ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ФОРМА НАВЧАННЯ ОЧНА

КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець
(підпис)

«_____»_____2020 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО ДИПЛОМНОЇ РОБОТИ

на тему

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТРЕНАЖЕРУ З ТЕМИ «СОРТУВАННЯ ФОН НЕЙМАНА» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ»

з спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Козодуб Владислав Сергійович _____ «___»_____2020 р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Ємець Олександра Олегівна

_____ «___»_____2020 р.
(підпис)

ПОЛТАВА 2020 р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

ЗАТВЕРДЖУЮ

Завідувач кафедри _____ О.О. Ємець
(підпис)

«_____» _____20_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК

ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ

Студента з спеціальності 122 «Комп'ютерні науки»

Прізвище, ім'я, по батькові Козодуб Владислав Сергійович

1. **Тема** «Розробка програмного забезпечення тренажеру з теми «Сортування фон Неймана» дистанційного навчального курсу «Алгоритми та структури даних» **затверджена наказом ректора № 150-Н від «2» вересня 2019 р.**

Термін подання студентом магістерської роботи 20 травня 2020 р.

2.Вихідні дані до магістерської роботи публікації за темою роботи; методичні рекомендації; стандарти.

3. Зміст пояснювальної записки (перелік питань, які потрібно розробити).

Вступ.
1. Постановка задачі.
2. Інформаційний огляд.
   2.1. Огляд тренажерів, присвячених сортуванню чисел.
   2.2. Позитивні аспекти оглянутих тренажерів.
   2.3. Вади оглянутих тренажерів.
   2.4. Необхідність та актуальність теми.
3. Теоретична частина.
   3.1. Зовнішнє сортування.
      3.1.1. Злиття впорядкованих послідовностей.
      3.1.2. Сортування фон Неймана.
   3.2. Алгоритм тренажеру.
      3.2.1. Приклад 1 (кількість елементів – парне число).
      3.2.2. Приклад 2 (кількість елементів – непарне число).
   3.3. Блок-схема алгоритму тренажера.
4. Практична частина.
   4.1. Інструкція по використанню програми.
   4.2. Опис програми.

Висновки.

4. Перелік графічного матеріалу (з точним визначенням кількості блок-схем, іншого графічного матеріалу) Блок-схема алгоритму (5-6 листів).

5. Консультанти розділів магістерської роботи

Розділ	П.І.Б., посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	Ємець Ол-ра О.	09.09.2019	09.09.2019
1. Постановка задачі	Ємець Ол-ра О.	09.09.2019	09.09.2019
2. Інформаційний огляд	Ємець Ол-ра О.	09.09.2019	09.09.2019
3. Теоретична частина	Ємець Ол-ра О.	09.09.2019	09.09.2019
4. Практична частина	Ємець Ол-ра О.	09.09.2019	09.09.2019

6. Календарний графік виконання магістерської роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ	04.05.20	
2. Вивчення методичних рекомендацій і стандартів та звіт керівнику	1.10.19	
3. Постановка задачі	10.10.19	
4. Інформаційний огляд джерел бібліотек та інтернету	1.11.19	
5. Теоретична частина	13.01.20	
6. Практична частина	15.04.19	
7. Закінчення оформлення	04.05.20	
8. Доповідь студента на кафедрі	25.05.20	
9. Доробка (за необхідності), рецензування	07.06.20	

Дата видачі завдання «9»вересня 2019 р.

Студент _____ Козодуб В.С.
(підпис)

Науковий керівник _____ к. ф.-м. н., доц. Ємець Ол-ра О.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту магістерської роботи

Магістерська робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 20_ р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Записка: 75 с., 83 рис., 4 таблиці, 2 додатки (на 32 сторінках), 20 джерел.

Предмет розробки – програма-тренажер з теми «Сортування фон Неймана».

Мета роботи – створити програму-тренажер, яка буде допомагати студентам засвоювати тему «Сортування фон Неймана» при її самостійному вивченні. Тренажер повинен базуватися на матеріалі дистанційного курсу «Алгоритми і структури даних» Полтавського університету та торгівлі.

Методи розробки – мова С++, програмне середовищі Borland Builder 5.

На базі прикладу сформульовано алгоритм тренажеру для випадку, коли сортується парна кількість елементів.

Розроблено ілюстративний приклад для випадку, коли сортується непарна кількість елементів. На базі цього прикладу сформульовано ще один алгоритм тренажеру.

Обидва алгоритми запрограмовано.

Ключові слова: ДИСТАНЦІЙНА ОСВІТА, ТРЕНАЖЕР, ЗОВНІШНЄ СОРТУВАННЯ, СОРТУВАННЯ ФОН НЕЙМАНА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, одиниці, скорочення, терміни	**Пояснення умовних позначень, символів, одиниць, скорочень, термінів**
$a_1, ..., a_n$, $b_1, ..., b_m$	Елементи вихідних послідовностей.
$c_1, ..., c_{n+m}$	Елементи результуючої послідовності
n	Кількість елементів послідовності a
m	Кількість елементів послідовності b
$n + m$	Кількість елементів результуючої послідовності
i	Поточний елемент послідовності a
j	Поточний елемент послідовності b
k	Поточний елемент результуючої послідовності
АА	Аналіз алгоритмів
АіСД	Алгоритми і структури даних
ММСІ	Математичне моделювання та соціальна інформатика
ПУЕТ	Полтавський університет економіки і торгівлі

ВСТУП

Актуальність. В зв'язку з широким розповсюдженням інтернету та персональних комп'ютерів прийшла ера нових технологій навчання, зокрема, дистанційної освіти.

Дистанційна освіта дозволяє отримувати нові знання, підвищувати кваліфікацію, отримувати інформацію у суміжних з основною професією галузях. Крім того навчання вдома за своїм комп'ютером дозволяє економити час, суміщати роботу та навчання, отримувати дві освіти одночасно, знизити витрати на навчання.

Разом з тим, оскільки біля учня немає викладача, готового в будь-який момент роботи усунути помилку чи ще раз пояснити матеріал, дистанційна освіта повинна мати додаткові технології, які допомагають засвоїти матеріал самостійно. До таких технологій відносять комп'ютерні тренажери.

Комп'ютерний тренажер – це програма, яка навчає користувача розв'язувати певні типові задачі з професійної області.

Отже, тематика даної магістерської роботи є актуальної та доцільною.

Мета магістерської роботи – створити програму-тренажер, яка буде допомагати студентам засвоювати тему «Сортування фон Неймана» при її самостійному вивченні. Тренажер повинен базуватися на матеріалі дистанційного курсу «Алгоритми і структури даних» Полтавського університету та торгівлі.

Для досягнення мети магістерської роботи необхідно виконати наступні ***задачі***:

1. Ознайомитись з теоретичними відомостями з теми «Сортування фон Неймана».
2. За матеріалом дистанційного курсу розробити два алгоритми тренажерів для парної та непарної кількості елементів, що сортуються.
3. Створити блок-схему алгоритмів(у).

4. Створити на базі розроблених алгоритмів програму-тренажер. Перевірити програму.

Об'єкт розробки – програма-тренажер.

Предмет розробки – програма-тренажер з теми «Сортування фон Неймана».

Методи розробки – мова С++, програмне середовище Borland Builder 5.

Нові практичні розробки – створена нова програма-тренажер для теми «Сортування фон Неймана» для двох випадків, коли кількість елементів в послідовності парна або непарна. Для даної теми інших тренажерів не знайдено.

Готовність до впровадження – програма повністю готова, протестована та передана на впровадження у сектор розробки електронних засобів навчання Полтавського університету та торгівлі.

1. ПОСТАНОВКА ЗАДАЧІ

Необхідно ознайомитись з теоретичним матеріалом з теми «Сортування фон Неймана», використовуючи дистанційний курс «Алгоритми і структури даних» Полтавського університету економіки і торгівлі, а також інші інформаційні джерела.

Провести огляд тренажерів подібної тематики.

За матеріалом з дистанційного курсу, використовуючи поданий в ньому ілюстративний приклад, розробити алгоритм тренажеру для парної кількості чисел, що сортуються, з теми «Сортування фон Неймана».

Для даного алгоритму створити блок-схему або її фрагмент (відобразити принаймні декілька типових кроків).

Створити ілюстративний приклад сортування методом фон Неймана для випадку, коли кількість чисел, що сортуються, непарна. Розробити алгоритм тренажеру для цього прикладу.

На основі розроблених алгоритмів створити програму з використанням будь-якої мови візуального програмування. При цьому в тренажері зазначити виконавця програми, керівника, випускову кафедру та вставити емблему спеціальності.

Програму протестувати на предмет помилок, коректності виконання та зручності інтерфейсу.

У пояснювальній записці представити повний хід роботи тренажеру, використовуючи скриншоти програми. А також описати, як створювалась програма.

У додатку представити програмний код створеної програми (повністю або фрагментарно).

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд тренажерів, присвячених сортуванню чисел

Кафедрою математичного моделювання та соціальної інформатики ПУЕТ у співпраці із ІТ-студентами було створено більше ста програм-тренажерів, мета яких – самостійне освоєння тієї чи іншої теми [1-13].

Розглянемо тренажери, що стосуються сортування чисел, які розроблені для дисциплін «Алгоритми та структури даних» та «Аналіз алгоритмів».

1. Тренажер з теми «Шейкер-сортування» дисципліни «Алгоритми та структури даних».

Ця програма, створена студентом Кильником Вадимом у 2017 р., являє собою веб-сторінку, яку можна відкрити в будь-якому браузері [1] (рис. 2.1).

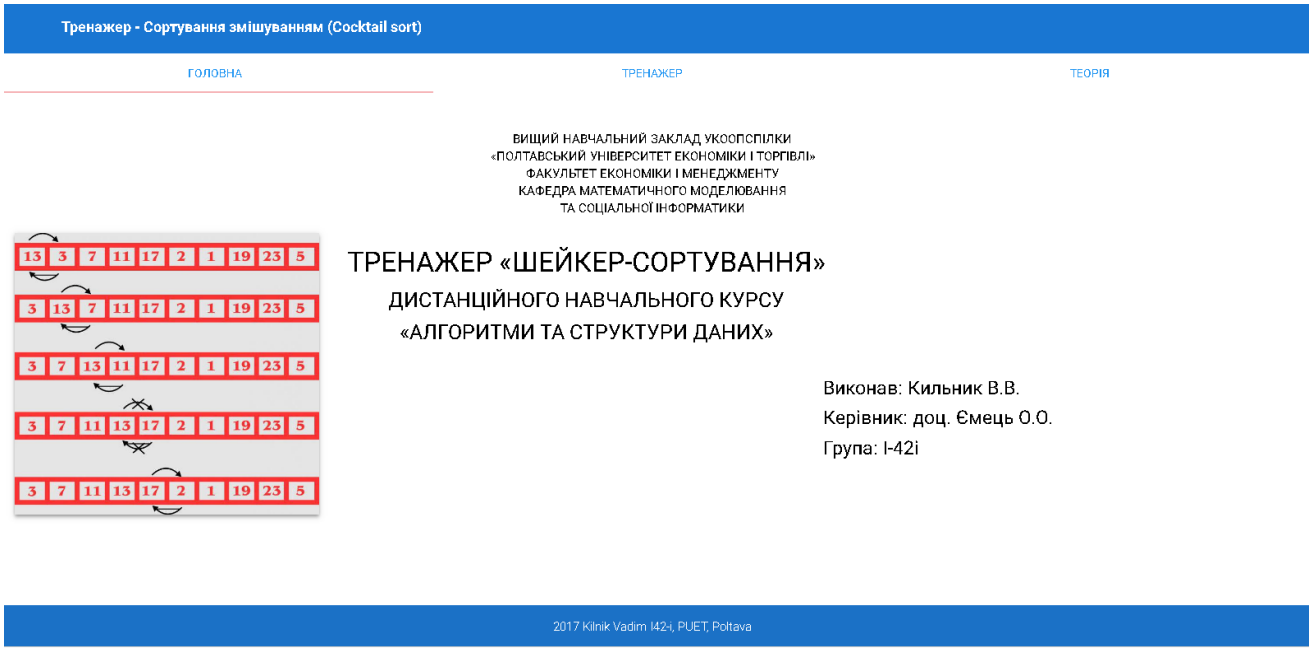

Рис. 2.1 – Тренажер «Шейкер-сортування»

Тренажер містить 3 рубрики: «Головна», «Тренажер», «Теорія». Перша – слугує в ролі титулу. Рубрика «Теорія» (рис. 2.2-2.4) містить опис методу

сортування; інформацію про складність методу; візуалізацію, що демонструє, як працює метод; та код мовою C++, що реалізує це сортування.

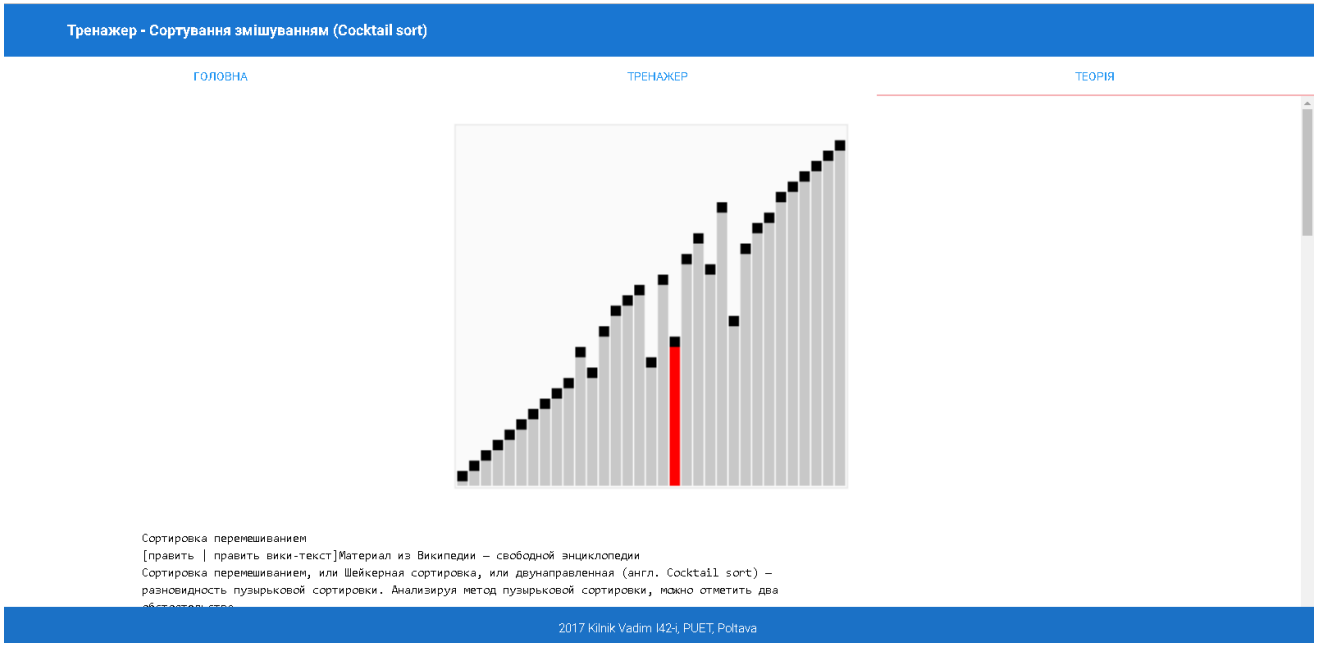

Рис. 2.2 – Тренажер «Шейкер-сортування», рубрика «Теорія»

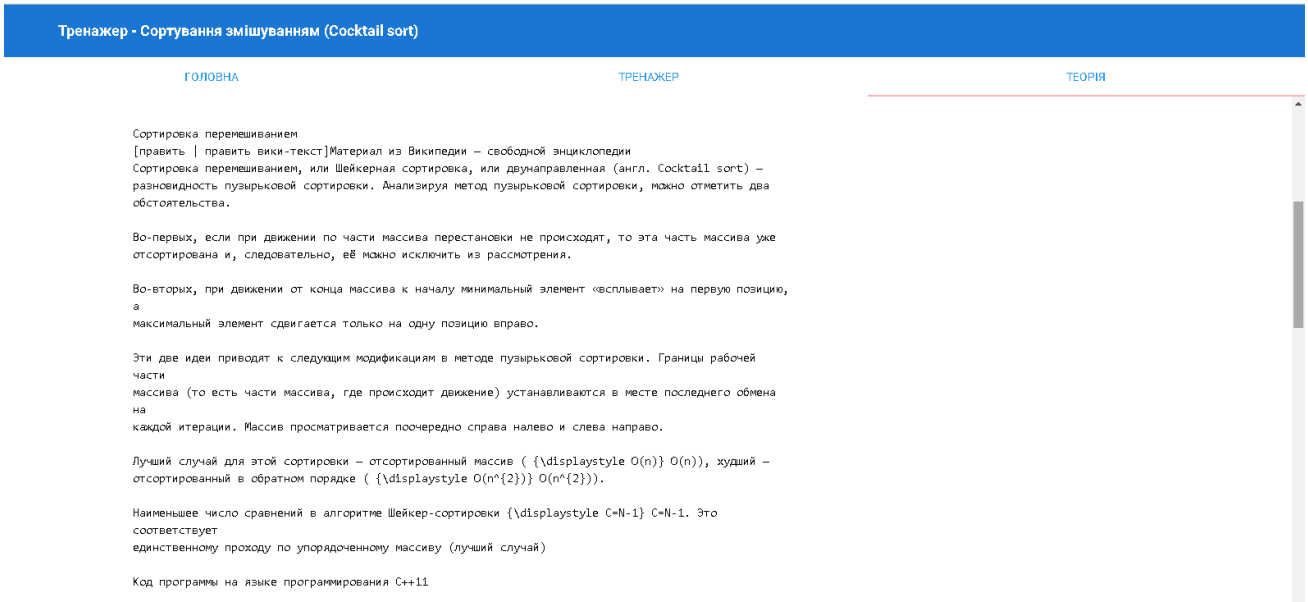

Рис. 2.3 – Тренажер «Шейкер-сортування», рубрика «Теорія»

Рубрика «Тренажер» – дозволяє ввести або згенерувати випадково (кнопка «Генерація») 5 чисел для подальшого сортування (рис. 2.5-2.6). Після натискання кнопки «Почати тренування» відбувається тренінг (рис.2.7). При правильній відповіді – йде перехід на наступний крок (рис. 2.8), при помилці – виникає

повідомлення про це (2.10). Користувач може скористатись підказкою (кнопка «Допомога», рис. 2.9).

Рис. 2.4 – Тренажер «Шейкер-сортування», рубрика «Теорія»

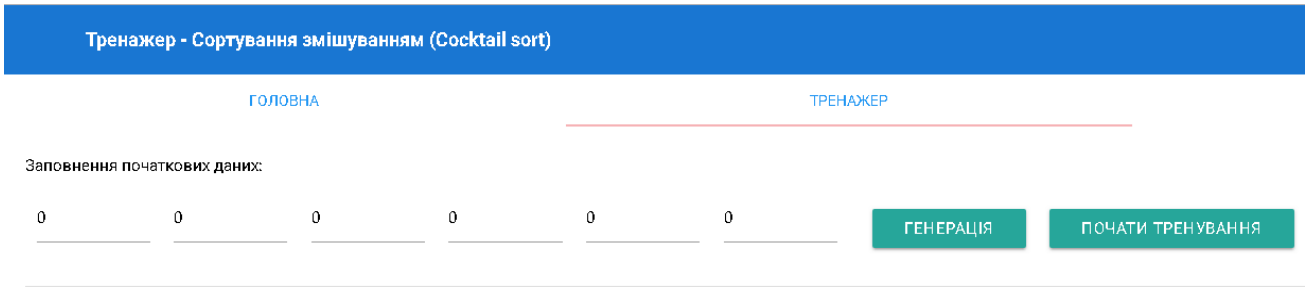

Рис. 2.5 – Тренажер «Шейкер-сортування», рубрика «Тренажер»

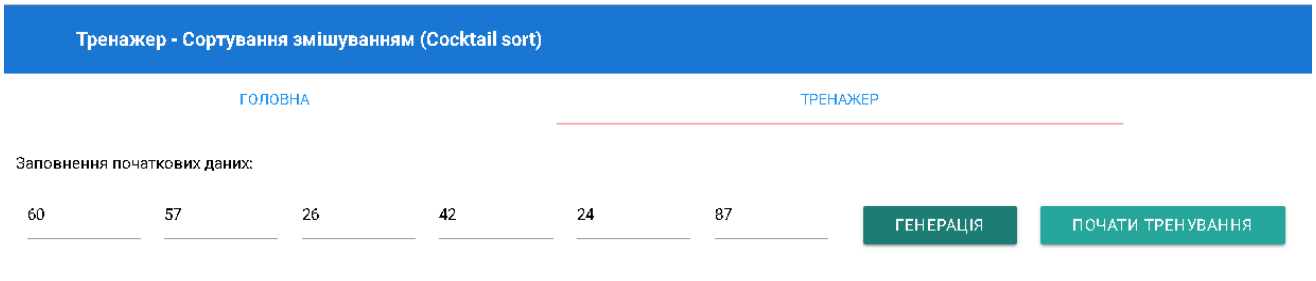

Рис. 2.6 – Згенеровані числа

При цьому числа, які слід переставляти, виділяються синім та червоним кольором та з’являється текст підказки (рис. 2.9).

