* – втрата фокусу введення. Це і є завершення введення значення.

Спробуємо скористатися цією подією. В модулі *Form1.Designer.cs* в секції *textBox1* набираємо оператор підключення обробника:

```
this.textBox1.Leave+=new System.EventHandler(textBox1_Leave);
```

Зверніть увагу: після *this* ми вказали ще й об'єкт, який втрачає фокус. Якщо б ми цього не зробили, то подією була б втрата фокусу вікном, а не полем введення. Створену заготовку обробника переносимо в модуль *Form1.cs*. В цей обробник нам і потрібно записати алгоритм обчислення. Але не весь, а тільки обчислення площі основи – адже ми оброблюємо втрату фокусу об'єктом *textBox1*, тобто завершення введення радіусу. Алгоритм такий:

```
double r;  
r = Convert.ToDouble(textBox1.Text);  
label8.Text =  
string.Format("{0,10:####.##}", Math.PI * r * r);
```

Перевірте правильність роботи програми (рис. 5.2).

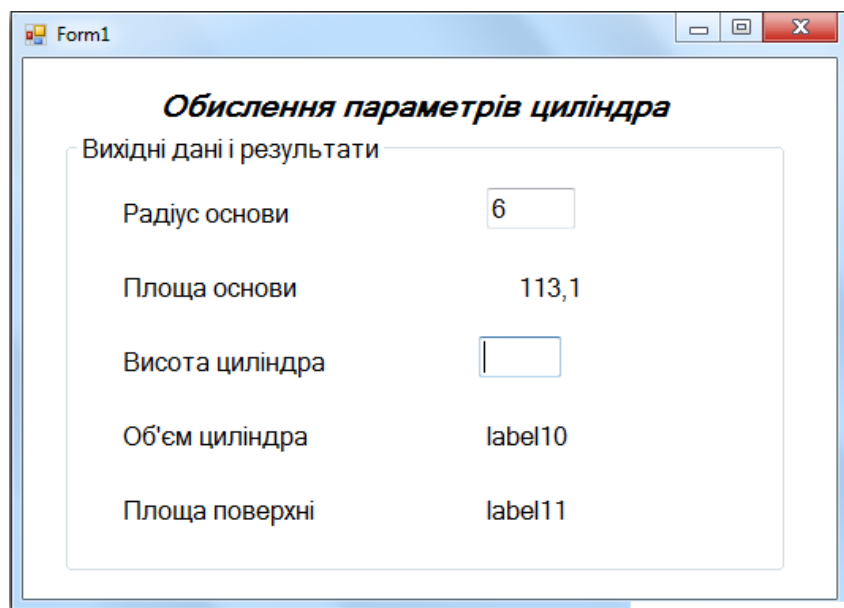


Рисунок 5.2 – Введення радіусу

### ***Завдання для самостійного виконання***

Забезпечте обчислення об'єму і поверхні циліндра, використовуючи аналогічну подію для об'єкта при введенні значення в об'єкт *textBox2*. Після виконання завдання перевірте правильність обчислень. Візьміть у якості вихідних даних для радіусу 6 і висоти значення 88 (рис. 5.3).

Обчислення параметрів циліндра	
Вихідні дані і результати	
Радіус основи	6
Площа основи	113,1
Висота циліндра	88
Об'єм циліндра	9952,57
Площа поверхні	3543,72

Рисунок 5.3 – Введення висоти

## 5.2 Доопрацювання і виправлення помилок введення

А тепер змініть значення радіуса на 100 (результат на рис. 5.4).

Обчислення параметрів циліндра	
Вихідні дані і результати	
Радіус основи	100
Площа основи	31415,93
Висота циліндра	88
Об'єм циліндра	9952,57
Площа поверхні	3543,72

Рисунок 5.4 – Змінений радіус

Але ж це зовсім не вірно! Якщо змінився радіус, то зміниться все: і площа основи, і об'єм, і поверхня. У нас же змінилася тільки площа основи. Бо при задані радіуса в методі-обробнику обчислюється тільки площа основи. Додамо в метод-обробник об'єкта *textBox1* оператори обчислення об'єму і поверхні. Для цього видалимо з першого обробника два перших оператори і додамо в нього все, що є в другому обробнику. Отримаємо:



```

double h, r;
h = Convert.ToDouble(textBox2.Text);
r = Convert.ToDouble(textBox1.Text);
label8.Text =
string.Format("{0,10:####.##}", Math.PI * r * r);
label10.Text =
string.Format("{0,10:####.##}", Math.PI * r * r * h);
label11.Text = string.Format
("{0,10:####.##}", 2*Math.PI*r*h + 2*Math.PI*r*r);

```

Запустимо програму. Наберемо значення 10 і перенесемо фокус введення. Отримаємо вікно аварійного завершення програми (рис. 5.5). Натиснемо кнопку Сведения і за допомогою движків знайдемо адресу помилки. Це 21 рядок в модулі *Form1.cs*.

