МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
«ІНСТИТУТ ПРОМИСЛОВИХ ТА БІЗНЕС ТЕХНОЛОГІЙ»
УКРАЇНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ НАУКИ І ТЕХНОЛОГІЙ

Л.І. ЛОЗОВСЬКА, Л.М. БАНДОРІНА,
Р.В. САВЧУК, Т.О. КЛИМКОВИЧ

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Дніпро 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
«ІНСТИТУТ ПРОМИСЛОВИХ ТА БІЗНЕС ТЕХНОЛОГІЙ»
УКРАЇНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ НАУКИ І ТЕХНОЛОГІЙ

Л.І. ЛОЗОВСЬКА, Л.М. БАНДОРІНА,
Р.В. САВЧУК, Т.О.КЛИМКОВИЧ

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Друкується за Планом видань навчальної та методичної літератури,
затвердженим Вченою радою УДУНТ
Протокол № 1 від 24.01.2022

Дніпро 2022

УДК 681.518
Лозовська Л.І., Бандоріна Л.М., Савчук Р.В., Климкович Т.О. Візуальне програмування : навч. посібник. Дніпро : УДУНТ, 2022. 147 с.

Містить методичні матеріали для вивчення основ програмування мовою високого рівня C#. Одночасне ознайомлення з можливостями системи програмування і конструкцій мови програмування дозволяє студентам з перших занять одержати уявлення про технологію розробки Windows-додатків.

Призначений для студентів спеціальності 051 – економіка, 126 – Інформаційні системи та технології.

Друкується за авторською редакцією.

Відповідальна за випуск Л. М. Бандоріна, канд. екон. наук, доц.

Рецензенти: Г.В. Корніч, д-р фіз.-мат. наук, проф., зав. каф. системного аналізу та обчислювальної математики Запорізького національного технічного університету;
В.А. Турчина, канд. фіз.-мат. наук, доц., зав. каф. обчислювальної математики та математичної кібернетики Дніпропетровського національного університету імені Олеся Гончара.

© Український державний університет науки і технологій, 2022
© Лозовська Л.І., Бандоріна Л.М., Савчук Р.В., Климкович Т.О., 2022

ЗМІСТ

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 розроблюються відповідні методи об’єктів.

В даному навчальному 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 вміння для проєктування 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.

Матеріал посібника може бути повністю або частково використано під час виконання лабораторних робіт.

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 ***«Программы»***. Порядок дій: ***Пуск – Все программы – Microsoft Visual Studio***.

Початок роботи

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

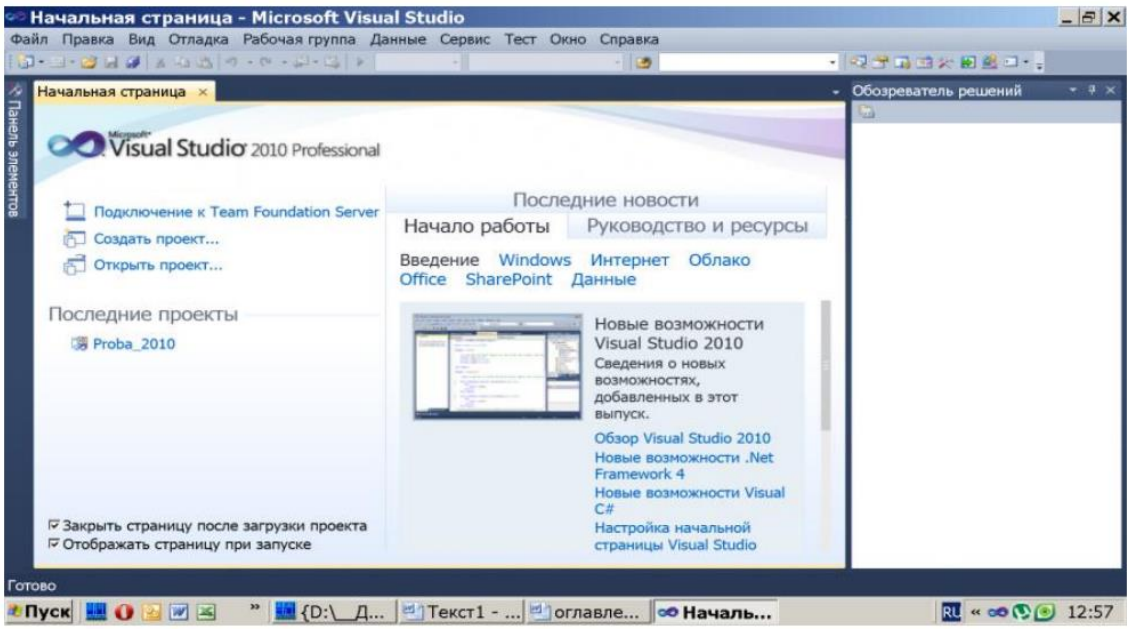

Рисунок 1.1 – Стартове вікно середовища

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 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 проєкту