На рис. 2.11-2.15 показано подальший хід роботи тренажеру.

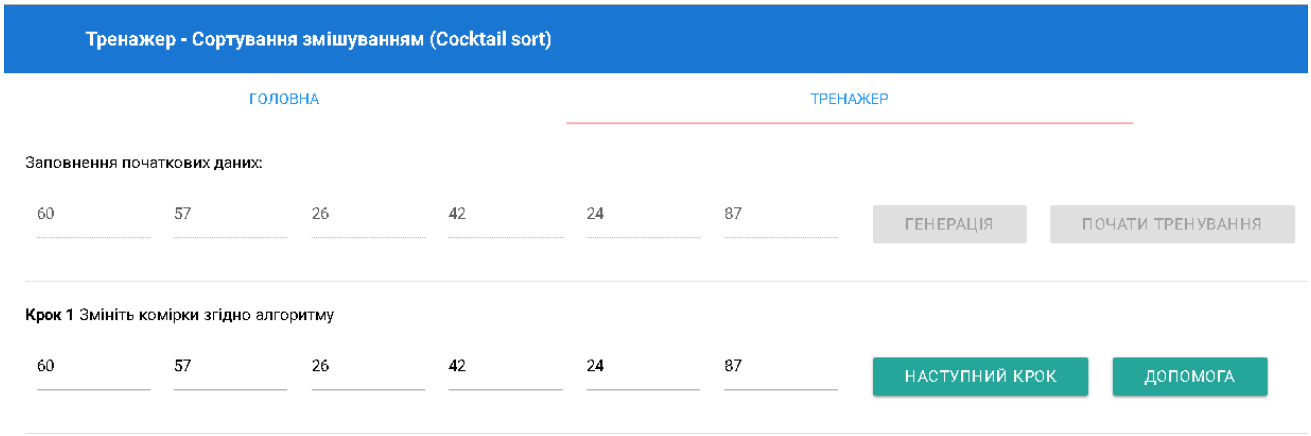

Рис. 2.7 – Початок першого кроку

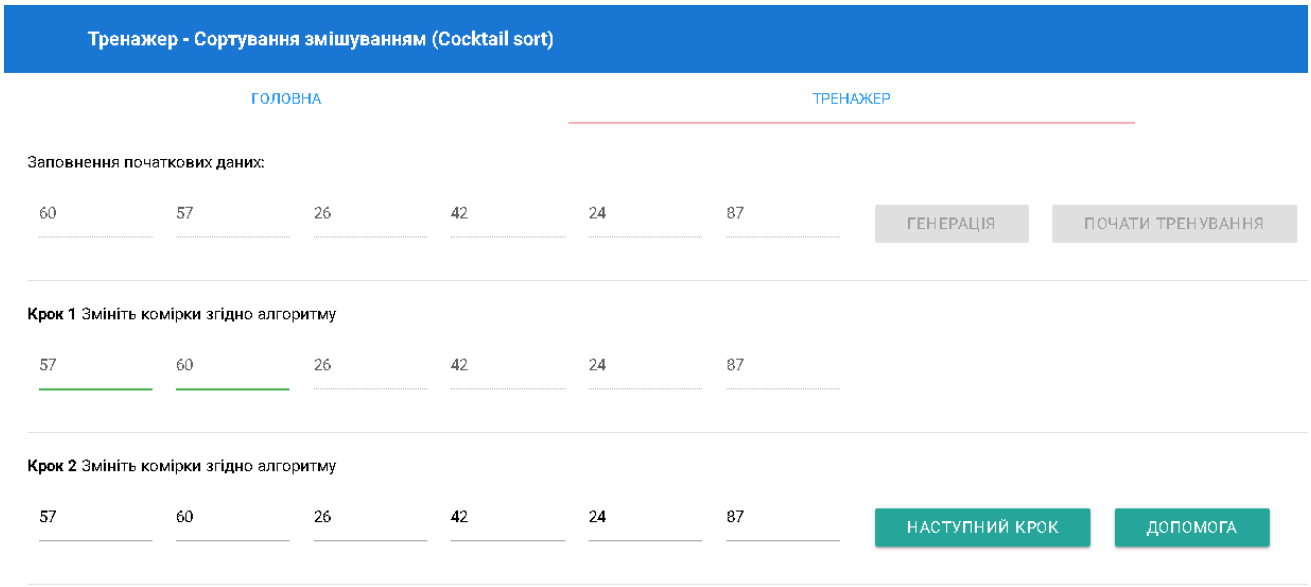

Рис. 2.8 – Після вірної дії на першому кроці

Рис. 2.9 – У випадку натиснення кнопки «Допомога»)

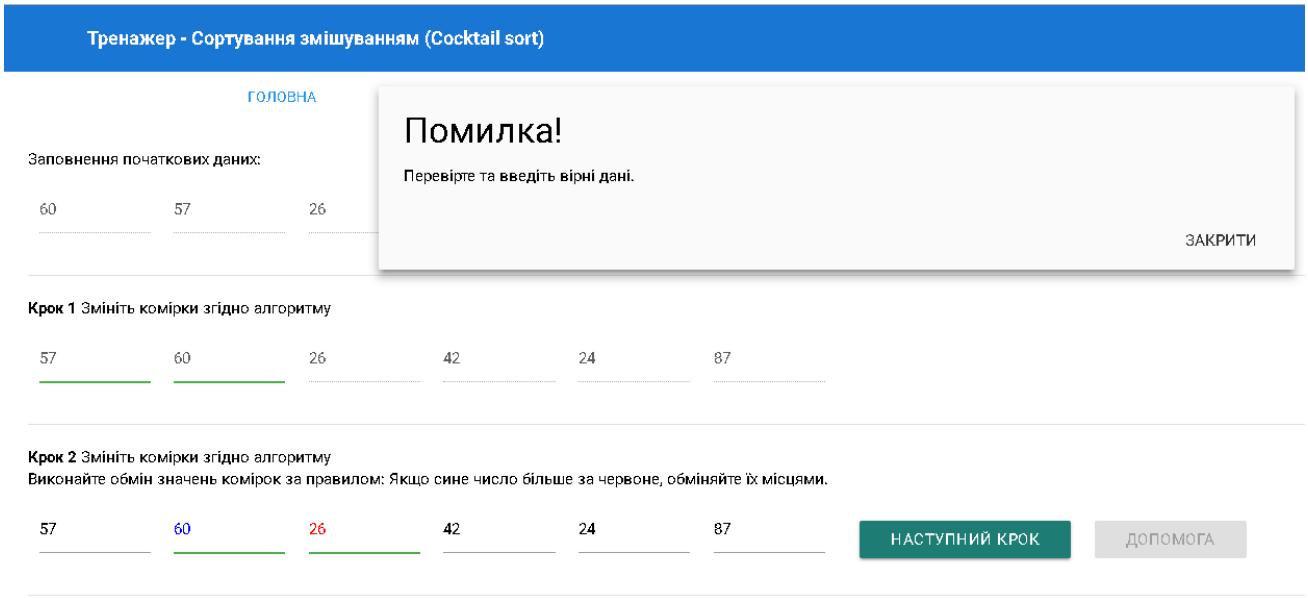

Рис. 2.10 – Після помилкової дії на другому кроці

Рис. 2.11 – Кроки 3-5 прикладу

Слід відмітити, що пара, яка порівнювалась на попередньому кроці підкреслюється зеленою рискою. Крайні елементи, як стали на своє місце і в подальшому сортуванню не приймають участь – блокуються та стають блідо-сірого кольору.

Рис. 2.12 – Кроки 6-8 прикладу

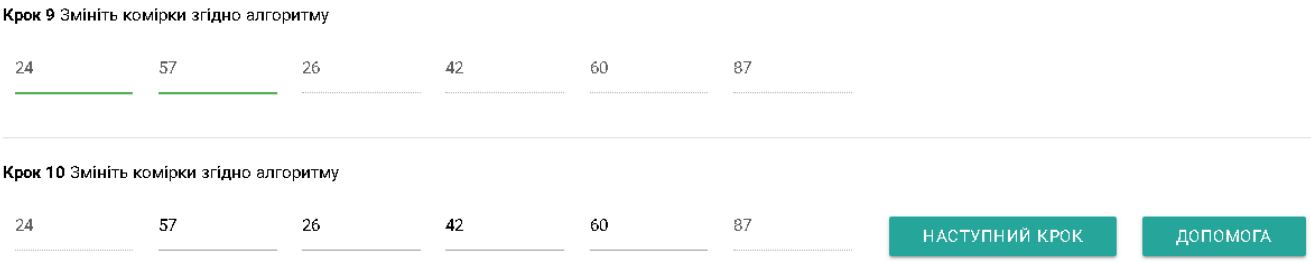

Рис. 2.13 – Кроки 9-10 прикладу

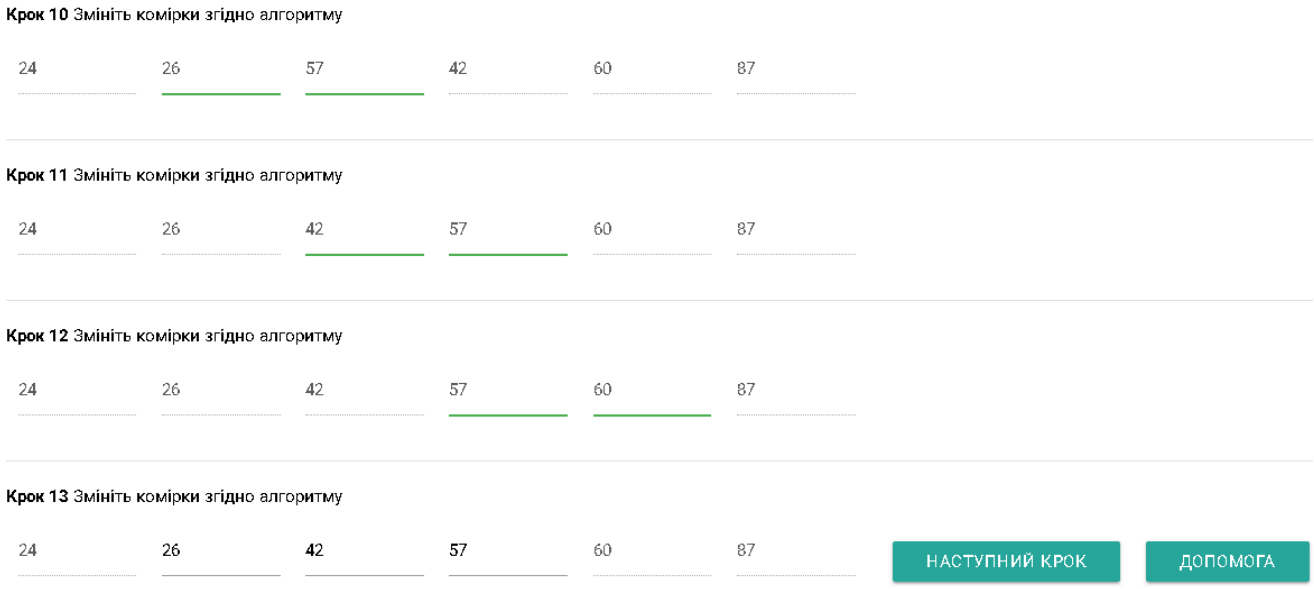

Рис. 2.14 – Кроки 10-13 прикладу

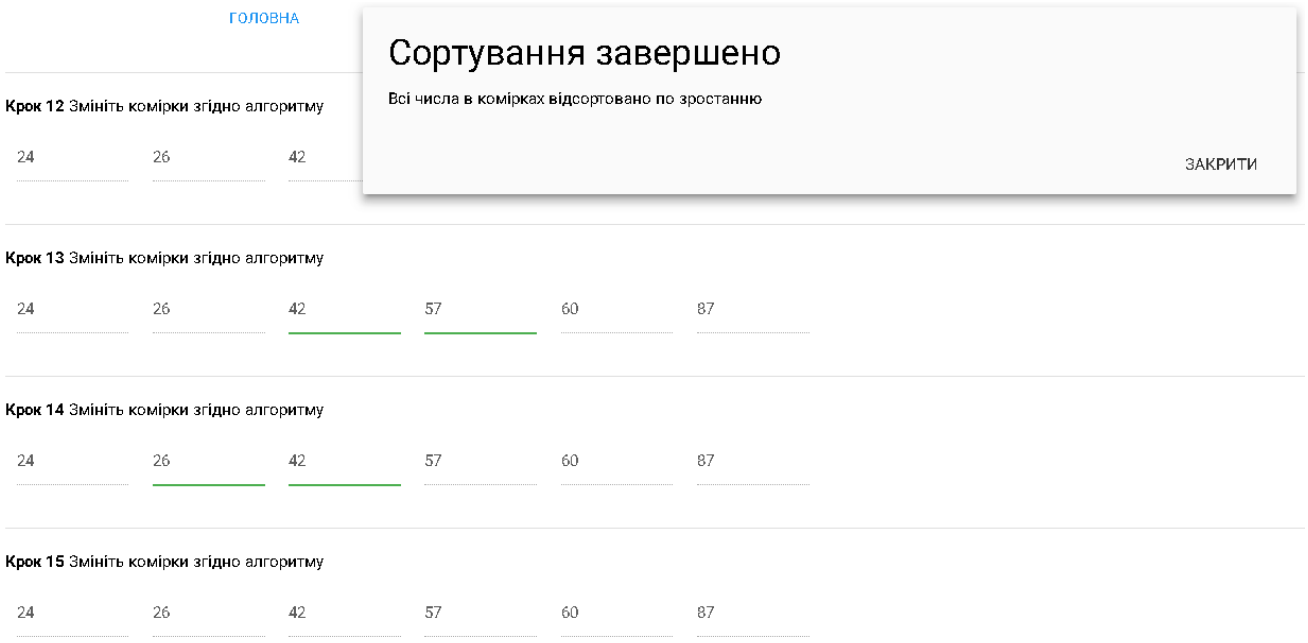

Рис. 2.15 – Кінець прикладу

2. Тренажер з теми «Аналіз алгоритму швидкого сортування» дисципліни «Аналіз алгоритмів».

Програма, створена студентом Русіном Владиславом у 2018 р., і також є веб-сторінкою (рис. 2.16).

Програма дозволяє провести трасування (покрокове виконання) алгоритму швидкого сортування для 4, 5, 6, 7 або 8 чисел, заданих користувачем або випадково згенерованих (рис. 2.17-2.18).

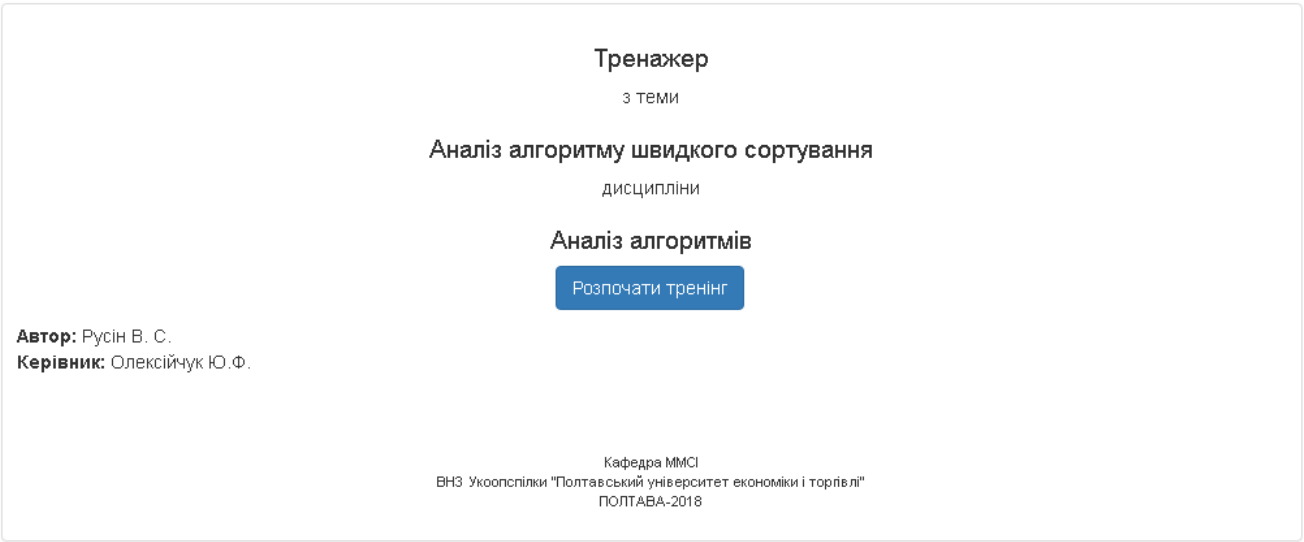

Рис. 2.16 – Тренажер «Аналіз алгоритму швидкого сортування»

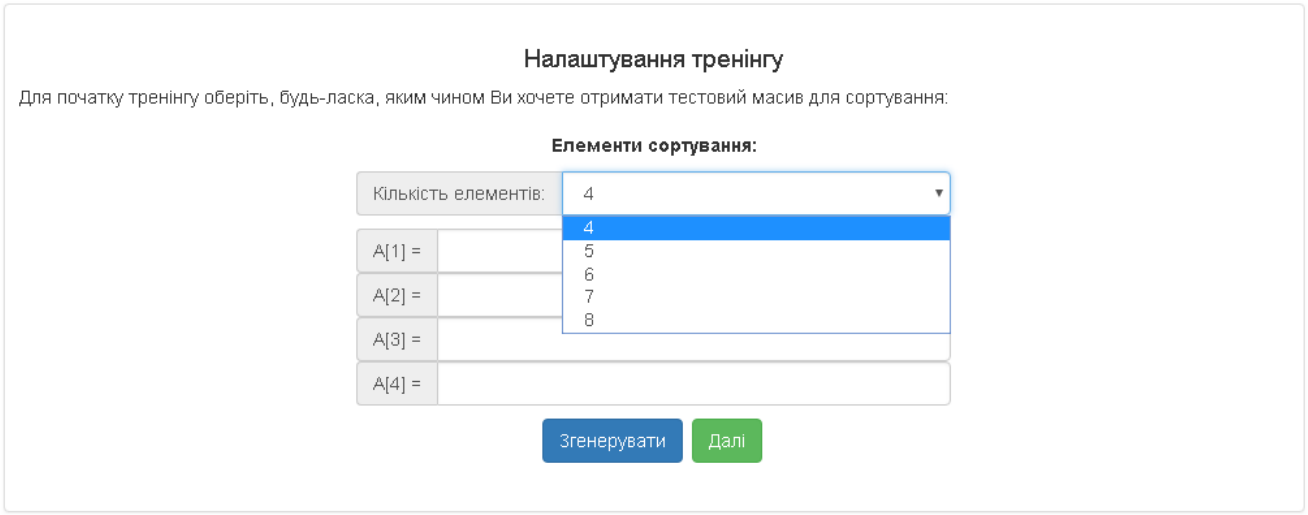

Рис. 2.17 – Задання набору чисел

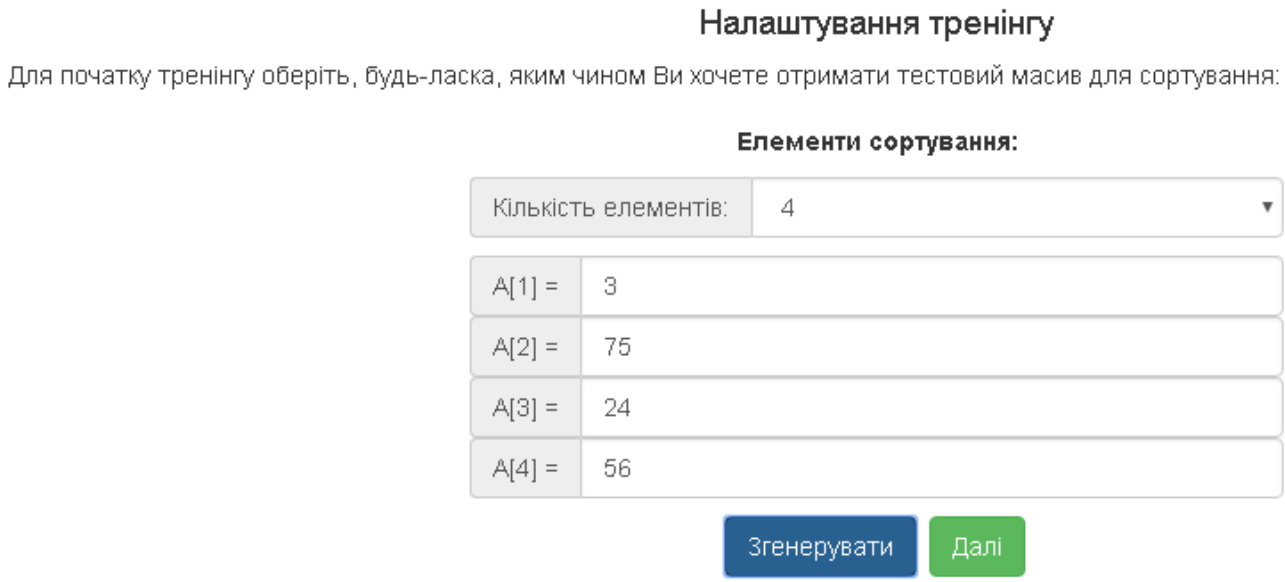

Рис. 2.18 – Згенеровані числа

Після чого з’являється перший крок (рис. 2.19-2.20), в якому у верхній частині вікна видно алгоритм сортування (псевдокод), вхідні дані, поточний вигляд послідовності (рис. 2.19).

