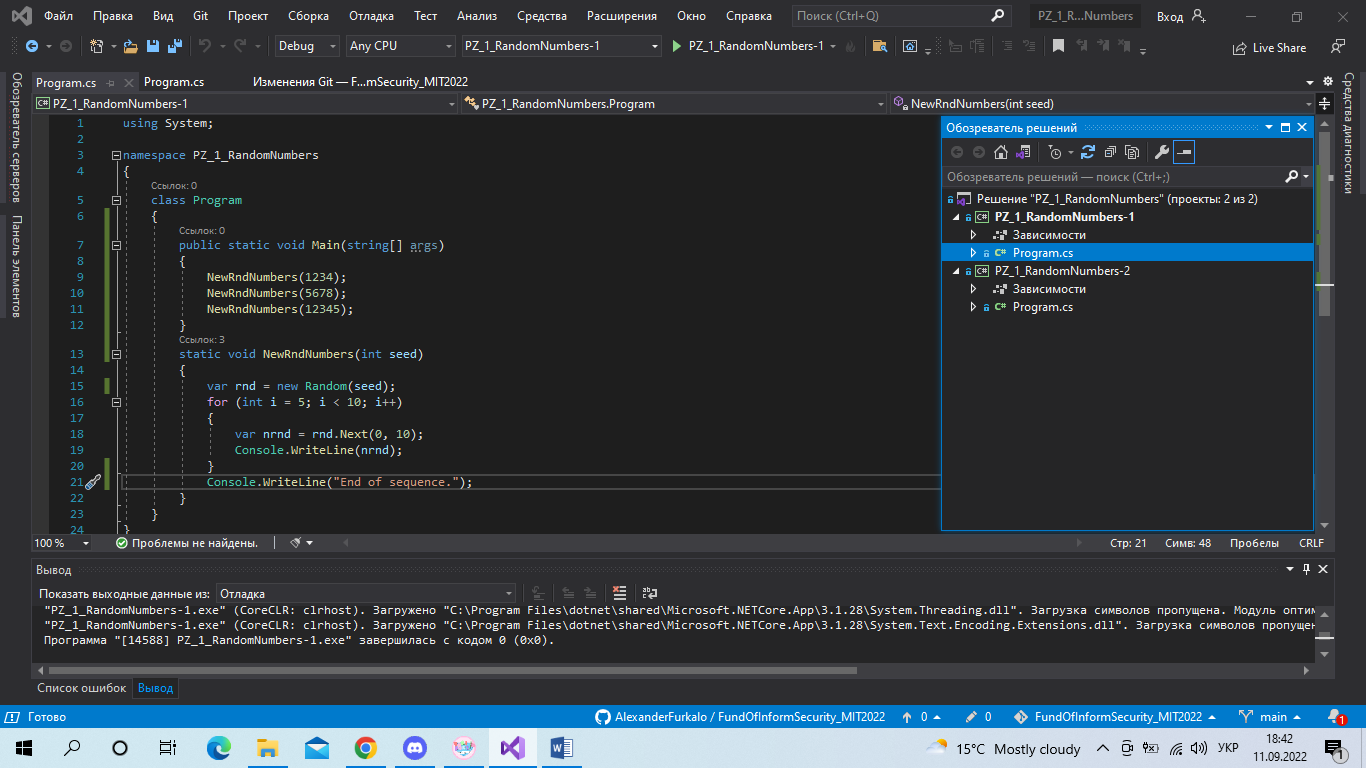
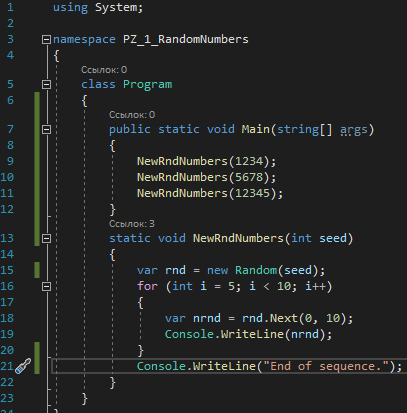
ПРАКТИЧНА РОБОТА №1  
З ОСНОВ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ  
СТУДЕНТА КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКО  
ФАКУЛЬТЕТУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
КАФЕДРИ МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ  
ДРУГОГО КУРСУ, ДРУГОЇ ПІДГРУПИ  
ОЛЕКСАНДРА ОЛЕКСАНДРОВИЧА ФУРКАЛА  
ЗА ТЕМОЮ “ГЕНЕРАЦІЯ ПОСЛІДОВНОСТІ ВИПАДКОВИХ ЧИСЕЛ”  
ЗВІТ  
11.09.2022

