***Практичне заняття №5***

*«Безпечне зберігання паролів»*

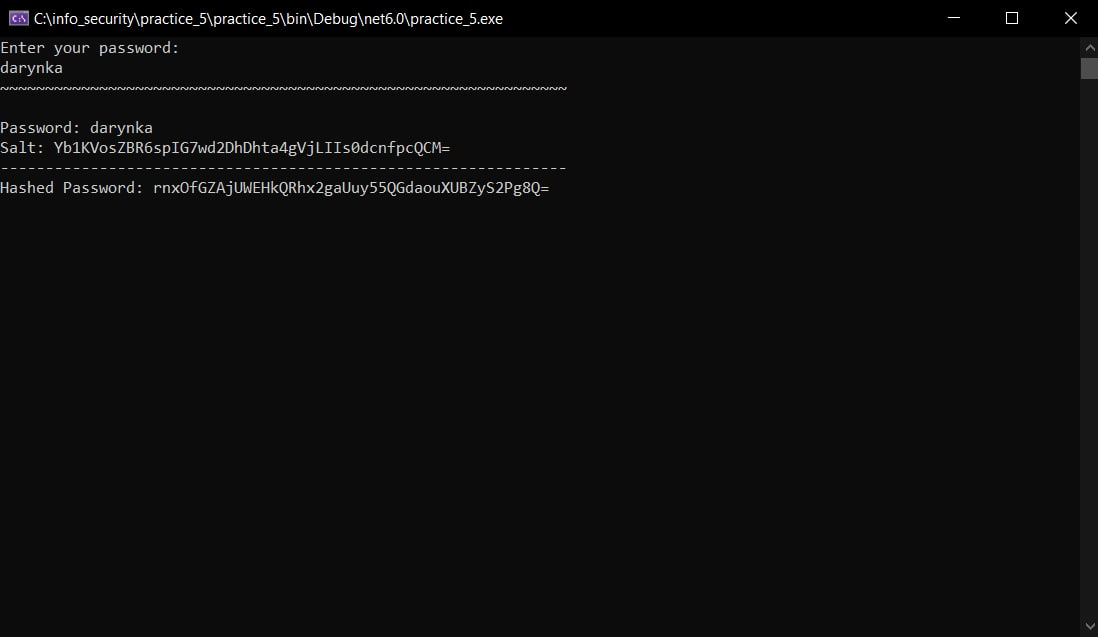
Виконала студентка групи МІТ-21 **Борук Дарина**

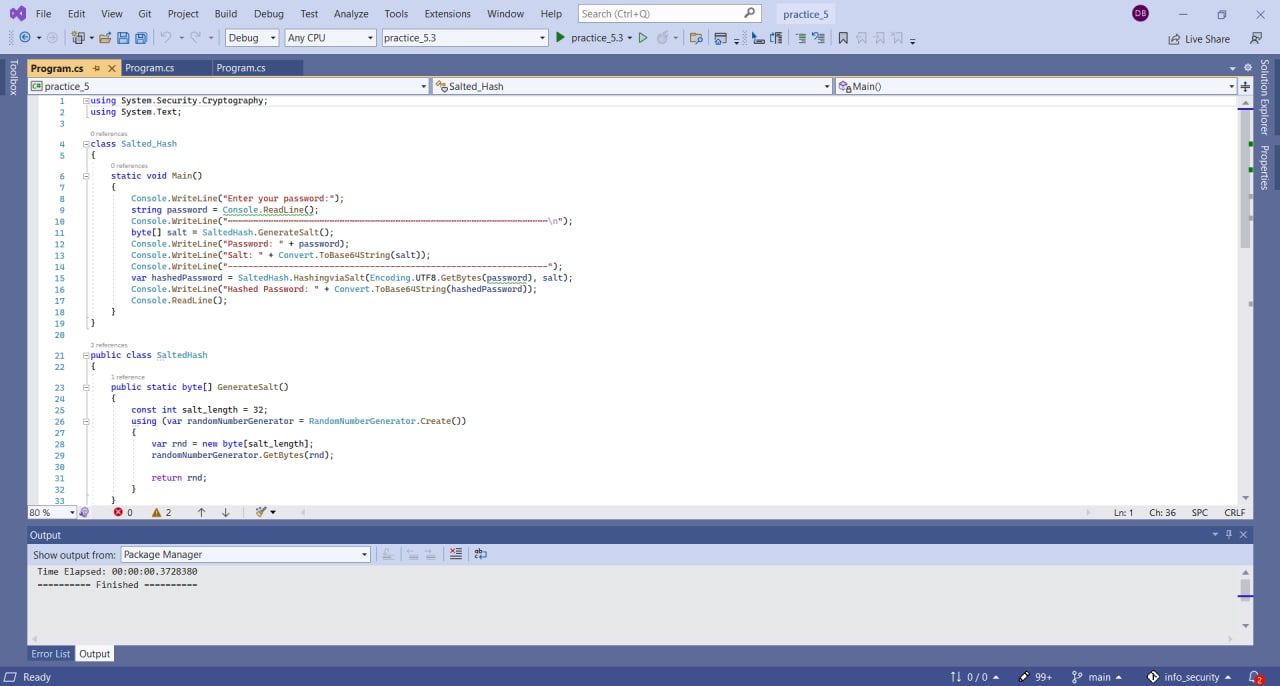
Мета: розібратися із сіллю для хешування, написати програму для реалізації хешування паролів із додаванням додаткової ентропії, програму для генерації солі з алгоритмом хешування (MD5, SHA1, SHA256, SHA384, SHA512), яка обчислює час, витрачений на обчислення хешу для певного числа ітерацій та програму, що реалізує хешування введеного пароля під час реєстрації користувача та зберігає логін, пароль та сіль з можливістю автентифікації.