А в нижній частині вікна (рис. 2.20) йде питання. При цьому можна вибрати назви підпрограм, ввести номер кроку або натиснути кнопку «Кінець алгоритму». Також видно, скільки всього буде питань (для різних входів. Кількість питань змінюється), скільки було помилок.

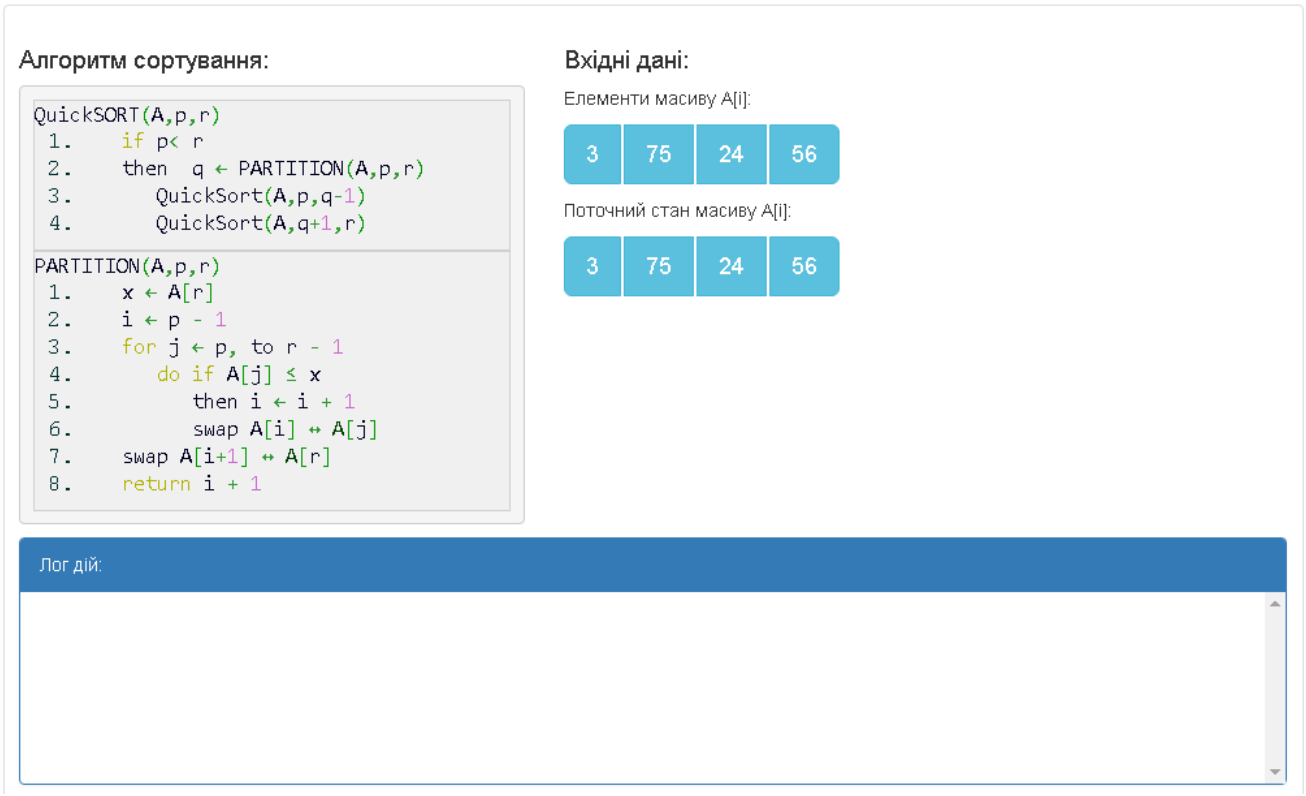

Рис. 2.19 – Перший крок тренажеру (верхня частина вікна)

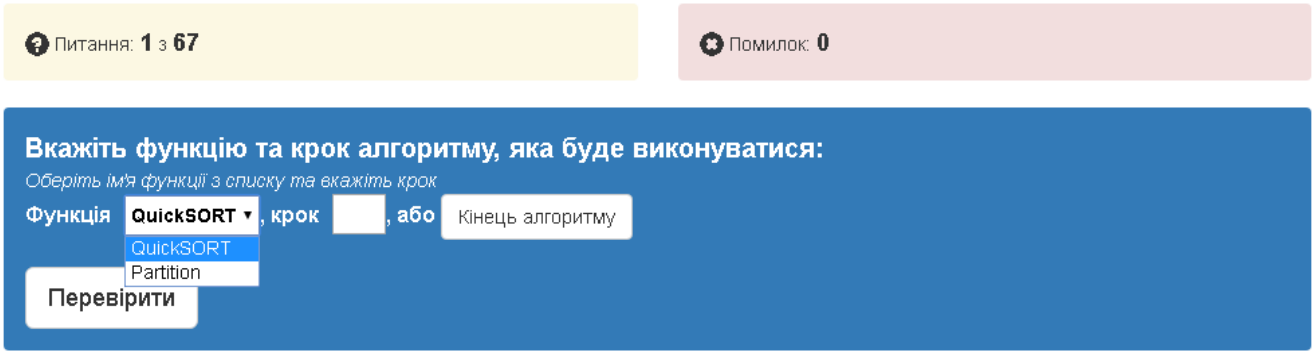

Рис. 2.20 – Перший крок тренажеру (нижня частина вікна)

Після вводу відповіді натискається кнопка «Перевірити». У випадку помилки, з'являється пояснення (рис. 2.21). Похибку слід виправити.

Також тренажер має панель «лог дій» (рис. 2.19, 2.22-2.24), який показує, що в алгоритмі вже відбулось.

В кінці (після відповіді на всі питання) відображається вхідних та вихідний набір чисел; кількість помилок, лог дій. Також є можливість пройти тренінг повторно.

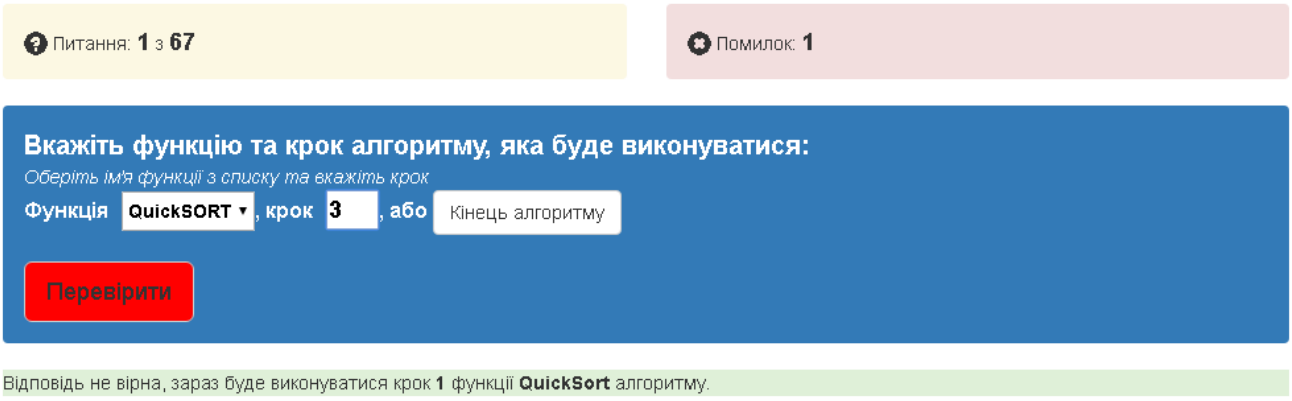

Рис. 2.21 – Вигляд тренажеру при помилці

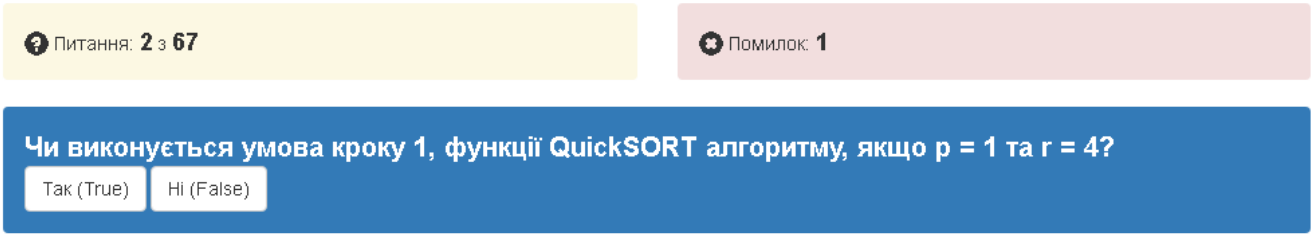

Рис. 2.22 – Другий крок тренажеру (нижня частина вікна)

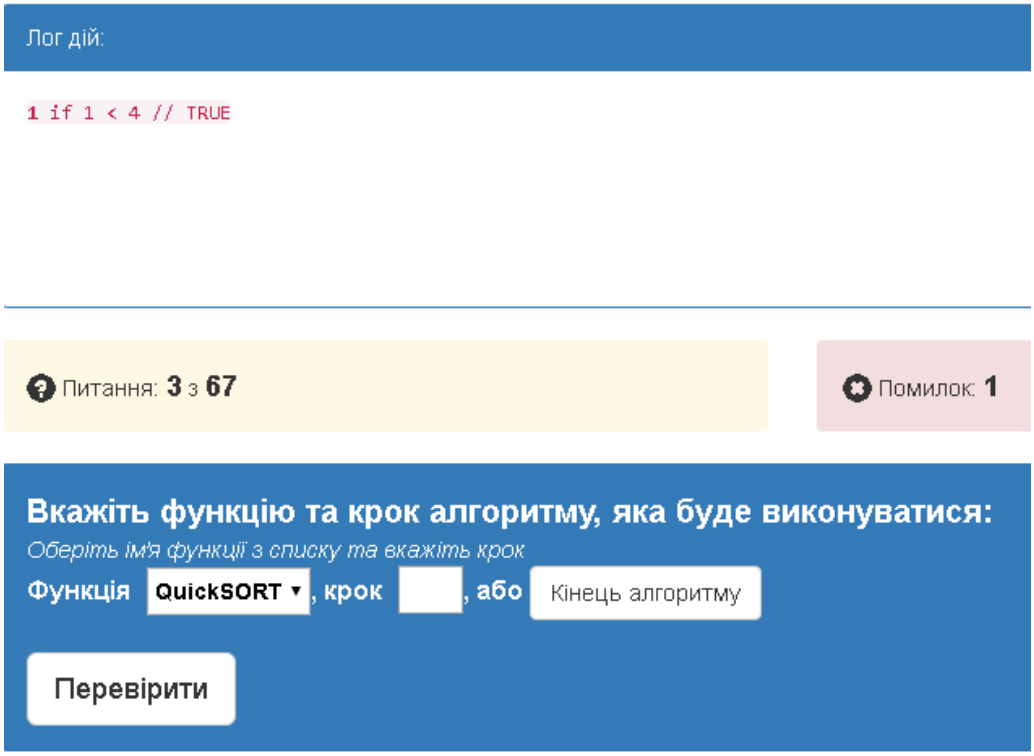

Рис. 2.23 — Третій крок тренажеру (нижня частина вікна)

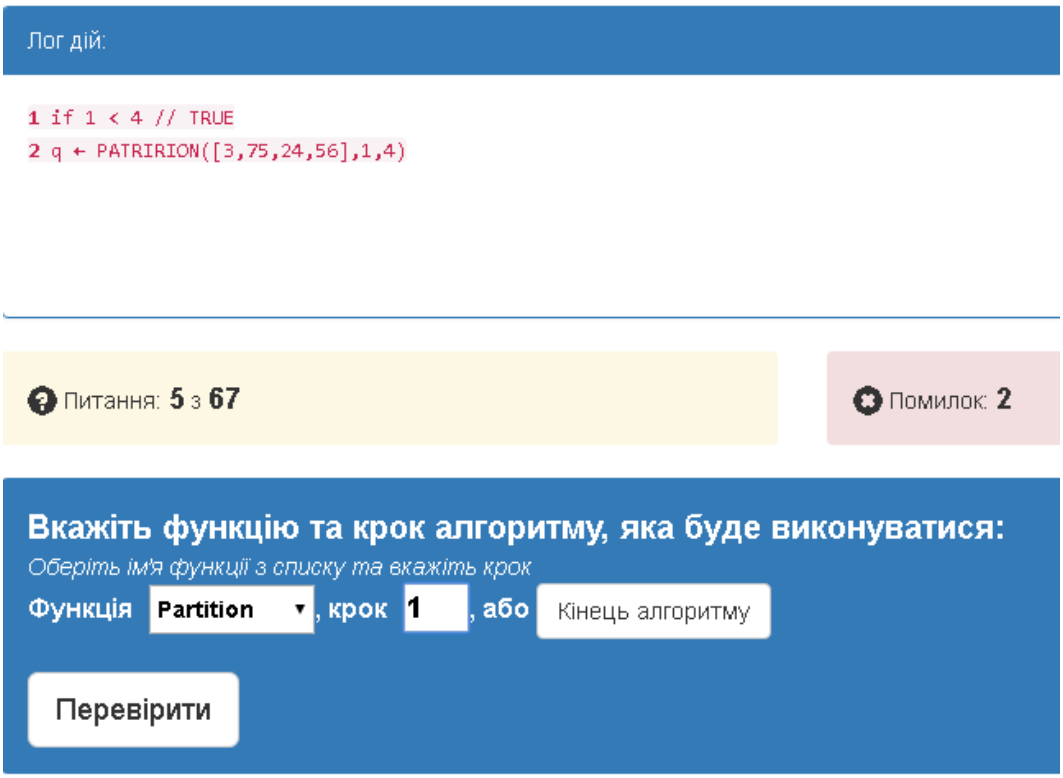

Рис. 2.24 – П’ятий крок тренажеру (нижня частина вікна)

3. Тренажер з теми «Сортування бульбашками» дисципліни «Аналіз алгоритмів».

Програма написана студентом Голубенко Владиславом у 2018 р. і являє собою тренінг із трасування алгоритму [2] (рис. 2.16).

Тренажер містить теоретичну інформацію (рис. 2.26).

Головне вікно (рис. 2.27, 2.29, 2.31, 2.32) містить вхідний масив, алгоритм на псевдокоді, питання, варіанти відповідей або поля для відповідей, кнопку для перевірки, а також вікно з логом дій та вікно, що показує, який вигляд на даному питанні має числова послідовність.

При помилці з’являються пояснення помилок (рис. 2.28, 2.30).

Після проходження тренінгу (рис. 2.32) можна, натиснувши кнопку «Виконати інший приклад», перейти до трасування алгоритму для іншої числової послідовності.

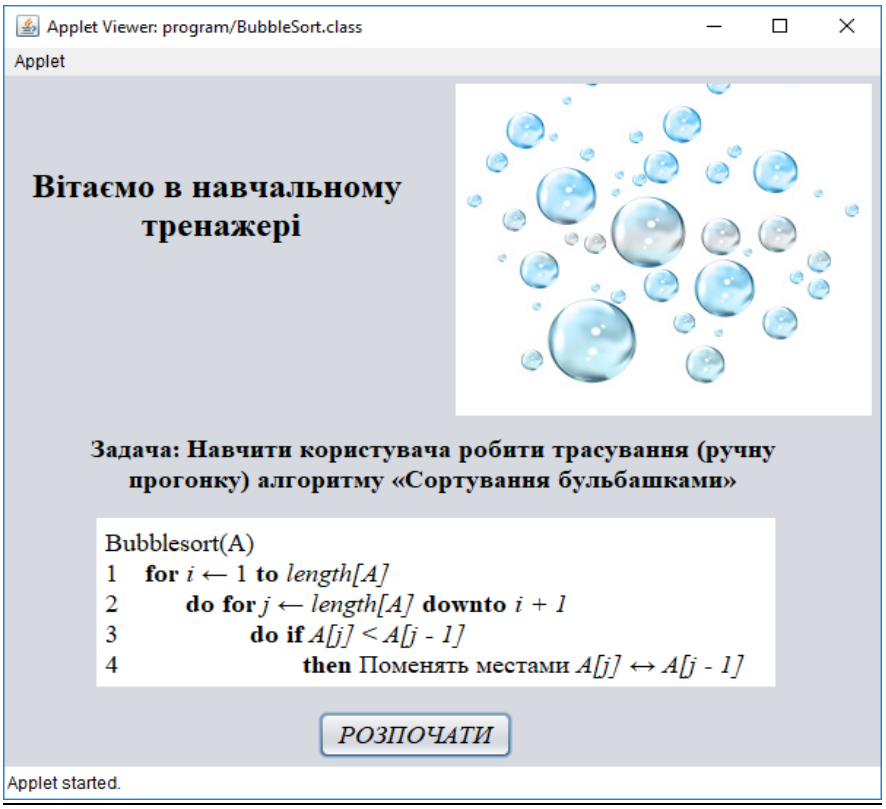

Рис. 2.25 – Тренажер «Сортування бульбашками»

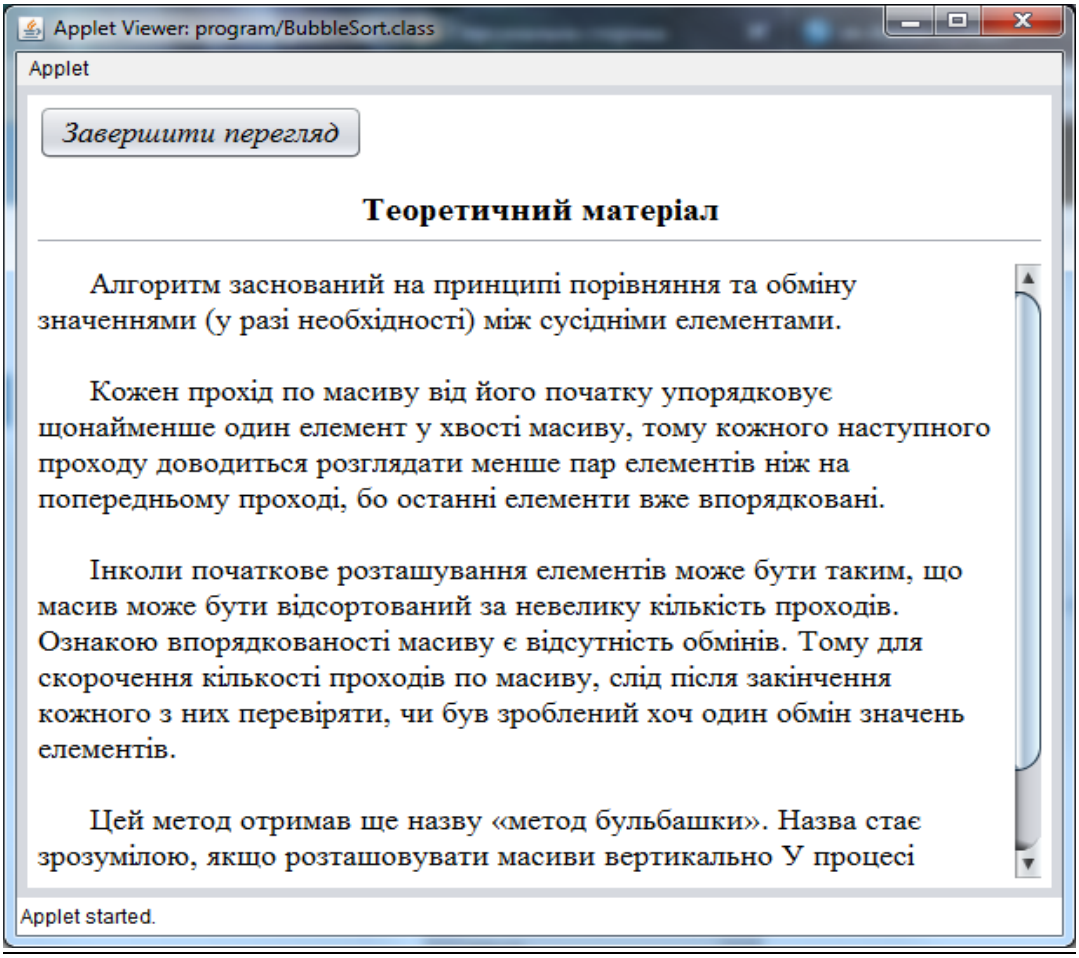

Рис. 2.26 – Тренажер «Сортування бульбашками», сторінка «Теорія»