Кожний додаток розробляється в середовищі ***Visual Studio*** як проєкт. В одному додатку може бути кілька проєктів. Створимо проєкт, 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 появиться вікно. Ліва секція в цьому вікні (рис. 1.2) містить список мов програмування, для яких можна будувати проєкти. Обираємо мову C#, зазначаючи при цьому, що розробка буде проводитися для роботи під Windows.

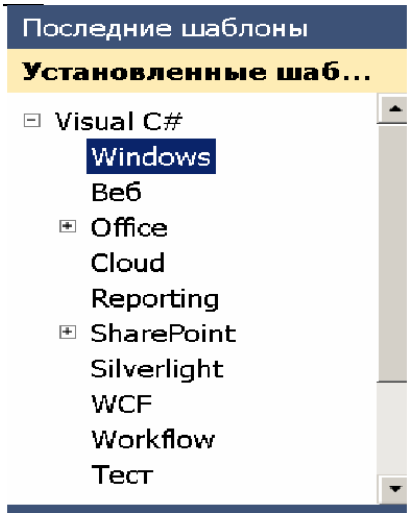

Рисунок 1.2 – Список мов

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

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 проєкт для створення служб Windows.

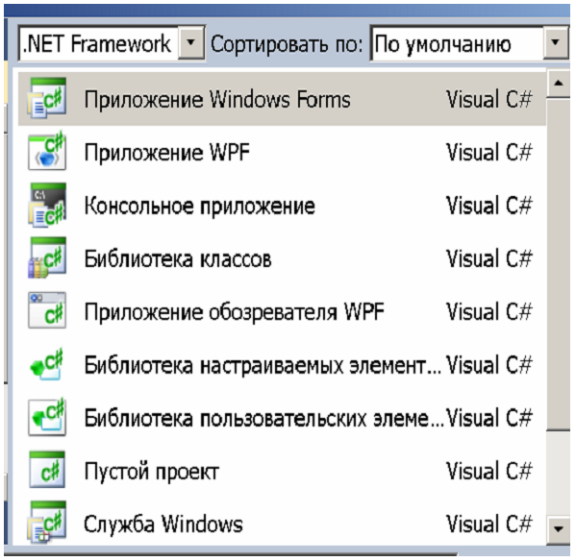

Рисунок 1.3 – Призначення додатка

Ім’я та місце розташування додатка

В нижній секції вікна є три поля для введення даних:

Имя. Це поле імені проєкту. Щоб задати ім’я проєкту, можна використовувати літери (кирилицю і латиницю) і цифри.

Имя решения. Це ім’я додатка, який створюється. Якщо в додатку один проєкт, то імена співпадають.

Расположение. В цьому полі вказується місце розміщення додатка. Вказується 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 проєкту ***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 ***«OK»***. Програмне середовище відкриє вікно с заготовкою консольного додатка. Тепер у вікні ***Обозреватель решений*** видно склад нашого проєкту. Майстер проєкту розмістив в них такі компоненти, яких достатньо для організації консольного додатка. Кожне рішення в C# може містити кілька проєктів. В нашому випадку – це один проєкт 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 C#. 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 с 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 C#. 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. Частину 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# використовується 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 проєкту (***консольний додаток***) підключило саме ці простори імен.

Перейдемо до розгляду другого блока. Він називається так, як і наш проєкт – ***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 підчас 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 відображуватися 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 типа. 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 типа. 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 вже 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 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 ‘c’ 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. Замінимо ***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 вводить 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 замінимо 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 завершені 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 теку ***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 проєкт, збережіть його і продемонструйте роботу з ним.

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).