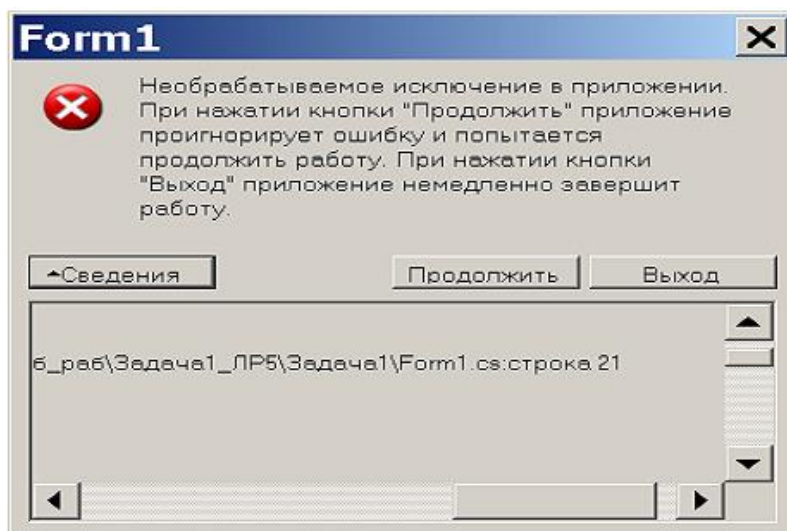


Рисунок 5.5 – Вікно аварійного завершення

Відкриємо модуль, знайдемо рядок:

```
h = Convert.ToDouble(textBox2.Text);
```

В цьому рядку виконується перетворення значення висоти з рядкового значення в чисельне. Але ж ми ще не ввели значення висоти — воно не визначено, і тому виконання оператора завершилось помилкою. Як виправити цю помилку? Спробуємо розібратися, як ця помилка виникла.

При виконанні програми система з'ясувала, що метод *ToDouble()* не може перетворити значення (воно не існує) і, як наслідок, програма не може виконуватися далі. В тому випадку, коли виконання програми неможливе, система аварійно завершує виконання програми. Кажуть, що система виробила виключення. Ситуація зовсім не безнадійна. Справа в тому, що програміст може на випадок виникнення виключення передбачити деякий алгоритм. Є механізм, що дозволяє включити в програму алгоритм обробки виключення,

який буде виконано при аварійному завершенні. Оформимо оператор перетворення висоти як окремий блок, а перед першою дужкою запишемо слово *try*. Отримаємо:

```
try  
{ h = Convert.ToDouble(textBox2.Text); }
```

Це означає, що даний оператор поставлено під контроль. Зразу після цього створимо ще один блок – блок пастки помилок:

```
catch  
{ }
```

Сенс двох блоків такий. Якщо при виконанні будь-якого оператора всередині блока *try* буде вироблене виключення, то виконання програми не перерветься – програма продовжить своє виконання операторами блока *catch*. Залишилось вирішити: що потрібно зробити, якщо буде вироблено виключення? Очевидно, що якесь значення висоти повинно бути сформовано. Наприклад, нульове. Всередині блока *catch* запишемо оператор: *h = 0*; Перевіримо роботу програми. Після набору радіуса маємо (рис. 5.6). Як видно, об'єм не обчислено, а повна поверхня дорівнює площі двох основ. Якщо набрати значення висоти, то все інше теж буде обчислено. Тепер неначе все добре. Але давайте зробимо помилку в наборі радіуса: наберемо замість цифри букву. Наприклад, так: *10a*. І ми отримаємо ту ж саму проблему, але уже з радіусом. Нецифрове значення також не може бути перетворене в число. А а значить, також виникає виключення.

Обчислення параметрів циліндра	
Вихідні дані і результати	
Радіус основи	6
Площа основи	113,1
Висота циліндра	
Об'єм циліндра	
Площа поверхні	226,19

Рисунок 5.6 – Виключення оброблено

### ***Завдання для самостійного виконання***

Забезпечте обробку виключень для радіуса. Те ж саме для радіуса і висоти необхідно зробити в другому методі-обробнику.

**Зауваження.** Після всіх доробок обидва обробники дуже схожі. Відмінність лише в тому, що в другому немає оператора обчислення площі основи. Але якщо б він і був, то все одно все працювало б правильно. Можливо варто зекономити на одному обробнику?

### ***Завдання для самостійної роботи***

1. Зробіть так, щоб обидві події (втрата фокуса будь-яким з об'єктів **textBox**) оброблював один обробник. Наприклад, **textBox1\_Leave**. Другий обробник видаліть з програми.

2. Переробіть програму. Використайте замість події **Leave** подію **TextChanged**.

**Зауваження.** Подія настає при наборі кожного символу. Це дозволить бачити розрахунки уже в процесі набору без втрати фокуса.

## **ТЕМА 6 ТИПОВІ АЛГОРИТМИ ОБРОБКИ МАСИВУ**

Створимо новий проект з іменем **Задача2\_T6**. В ньому єдине вікно наступного вигляду (рис. 6.1). У вікні – 16 об'єктів **label** і один об'єкт **textBox1** з установленим багаторядковим режимом (властивість **MultiLine=true**). В об'єктах **label** з 10 по 16 будуть відображатися значення у відповідності з написами в об'єктах. В об'єкт **textBox1** користувач буде набирати значення елементів масиву. Кнопка забезпечує запуск алгоритму обчислень. Порядок роботи користувача такий. Спочатку він вводить значення в масив, потім натискає кнопку і отримує результати одразу всіх обчислень.

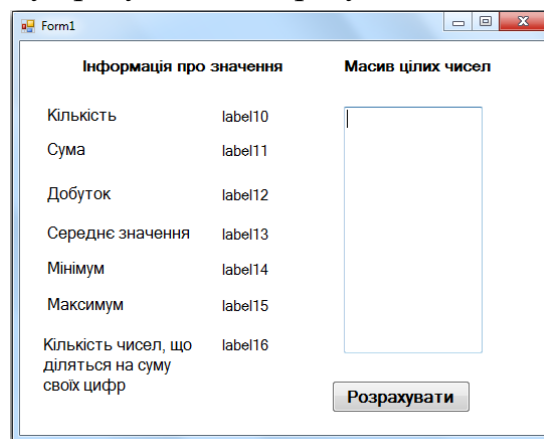


Рисунок 6.1 – Вікно задачі

При натисканні на кнопку виникає подія *Click*, обробник якої будується подвійним кліком по кнопці (заготовка добавиться в модуль *Form1.cs*):

```
private void button1_Click(object sender, EventArgs e)  
{ }
```