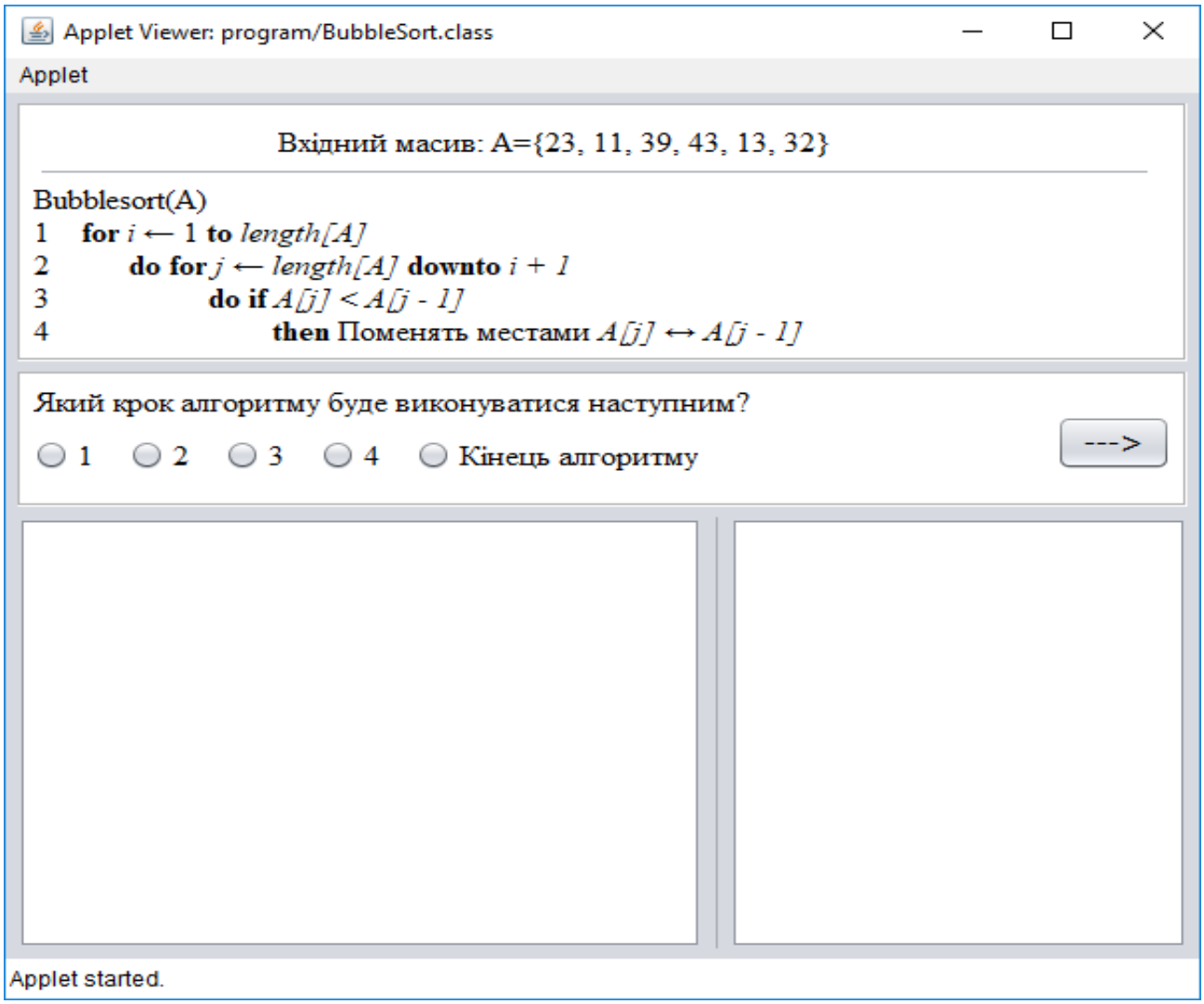

Рис. 2.27 – Тренажер «Сортування бульбашками», перший крок

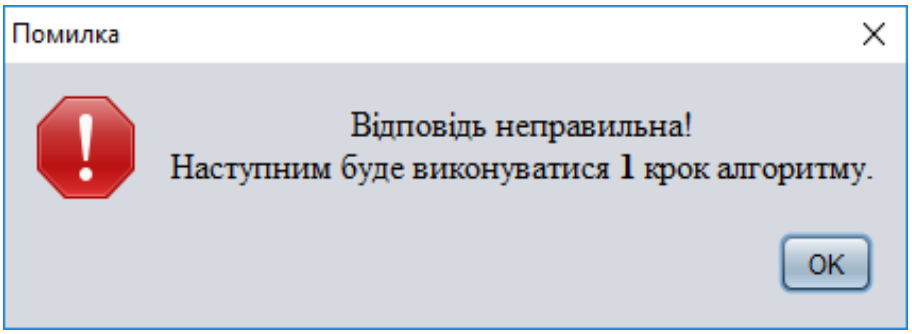

Рис. 2.28 – Пояснення помилки на першому кроці

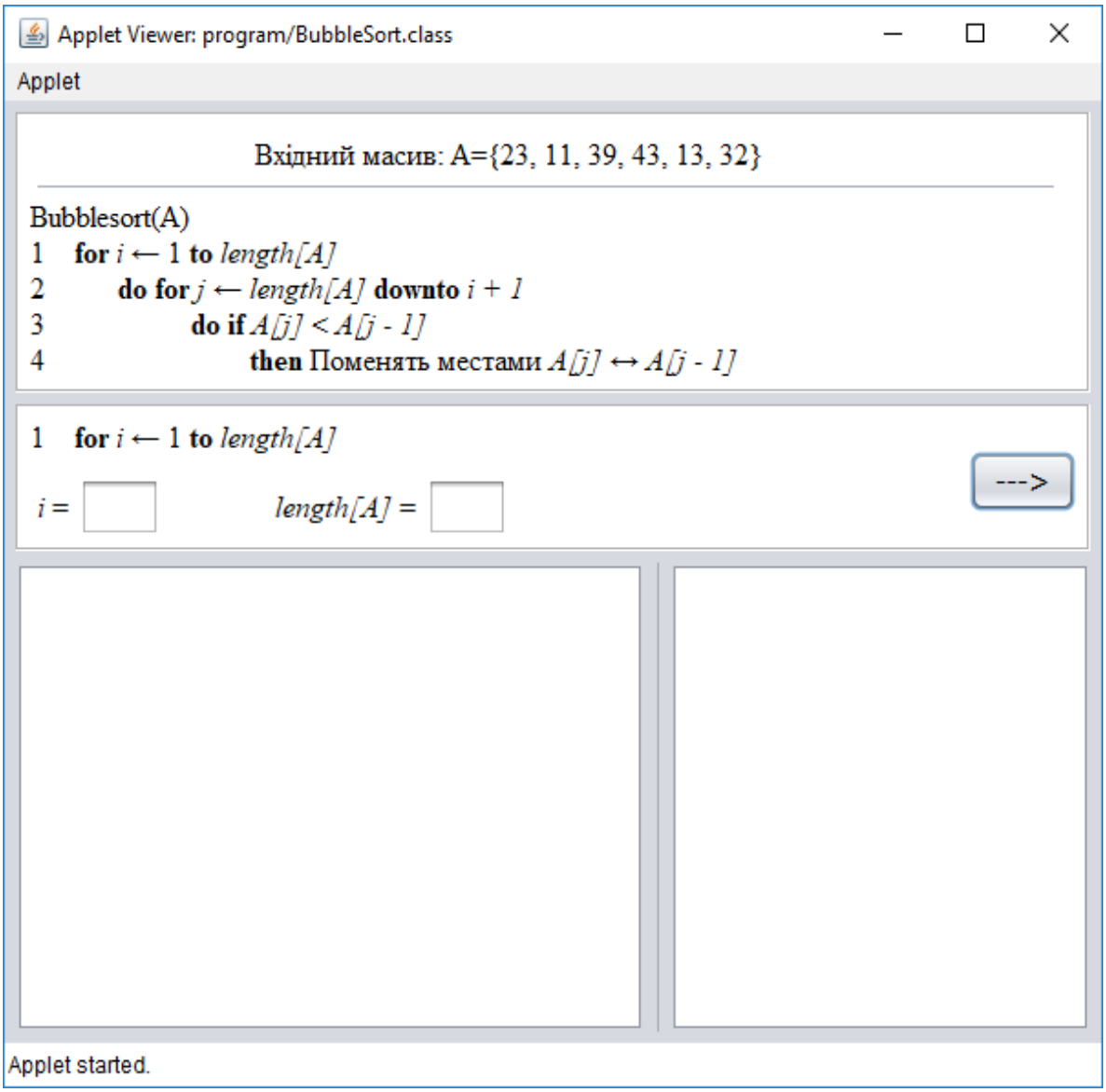

Рис. 2.29 – Тренажер «Сортування бульбашками», другий крок

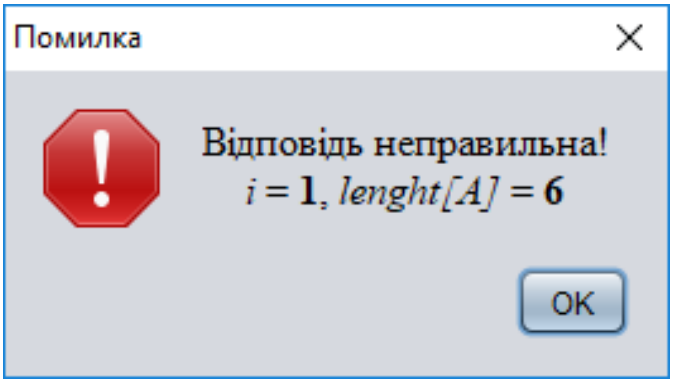

Рис. 2.30 – Пояснення помилки на другому кроці

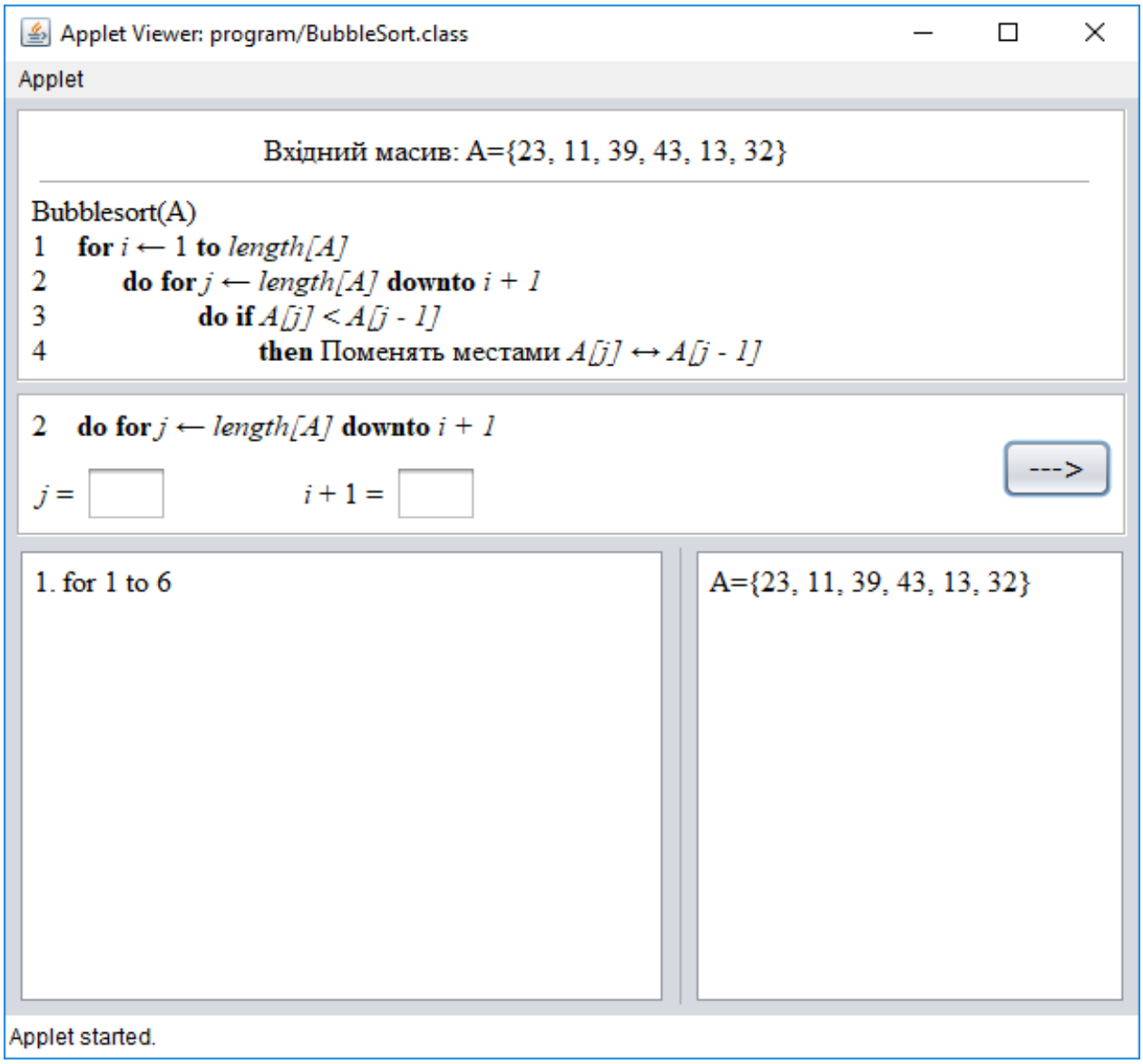

Рис. 2.31 – Тренажер «Сортування бульбашками», третій крок

2.2. Позитивні аспекти оглянутих тренажерів

Перерахуємо позитивні моменти розглянутих тренажерів.

Тренажер з теми «Шейкер-сортування»:

а) форма веб-сторінки зручна для впровадження та використання;

б) тренажер гарно структурований;

в) наявність теорії та коду сортування;

г) можна перевірити роботу сортування на будь-яких 5 числах;

д) за рахунок випадкової генерації утворюється безліч різних прикладів;

е) простий та зрозумілий інтерфейс тренажеру;

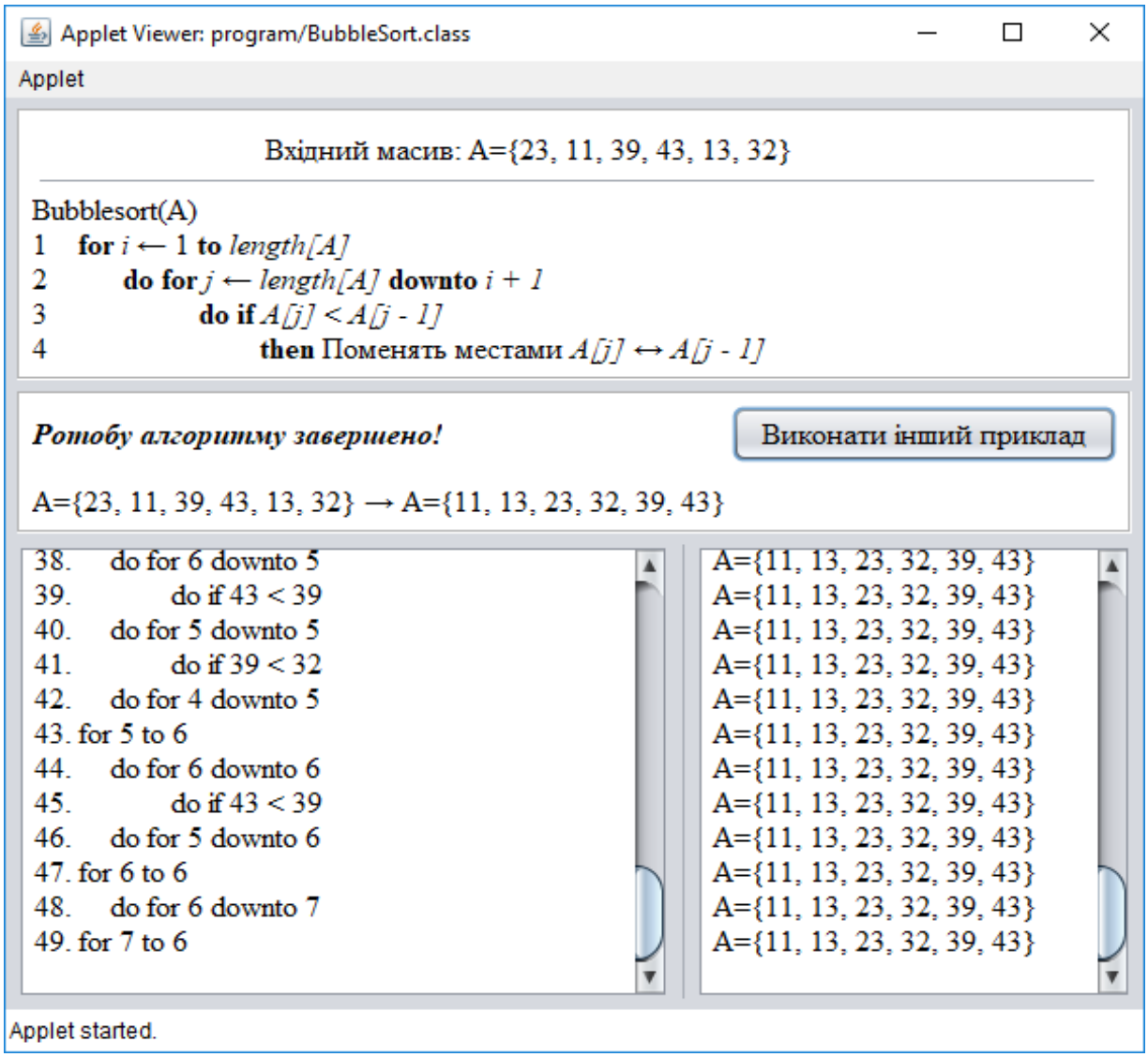

Рис. 2.32 – Тренажер «Сортування бульбашками», кінець

є) використання візуальних прийомів для покращення сприйняття (синій та червоний колір цифр, що порівнюються, при підказці; підкреслення пари, що порівнювалась на попередньому кроці; блідо-сірий колір та блокування чисел, що вже не використовуються);

ж) тренажер можна використовувати як візуалізатор (для викладачів, зокрема) для пояснення, як працює сортування;

з) наявність інформації про розробника.

Тренажер з теми «Аналіз алгоритму швидкого сортування»:

а) форма веб-сторінки зручна для впровадження та використання;

б) можна перевірити роботу сортування на будь-яких 4, 5, 6, 7, 8 числах;

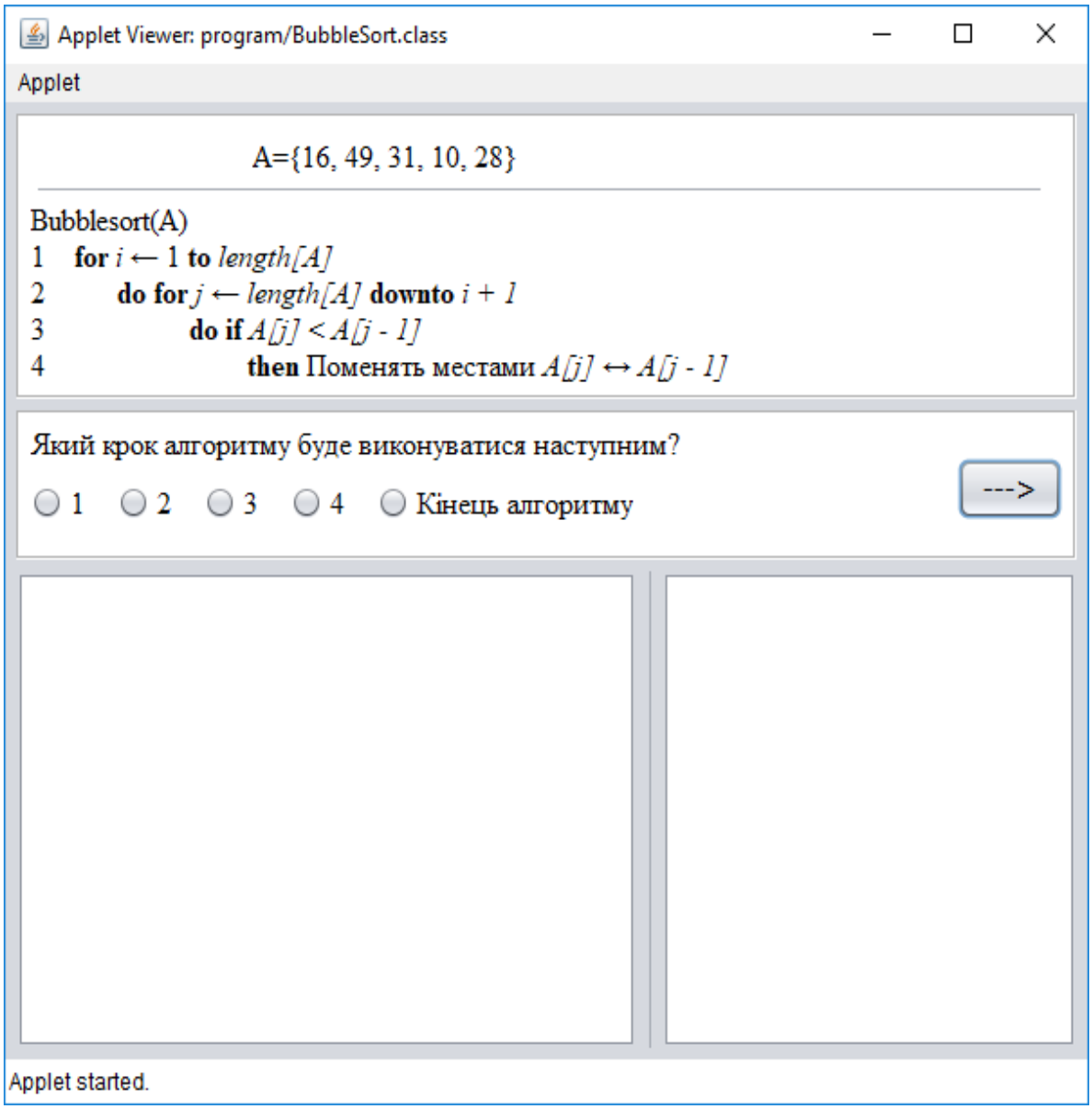

Рис. 2.33 – Тренажер «Сортування бульбашками», інший приклад

в) за рахунок випадкової генерації утворюється безліч різних прикладів;

г) зрозумілий інтерфейс тренажеру;

д) наявність підрахунку помилок;

е) наявність логу дій дає можливість краще зрозуміти алгоритм, провести його аналіз;

є) тренажер можна використовувати як візуалізатор трасування алгоритму швидкого сортування;

ж) тренажер можна використовувати для проведення самостійних та контрольних робіт, заліків, іспитів, надавши студентам різні вхідні данні (оскільки є індивідуальні завдання та підрахунок помилок);

з) наявність інформації про розробника.