Завдання цієї практичної роботи були такі:

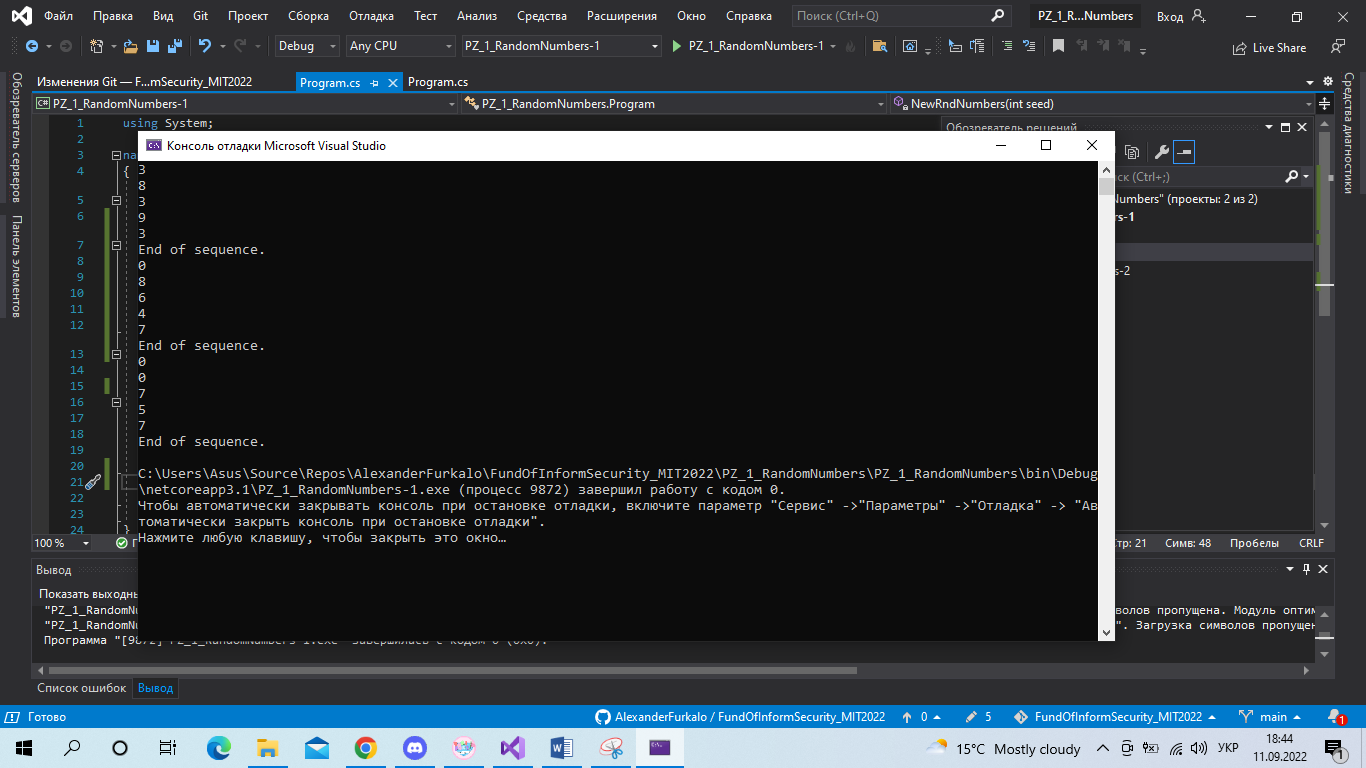
- Опрацювати розділ 3 з джерела Haunts Stephen. Applied Cryptography in .NET and Azure Key Vault (сторінки 24-30);  
- Створити рішення (Visual Studio Solution), у якому містяться проекти, що реалізують наступні задачі:  
- Написати програму, яка забезпечує генерацію та виведення на екран послідовності псевдовипадкових чисел. Порівняти елементи послідовності для однакових та різних початкових значень. Зробити висновки;  
- Написати програму, яка забезпечує генерацію та виведення на екран криптографічно стійкої послідовності випадкових чисел. Порівняти елементи послідовності для декількох послідовних спроб. Зробити висновки;  
- Оформити звіт.