Підключати цей обробник до події немає необхідності – це зробить сама система. Відкрийте модуль *Form1.Designer.cs* і переконайтесь в цьому. Всі характеристики масиву (сума, середнє та інші) обчислюються за власними типовими алгоритмами роботи з масивами. Виключення – кількість чисел, які діляться на суму своїх цифр. Всі алгоритми повинні бути записані всередині обробника. До натискання кнопки дані зберігаються в об'єкті *textBox1*. Для цього об'єкт має колекцію рядків *Lines* (звичайний масив типу *string*). Перетворимо елементи цього масиву з рядкових значень в числові і збережемо їх у цілочисельному масиві, вважаючи, що користувач задає в одному рядку тільки одне число:

```
int[] m = new int[1000];
```

Тисячі елементів достатньо. Потім обчислимо кількість чисел в колекції:

```
int n = textBox1.Lines.Length;
```

Оголосимо змінну циклу – індекс поточного значення:

```
int i;
```

Оголосимо змінну, в якій буде зберігатися кількість значень в масиві:

```
int k = 0;
```

Напишемо цикл, який формує масив *m* шляхом перетворення рядкових значень з колекції:

```
for (i = 0; i < n; i++)
```

```
{ m[k] = Convert.ToInt32(textBox1.Lines[i]); k++; }
```

При додаванні значення збільшуємо лічильник чисел на одиницю. Перевіримо роботу програми. Набираємо кілька чисел. При натисканні на кнопку нічого не відбувається, якщо числа набрано вірно – цифровими символами. Але, якщо десь випадково замість цифри попала літера або один з рядків колекції виявився пустим, то програма завершується аварійно. Це означає, що генерується виключення під час виконання методу *ToInt32()*.

### ***Завдання для самостійної роботи***

1. Обробіть виключення. Якщо в рядку некоректне значення (не цифра або пустий рядок), то ніяке значення в масив не включається.
2. Запишіть алгоритм видачі на екран кількості чисел в масиві.
3. Запишіть алгоритми видачі на екран суми, добутку елементів масиву.

4. Запишіть алгоритми видачі на екран середнього арифметичного елементів масиву. мінімального та максимального елемента масиву.

5. Запишіть алгоритм підрахунку кількості чисел, які діляться націло на суму своїх цифр.

6. Доробіть програму. Виведіть на екран номери рядків, які мають помилки введення і не включені в обчислення (використайте вікно **MessageBox**).

**Вказівка.** Збережіть номери рядків в окремому масиві.

## ТЕМА 7 РОБОТА С МАСИВАМИ ВИПАДКОВИХ ЧИСЕЛ. МЕТОДИ КЛАСІВ RANDOM І MATH

### 7.1 Інтерфейс і постановка задачі

Створимо новий проєкт з іменем *Задача3\_T7*.

#### Завдання для самостійного виконання

Створіть вікно наступного вигляду (рис 7.1).

1. Два об'єкта **ListBox**:

- **listBox1** – для аргументів функції;
- **listBox2** – для значень функції.

2. Три об'єкта **textBox**:

- **textBox1** – поле для джерела повторення;
- **textBox2** – поле для мінімуму;
- **textBox3** – поле для максимуму.

3. Об'єкт **comboBox1** – для вибору функції.

4. Два об'єкта **checkBox**:

- **checkBox1** – для генерації серії чисел, що повторюється;
- **checkBox2** – для генерації цілих чисел.

5. Три об'єкта **radioButton**:

- **radioButton1** – всі цілі;
- **radioButton2** – цілі в діапазоні [0, max];
- **radioButton3** – цілі в діапазоні [min, max].

6. Об'єкт **groupBox1** – групує характеристики генератора випадкових чисел.

7. Кнопка **button1** – кнопка с написом «Генерація».

8. Об'єкт **label3** – це надпис «Функції».

9. Об'єкт **label2** – це надпис «*Джерело повторення*».

Всі інші надписи – це об'єкти **label** під довільними номерами.

Запустіть програму і уважно прочитайте те, що описано нижче.

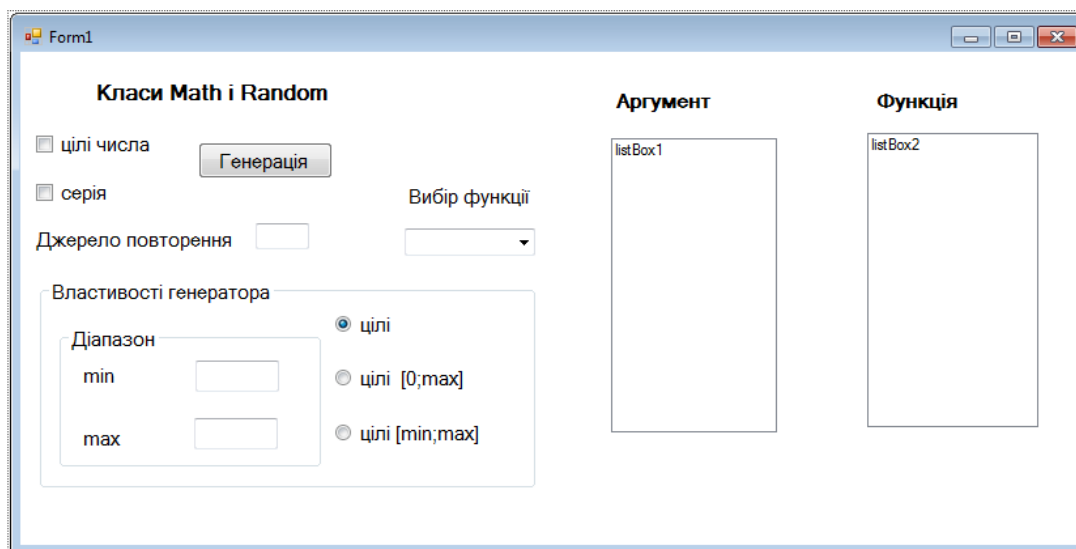


Рисунок 7.1 – Вікно задачі № 3

Пропонується реалізувати наступний сценарій.