Тренажер з теми «Сортування бульбашками

а) можна перевірити роботу алгоритмі на різних прикладах;

б) компактний інтерфейс;

в) наявність логу дій дає можливість краще зрозуміти алгоритм, провести його аналіз;

г) наявність вікна, в якому показано порядок чисел в послідовності на кожному кроці.

2.3. Вади оглянутих тренажерів

Тренажер з теми «Шейкер-сортування»:

вад не виявлено.

Тренажер з теми «Аналіз алгоритму швидкого сортування»:

а) з псевдокоду не зразу зрозуміло, що означають аргументи p, r,

б) не зберігаються явно значення p, r, які були згадані на попередніх кроках, можна згадати їх значення лише з логу дій.

Тренажер з теми «Сортування бульбашками:

а) псевдокод з використанням російської мови;

б) є орфографічні помилки.

2.4. Необхідність та актуальність теми

Після проведення огляду видно, що гарно розроблений тренажер дає можливість краще зрозуміти тему. Це особливо актуально для тем з алгоритмізації, оскільки, даний матеріал є абстрактним, а отже без наочності погано зрозумілим.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Зовнішнє сортування

Історично зовнішнє сортування використовувалося для впорядкування даних, що розміщені у файлах на диску [14-15]. Таким чином, по-перше, відсутні обмеження на використання додаткової пам'яті для збереження проміжних результатів, оскільки ємність диска істотно перевищує ємність оперативної пам'яті, і, по-друге, доступ до даних здійснюється послідовно (хоча у мовах програмування є засоби підтримки прямого доступу до інформації у файлах, основний вид доступу до інформації у файлах – послідовний).

Отже, методи зовнішнього сортування можна використовувати і для даних, розміщених в оперативній пам'яті, якщо ємнісна трудомісткість методу не істотна і відповідна структура даних допускає послідовний доступ (масив, список тощо). Усі методи зовнішнього сортування використовують алгоритм злиття послідовностей [14-15].

3.1.1. Злиття впорядкованих послідовностей

Нехай є дві впорядковані послідовності: $a_1, ..., a_n$ $(a_1 \le a_2 \le ... \le a_n)$; $b_1, ..., b_m$ $(b_1 \le b_2 \le ... \le b_m)$ [14-15].

Створимо з них нову впорядковану послідовність $c_1, ..., c_{n+m}$ довжини $n + m$.

Алгоритм побудови результуючої послідовності [14-15] полягає ось у чому: на кожному кроці порівнюються поточні елементи першої і другої послідовностей. При цьому в тій послідовності, звідки був „узятий" менший елемент, поточним стає наступний елемент. Якщо наступного немає, тобто одна з вихідних послідовностей закінчилася, залишок другої (від поточного елемента до кінця) приписується у кінець результуючої послідовності. Наприклад, (рис. 3.1-3.2):

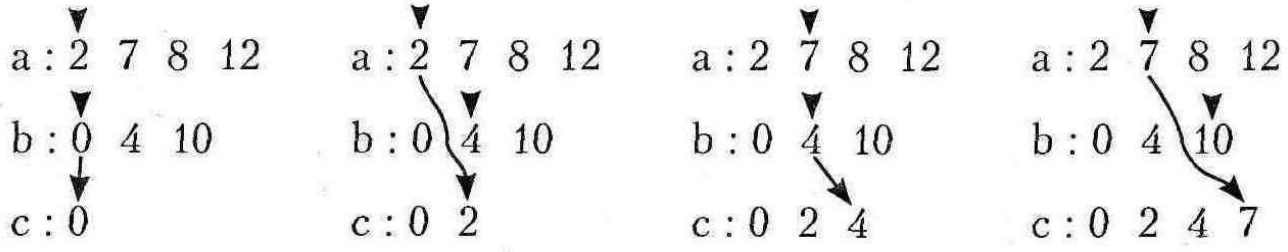

Рис. 3.1 – Приклад злиття впорядкованих послідовностей

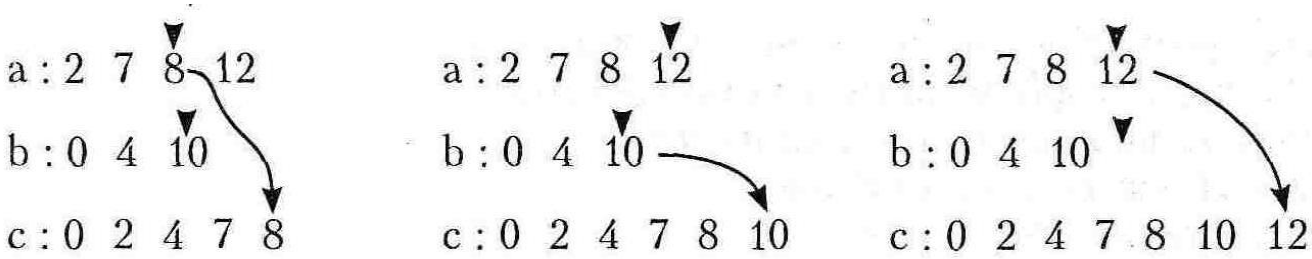

Рис. 3.2 – Продовження прикладу злиття впорядкованих послідовностей

Цей алгоритм можна записати у вигляді такої блок-схеми (рис. 3.3).

Позначимо i – поточний елемент першої послідовності;

j – поточний елемент другої послідовності;

k – поточний елемент результуючої послідовності.

Дано $a_1, ..., a_n$; $b_1, ..., b_m$ – упорядковані послідовності.

Знайдено $c_1, ..., c_{n+m}$ – упорядкована послідовність.

3.1.2. Сортування фон Неймана

Нехай є послідовність елементів $a = \{a_1, ..., a_n\}$. Необхідно відсортувати її в порядку зростання (не убування). Ідея алгоритму фон Неймана полягає ось в чому [14-15].

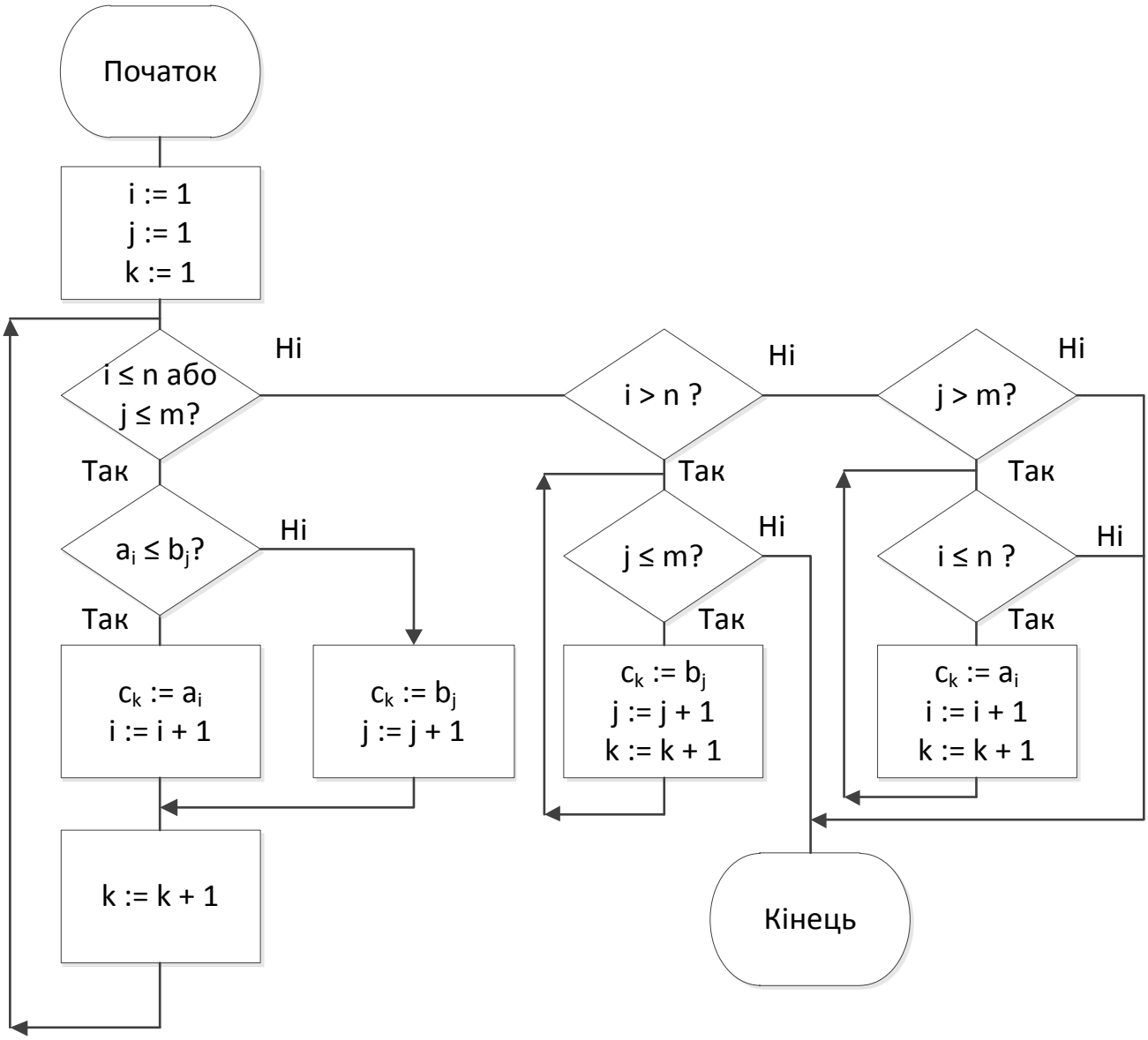

Рис. 3.3 – Блок-схема алгоритму злиття впорядкованих послідовностей

Вихідна послідовність ділиться на дві частини b і c. Послідовності b і c об’єднуються в одну за допомогою злиття елементів у впорядковані пари (крім, можливо, останньої, якщо n непарне). Нова послідовність a знову ділиться на дві частини b і c, що складаються вже з упорядкованих пар. У процесі об’єднання b і c в нову послідовність a пари зливаються в упорядковані четвірки і т.д. Наприклад, (рис. 3.4):

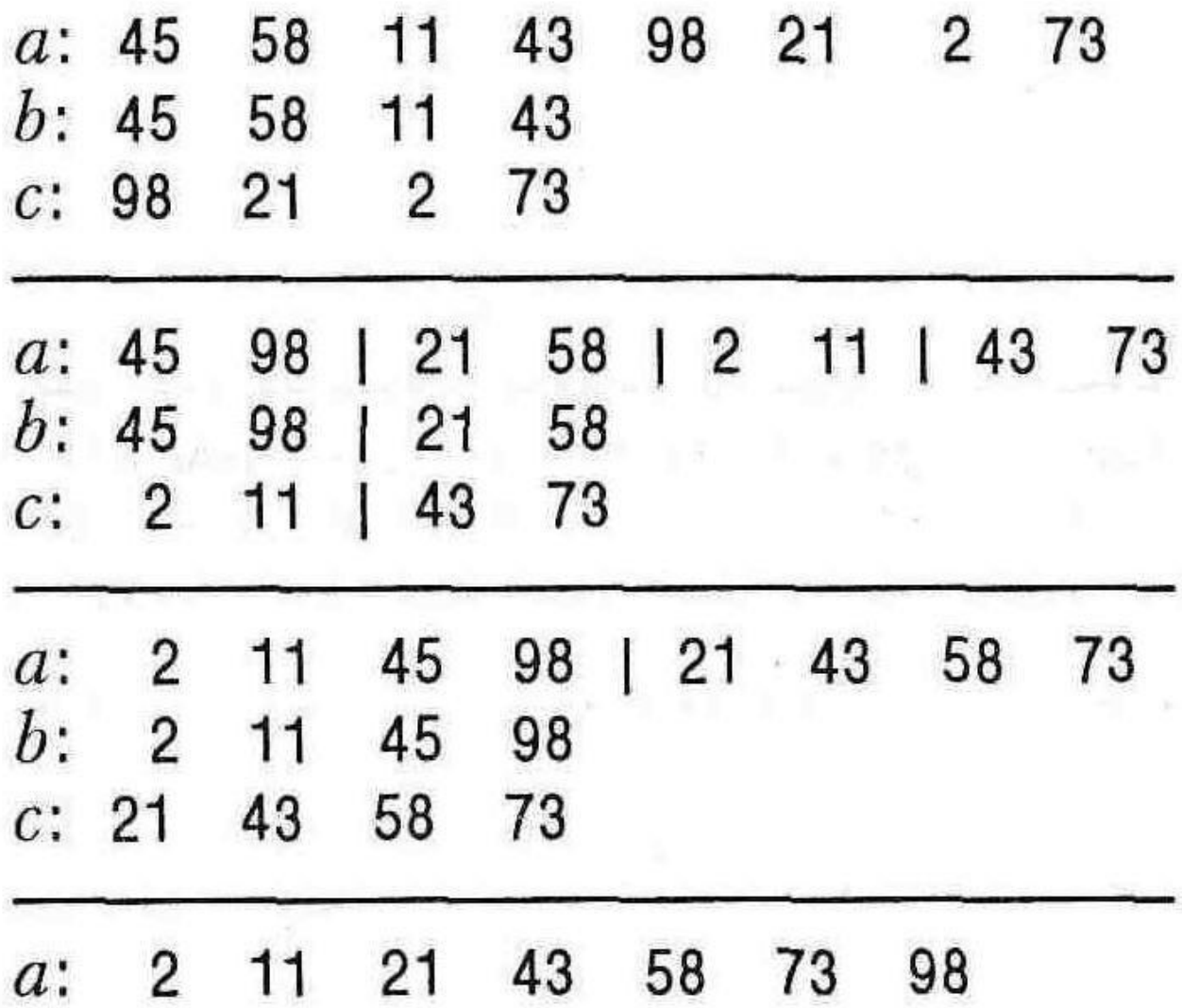

Рис. 3.4 – Приклад сортування фон Неймана

Таким чином, алгоритм фон Неймана складається з двох фаз розділення і злиття. При цьому використовуються дві допоміжні послідовності (стрічки). Тому цей алгоритм часто називають ***двофазним тристрічковим злиттям***. Фази розділення злиття можна об'єднати, якщо виконувати одразу в процесі злиття. Тоді буде потрібна ще одна допоміжна послідовність. Відповідний метод отримав назву ***однофазного чотири стрічкового злиття***.

Оцінімо трудомісткість методу фон Неймена. Оскільки він призначений для сортування даних на зовнішньому носії, оцінювати слід кількість пересилань елементів: час обміну інформацією із зовнішньою будовою істотно перевищує час

порівняння двох елементів. Кількість проходів (фаз) дорівнює $[\log_2 n]$, оскільки на кожному проході довжина впорядкованої послідовності подвоюється. При цьому відбувається копіювання всіх n елементів у послідовності. Отже, загальне число пересилок $M = n[\log_2 n]$.

3.2. Алгоритм тренажеру

3.2.1. Приклад 1 (кількість елементів – парне число).

На екрані умова прикладу: є послідовність чисел *a={45, 58, 11, 43, 98, 21, 2, 73}*. Відсортувати послідовність за методом фон Неймана.

Умова завдання видима протягом роботи всього тренажеру.

Крок 1. *Поділіть послідовність a навпіл так, щоб перша половина послідовності a утворила послідовність b, а друга половина послідовності a утворила послідовність c.*

Введіть необхідні числа для послідовностей b та c.

Правильна відповідь: *b: 45 58 11 43; c: 98 21 2 73.*

Спочатку перевіряється послідовність *b*. У випадку помилки з'являється повідомлення: «Помилка! Послідовність *b* – це перша половина послідовності *a*, тобто числа *45 58 11 43.*». Користувачу потрібно виправити помилки.

Якщо послідовність *b* зазначена вірно, то з'являється повідомлення: «Послідовність *b* правильна!». Після цього здійснюється перевірка послідовності *c*.

У випадку помилки для послідовності *c* з'являється повідомлення: «Не вірно! Послідовність *c* – це друга половина послідовності *a*, тобто числа *98 21 2 73.*». Користувачу необхідно виправити помилки.

Якщо послідовність *c* зазначена вірно, то з'являється повідомлення: «Послідовність *c* правильна!». Після цього відбувається перехід на крок 2.

Крок 2. *Об'єднайте послідовності b={45, 58, 11, 43} та c={98, 21, 2, 73} в одну. При цьому утворіть* ***впорядковані пари*** *елементів. Перша пара – перший елемент послідовності b та перший елемент послідовності c, які записуються за зростанням; друга пара – другий елемент послідовності b та другий елемент послідовності c, які записуються за зростанням, і т.д.*

Введіть необхідні числа для об'єднаної послідовності.

Правильна відповідь: *a: | 45 98 | 21 58 | 2 11 | 43 73 |*.

Перевірка здійснюється спочатку для першої пари. У випадку помилки, з'являється повідомлення: «Помилка! Слід взяти перший елемент послідовності *b* – це *45*, та перший елемент послідовності *c* – це *98*, та порівняти їх. Оскільки *45<98,* то перша впорядкована пара – це | *45 98* |.». Користувачу слід виправити помилки. Якщо помилки в першій парі чисел немає, то з'являється повідомлення: «Перша пара вірна!» та здійснюється перевірка другої пари.

Перевірка другої пари: при помилці з'являється повідомлення: «Не вірно! Слід взяти другий елемент послідовності *b* – це *58,* та другий елемент послідовності *c* – це *21*, та порівняти їх. Оскільки *58>21*, то друга впорядкована пара – це | *21 58* |.». Користувачу необхідно виправити похибки. Якщо помилки в другій парі чисел немає, то з'являється повідомлення: «Друга пара вірна!» та здійснюється перевірка третьої пари.

Перевірка третьої пари: при невірній відповіді з'являється повідомлення: «Хибна відповідь! Слід взяти третій елемент послідовності *b* – це *11*, та третій елемент послідовності *c* – це *2*, та порівняти їх. Оскільки *11>2*, то третя впорядкована пара – це | *2 11* |.». Користувачу необхідно виправити похибки. Якщо помилки в третій парі чисел немає, то з'являється повідомлення: «Третя пара вірна!» та здійснюється перевірка четвертої пари.

Перевірка четвертої пари: при невірній відповіді з'являється повідомлення: «Похибка! Слід взяти четвертий елемент послідовності *b* – це *43*, та четвертий елемент послідовності *c* – це *73*, та порівняти їх. Оскільки *43<73*, то четверта впорядкована пара – це | *43 73* |.». Користувачу необхідно виправити помилки.

Якщо помилки в четвертій парі чисел немає, то з'являється повідомлення: «Четверта пара вірна!» та здійснюється перехід на крок 3.

Крок 3. *Поділіть послідовність a: | 45 98 | 21 58 | 2 11 | 43 73 | навпіл так, щоб перша половина послідовності a утворила послідовність b, а друга половина послідовності a утворила послідовність c. Впорядкованість пар при цьому зберігається.*

Введіть необхідні числа для послідовностей b та c.

Правильна відповідь: *b: | 45 98 | 21 58 |; c: | 2 11 | 43 73 |*.