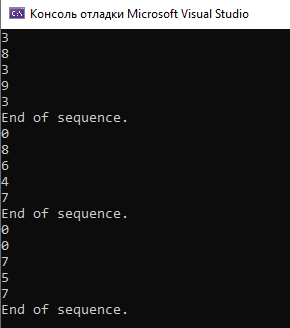
Перейдемо до першого (з двох) кодів:



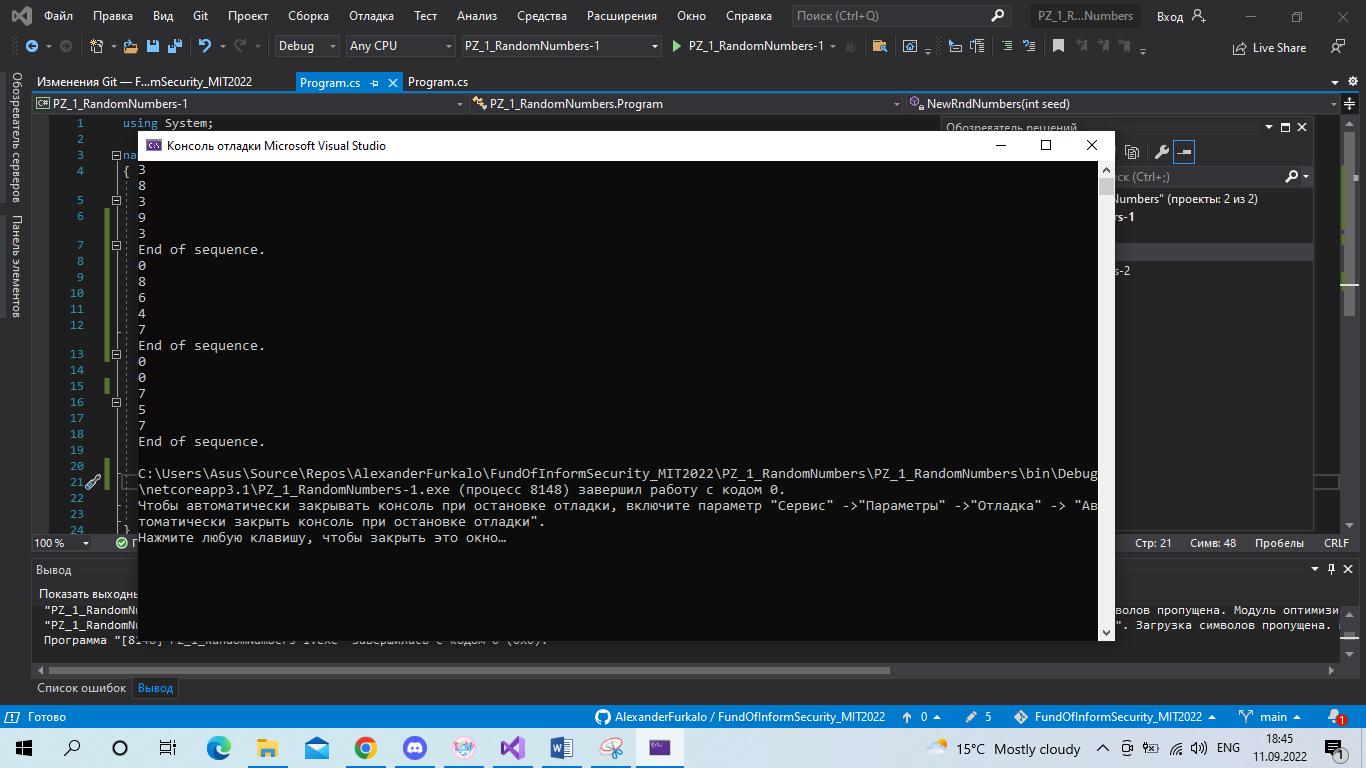


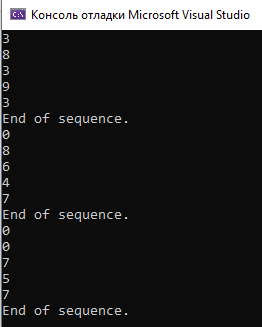
Результати роботи програми:



