**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

До лабораторної роботи №2

**На тему: “Основи модульного тестування програм з використанням мови С++”**

**З дисципліни:** *“Об’єктно-орієнтовне програмування”*

**Лектор:**

доц. каф. ПЗ

Тетяна КОРОТЄЄВА

**Виконав:**

ст. гр. ПЗ-16

Тарас КАБАЧОК

**Прийняв:**

асист. каф. ПЗ

Владислав ДАЛЯВСЬКИЙ

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

Львів – 2025

**Тема роботи:** Основи модульного тестування програм з використанням мови програмування С++

**Мета роботи:** Навчитись створювати та виконувати модульні тести з використанням бібліотеки Google Test

**Теоретичні відомості**

Google Test – це бібліотека для модульного тестування коду мовою C++. Вона поширюється за ліцензією BSD та може компілюється під багато POSIX та Windows платформ та дозволяє тестувати код C та C++ з мінімальними змінами.

Розробка тестів полягає у створенні функцій-тестів, що містять виклик тестованого коду та спеціальних макросів стверджень (EXPECT\_\*, ASSERT\_\*, і тд.). Для створення функцій використовують спеціальні макроси (TEST, TEST\_F, TEST\_P).

**Хід роботи**

1. **Встановлення бібліотеки Google Test**

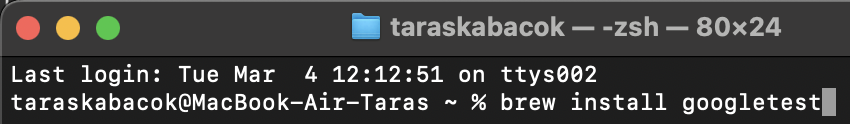


Рис. 1. Використовуючи Homebrew встановимо бібліотеку за допомогою команди

1. **Налаштування Cmake**

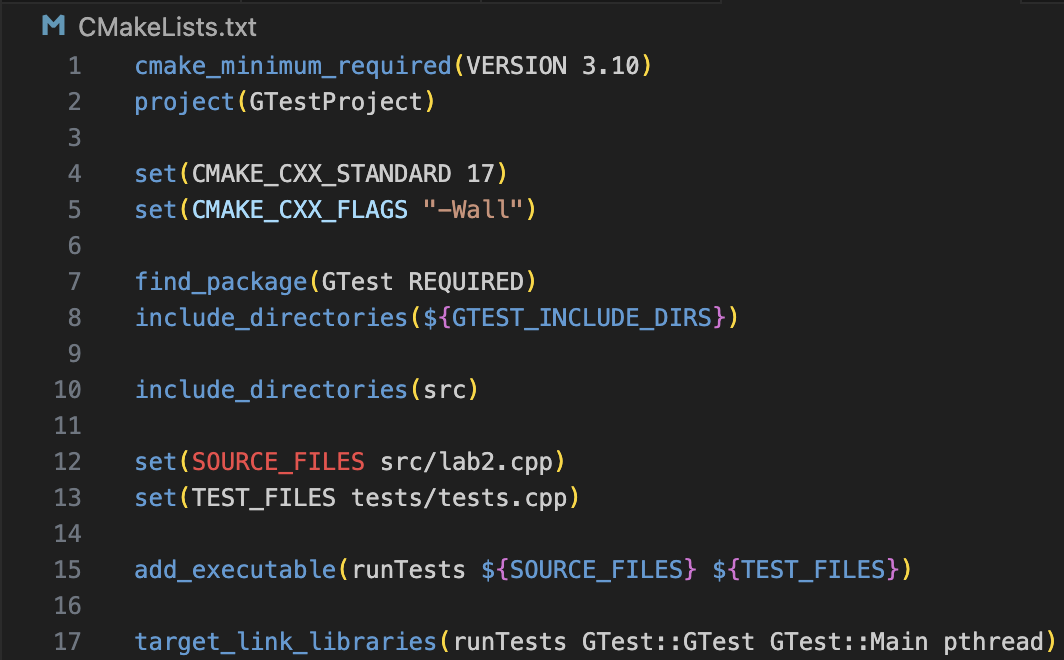
****

Рис. 2. Файл **CMakeLists.txt**

CMake потрібен для спрощення компіляції проекту, він автоматично знаходить потрібні біліотеки, та наліштовує збірку для різних систем. Найголовніше – він спрощує написання довгих команд компіляції до запуску make файлу.