Користувачу потрібна програма для вивчення функцій класу **Math**. Функцію він зможе обрати зі списку об'єктів **comboBox1**. Переконайтесь, що доки список пустий. При виборі функції надпис «**Функція**» повинен замінитися на ім'я обраної функції.

Аргументи для обчислення функції будуть розміщуватися в об'єкті **listBox1**, а результати обчислення – в об'єкті **listBox2**. Обчислення повинні виконуватися одразу, як тільки користувач обере функцію.

Аргументи формуються в списку об'єкта **listBox1** при натисканні кнопки «**Генерація**». Значення, що генеруються – це випадкові числа.

Які значення будуть генеруватися, залежать від налаштувань. Це можуть бути дійсні числа в інтервалі від 0 до 1 (за замовчуванням). Можна установити прапорець – тоді будуть генеруватися цілі числа. Серії чисел, що генеруються, можуть бути різними, а при установленні прапорця повторення серії – кожного разу однаковими для кожного заданого числа (джерела повторення).

В випадку цілих чисел можлива генерація:

- з усіх можливих додатних цілих чисел;
- з відрізка від 0 до максимального значення;
- з відрізка від мінімального до максимального ( в залежності від обраного перемикача).

Значення мінімуму і максимуму повинні задаватися тільки за необхідності. Бажано, щоб в кожний момент часу виконання програми в вікні відображалось тільки те, що необхідно. Наприклад, якщо не встановлено прапорець цілих чисел, то і всі настройки для цілих відображуватися не повинні. Те ж саме і з повторенням серії: якщо прапорець не встановлено, то і джерело відображатися не повинен. Закрийте запущену програму.

Переходимо до реалізації сценарію.

## **7.2 Формування вихідних даних**

За умовами задачі при запуску програми прапорець «цілі числа» не встановлено. Значить, не повинні відображатися ніякі з настройок, тобто всі об'єкти, вміщені в **groupBox1**, повинні бути невидимі. Знайдемо в властивостях об'єкта **groupBox1** властивість **Visible** і встановимо в ньому значення **false**. Запустіть програму і переконайтесь, що об'єкти настройки не відображаються.

### **Завдання для самостійного виконання**

*При запуску програми прапорець повторення серії також не встановлено. Зробіть так, щоб під час запуску програми поле джерела повторень і надпис не відображались на екрані. Перевірте запуском програми.*

При установці прапорця «цілі числа» встановлюється режим генерації цілих чисел. При цьому повинні показатися всі режими настройки, вміщені в об'єкті **groupBox1**. І навпаки, коли прапорець знято, режими настройки знову повинні стати невидимими. Зробимо це. Подвійним кліком по об'єкту **checkBox2** відкриємо обробник події **Click**:

```
private void checkBox2_CheckedChanged  
(object sender, EventArgs e) { }
```

Додамо всередині методу такий алгоритм: якщо прапорець встановлено, то зробимо групу об'єктів видимою, а якщо не встановлено, зробимо групу невидимою:

```
if (checkBox2.Checked == true)  
{ groupBox2.Visible = true; }  
else groupBox2.Visible = false;
```

Підключати обробник до події не потрібно – це зроблено автоматично. Перевірте, як це працює.

### ***Завдання для самостійного виконання***

При установці прапорця «**повторяюцяся серия**» поле джерела і надпис повинні відображатися. Навпаки, при знятті прапорця обидва об'єкти повинні стати невидимими. Запрограмуйте цю логіку і перевірте роботу.

Тепер наповнимо список об'єкта **comboBox1**.

Включимо в цей список імена тих функцій класу **Math**, які користувач зможе тестувати. Наприклад, такі:

**Геометричні:** **Atan** (арктангенс), **Cos** (косинус), **Exp** (експонента, число Е в ступені), **Log** (натуральний логарифм), **Sin** (синус), **Tan** (тангенс).

**Математичні:** **Floor** (округлення), **Ceiling** (округлення), **Pow(x,3)** (піднесення числа в третю ступінь), **Sqrt** (квадратний корінь).

Для занесення в список відкриємо властивості об'єкта **comboBox1** і знайдемо властивість **Items**. Ця властивість являється колекцією значень. Занесемо їх вручну. Перевіримо, запустивши програму.

За умовами задачі при виборі імені функції це вибране ім'я повинне замінити собою надпис «**Функція**».

### ***Завдання для самостійного виконання***

Замініть надпис «**Функція**» при виборі імені зі списку.

#### ***Вказівки***

1. Обране ім'я функції – у властивості **Text** об'єкта **comboBox**. Подія, яка настає при виборі імені – **SelectedIndexChanged**.