Спочатку перевіряється послідовність *b*, у випадку помилки з'являється повідомлення: «Хибна відповідь! Послідовність *b* – це перша половина послідовності *a*, тобто впорядковані пари *| 45 98 | 21 58 |*.». Користувачу потрібно виправити помилки. У випадку вірності з'являється повідомлення: «Послідовність *b* вірна!».

Потім перевіряється послідовність *c*, у випадку помилки з'являється повідомлення: «Помилкова відповідь! Послідовність *c* – це друга половина послідовності *a*, тобто впорядковані пари *| 2 11 | 43 73 |*.». Користувачу слід виправити помилки. У випадку правильності з'являється повідомлення: «Послідовність *c* вірна!» та відбувається перехід на крок 4.

Крок 4. *Об'єднайте послідовності b: | 45 98 | 21 58 | та c: | 2 11 | 43 73 | в одну. При цьому утворіть* ***впорядковані четвірки*** *елементів. Перша четвірка – перша пара послідовності b та перша пара послідовності c, елементи яких записуються за зростанням, і т.д.*

Введіть необхідні числа для об'єднаної послідовності.

Правильна відповідь: *a: | 2 11 45 98 | 21 43 58 73 |*.

Перевірка здійснюється спочатку для першої четвірки. У випадку помилки, з'являється повідомлення: «Помилка! Слід розглянути першу пару послідовності *b* – це *| 45 98 |* та першу пару послідовності *c* – це *| 2 11 |*. Впорядкована четвірка з цих

елементів – це | *2 11 45 98* |.». Користувачу необхідно виправити помилки. У випадку правильної відповіді з'являється повідомлення «Перша четвірка вірна!».

Якщо помилки в першій четвірці чисел немає, то здійснюється перевірка другої четвірки. У випадку помилки, з'являється повідомлення: «Помилка! Слід розглянути другу пару послідовності *b* – це | *21 58* | та другу пару послідовності *c* – це | *43 73* |. Впорядкована четвірка з цих елементів – це | *21 43 58 73* |.» Користувачу необхідно виправити помилки. У випадку правильної відповіді з'являється повідомлення «Друга четвірка вірна!» та відбувається перехід на крок 5.

Крок 5. *Поділіть послідовність a: | 2 11 45 98 | 21 43 58 73 | навпіл так, щоб перша половина послідовності a утворила послідовність b, а друга половина послідовності a утворила послідовність c. Впорядкованість четвірок при цьому зберігається.*

Введіть необхідні числа для послідовностей b та c.

Правильна відповідь: *b: | 2 11 45 98 |; c: | 21 43 58 73 |.*

Спочатку перевіряється послідовність *b*, у випадку помилки з'являється повідомлення: «Неправильно! Послідовність *b* – це перша половина послідовності *a*, тобто впорядкована четвірка | *2 11 45 98* |.». Користувачу потрібно виправити помилки. У випадку правильної відповіді з'являється повідомлення «Послідовність *b* вірна!».

Потім перевіряється послідовність *c*, у випадку помилки з'являється повідомлення: «Не вірна відповідь! Послідовність *c* – це друга половина послідовності *a*, тобто впорядкована четвірка | *21 43 58 73* |.». Користувачу слід виправити помилки. У випадку правильності з'являється повідомлення: «Послідовність *c* вірна!» та відбувається перехід на крок 6.

Крок 6. *Об'єднайте послідовності b: | 2 11 45 98 | та c: | 21 43 58 73 | в одну. При цьому утворіть* ***впорядковану вісімку*** *елементів. Тобто береться впорядкована четвірка послідовності b та впорядкована четвірка послідовності c, елементи яких записуються за зростанням.*

Введіть необхідні числа для об'єднаної послідовності.

Правильна відповідь: *a: | 2 11 21 43 45 58 73 98 |.*

У випадку помилки з'являється повідомлення «Помилка! Впорядкована вісімка з четвірок | *2 11 45 98* | та | *21 43 58 73* | – це | *2 11 21 43 45 58 73 98* |.».

У випадку правильної відповіді з'являється повідомлення «Впорядкована вісімка правильна!».

Далі виводиться повідомлення «Послідовність з парною кількістю елементів відсортована! Робота з тренажером завершена!».

3.2.2. Приклад 2 (кількість елементів – непарне число).

На екрані умова прикладу: є послідовність чисел *a={5, 69, 90, 32, 96, 75, 77, 89, 95}*. Відсортувати послідовність за методом фон Неймана.

Умова завдання видима протягом роботи всього тренажеру.

Крок 1. *Поділіть послідовність a приблизно навпіл так, щоб перша частина послідовності a утворила послідовність b, а друга частина послідовності a утворила послідовність c.*

Оскільки чисел 9, то в послідовності a буде 4 числа, а в b – 5 чисел.

Введіть необхідні числа для послідовностей b та c.

Правильна відповідь: *b: 5 69 90 32; c: 96 75 77 89 95.*

Спочатку перевіряється послідовність *b*. У випадку помилки з'являється повідомлення: «Помилка! Послідовність *b* – це перша частина послідовності *a*, тобто числа *5 69 90 32*.». Користувачу потрібно виправити помилки.

Якщо послідовність *b* зазначена вірно, то з'являється повідомлення: «Послідовність *b* правильна!». Після цього здійснюється перевірка послідовності *c*.

У випадку помилки для послідовності *c* з'являється повідомлення: «Не вірно! Послідовність *c* – це друга частина послідовності *a*, тобто числа *96 75 77 89 95*.». Користувачу необхідно виправити помилки.

Якщо послідовність *c* зазначена вірно, то з'являється повідомлення: «Послідовність *c* правильна!». Після цього відбувається перехід на крок 2.

Крок 2. *Об'єднайте послідовності b={5, 69, 90, 32} та c={96, 75, 77, 89, 95} в одну. При цьому утворіть* ***впорядковані пари*** *елементів та* ***впорядковану трійку*** *(останню). Перша пара – перший елемент послідовності b та перший елемент послідовності c, які записуються за зростанням; друга пара – другий елемент послідовності b та другий елемент послідовності c, які записуються за зростанням, і т.д.*

Введіть необхідні числа для об'єднаної послідовності.

Правильна відповідь: *a: | 5 96 | 69 75 | 77 90 | 32 89 95 |.*

Перевірка здійснюється спочатку для першої пари. У випадку помилки, з'являється повідомлення: «Помилка! Слід взяти перший елемент послідовності *b* – це *5*, та перший елемент послідовності *c* – це *96*, та порівняти їх. Оскільки *5<98*, то перша впорядкована пара – це | *5 96* |.». Користувачу слід виправити помилки. Якщо помилки в першій парі чисел немає, то з'являється повідомлення: «Перша пара вірна!» та здійснюється перевірка другої пари.

Перевірка другої пари: при помилці з'являється повідомлення: «Не вірно! Слід взяти другий елемент послідовності *b* – це *69*, та другий елемент послідовності *c* – це *75*, та порівняти їх. Оскільки *69<75*, то друга впорядкована пара – це | *69 75* |.». Користувачу необхідно виправити похибки. Якщо помилки в другій парі чисел немає, то з'являється повідомлення: «Друга пара вірна!» та здійснюється перевірка третьої пари.

Перевірка третьої пари: при невірній відповіді з'являється повідомлення: «Хибна відповідь! Слід взяти третій елемент послідовності *b* – це *90*, та третій елемент послідовності *c* – це *77*, та порівняти їх. Оскільки *90>77*, то третя впорядкована пара – це | *77 90* |.». Користувачу необхідно виправити похибки. Якщо помилки в третій парі чисел немає, то з'являється повідомлення: «Третя пара вірна!» та здійснюється перевірка впорядкованої трійки.

Перевірка впорядкованої трійки: при невірній відповіді з'являється повідомлення: «Похибка! Слід взяти четвертий елемент послідовності *b* – це *32*, та четвертий елемент послідовності *c* – це *89*, та порівняти їх. Оскільки *32<89*, то перший елемент впорядкованої трійки – це *32*. Далі слід взяти четвертий елемент послідовності *c* – це *89*, та п'ятий елемент послідовності *c* – це *95*, та порівняти їх. Оскільки *89<95*, то впорядкована трійка – це | *32 89 95* |.». Користувачу необхідно виправити помилки. Якщо помилок тут немає, то з'являється повідомлення: «Впорядкована трійка вірна!» та здійснюється перехід на крок 3.

Крок 3. *Поділіть послідовність a: | 5 96 | 69 75 | 77 90 | 32 89 95 | приблизно навпіл так, щоб перша частина послідовності a утворила послідовність b, а друга частина послідовності a утворила послідовність c. Впорядкованість пар та трійки при цьому зберігається.*

Оскільки чисел 9, то в послідовності a буде 4 числа, а в b – 5 чисел.

Введіть необхідні числа для послідовностей b та c.

Правильна відповідь: *b: | 5 96 | 69 75 |; c: | 77 90 | 32 89 95 |.*

Спочатку перевіряється послідовність *b*, у випадку помилки з'являється повідомлення: «Хибна відповідь! Послідовність *b* – це перша частина послідовності *a*, тобто впорядковані пари | *5 96* | *69 75* |.». Користувачу потрібно виправити помилки. У випадку вірності з'являється повідомлення: «Послідовність *b* вірна!».

Потім перевіряється послідовність *c*, у випадку помилки з'являється повідомлення: «Помилкова відповідь! Послідовність *c* – це друга частина послідовності *a*, тобто | *77 90* | *32 89 95* |.». Користувачу слід виправити помилки. У випадку правильності з'являється повідомлення: «Послідовність *c* вірна!» та відбувається перехід на крок 4.

Крок 4. *Об'єднайте послідовності b: | 5 96 | 69 75 | та c: | 77 90 | 32 89 95 | в одну. При цьому утворіть* ***впорядковані четвірку*** *та* ***п'ятірку*** *елементів. Перша четвірка – перша пара послідовності b та перша пара послідовності c, елементи яких записуються за зростанням, і т.д.*

Введіть необхідні числа для об'єднаної послідовності.

Правильна відповідь: *a: | 5 77 90 96 | 32 69 75 89 95 |.*

Перевірка здійснюється спочатку для четвірки. У випадку помилки, з'являється повідомлення: «Помилка! Слід розглянути першу пару послідовності *b* – це | *5 96* | та першу пару послідовності *c* – це | *77 90* |. Впорядкована четвірка з цих елементів – це | *5 77 90 96* |.». Користувачу необхідно виправити помилки. У випадку правильної відповіді з'являється повідомлення «Четвірка вірна!».

Якщо помилки в четвірці чисел немає, то здійснюється перевірка п'ятірки. У випадку помилки, з'являється повідомлення: «Помилка! Слід розглянути другу пару послідовності *b* – це | *69 75* | та трійку послідовності *c* – це | *32 89 95* |. Впорядкована п'ятірка з цих елементів – це | *32 69 75 89 95* |.» Користувачу необхідно виправити помилки. У випадку правильної відповіді з'являється повідомлення «П'ятірка вірна!» та відбувається перехід на крок 5.

Крок 5. *Поділіть послідовність a: | 5 77 90 96 | 32 69 75 89 95 | приблизно навпіл так, щоб перша частина послідовності a утворила послідовність b, а друга частина послідовності a утворила послідовність c. Впорядкованість четвірки та п'ятірки при цьому зберігається.*

Введіть необхідні числа для послідовностей b та c.

Правильна відповідь: *b: | 5 77 90 96 |; c: | 32 69 75 89 95 |.*

Спочатку перевіряється послідовність *b*, у випадку помилки з'являється повідомлення: «Неправильно! Послідовність *b* – це перша частина послідовності *a*, тобто впорядкована четвірка | *5 77 90 96* |.». Користувачу потрібно виправити помилки. У випадку правильної відповіді з'являється повідомлення «Послідовність *b* вірна!».

Потім перевіряється послідовність *c*, у випадку помилки з'являється повідомлення: «Не вірна відповідь! Послідовність *c* – це друга частина послідовності *a*, тобто впорядкована п'ятірка | *32 69 75 89 95* |.». Користувачу слід виправити помилки. У випадку правильності з'являється повідомлення: «Послідовність *c* вірна!» та відбувається перехід на крок 6.

Крок 6. *Об'єднайте послідовності b: | 5 77 90 96 | та c: | 32 69 75 89 95 | в одну. При цьому утворіть* ***впорядковану дев'ятку*** *елементів. Тобто береться впорядкована четвірка послідовності b та впорядкована п'ятірка послідовності c, елементи яких записуються за зростанням.*

Введіть необхідні числа для об'єднаної послідовності.

Правильна відповідь: *a: | 5 32 69 75 77 89 90 95 96 |*.

У випадку помилки з'являється повідомлення «Помилка! Впорядкована дев'ятка з | *5 77 90 96* | та | *32 69 75 89 95* | – це | *5 32 69 75 77 89 90 95 96* |.».

У випадку правильної відповіді з'являється повідомлення «Впорядкована дев'ятка правильна!».

Далі виводиться повідомлення «Послідовність з непарною кількістю елементів відсортована! Робота з тренажером завершена!».

3.3. Блок-схема алгоритму тренажера

На рис. 3.5-3.9 представлено блок-схему [16] алгоритму тренажеру для першого та другого кроків прикладу 1.

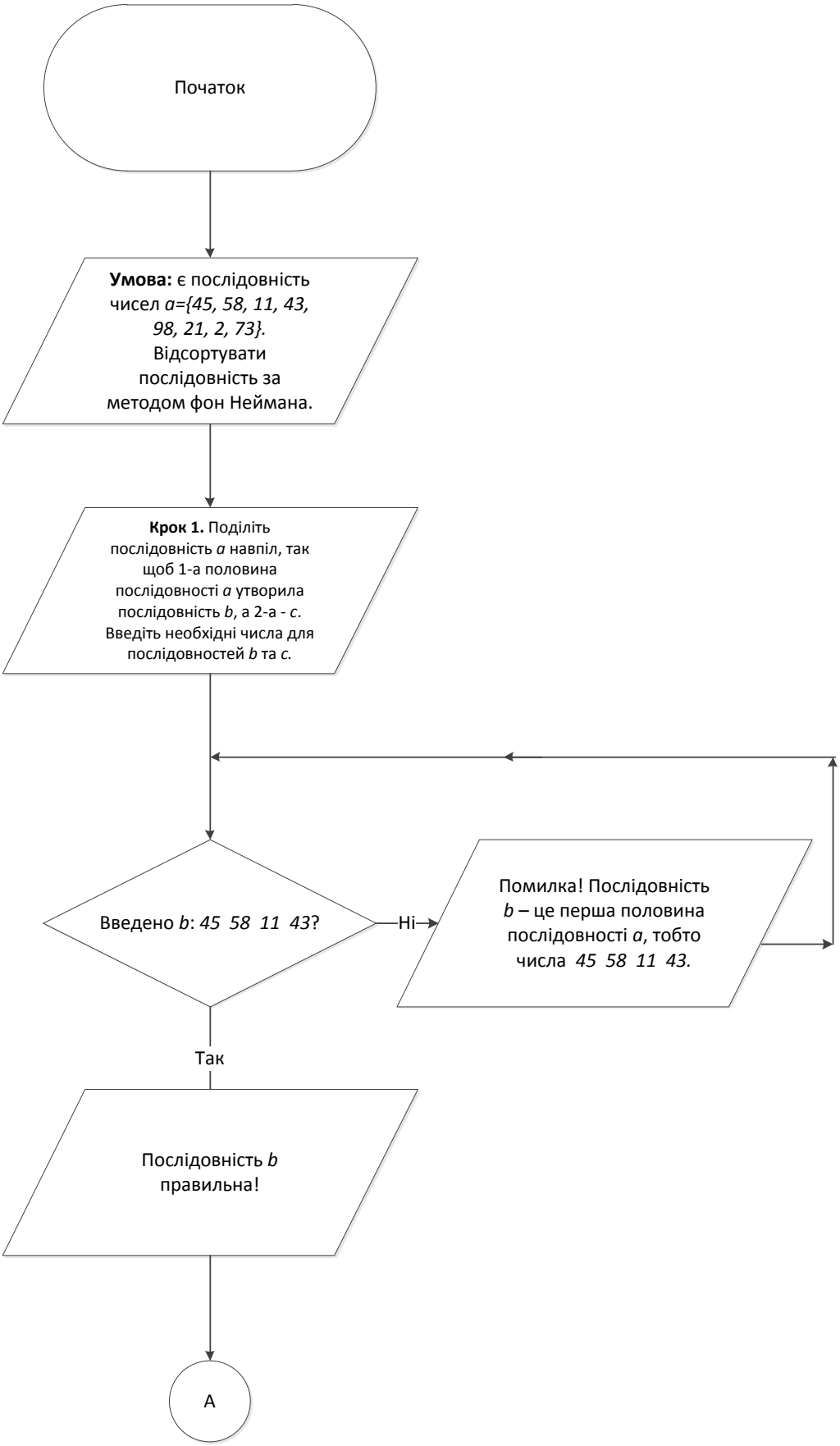

Рис. 3.5 – Блок-схема алгоритму (приклад l, крок 1)

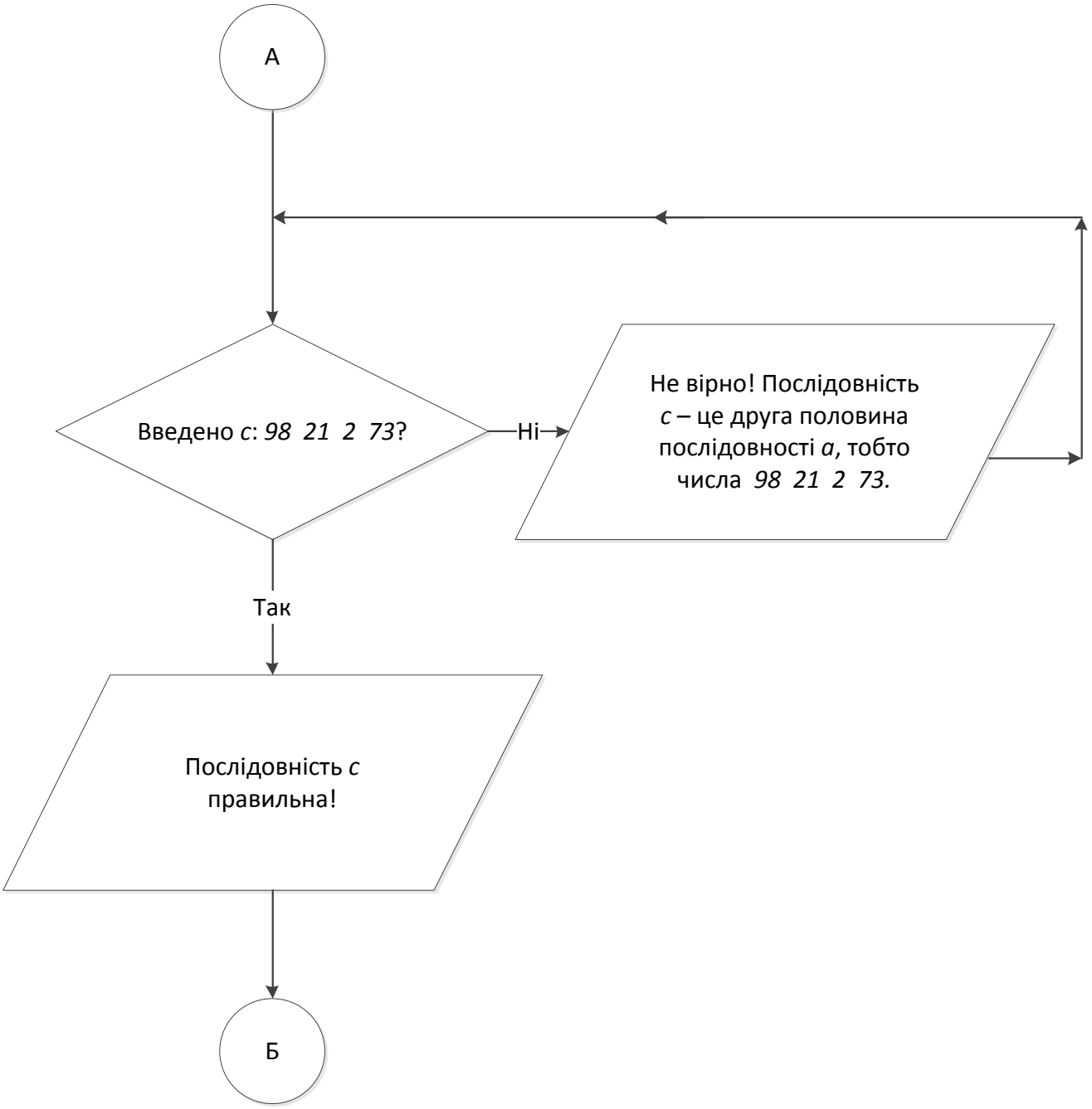

Рис. 3.6 – Блок-схема алгоритму (приклад l, продовження кроку 1)