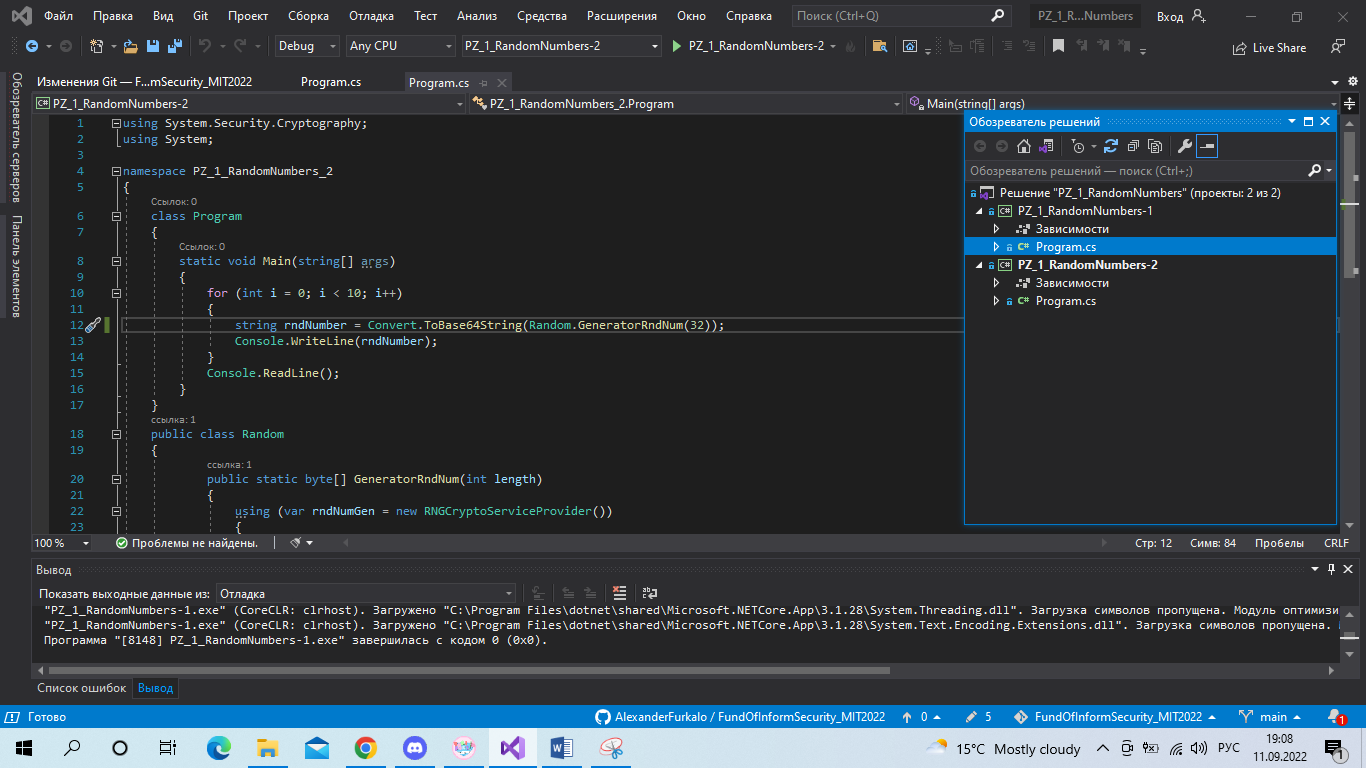
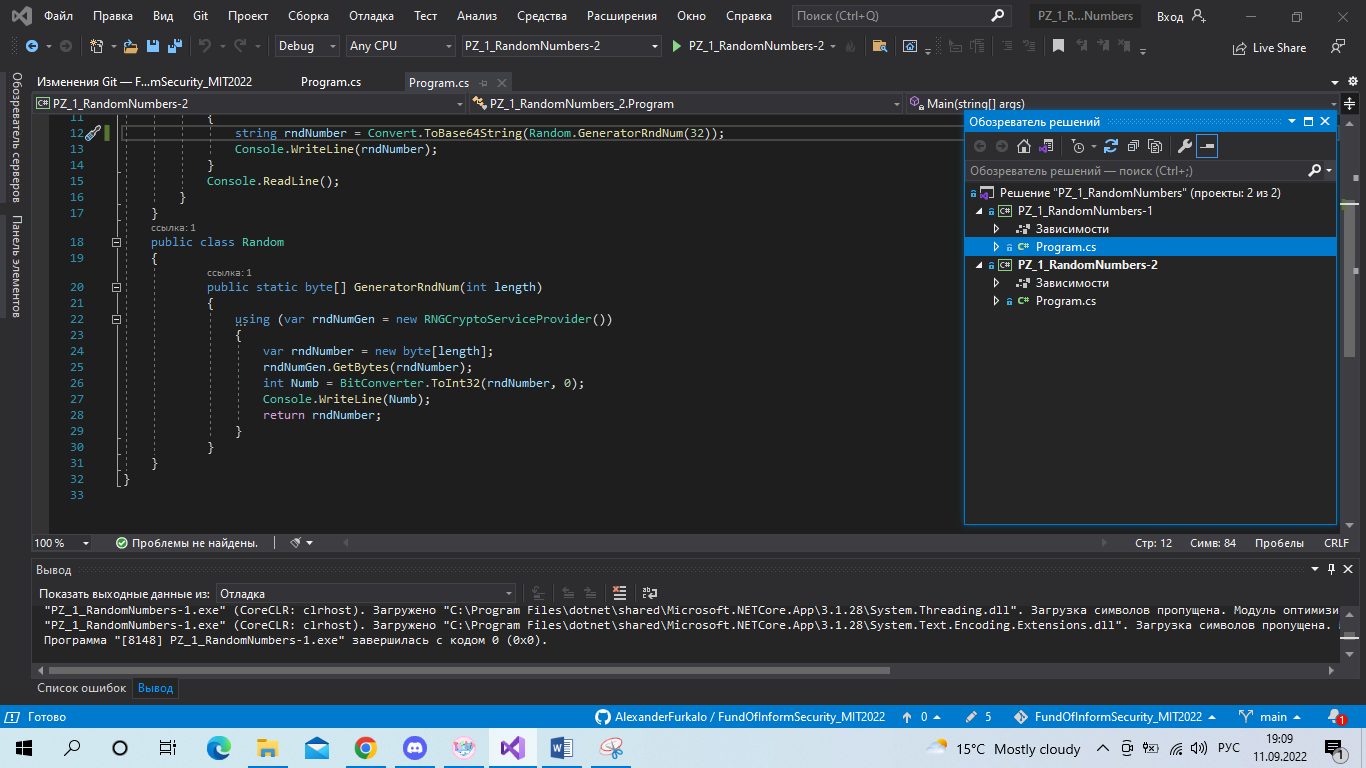