2. Створіть відповідний метод-обробник – для створення заготовки достатньо подвійного кліку на об'єкті **comboBox1**. Підключати до події його не потрібно. Тепер можна зайнятися кнопкою. Подвійним кліком по кнопці створимо обробник події. Алгоритм тут достатньо складний. Адже ми повинні забезпечити різні способи генерації послідовності чисел – їх повинно бути 10 штук. Нам можуть знадобитися два генератора – два об'єкта класу **Random**. Один – для випадку послідовності, що не повторюється, інший – для послідовності, що повторюється. Створимо той, який потрібний. А щоб узнати, який саме потрібний, перевіримо прапорець **checkBox1**. В залежності від того обраний він чи ні, у нас будуть дві гілки алгоритму. Перша гілка **if** відповідає послідовності, що не повторюється. Тут повинен бути створений простий об'єкт класу **Random**:



**Random G = new Random();**

Гілка *else* відповідає послідовності, що повторюється. Тут повинен бути створеним складний об'єкт, що враховує задане значення в джерелі повторення *textBox3.Text*. Але, це значення типу *string*, його потрібно перетворити в ціле число, а це можливо тільки тоді, коли це цифровий вираз! Таким чином, необхідно ще обробити виключення: якщо значення не задано або задано не цифрою, то взяти в якості джерела будь-яке число. Наприклад, нуль:

```
Random G;  
if (checkBox1.Checked == false) G = new Random();  
    else  
        { int n;  
        try { n = Convert.ToInt32(textBox1.Text); }  
        catch { n = 0; }  
        G = new Random(n);  
    }
```

Об'єкт-генератор створено. Тепер можна доставати з генератора випадкові числа, використовуючи метод *Next()* або метод *NextDouble()*. Яким методом – залежить від стану прапорця «*цілі числа*». Якщо він не обраний, то використовуємо *NextDouble()*, а якщо обрано, то *Next()*. Якщо послідовність цілочисельна, то потрібно іще врахувати, який інтервал вибору: всі числа, от нуля до максимуму або від мінімуму до максимуму. А це залежить від того, який з перемикачів (*radioButton1*, *radioButton2*, *radioButton3*) обрано. При обранні двох останніх перемикачів повинні бути обрані допустимі (цифрові) значення мінімуму і максимуму! І весь цей вибір – в циклі. Кожне отримане випадкове число потрібно розмістити в *listBox1* як окремий рядок!

Створимо цикл і розгалуження в залежності від стану прапорця:

```
for (n = 0; n < 10; n++)  
{ if (checkBox2.Checked == false)  
    { }  
    else  
        { }  
}
```

В секцію «*if*» запишемо два оператори:

```
double d = G.NextDouble(); //1  
listBox1.Items.Add(d.ToString()); //2
```

Перший оператор забезпечує отримання випадкового дійсного числа. Другий за допомогою методу *Add()* додає це число в колекцію об'єкта, з попереднім перетворення його у в рядок.

### ***Завдання для самостійного виконання***

1. Запустіть програму. Перевірте роботу без прапорця – генерація виконується. А з прапорцем не виконується – ще не написано алгоритм!

2. Перезапустіть програму. Натисніть на кнопку. Потім натисніть ще раз. Це не порядок! Повинні бути нові значення, але старі теж залишилися. Підправте програму: перед формуванням списку потрібно його очистити.

**Вказівка.** Потрібно очистити поле **Items** до початку циклу заповнення об'єкта **listBox1**.

3. Перевірте роботу при установленому прапорці повтору серії, переконайтеся, що при різних значеннях джерела повтору отримуються різні послідовності. Запустіть програму, установіть прапорець «**цілі числа**». Відобразяться характеристики генератора для випадку цілих чисел. Зверніть увагу: не обрано жоден з перемикачів. Взагалі це неправильно.

### ***Завдання для самостійного виконання***

Будемо вважати, що при виборі прапорця потрібно установити значення «обрано» в перемикач «всі цілі». Це досягається зміною властивості **Checked** даного перемикача. Внесіть потрібний оператор в програму.

Тепер можна зайнятися гілкою **else** обробника натискання на кнопку. Додамо в цю гілку наступні оператори:

```
int m;  
if (radioButton1.Checked == true)  
{ }  
if (radioButton2.Checked == true)  
{ }  
if (radioButton3.Checked == true)  
{ }
```

Змінна **m** призначена для випадкового цілого числа. Кожний блок призначено для свого перемикача: якщо він обраний, то буде обробка. З першим все просто. З генератора береться ціле число з усього діапазону цілих додатних чисел:

```
m = G.Next();
```

В другому блоці потрібно спочатку обчислити максимальне значення. Потрібно врахувати, що воно може бути з помилкою або не задане – в цьому випадку можна взяти якесь значення як максимально можливе, наприклад, 100:

```
int max;  
try { max = Convert.ToInt32(textBox3.Text); }  
    catch { max = 100; }  
m = G.Next(max);
```

### ***Завдання для самостійного виконання***

1. Доробіть третій блок. Тут метод *Next()* використовується з двома параметрами – мінімальне і максимальне значення. Не забудьте їх також перевірити на цифрове значення.

2. Запишіть зразу після третього блока оператор, що добавляє отримане ціле число в список *listBox1*.

**Зауваження.** Якщо після додавання компілятор видасть помилку «Використання локальної змінної «*m*», якій не привласнено значення», то просто замініть оператор «*int m;*» на оператор «*int m = 0;*».

### ***7.3 Обчислення значень функції***

Необхідно розробити алгоритм обчислення значень функції. Ми обрали 10 функцій, які є в списку об'єкта *comboBox1*. При виборі функції у нас уже змінюється надпис, тобто подія уже оброблюється. Залишилось в цей же обробник додати блок обчислень. Фактично це 10 різних варіантів обчислень, але потрібно ще розпізнати, який саме варіант обрано – яка функція. Для розпізнавання варіанта не будемо аналізувати ім'я обраної функції. Нам відомо, в якому порядку вони розміщені в списку. Наприклад, функція *Atan()* знаходиться в списку під індексом 0, а функція *Sqrt()* – під індексом 9. Проще всього скористатися оператором *switch*, який буде знаходитися всередині циклу, що обчислює функцію для 10 значень з *listBox1*. Значення відправляються в *listBox2*.