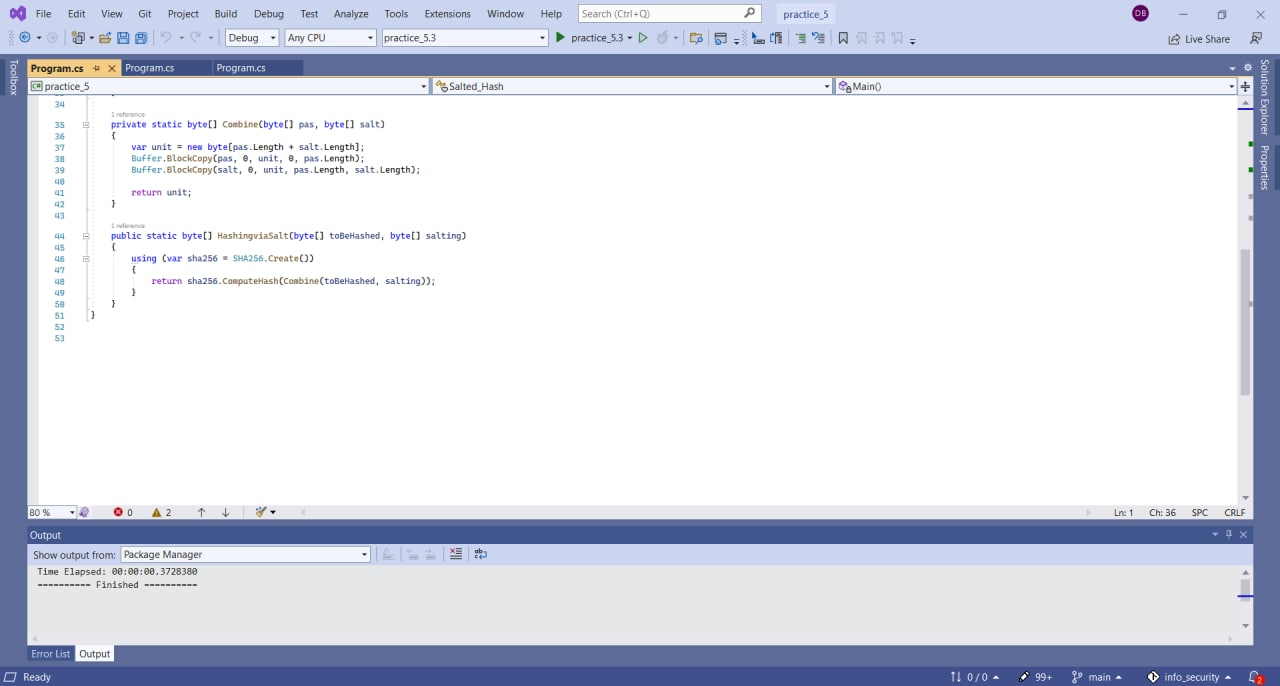
Хід роботи:

* Розпочнемо з написання першої програми для реалізації хешування паролів із додаванням додаткової ентропії.

1. Для початку напишемо інтерфейс для користувача (треба буде ввести пароль, а для результату виведемо пароль, сіль і захешований пароль)
2. Створимо клас SaltedHash, в якому пропишемо усі наші функції для хешування паролю. Пропишемо функцію GenerateSalt, яка згенерує рандомну послідовність цифр, розміром 32 байти, для створення солі.
3. Далі напишемо 2 функції: HashingviaSalt, яка захешує наш пароль за допомогою солі, використовуючи алгоритм SHA256, і Combine, яка згенерує остаточний результат нашого хешування за допомогою комбінування результатів копіювання нашого паролю з сіллю.
4. Отримаємо результат (у прикладі використано пароль «darynka»)