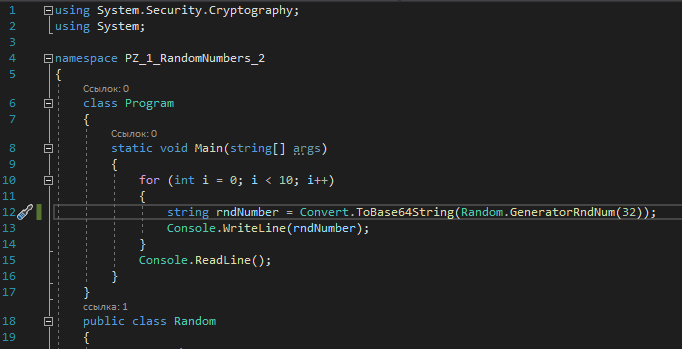
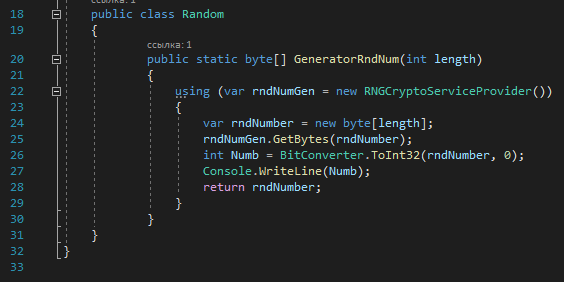