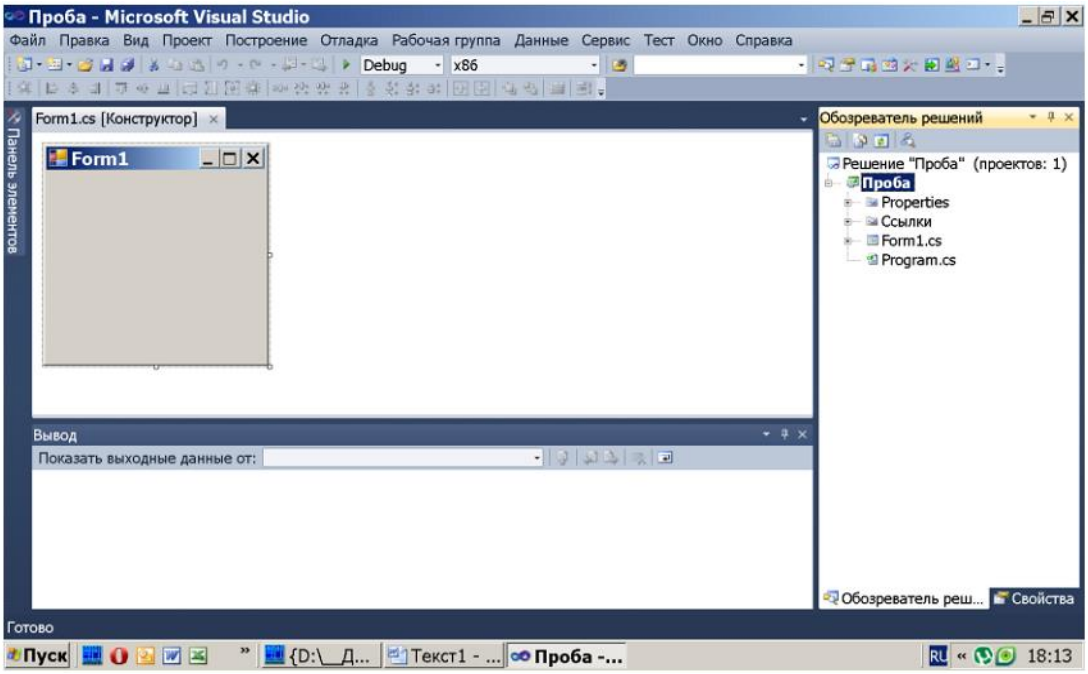

Рисунок 2.1 – Заготовка Windows-додатка

У вікні ***Обозреватель решений*** видно склад нашого проєкту. Він не пустий – в папках є компоненти, необхідні для організації ***Windows***-додатка. Кожне рішення в C# може містити кілька проєктів. Зараз існує один проєкт під назвою ***Проба,*** у складі якого є два програмних файла – ***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 вже 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 й 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'єкта вікно ***Form1***.

Властивість ***Text*** містить заголовок вікна. Її можна змінити, це призведе 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 змінені значення властивості, а щось – тільки після запуску програми. Спробуйте змінити значення деяких властивостей, запустивши програму (**Ctrl-F5**), подивіться, як різні значення властивостей проявляються при виконані програми.*

Дослідіть:

1. ***AutoSizeMode*** *(переконайтеся, що при одному зі значень вікно неможливо розтягти).*

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).*

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. Запитання: 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 іще 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 проєкту ***Задача1***;
- вказали, що проєкт повинен бути ***Windows-додатком***;
- вказали, що місцем його розташування буде папка ***USERS***.

Після цього отримали проєкт з одним вікном. Вікно, правда, доки є пустим. Але в склад нашої програми середовище програмування вже 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 створені 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 встановити 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 оголошені 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 элементов***. На екрані з'явиться ще одне вікно – ***Панель элементов*** або, як її називали в більш ранніх версіях середовища, ***Панель инструментов***. 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 рисунку 3.1.*