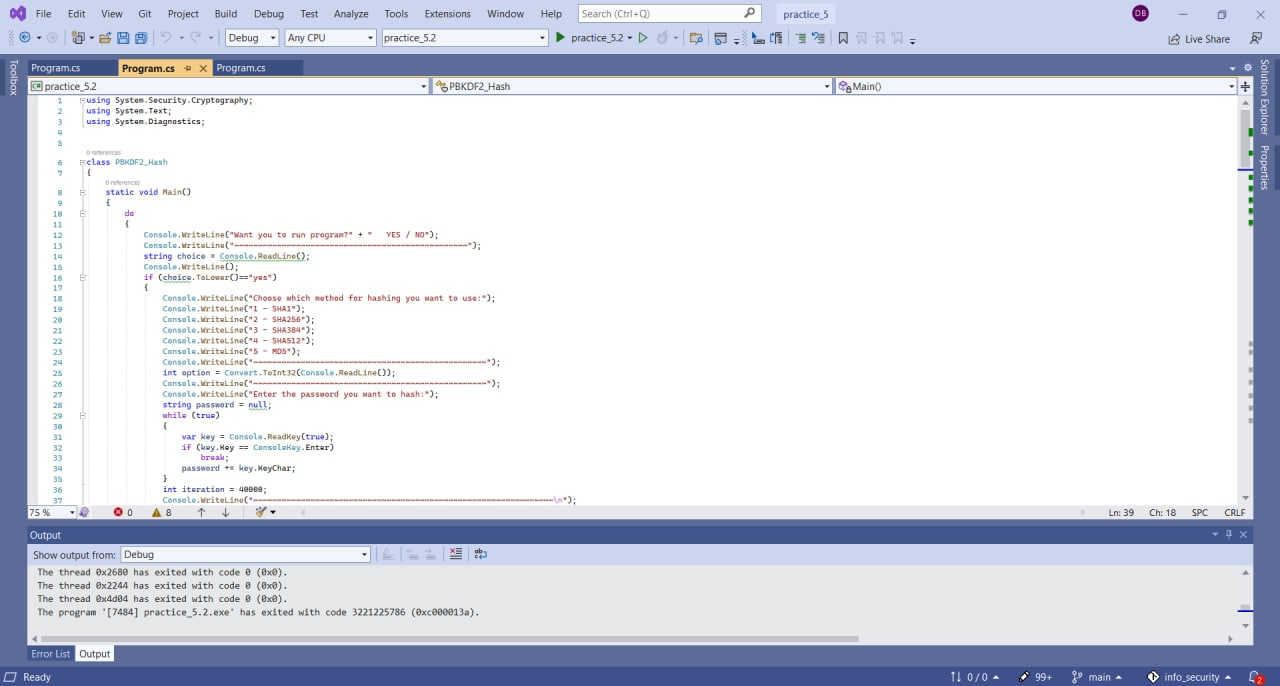


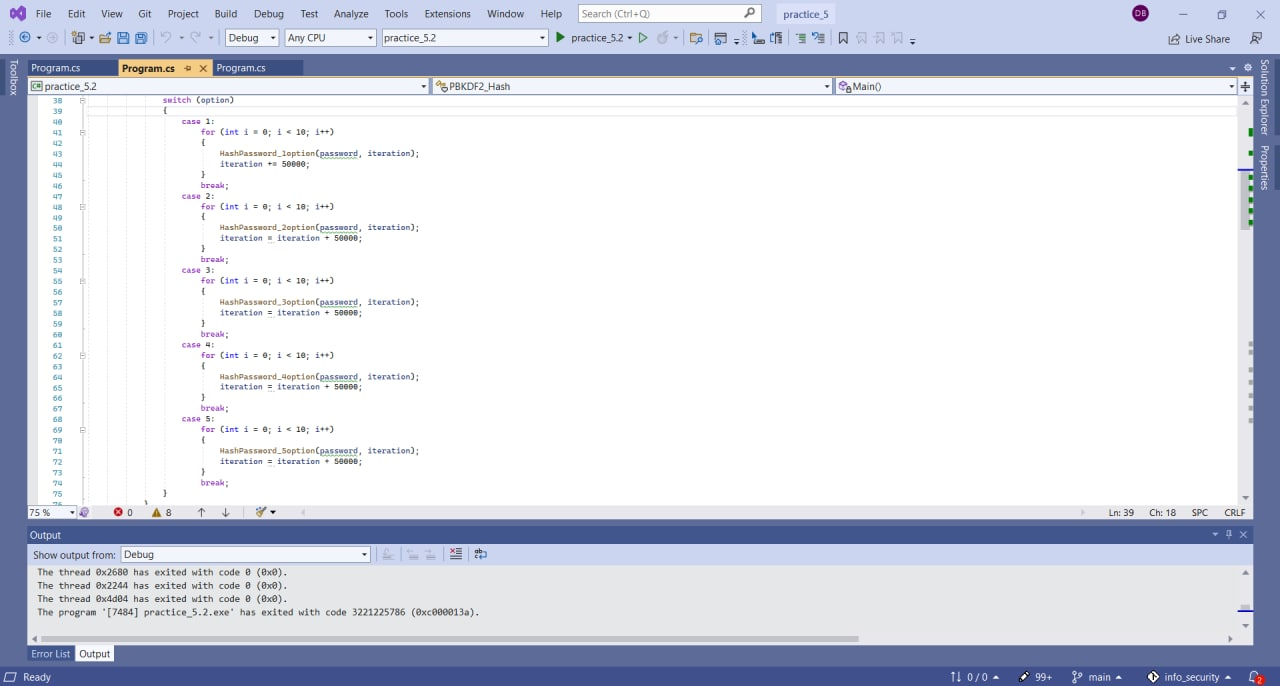


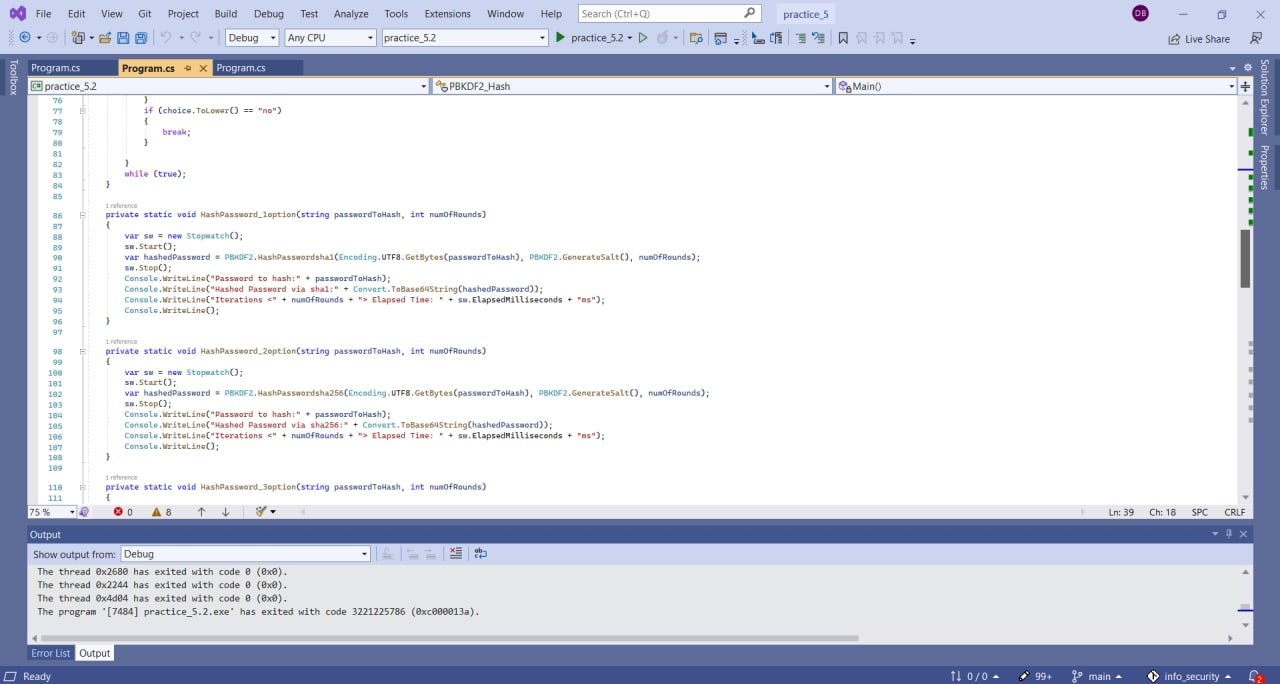


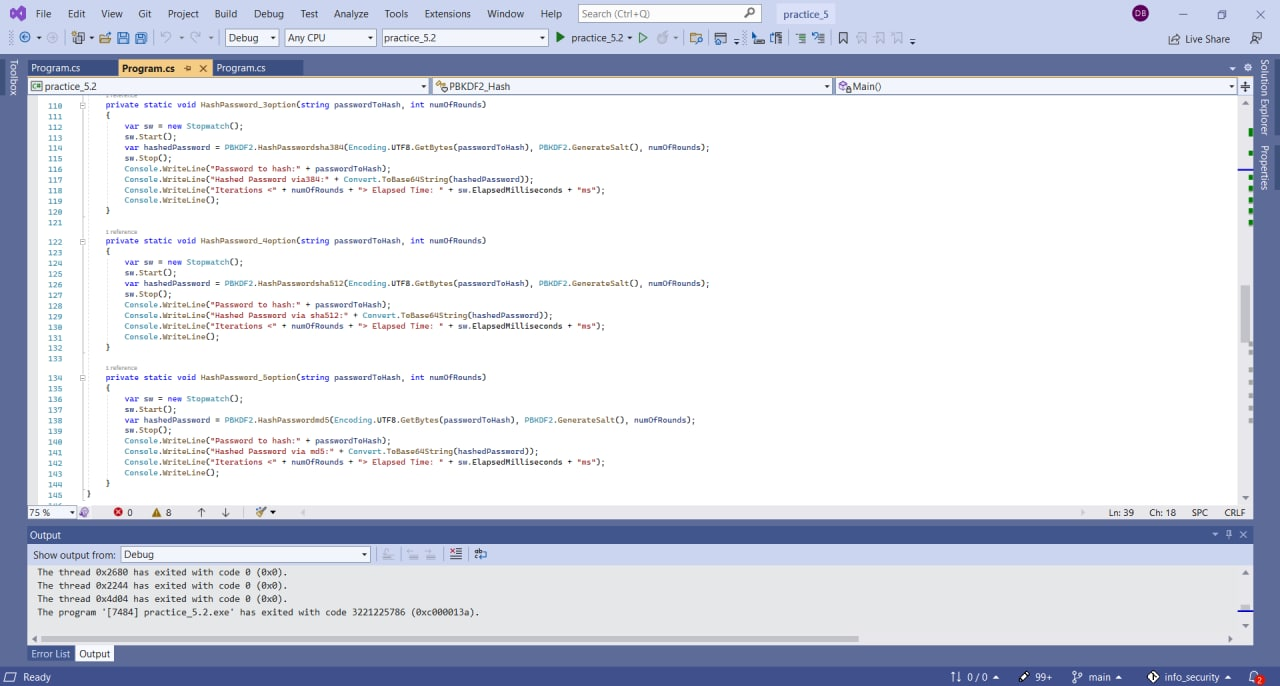
* Далі напишемо програму для генерації солі з вибором алгоритму хешування (MD5, SHA1, SHA256, SHA384, SHA512), яка обчислює час, витрачений на обчислення хешу для певного числа ітерацій (10 ітерацій з кроком 50000, перше значення 40000).