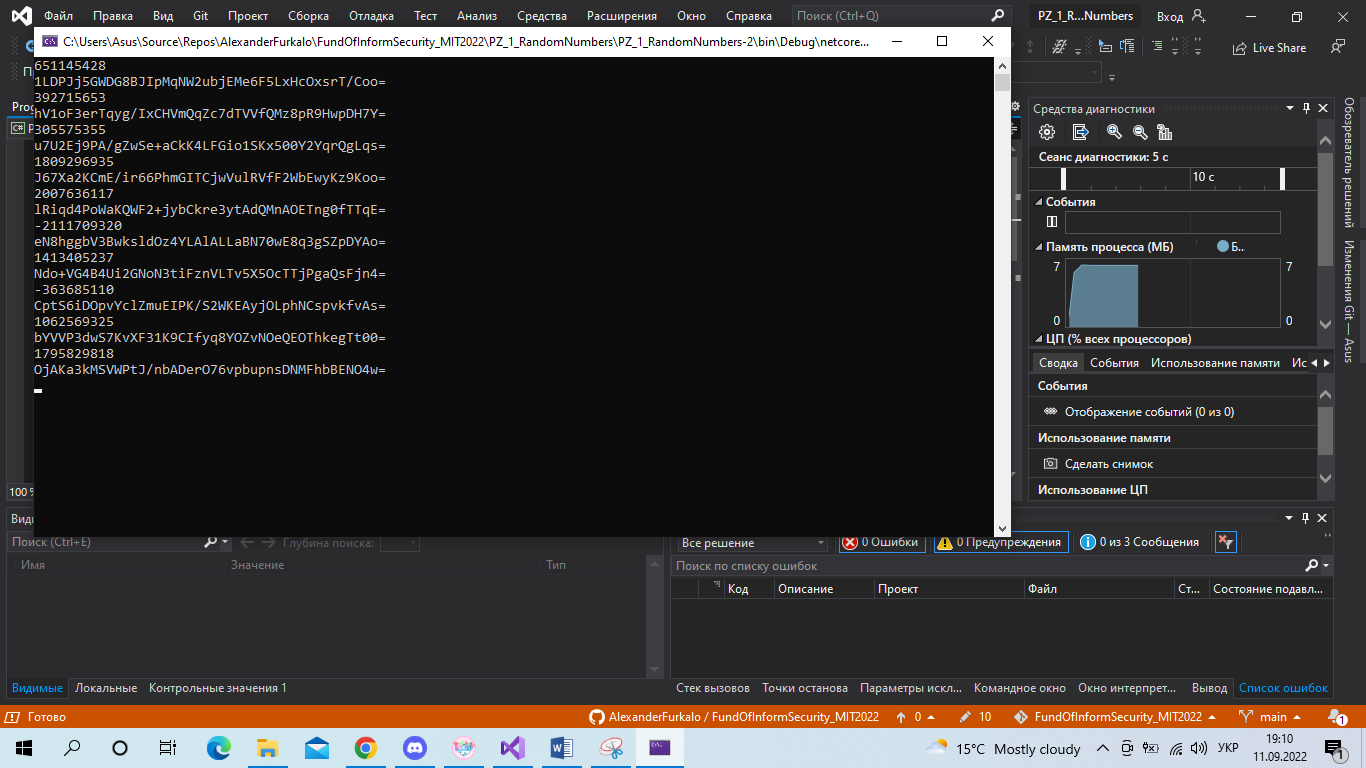
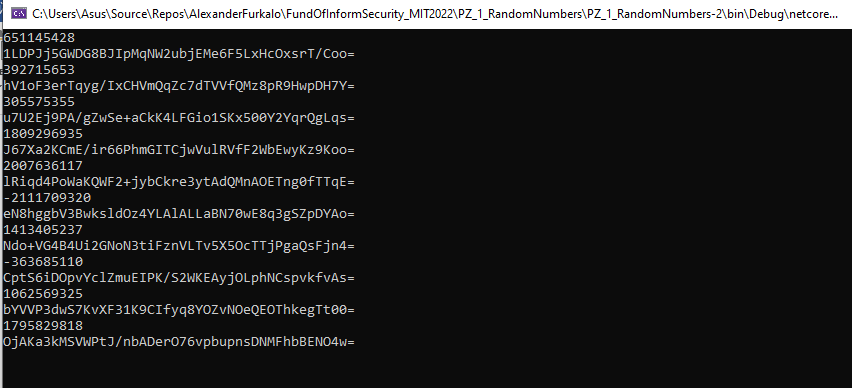
Якщо ми запустимо той же код повторно:

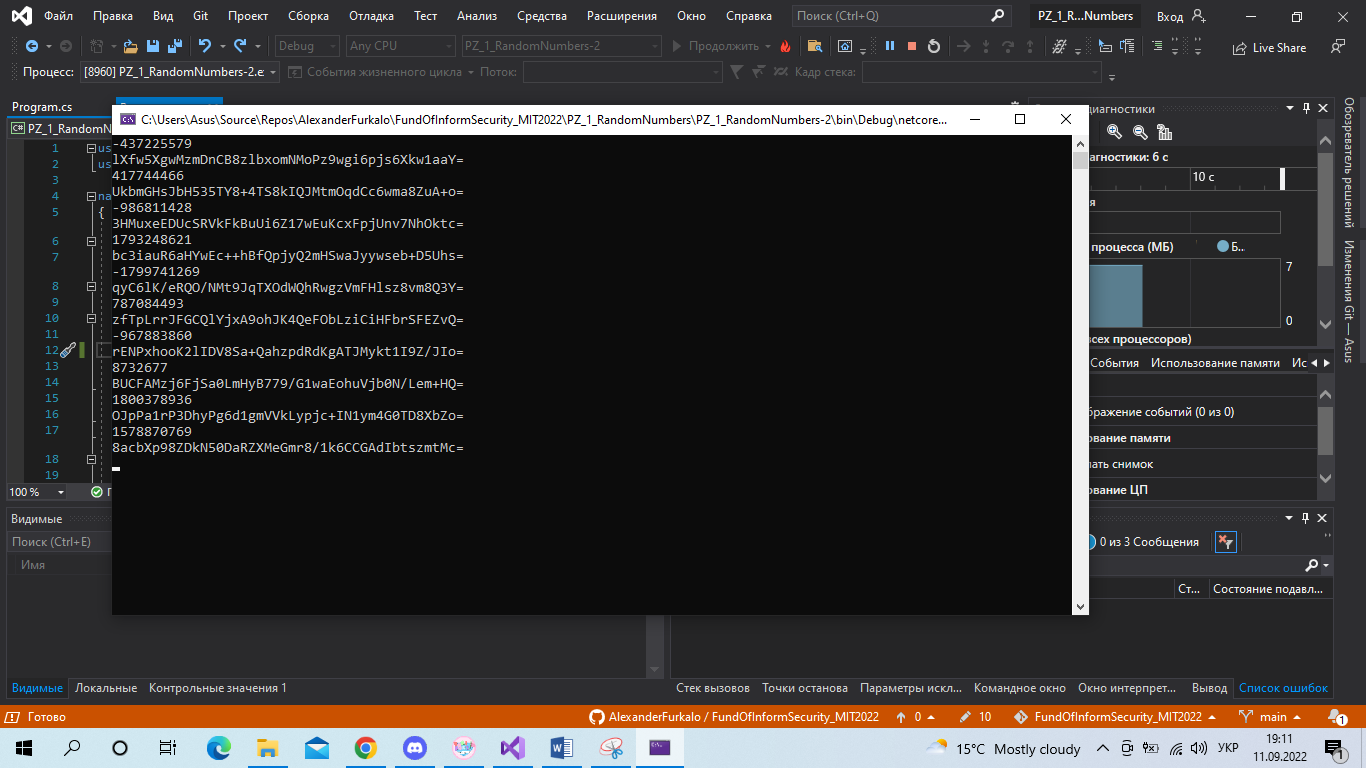
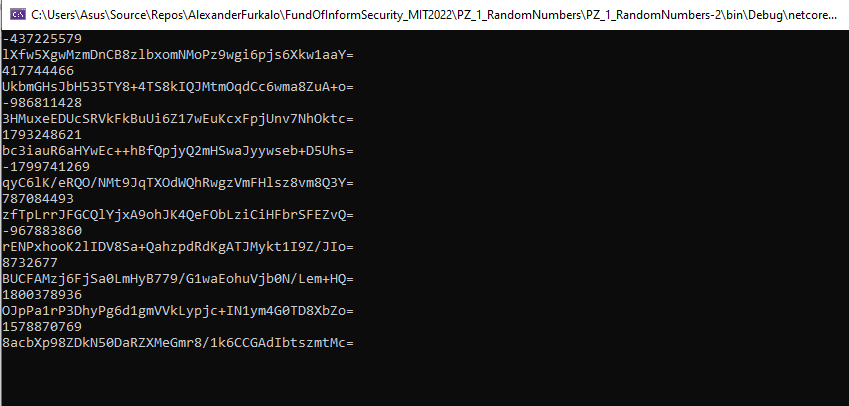