1. **Створення файлів**

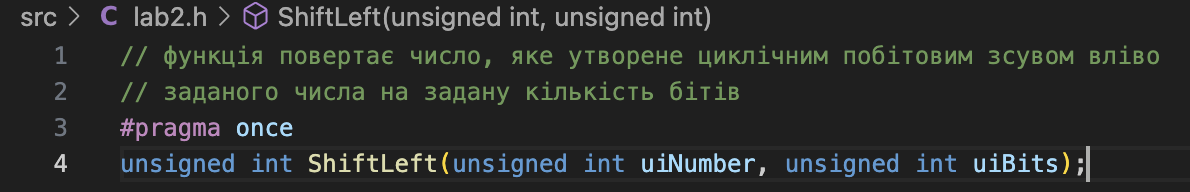


Рис. 3. **lab2.h**

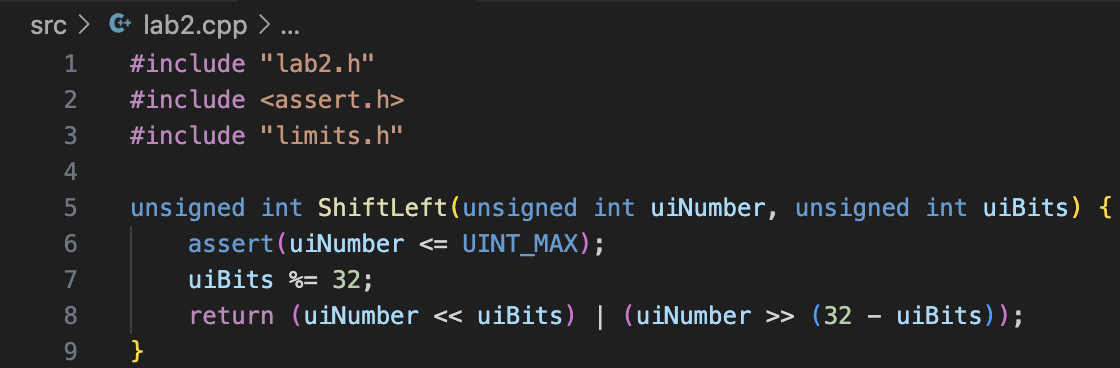


Рис. 4. **lab2.cpp**

Принцип роботи функції зводиться до об’єднання результатів побітового зсуву вліво та вправо операцією логічного АБО (OR). Зсунувши вліво дане число на задану кількість бітів, ми втрачаємо частину, яка може вийти за межу 32-бітного `unsigned int`. Тому потрібно ще зсунути число вправо на `32 – (задана кількість зсуву)` бітів, щоб відкинути саме ту частину, яка залишилась при зсуві вліво. В результаті отримаємо два значення, які містять різні частини початкового числа. Застосувавши між ними логічну операцію АБО, отримаємо число, в якому на кожному розряді стоятиме `1`, якщо хоча б в одному з проміжних результатів там була `1`. Це дозволяє реалізувати циклічний зсув, де втрачені біти з одного боку компенсуються бітами з іншого.