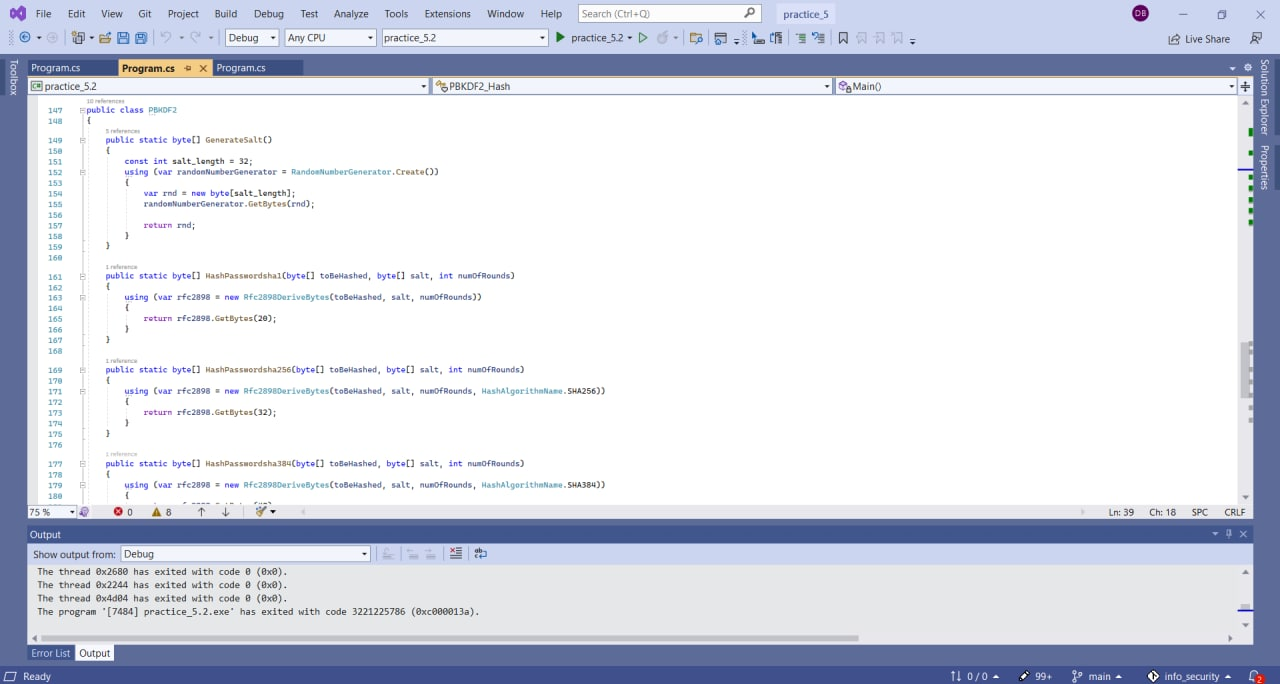
1. Напишемо інтерфейс для користувача: введення паролю і вибір алгоритму.
2. Далі пропишемо оператор switch для обраного кейсу з відповідним алгоритмом. У кожному кейсі пропишемо цикл для ітерації, у якому буде викликатись наша функція хешування і наступне значення ітерації збільшуємо на 50000.
3. Перейдемо до написання функцій. Перша функція HashPassword\_«n»option, де «n» номер кейсу, використовується для хешування, підрахунку часу та виведення на екран результатів: самого паролю, захешованого паролю, кількість ітерацій і витрачений на них на час. Створюємо клас PBKDF2, у якому пропишемо другу і третю функції. Друга функція буде для генерації солі, вона нам вже відома з написання першої програми, а третя HashPassword«algorithm», де «algorithm» назва алгоритму хешування. Всі функції будуть аналогічні одна одній, тому розглянемо одну функцію: у нашу функцію ми передаємо наш пароль, сіль і кількість ітерацій. Використовуючи клас Rfc2898DeriveBytes, викличемо функцію, яка реалізує формування ключа на основі пароля (PBKDF2) за допомогою генератора випадкових чисел HMACSHA1 (єдина різниця для кожної функції полягає у тому, що викликаючи цей клас, ми уточнюємо, який саме алгоритм ми хочемо використати за допомогою структури HashAlgorithmName.«name», де «name» назва алгоритму (MD5, SHA1, SHA256, SHA384, SHA512)). Повертаємо результат у вигляді перших k байтів нашого ключа, де k окрема кількість байтів для відповідної функції (20 – SHA1, 32 – SHA256, 48 – SHA483, 64 - SHA512, 16 – MD5).