Додайте в обробник наступний код:

```

int i;
double d=0;
double r=0;
for (i = 0; i < 10; i++)
{ d = Convert.ToDouble(listBox1.Items[i]);
  switch (comboBox1.SelectedIndex)
  { case 0: r= Math.Atan(d); break;
    }
  listBox2.Items.Add(r.ToString());
}

```

### ***Завдання для самостійного виконання***

1. *Перевірте роботу програми для функції  $\text{Atan}()$ .*
2. *При повторному виборі функції старі значення не зникають. Внесіть виправлення.*
3. *Після генерації нових аргументів у вікні результатів залишаються старі значення. Забезпечте очистку вікна результатів при генерації нових аргументів.*
4. *Зараз дійсні аргументи і результати зберігаються з великою кількістю знаків після коми. Відформатуйте виведення значень: аргументи – для дійсних два знака після коми, результати – чотири.*
5. *Доробіть програму. Забезпечте обчислення інших функцій.*

## **ТЕМА 8 ПОРЯДОК ТА ОСОБЛИВОСТІ РОЗРОБКИ ФУНКЦІЙ**

### ***8.1 Розробка алгоритму програми***

Розробити програму знаходження коренів рівняння наступного вигляду:  
 $Ax^2 + Bx + C = 0$ . Коефіцієнти рівняння задаються користувачем і відображаються на екрані. Процес розв'язання запускається натисканням кнопки. Розв'язання відображається на екрані і містить:

- виведення про кількість дійсних коренів або повідомлення про їх відсутність;
- значення коренів (показувати тільки знайдені значення).

Варіантів розв'язання даної задачі може бути багато – в залежності від умінь і фантазії програміста. Але в будь-якому випадку потрібно так

організувати інтерфейс користувача, щоб йому було максимально зручно і комфортно працювати.

Спробуємо розібратися з умовами задачі. За формою запису – це квадратне рівняння. Але воно буде квадратним тільки в тому випадку, коли  $A$  не дорівнює нулю. В протилежному випадку воно перетворюється в звичайне лінійне:  $Bx + C = 0$ .

Алгоритм розв'язання квадратного і лінійного рівняння різні. Так як в умові задачі нічого не сказано про можливі значення коефіцієнтів, то значення  $A=0$  необхідно вважати допустимим. Значить, необхідно передбачити алгоритми розв'язання для двох видів рівняння. Для лінійного рівняння маємо наступний алгоритм:

$$x = -C / B.$$

Для квадратного рівняння – трішки складніше:

1. Спочатку обчислюється дискримінант за формулою:

$$D = B^2 - 4AC.$$

2. Далі можливі три випадки:

- якщо  $D < 0$ , то дійсних коренів немає;
- якщо  $D = 0$ , то рівняння має два однакових кореня, але прийнято говорити, що корінь один, і обчислюється він за формулою:

$$x = \frac{-B}{2A};$$

- якщо  $D > 0$ , то рівняння має два різних кореня, які обчислюються за формулами:

$$x_1 = \frac{-B + \sqrt{D}}{2A}, \quad x_2 = \frac{-B - \sqrt{D}}{2A}.$$

Тепер спробуємо уявити, як буде виглядати вікно програми. Для цього створимо проект з іменем **Задача4\_T8**.

Створимо заголовок у вигляді об'єкта **Label** з текстом «**Решение уравнения**». Відобразимо також саме рівняння в вигляді  $Ax^2 + Bx + C = 0$  (рис. 8.1). Єдиний об'єкт, який нам тут допоможе – це об'єкт **pictureBox** (картинка). Порядок дій приблизно такий. Скопіюємо формулу в буфер обміну, відкриємо **Paint**, вставимо з буфера обміну, збережемо в файл. Перенесемо з **ToolBox** компонент **pictureBox** на вікно, а потім, використовуючи властивість **Image** об'єкта **pictureBox1**, візуально додамо файл з картинкою в об'єкт.

$$Ax^2 + Bx + C = 0$$

Рисунок 8.1 – Вікно задачі

Для введення коефіцієнтів використаємо компоненти *TextBox*. Домовимося, що це будуть:

*textBox1* – поле введення *A*; *textBox2* – поле *B*; *textBox3* – поле *C*.

Для виведення розв'язку використовуємо компоненти *Label*. Також домовимося, що це будуть:

- *label11* – значення дискримінанта;
- *label5* – повідомлення про кількість коренів або їх відсутність;
- *label8* – перший корінь (або єдиний корінь);
- *label9* – другий корінь.

Буде в вікні і єдина кнопка с текстом «*Знайти корені*». Всі інші пояснюючі надписи оформимо через компонент *Label*. Приблизний вигляд (з урахуванням наведених вище домовленостей) буде таким (див. рис. 8.1). Запустіть програму. Переконайтесь, що вихідні дані набираються без проблем.

Залишилось «оживити» кнопку. Відкриваємо обробник події «натискання на кнопку». Оголошуємо три дійсних змінних для коефіцієнтів:

**double a, b, c;**

І ще три: для дискримінанта і двох коренів:

**double d, x1, x2;**

Обчислюємо значення коефіцієнтів за значеннями в об'єктах *textBox*:

```
a = Convert.ToDouble(textBox1.Text);  
b = Convert.ToDouble(textBox2.Text);  
c = Convert.ToDouble(textBox3.Text);
```

Визначаємося, яке у нас рівняння – все залежить від значення  $A$ :

```
if (a == 0) { }  
    else { }
```

В випадку *if* у нас лінійне рівняння, інакше – квадратне.

**Розв’язуємо лінійне рівняння.** Ми можемо скористатися формулою за умови, що знаменник не дорівнює нулю. Якщо же він нуль, то розв’язання немає:

```
if (b==0)  
{ label5.Text = "решения нет "; label8.Text=""; }  
    else  
  
    { x1=(-c)/b;  
      label8.Text=string.Format("{0,10:##.##}",x1);  
      label5.Text = "один корень";  
    }
```

Перевірте, як це працює. Не забудьте, що  $A = 0$ . І  $B$  також може бути нулем.