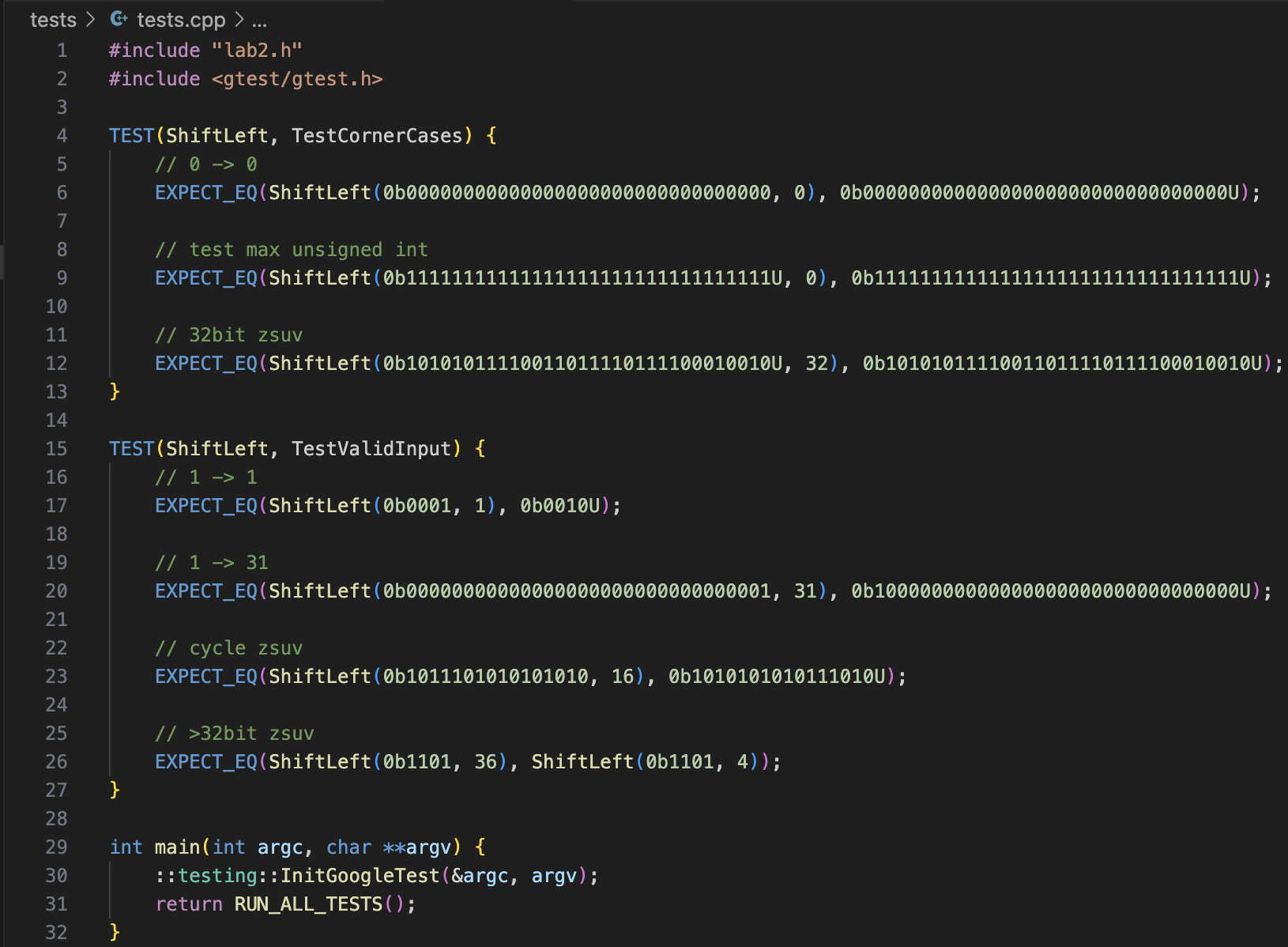


Рис. 5. **tests.cpp**

* У тестах ShiftLeft, TestCornerCases:
  + Перевіряється, що зсув нуля на 0 бітів не змінює число.
  + Перевіряється, що максимальне unsigned int (0b11111111111111111111111111111111U) при зсуві на 0 також залишається без змін.
  + Перевіряється зсув на 32 біти, що має дати те саме число, оскільки циклічний зсув на повний розряд нічого не змінює.
* У тестах ShiftLeft, TestValidInput:
  + Перевіряється, що 1 при зсуві на 1 біт стає 2 (тобто 0b0001 -> 0b0010).
  + Перевіряється, що 1 зсунуте на 31 біт стає 0b10000000000000000000000000000000.
  + **Помилковий тест:** циклічний зсув 16-бітного числа 0b1011101010101010, який має повернути 0b1010101010111010.
  + Останній випадок перевіряє, що зсув на 36 біт має бути рівним зсуву на 4 біти (ShiftLeft(0b1101, 36) == ShiftLeft(0b1101, 4)).