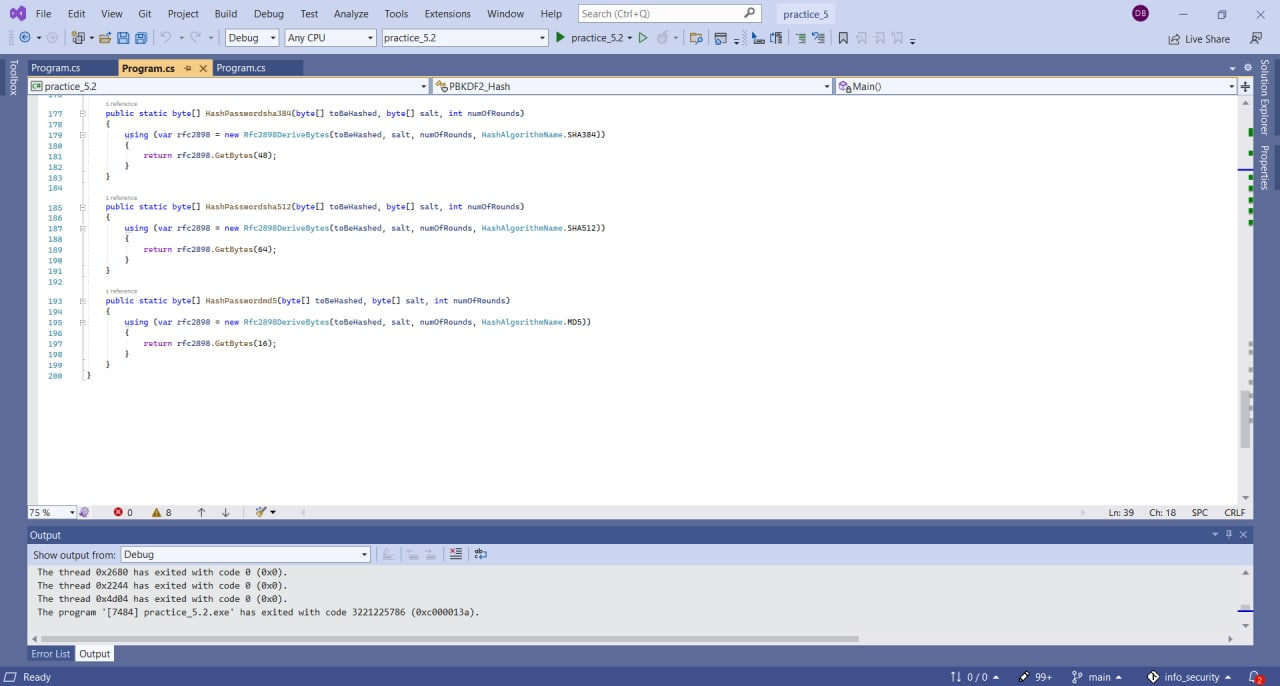




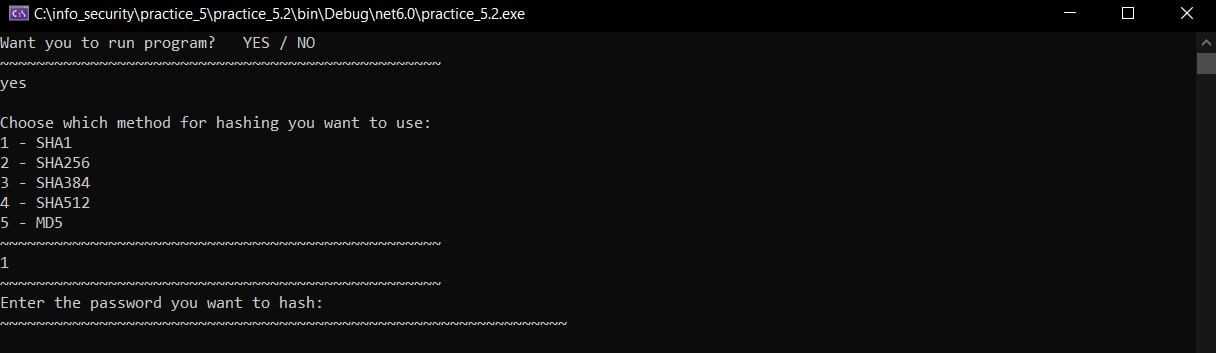


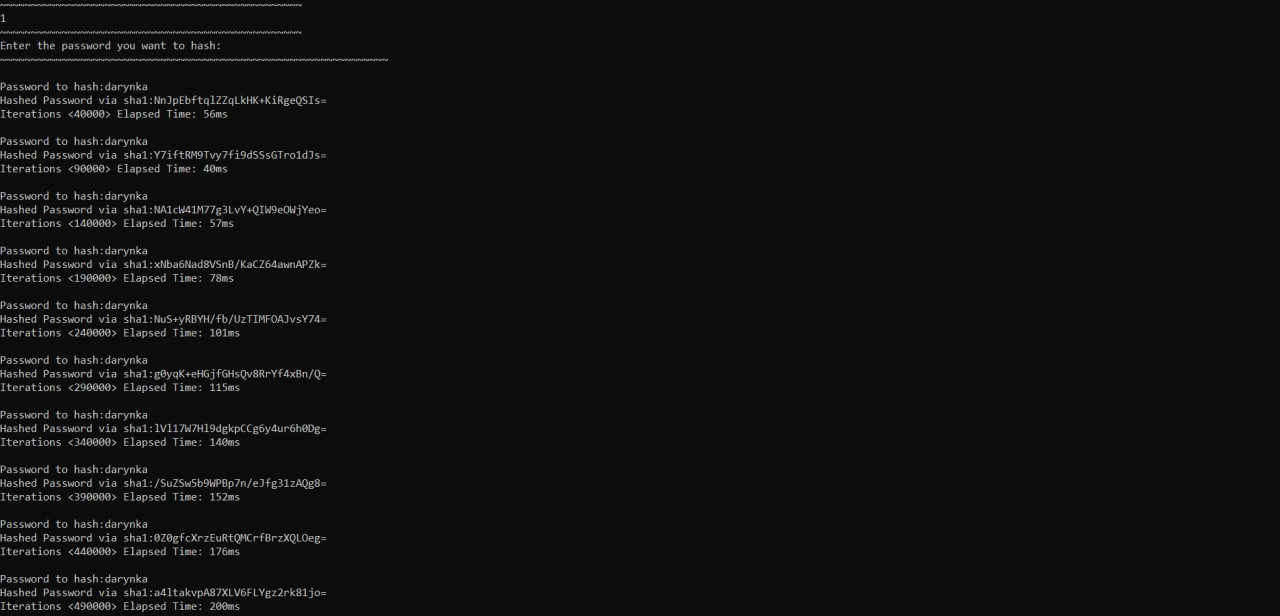




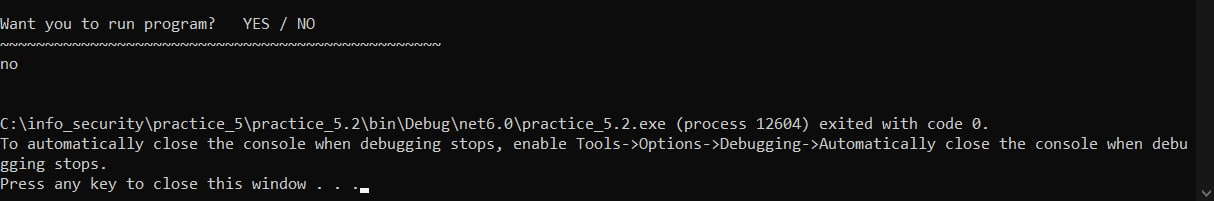


Маємо такий результат програми (пароль «darynka»):