**Розв’язуємо квадратне рівняння.** Обчислюємо дискримінант:

```
d = b * b - 4 * a * c;
```

Розглянемо перший випадок:

```
if (d < 0)  
{ label11.Text = "менше нуля";  
  label5.Text = "коренів не існує";  
}
```

Перевірте, як це працює.

Розглянемо другий випадок:

```
if (d == 0)  
{ label11.Text = "0";  
  label5.Text = "один корінь ";  
  x1 = (-b) / (2 * a);  
  label8.Text = string.Format("{0,10:##.##}", x1);  
}
```

Перевірте, як це працює.

### ***Завдання для самостійного виконання***

*Самостійно реалізуйте алгоритм для третього випадку.*

Тепер виправимо помилки і недоліки. Очевидно, що все працює нормально, якщо коефіцієнти набрані коректно. А якщо якийсь не набраний або замість цифри – літера?

### ***Завдання для самостійного виконання***

*А. Забезпечте коректну роботу програми в випадку неправильного набору вихідних даних.*

**Вказівка 1.** Не потрібно перевіряти на коректність кожне значення. Забезпечте контроль всіх зразу, і якщо буде виключення, то виведіть про це повідомлення на екран «перевірте вихідні дані» з наступним завершенням методу-обробника.

**Вказівка 2.** Для завершення методу використайте оператор ***return***;

*Б. Перевірте роботу програми зразу за кількома варіантами. Переконайтесь, що при виконанні за варіантами 1 і 2, а також при розв'язанні лінійного рівняння на екрані залишаються непотрібні значення і тексти. Приберіть непотрібну інформацію з екрана в кожному варіанті, але не забудьте її поновити для другого варіанта.*

## ***8.2 Виділення функцій***

Запустіть проект. У вас повинні бути виконані абсолютно всі завдання. Відкрийте програмний код вікна. Проаналізуємо вміст методу-обробника ***button1\_Click()***. Це достатньо довгий текст. Але он правильний, забезпечує зручну роботу користувача – і в цьому сенсі (від користувача) ніяких претензій до програміста немає. А ось у програміста до самого себе є. Коли «суцільний» програмний текст достатньо великий, то його стає важко розуміти. Особливо з часом – чим більше проходить часу з моменту його написання, тим менше залишається в пам'яті. Текст не повинен бути дуже великим. Текст можна зменшити, якщо розбити його на окремі підпрограми (маленькі програми), дати цим підпрограмам імена і звертатися до них, коли потрібно, за допомогою всього однієї команди. В С# такі підпрограми, виділені програмістом, називаються функціями (або методами).

Спробуємо перебудувати текст обробника, виділивши з нього окремі фрагменти як функції. Винесемо в окрему функцію наступну групу:



```

try
{ a = Convert.ToDouble(textBox1.Text);
  b = Convert.ToDouble(textBox2.Text);
  c = Convert.ToDouble(textBox3.Text);
}
catch
{ MessageBox.Show("проверьте исходные дані"); return;
}

```

Сенс групи також зрозумілий: перетворення вихідних рядкових даних в числові значення. Назвемо цю функцію *Перетворення()*. Але, простого винесення операторів буде недостатньо. Справа в тому, що результат роботи цього блока двоякий. Може бути нормальне завершення, коли всі дані – числові. Але можливий і другий варіант – якщо виробляється виключення. Таким чином, потрібно передбачити обидва варіанти завершення функції. Як?

Наприклад, так. Нехай функція при нормальному завершенні виробляє значення «істина», а при другому варіанті – значення «неістина». В обробнику ми викликаємо цю функцію, а після її завершення перевіримо, що вона виробила (говорять, повернула). Таким чином, функція буде повертати значення типа *bool*, тобто тип функції буде уже не *void*, а *bool*. Запишемо проект нашої функції зразу перед обробником в такому вигляді:

```

private bool Перетворення()
{ try
{ a = Convert.ToDouble(textBox1.Text);
  b = Convert.ToDouble(textBox2.Text);
  c = Convert.ToDouble(textBox3.Text);
}
catch
{ MessageBox.Show("проверьте исходные дані "); return;
}
}

```

Всі винесені команди видалимо з обробника, а замість них запишемо такий оператор:

```

if (Перетворення() == false) return;

```

Сенс цього оператора такий: якщо в результаті роботи функції повернулось значення «*неістина*», то це значить, що в даних є помилка і потрібно завершити роботу обробника оператором *return* (повернення). Спробуємо запустити програму. Нажаль, отримаємо кілька помилок (рис. 8.2).

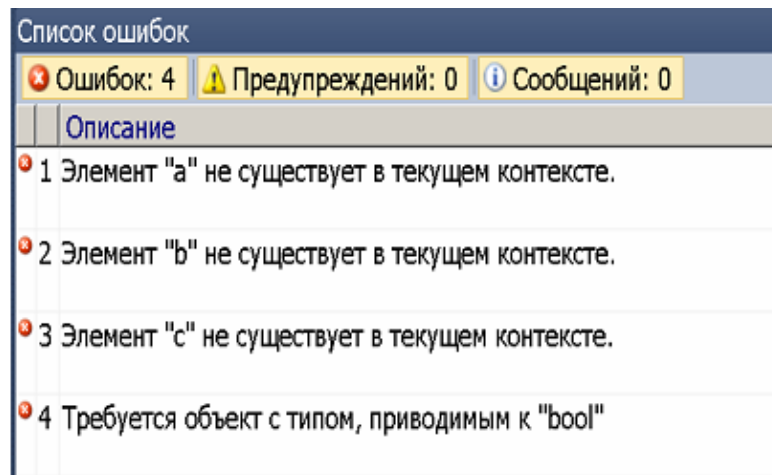


Рисунок 8.2 – Вікно помилок

Три перших помилки мають одну і ту ж природу. Текст пояснення до помилки говорить про те, що змінні не визначені всередині функції *Перетворення()*.