1. **Результат виконання програми**

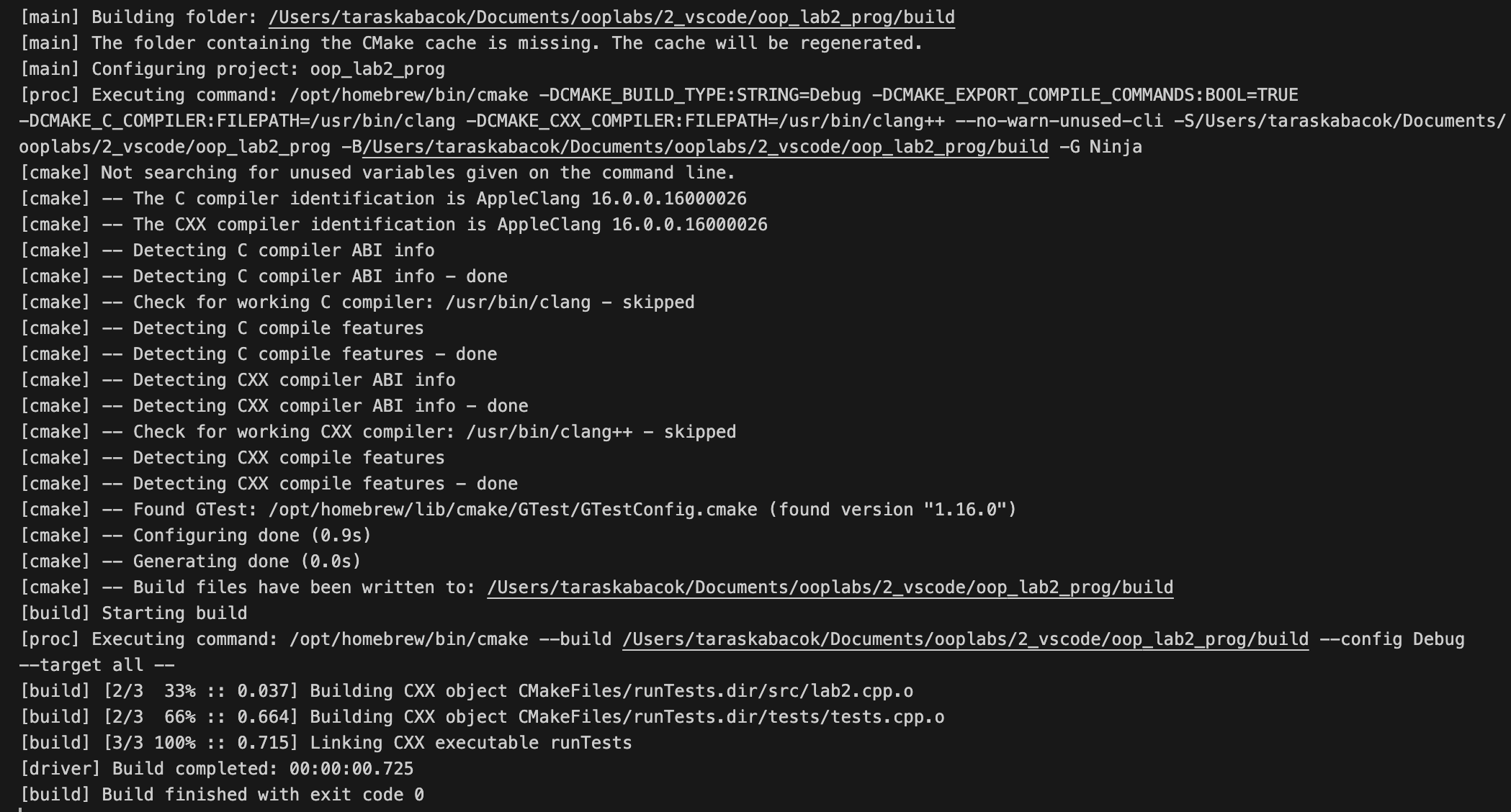


Рис. 6. Результат запуску CMake, створення MakeFile та компіляція проекту

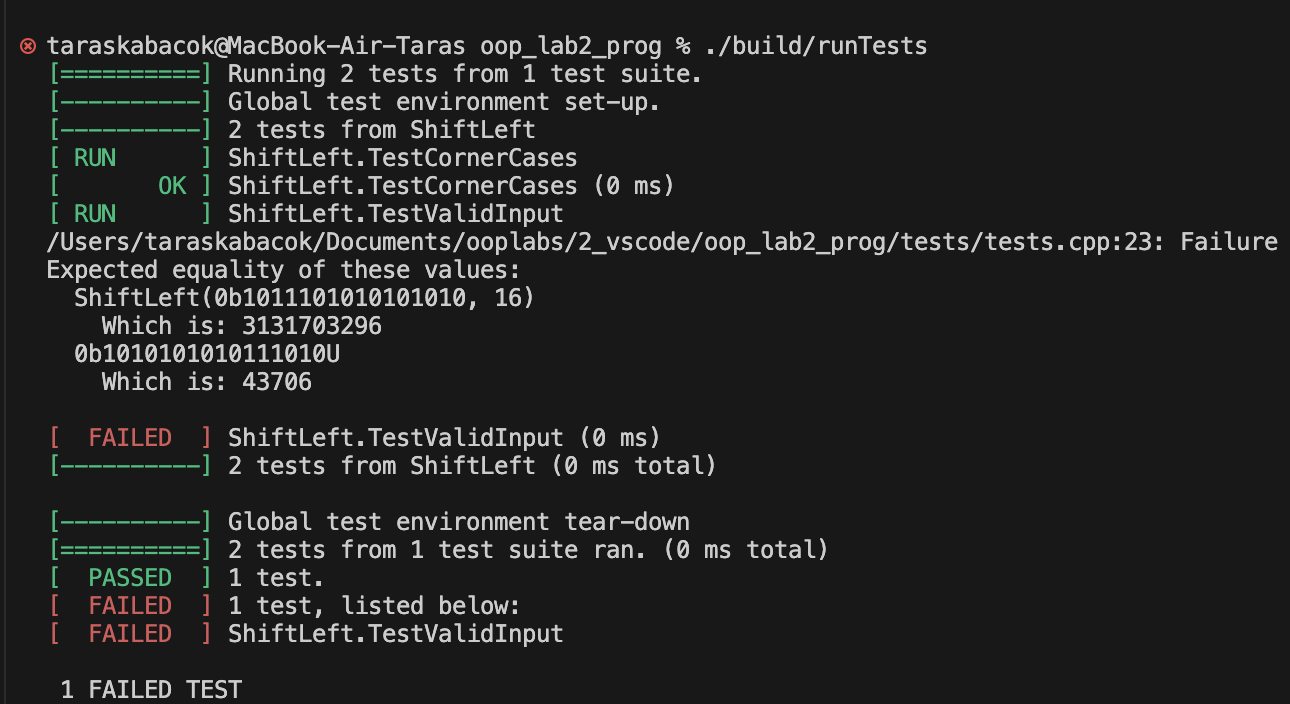


Рис. 7. Результат запуску виконуваного фалу runTests

**Висновки**

В цій лабораторній роботі я познайомився з використанням бібліотеки google tests, та створенням Unit тестів для програм. Окрім цього я протестував програму на правильні та завчасно неправильні тести.