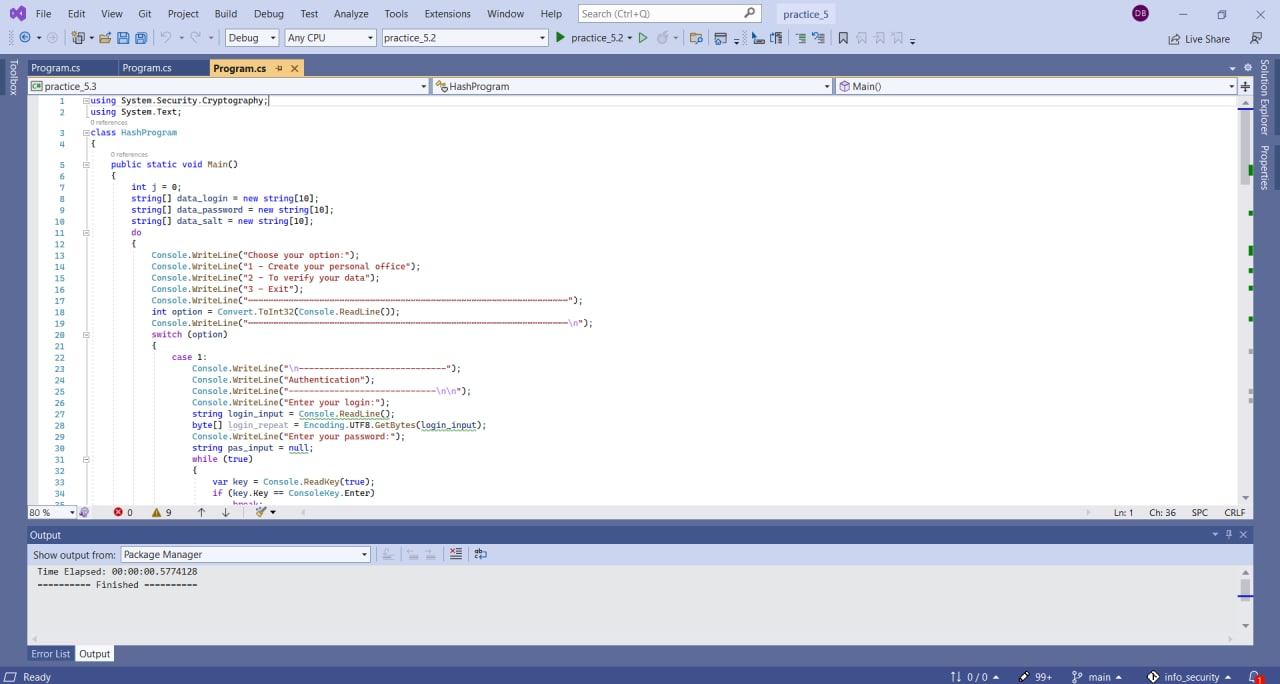


Для інших кейсів вигляд результатів буде такий самий але з іншими результатами хешування. У випадку з вибором «no» програма припиняє роботу.

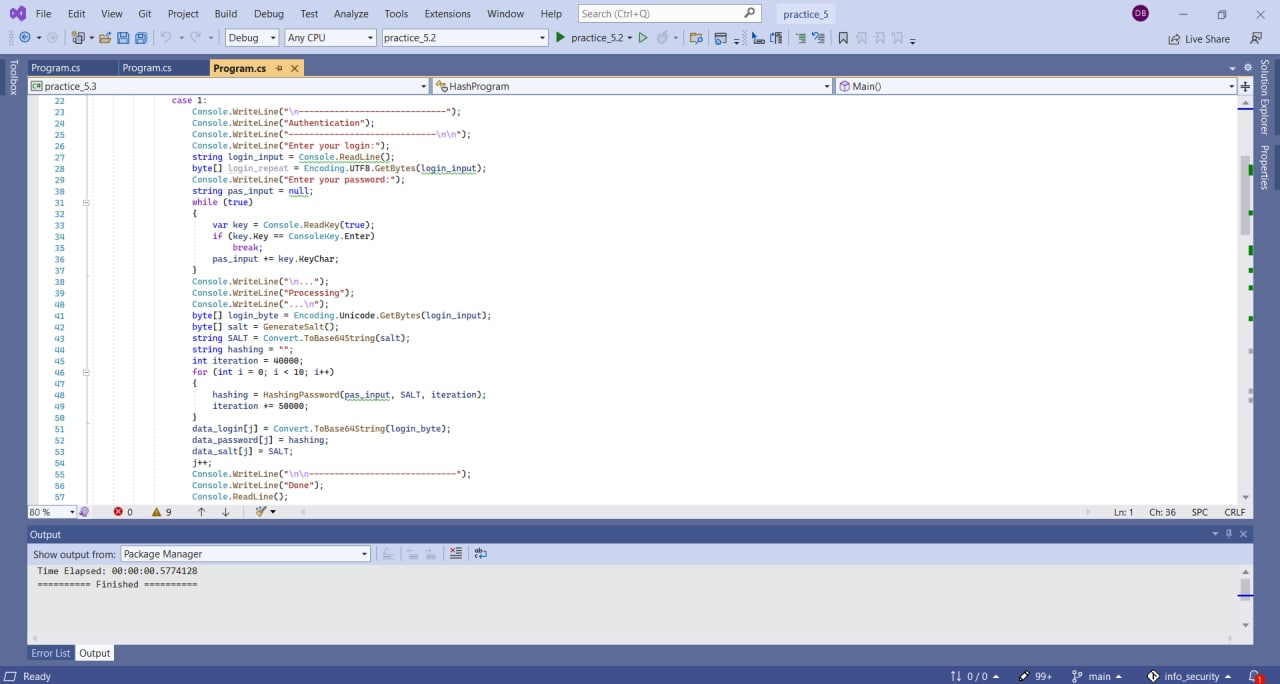


* Останньою напишемо програму, що реалізує хешування введеного пароля під час реєстрації користувача та зберігає логін, пароль та "сіль" у пам'яті і реалізуємо можливість автентифікації за логіном і паролем.