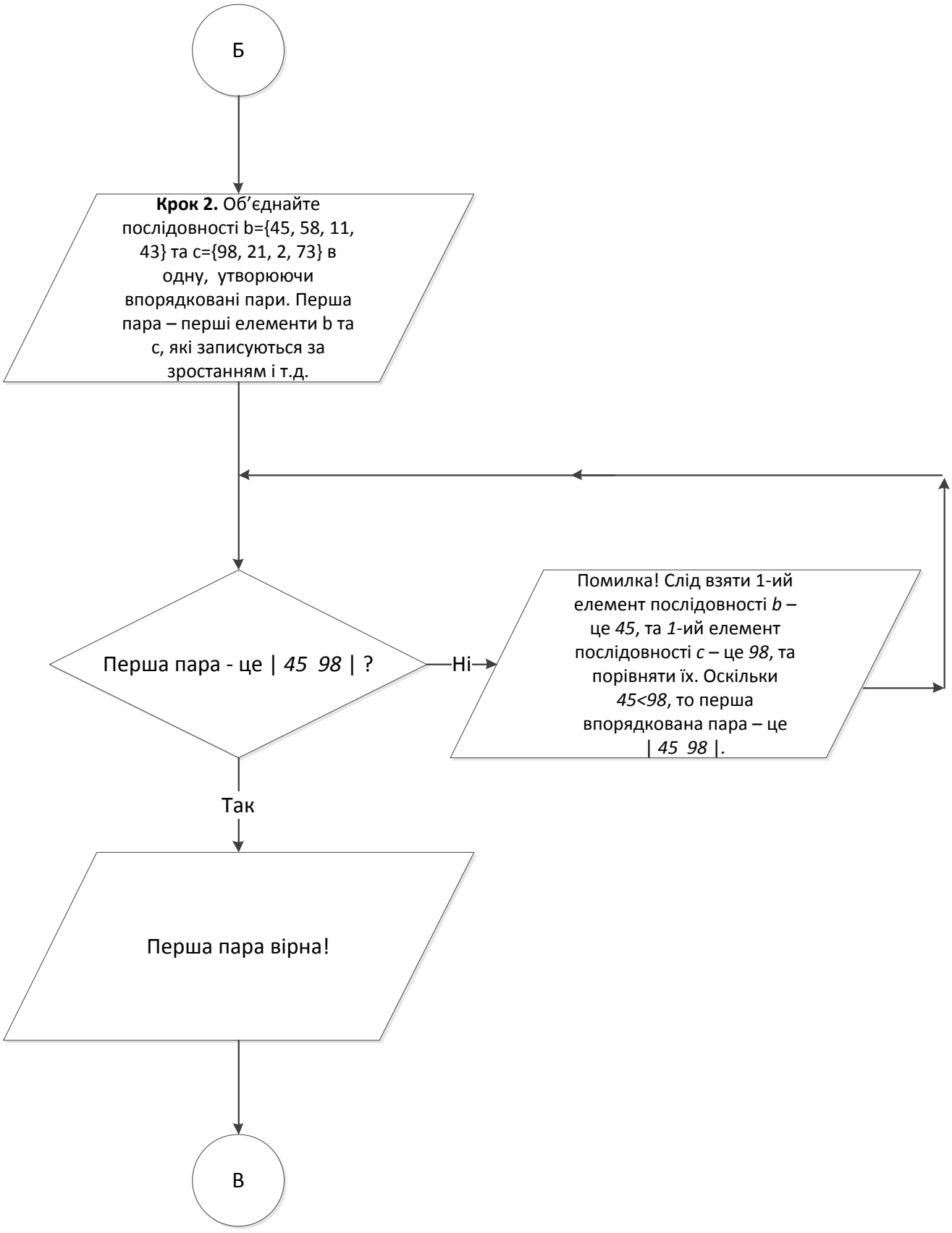

Рис. 3.7 – Блок-схема алгоритму (приклад l, крок 2)

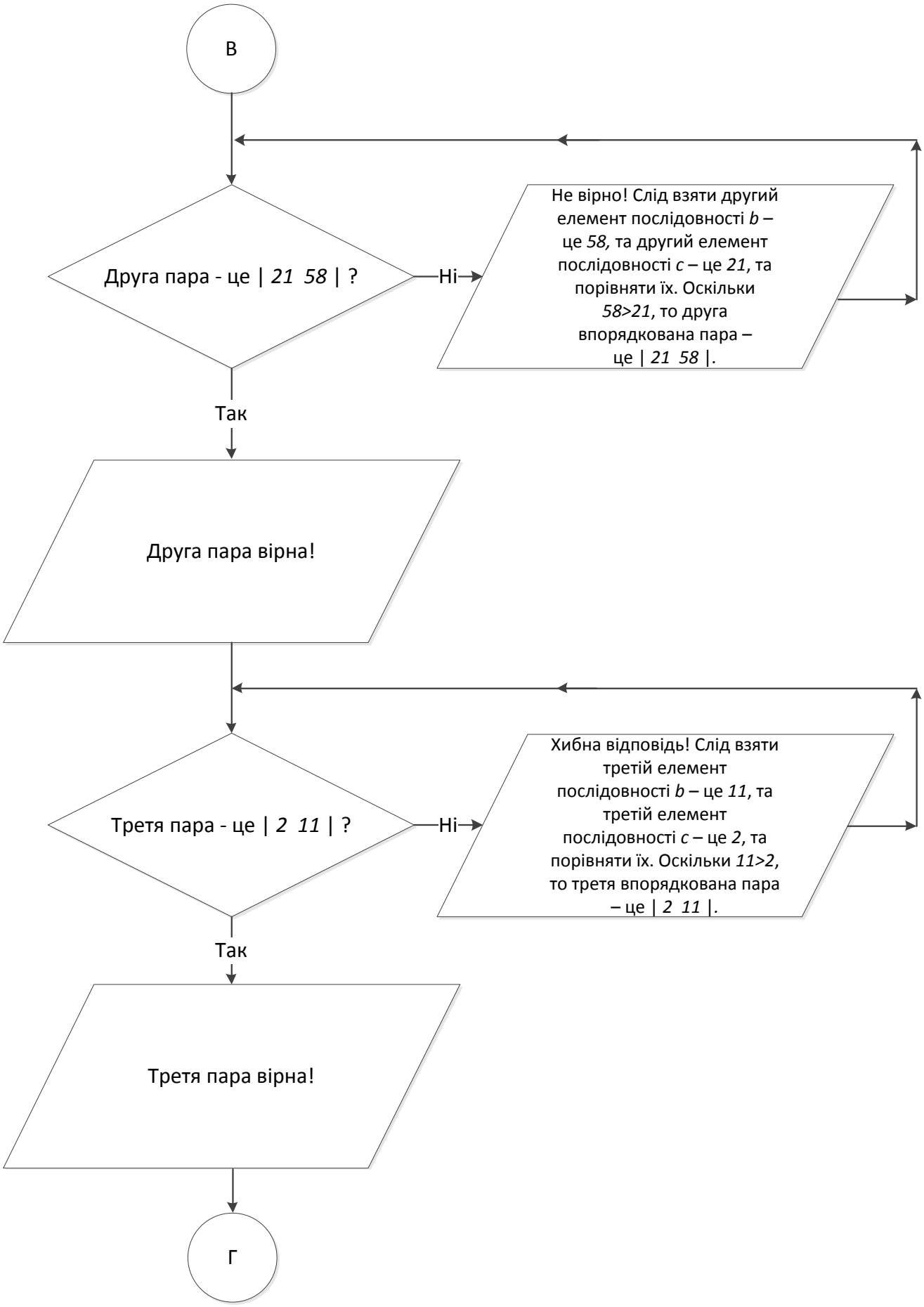

Рис. 3.8 – Блок-схема алгоритму (приклад l, продовження кроку 2)

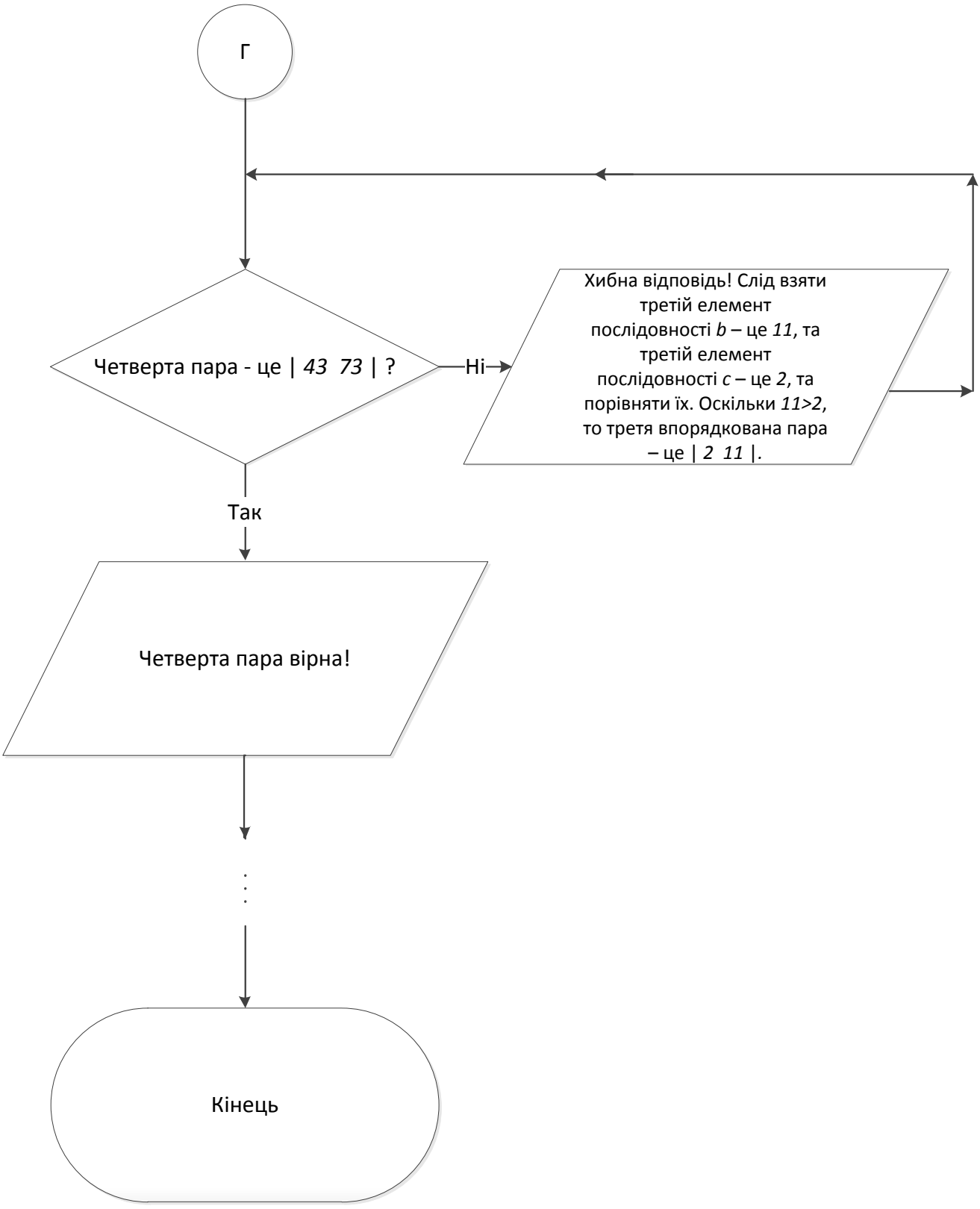

Рис. 3.9 – Блок-схема алгоритму (приклад l, продовження кроку 2)

4. ПРАКТИЧНА ЧАСТИНА

4.1. Інструкція по використанню програми

На рис. 4.1-4.41 представлено, як працює тренажер на прикладі прикладу 1. Приклад 2 подано у додатку А (рис. А.1-А.34).