Як бачимо, значення не змінились.  
Наші випадкові числа генеруються за допомогою команди Random(int Seed), де Seed – це число, яке використовується для обчислення початкового значення послідовності псевдовипадкових чисел. Тобто наша послідовність залежить від обраного нами числа і не змінюватиметься, незалежно від того, скільки разів ми запустимо програму.  
Інша проблема – в System.Random, який не є потокобезпечним (Код потокобезпечний, якщо він функціонує справно при використанні його з кількох потоків одночасно). Викликаючи його, потрібно використовувати блокування потоку.  
Через деяку передбачуваність набору чисел такий метод не підходить для вирішення криптографічних завдань, де потрібні сильніші методи захисту.  
Однак такий код буде корисний для будь-яких нескладних завдань як симуляція кидка кубика або як псевдовипадковий набір даних для перевірки чогось, тобто, де нам не потрібно замислюватися над засекречуванням даного набору цифр.  
  
В цьому коді також використовується:  
- for (int i = 5; i < 10; i++) - для генерації саме 5 псевдовипадкових чисел (параметр починається з 5, потім збільшується на 1 - цифри будуть виводитися до тих пір, поки значення не буде рівним десяти)  
- var nrnd = rnd.Next(0, 10) - псевдовипадкові числа будуть знаходитися в діапазоні від 0 до 10

Перейдемо до наступного коду:  
  
  
  


Результати виконання:  
  


Якщо спробуємо запустити ще раз:  
  


Як бачимо, набір цифр змінюється після повторного запуску програми. Це можливо завдяки RNGCryptoServiceProvider – цей клас реалізує криптографічний генератор випадкових чисел, використовуючи реалізацію, яку надає постачальник служб шифрування (CSP). RNGCryptoServiceProvider внутрішньо використовує реалізацію під назвою CryptGenRandom і функція під назвою RtlGenRandom для генерації випадкових чисел. Такий набор вже можна використовувати як ключі шифрування.  
RNGCryptoServiceProvider набагато повільніше за System.Random, проте цей недолік у достатку компенсується набагато вищою захищеністю.  
Використовується:  
- RNGCryptoServiceProvider.GetBytes - заповнює масив байтів криптостійкої послідовності випадкових значень.  
- BitConverter - перетворює базові типи даних на масив байтів, а масив байтів — на базові типи даних (у коді потрібен для того, щоб користувач бачив незакодовані числа - прибравши цей ряд, залишиться лише криптографично стійкий висновок)  
- Convert.ToBase64String(Random.GeneratorRndNum(32)) - дана команда використовується в коді для того, щоб перетворити отримані випадкові числа на зашифрований вигляд. Convert перетворює значення базового типу даних до іншого базового типу даних, Convert.ToBase64String - перетворює значення масиву 8-розрядних цілих чисел без знаку на еквівалентне рядкове уявлення, що складається з цифр у кодуванні Base64. Підставивши інше значення замість 32 у дужках, ми збільшимо кількість символів у криптографічно стійкому рядку.

Висновки

Під час виконання цієї лабораторної роботи ми вивчили пару підходів до виконання завдання з генерації послідовності випадкових чисел. Спочатку, ми написали програму, яка забезпечує генерацію та виведення на екран послідовності псевдовипадкових чисел за допомогою System.Random - цей метод простий у виконанні і швидкий у роботі, дозволяючи нам отримати певний набір псевдовипадкових чисел, проте передбачуваність у генерації та відсутність потокобезпеки робить його непридатним для криптографічних завдань, однак все ж має застосування в, наприклад, емуляції лотереї або кидання гральних кубиків. У наступному завданні ми написали програму, яка забезпечує генерацію та виведення на екран криптографічно стійкої послідовності випадкових чисел, використовуючи RNGCryptoServiceProvider - він повільніший за попередній варіант, але ряд чисел більш випадковий, не залежить від одного параметра, не повторюється в повторних генераціях і криптографічно безпечний, тому це найкращий варіант для криптографічних завдань.