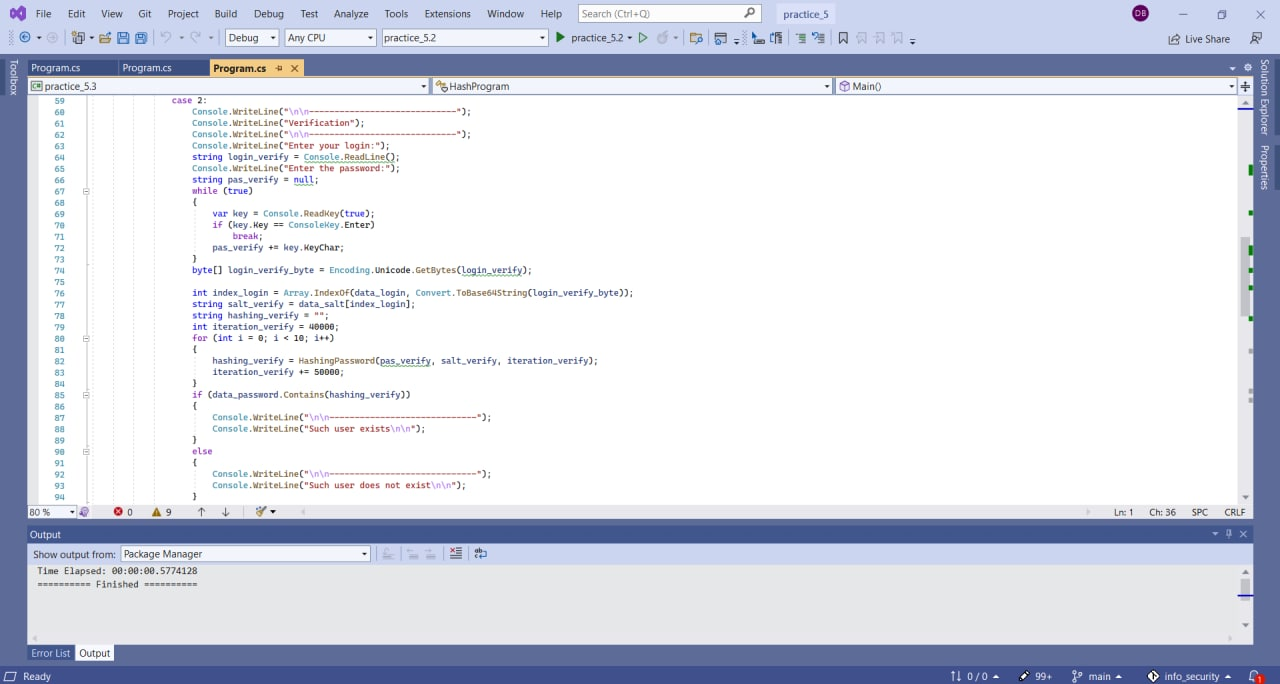
1. Напишемо інтерфейс для користувача і оголосимо 3 масиви байтів, що можуть містити 10 елементів, для збереження певних даних: логін користувача, пароль і сіль.



1. Далі напишемо кейс, який буде обробляти дані для створення хешованого пароля та зберігати необхідні для цього дані у наші масиви. Загалом, робимо так званий кабінет користувача з логіном і паролем. Дії, схожі до попередніх програмах: перетворення введених даних у байти, створення солі і нашого хешу, ітерація. В кінці додаємо наш захешований пароль, логін і сіль у масиви (кожен елемент має індекс, спільний для всіх цих трьох даних), збільшуючи змінну j, яка виконує роль значення індексу, на одиницю.

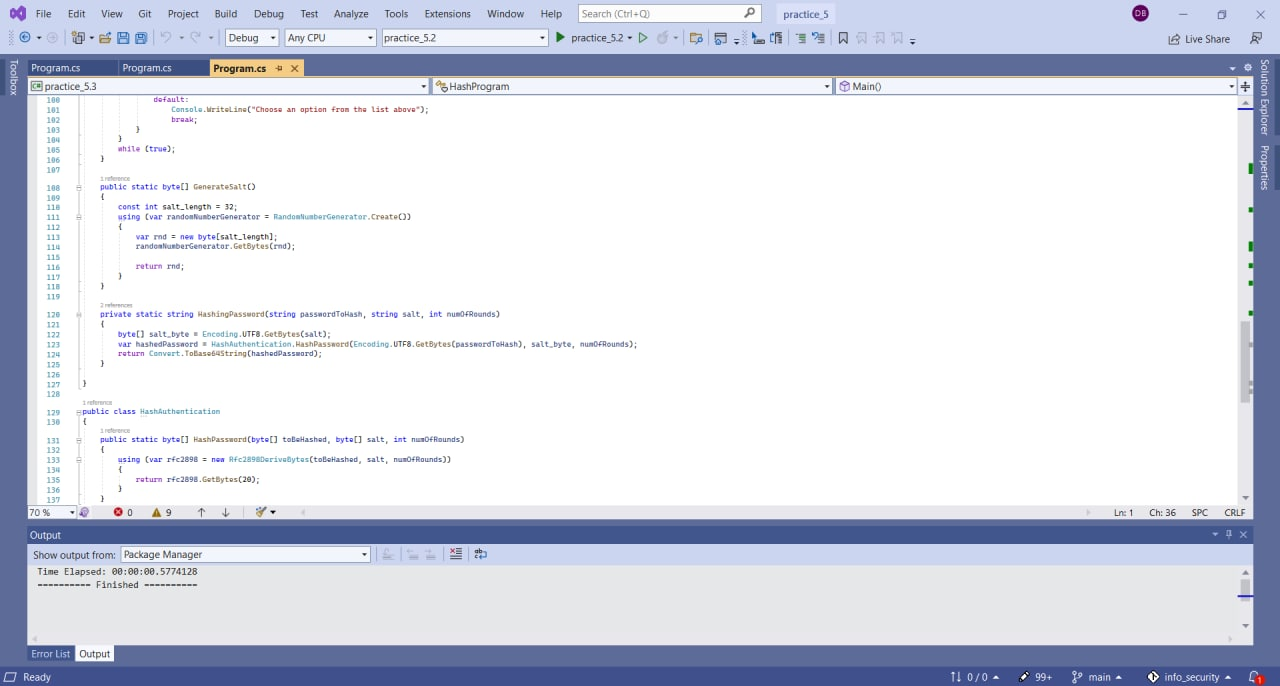


1. Далі пропишемо автентифікацію користувача, яка вимагає введення паролю і логіна. Ми шукаємо індекс нашого логіну у масиві з логінами і потім шукаємо сіль, елемент масиву якої має такий самий індекс. Потім проводимо хешування паролю з цією сіллю з ітераціями і перевіряємо чи існує у нашому масиві паролів такий пароль. Якщо існує, то виводимо повідомлення про те, що користувач існує, якщо ні – то відповідне повідомлення з інформацією про те, що такого користувача немає.