Насправді, ці змінні оголошені всередині обробника. Тепер ми так побудували роботу, що значення в ці змінні повинні сформуватися всередині іншої функції (*Перетворення*), і ці значення повинні бути передані з неї в обробник. Сформовані значення – це вихід функції *Перетворення()*. В цьому випадку потрібно використання вихідних параметрів. Змінимо заголовок функції і вкажемо список з трьох вихідних параметрів:

**private bool Перетворення(out double a, out double b, out double c)**

Але коли змінився заголовок, то потрібно змінити і оператор виклику:

**if (Перетворення (out a, out b, out c) == false) return;**

Тепер ми можемо бути впевненими, що якщо значення сформуються в функції *Перетворення*, то вони будуть передані в змінні *a*, *b* і *c* в обробник.

Знову спробуємо запустити програму. І знову помилка, зате яка цікава! Зверніть увагу: під підозру компілятора попала тепер уже вся функція. В повідомленні про помилку говориться, що не всі варіанти виконання функції завершуються поверненням значення. Є два варіанти. Один варіант, коли перетворення не виконано, ми завершимо оператором *return false*. А другий варіант, коли все нормально? Додамо і туди оператор повернення, але уже повернемо «істину»:

```

try
{
    a = Convert.ToDouble(textBox1.Text);
    b = Convert.ToDouble(textBox2.Text);
    c = Convert.ToDouble(textBox3.Text); return true;
}

```

Знову виконаємо компіляцію і отримаємо уже три помилки, що мають однакову природу. Вихідні параметри повинні обов'язково отримувати якісь значення. А якщо буде вироблено виключення? Компілятор це «побачив». Давайте привласнимо змінним якісь значення. Наприклад, нульові, в самому початку функції:

```

private bool Перетворення(out double a, out double b, out double c)
{
    a = 0; b = 0; c = 0;
    try

```

Виправимо і знову запусимо програму. Тепер все в порядку! Наступний великий фрагмент тексту (при  $a = 0$ ):

```

if (b==0)
{
    label5.Text = "решения нет "; label8.Text="";
}
else
{
    x1 = (-c) / b;
    label8.Text=string.Format("{0,10:##.##}",x1);
    label5.Text = "один корень ";
}
label9.Visible = false; label6.Visible = false;
label11.Visible = false;
label10.Visible = false; label7.Visible = false;

```

Проаналізуємо, що тут робиться?

**Обчислюється  $x1$ .** Це означає, це буде вихідний параметр. Він обчислюється на основі  $b$  і  $c$ . Це означає, що ці значення повинні бути передані в нову функцію. Таким чином, буде три параметра: два звичайних (значення) і один вихідний (*out*). Повертати функція нічого не буде, тобто тип функції *void*. Назвемо функцію *Линейное()*:

```

private void Линейное(double b, double c, out double x1)
{ if (b==0)
  { label5.Text = "решения нет "; label8.Text=""; }
  else
  { x1 = (-c) / b;
    label8.Text=string.Format("{0,10:##.##}",x1);
    label5.Text = "один корень ";
  }
  label9.Visible = false; label6.Visible = false;
  label11.Visible = false; label10.Visible = false;
  label7.Visible =false ;
}

```

Замість винесеного тексту – один оператор:  
**Линейное (b, c, out x1);**

Запускаємо програму. Знову працює! А тепер винесемо в окрему функцію всю логіку розв'язання квадратного рівняння, тобто весь текст з гілки *else*. Очевидно, що в даному алгоритмі використовуються для обчислення всі коефіцієнти. Це будуть параметри-значення. Далі, формуються результати *x1* і *x2* – це будуть вихідні параметри. Функція все обчислює і «розкидає» по об'єктах *label*, тобто їй просто нічого повертати – значить, тип *void*. Назвемо її – *Квадратное()*:

```

private void Квадратное(double a, double b, double c, out double x1,
out double x2)

```

Відповідно, виклик функції:

```

Квадратное(a, b, c, out x1, out x2);

```

Запускаємо! Вісім помилок! Перші сім помилок пов'язані зі змінною *d*. Правильно, вона ж в нові функції не оголошена. Оголошуємо в самому початку функції:

```

double d;

```

А восьма помилка – це попередження. Клікнемо по тексту помилки двічі і побачимо, що компілятор «придрався» до оголошення цієї змінної в обробнику. Насправді змінна оголошена, але ніде не використовується. Видаємо її.

Запускаємо! І знову бачимо: не за всіма гілками алгоритму вихідні змінні отримують значення (а повинні). Надамо їм в самому початку нульові значення (фрагмент):

```
double d;  
x1 = 0; x2 = 0;  
d = b * b - 4 * a * c;
```

І знову запускаємо програму. Тепер все. Зверніть увагу на метод-обробник: він став зовсім короткий. І зрозумілий. Правда, функцій стало більше, але знову ж кожна – невелика і зрозуміла.

### ***Завдання для самостійної роботи***

*Виділіть в окремі функції фрагменти функції **Квадратное()**:*

- 1) оператор обчислення дискримінанта;*
- 2) фрагмент операторів всередині фігурних дужок після перевірки  $d < 0$ ;*
- 3) аналогічно після перевірки  $d == 0$ ;*
- 4) аналогічно після перевірки  $d > 0$ .*