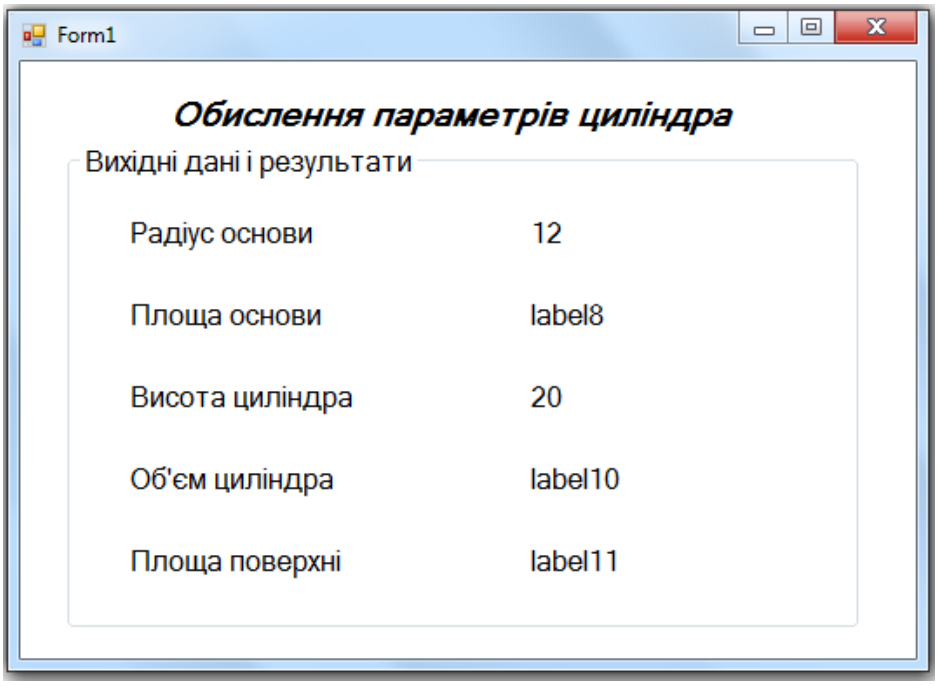

Рисунок 3.1 – Вікно задачі

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 додались 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.*

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_Т4.*** Відкрийте скопійований проєкт. Упевніться, що відкрилося таке ж саме вікно, як і фінальне вікно попереднього проєкту.

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 вже повинні бути. Постає запитання: 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 запитання: «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 запитання «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'єкта ***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), виконаються до появи вікна на екрані.

Таким чином, ми отримали відповідь на запитання «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 π – 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 у 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).

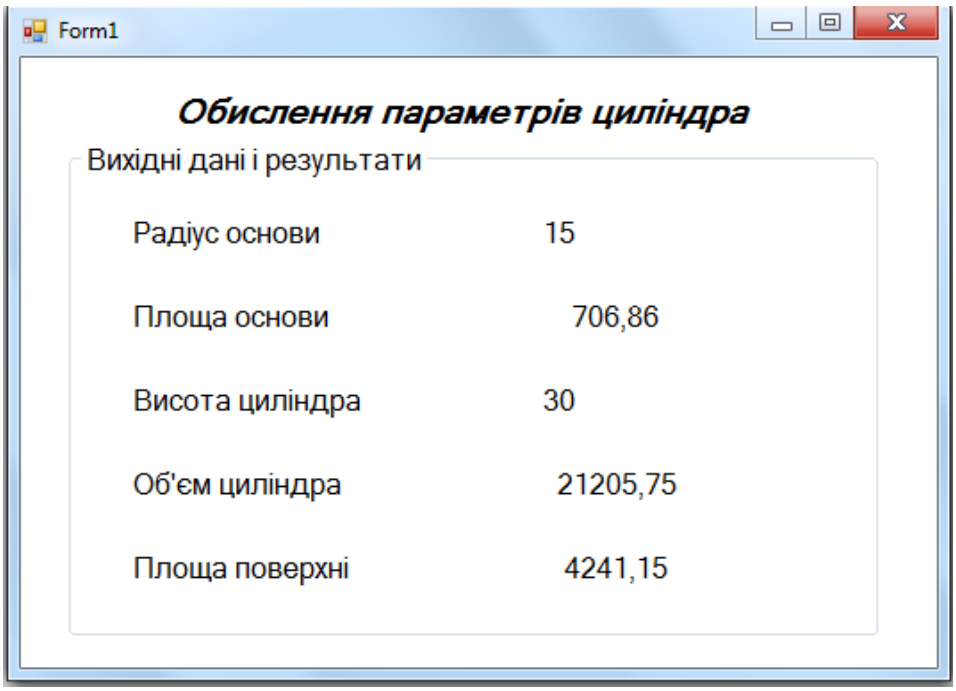

Рисунок 4.2 – Виконання програми

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 вказано 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 наступає подія ***Load***.

І останнє запитання: 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 запитання: 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 й інші події. Крім події ***Load***, з об'єктом ***Form1*** пов'язано ще кілька подій. Наприклад:

- ***FormClosing*** – наступає при закритті вікна;
- ***FormClosed*** – наступає після закриття вікна;
- ***Click*** – наступає, 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 вже буде один оператор, але він нам не потрібний: видалимо його. А потім перенесемо обробник в модуль ***Form1.cs***. Після цього запустимо програму і упевнимося, 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).

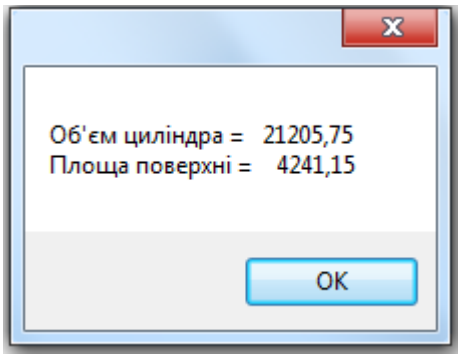

Рисунок 4.3 – Результати виконання

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»****.*