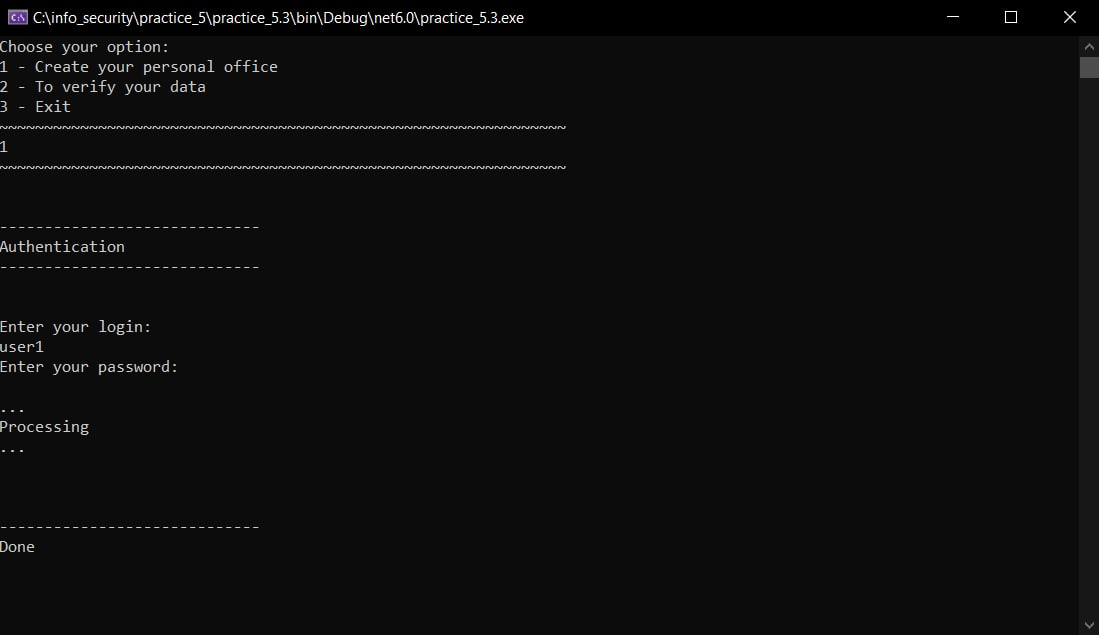
1. Розглянемо функції, які використовуються у даній програмі.

Всі функції нам відомі з попередніх програм: функція для генерації солі, функція хешування і функція, яка безпосередньо хешує і повертає результат.

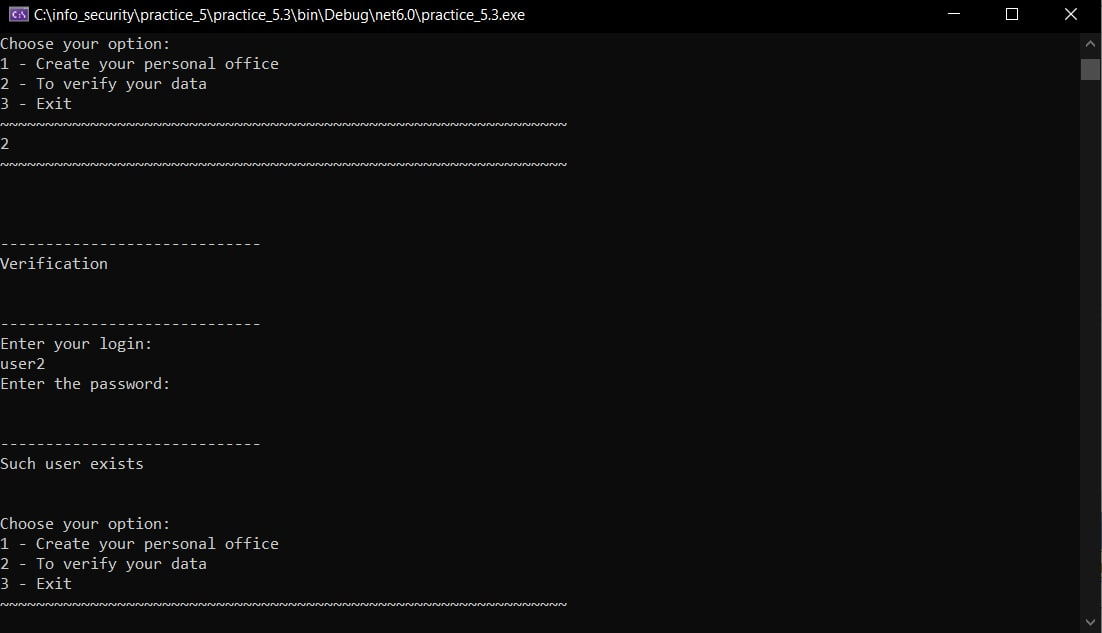


Розглянемо результат виконання цієї програми.

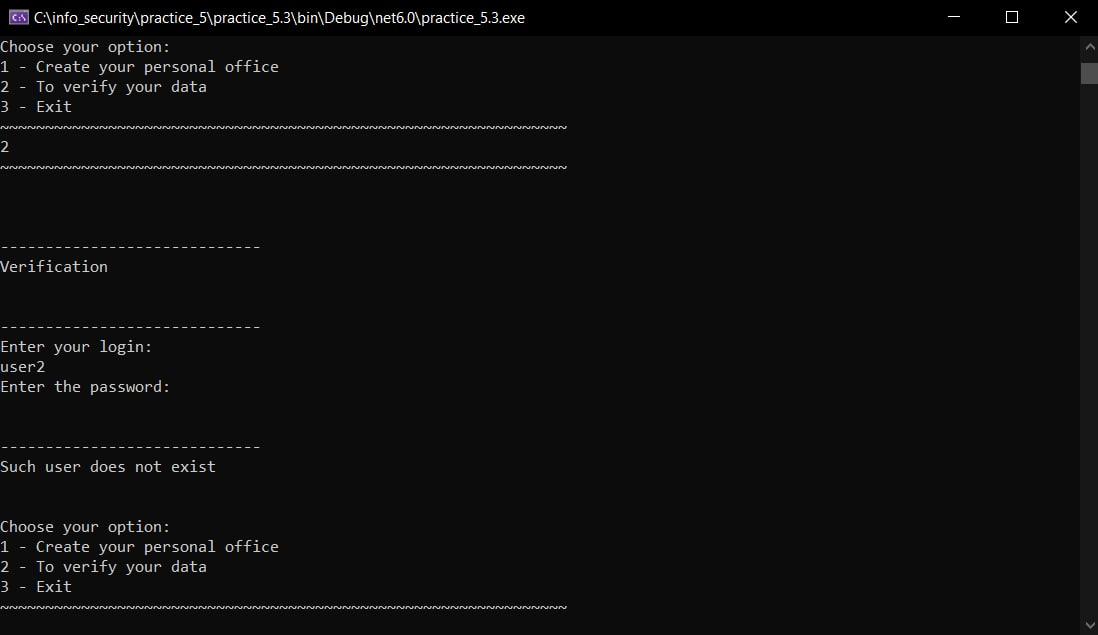
Створимо таких користувачів: user1 – password1, user2 – password2, user3 – password3.



Перевіримо чи існує наш другий користувач (введемо правильні дані) і побачимо, що такий користувач дійсно існує.



Зробимо ті ж самі дії, але введемо неправильні дані (user2 – password3) і побачимо, що такого користувача немає.



Висновки: у ході практичної роботи були отримані знання про сіль для хешування, хешування з додатковою ентропією, використовуючи алгоритми MD5, SHA1, SHA256, SHA384 та SHA512.