Рис. 4.1 – Титульне вікно

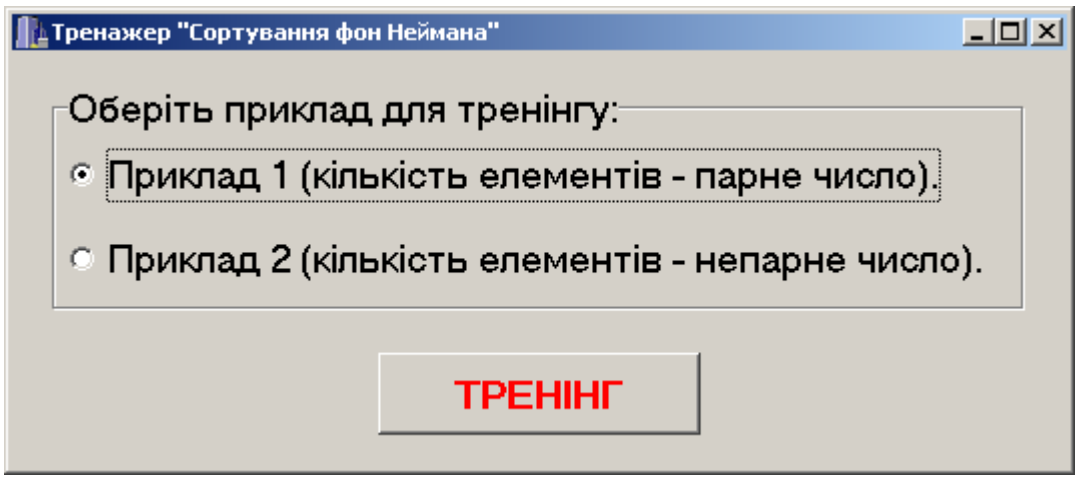

Рис. 4.2 – Друге вікно програми

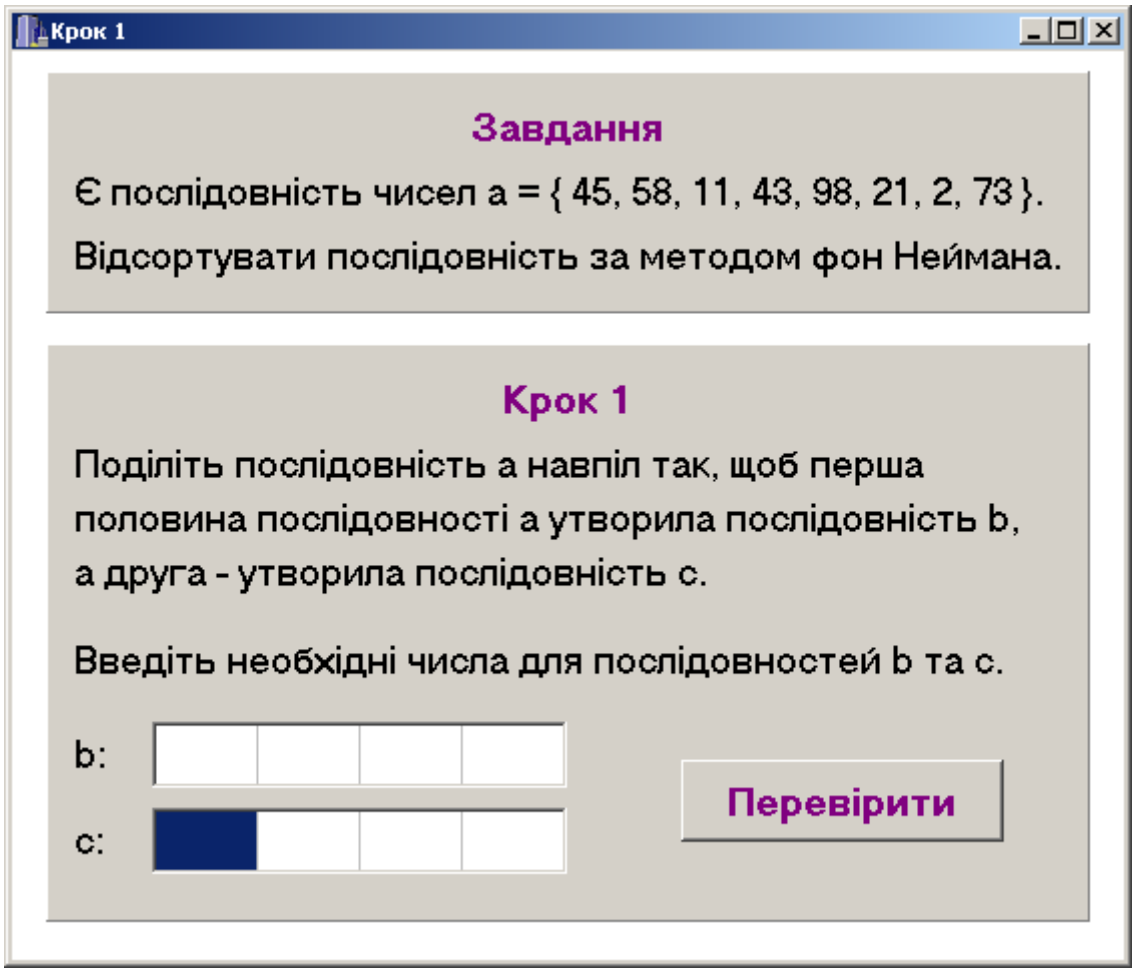

Рис. 4.3 – Перший крок

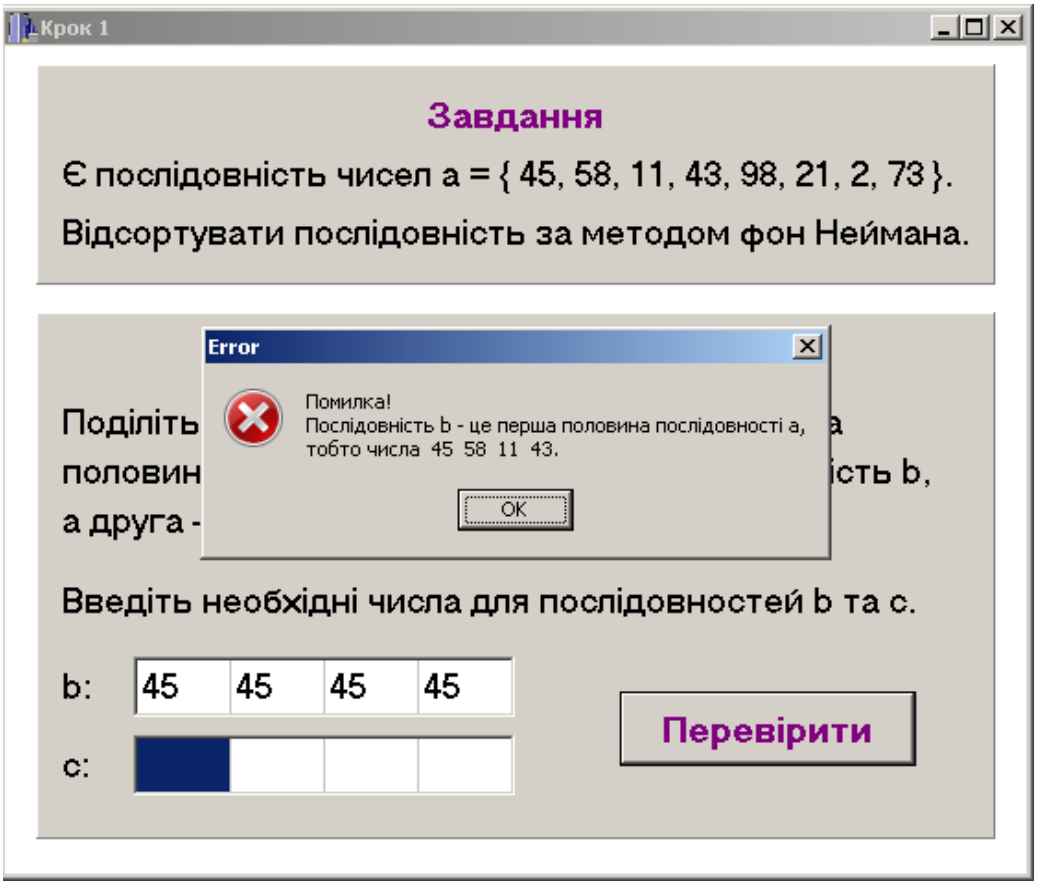

Рис. 4.4 – Помилка у послідовності *b* на першому кроці

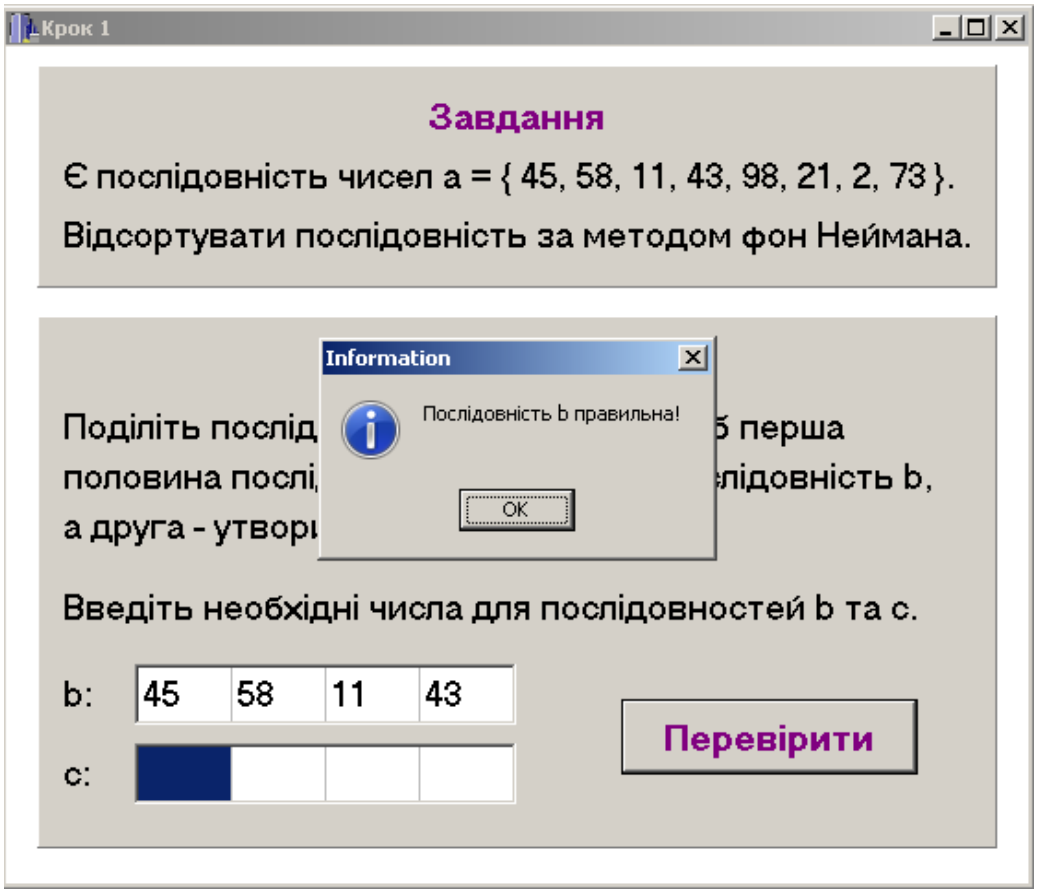

Рис. 4.5 – Підтвердження правильності послідовності *b* на першому кроці

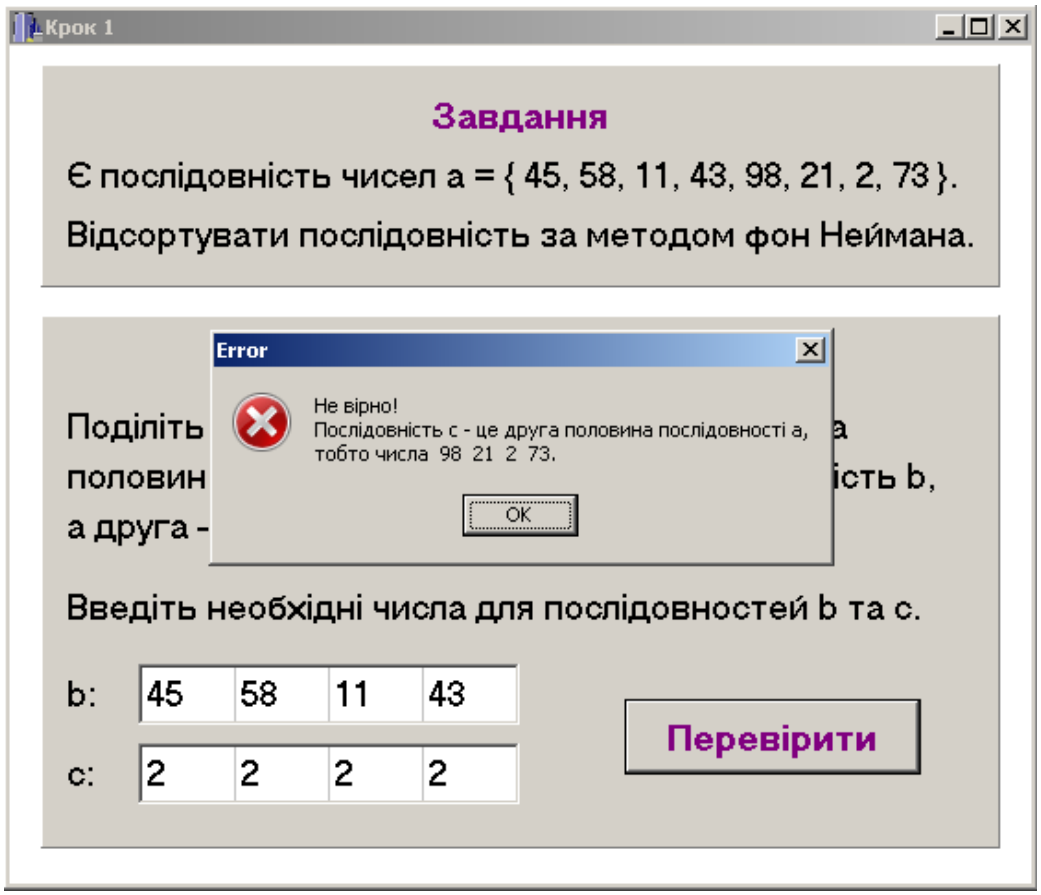

Рис. 4.6 – Помилка у послідовності *c* на першому кроці

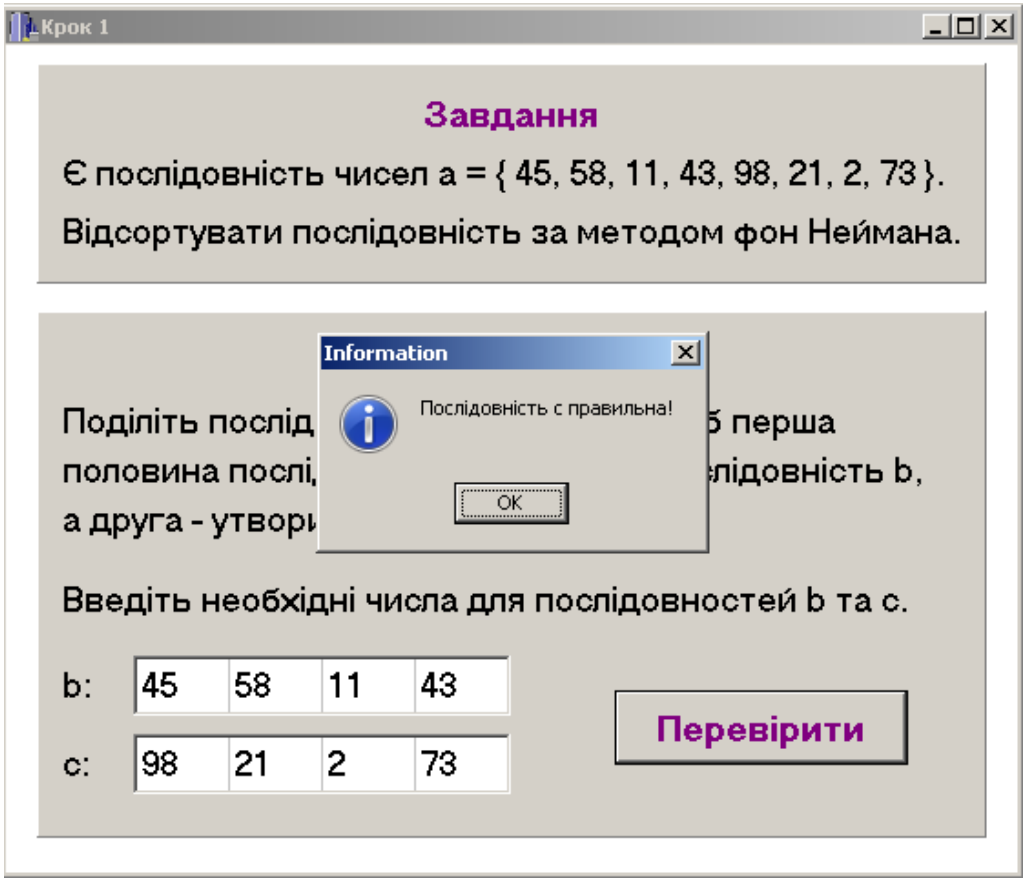

Рис. 4.7 – Підтвердження правильності послідовності *c* на першому кроці

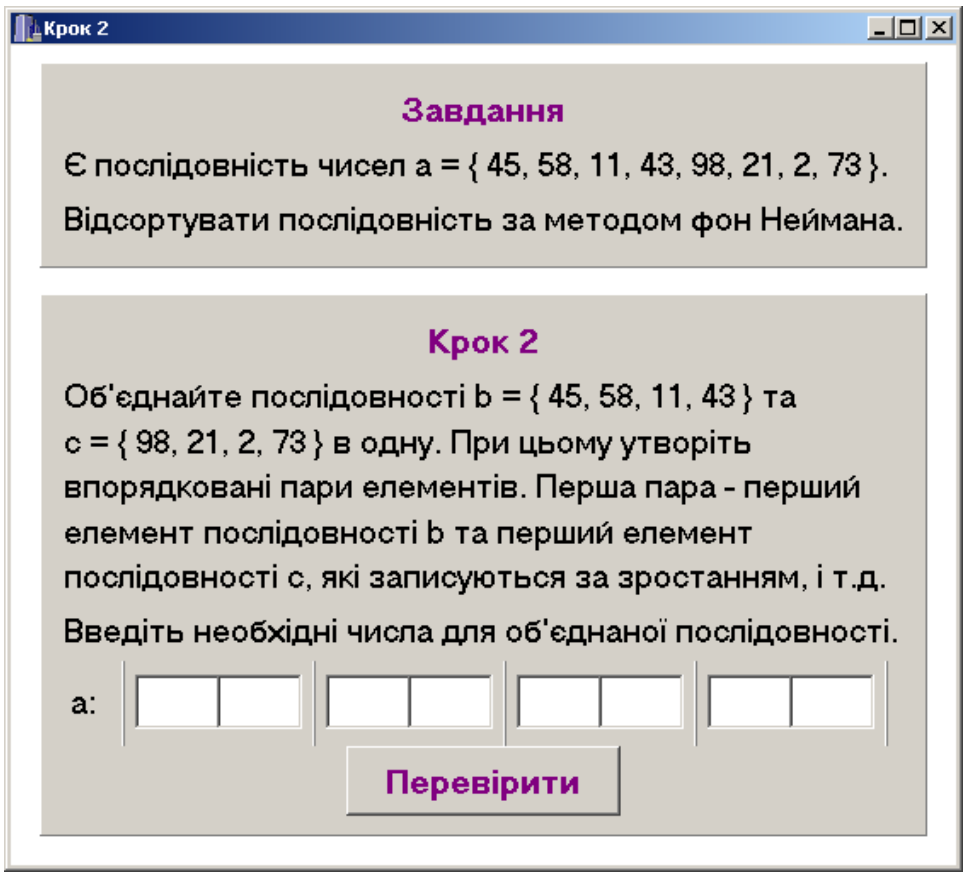

Рис. 4.8 – Другий крок

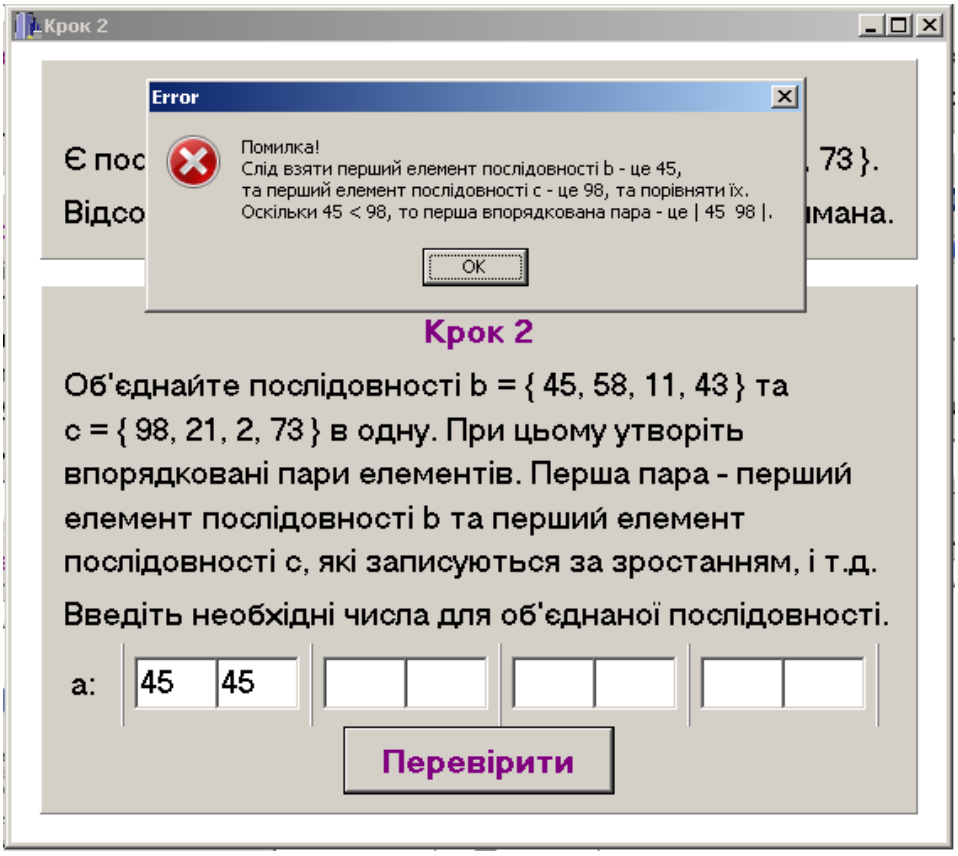

Рис. 4.9 – Помилка у першій парі на другому кроці

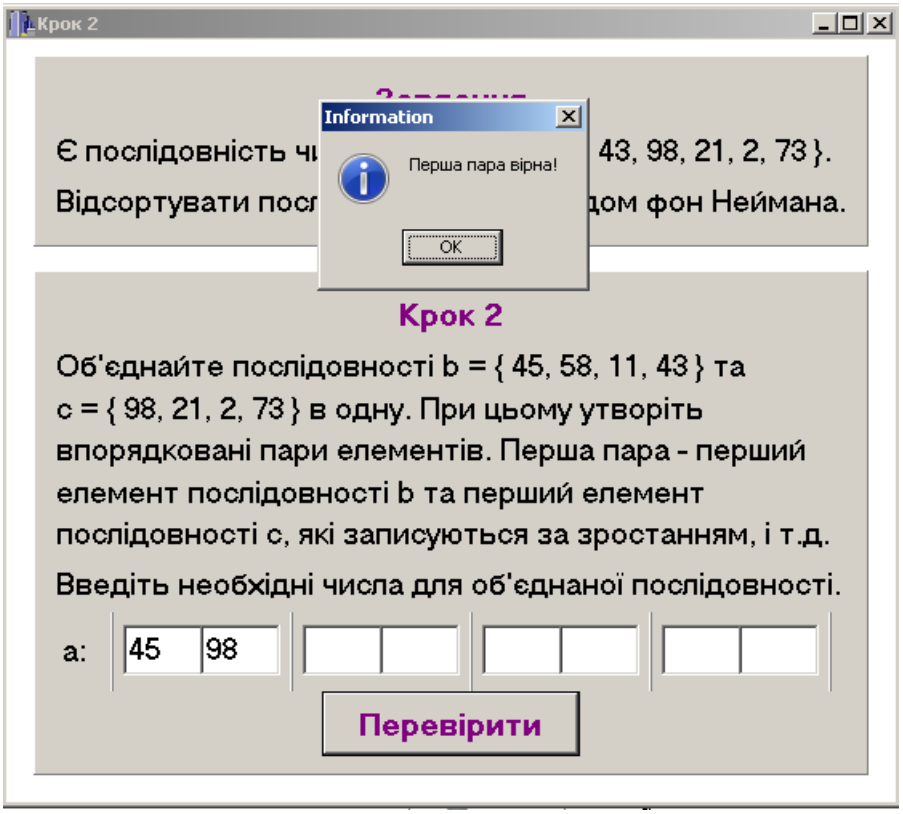

Рис. 4.10 – Підтвердження правильності першої пари на другому кроці

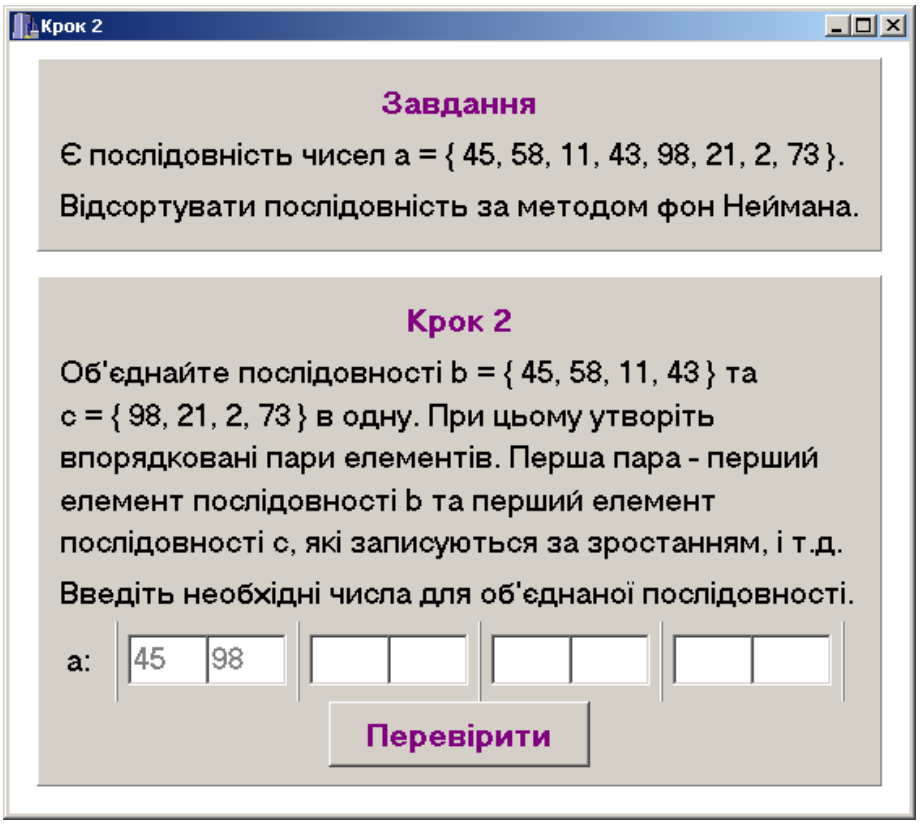

Рис. 4.11 – Перехід до заповнення другої пари на другому кроці

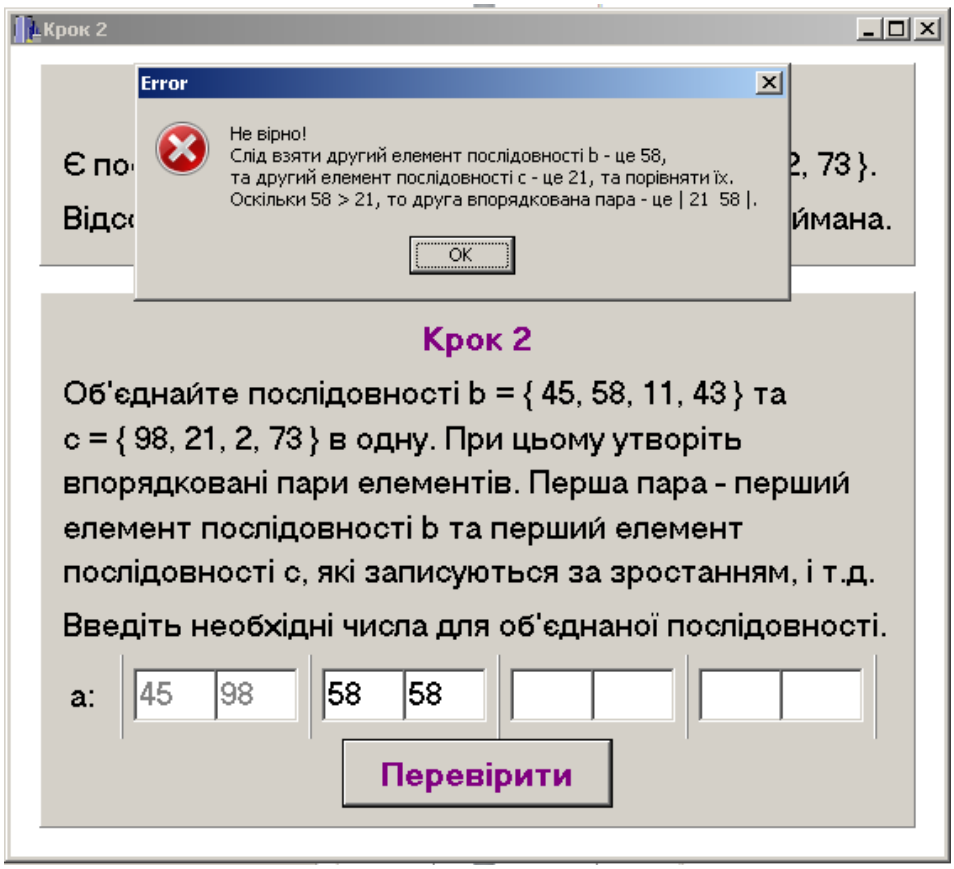

Рис. 4.12 – Помилка у другій парі на другому кроці

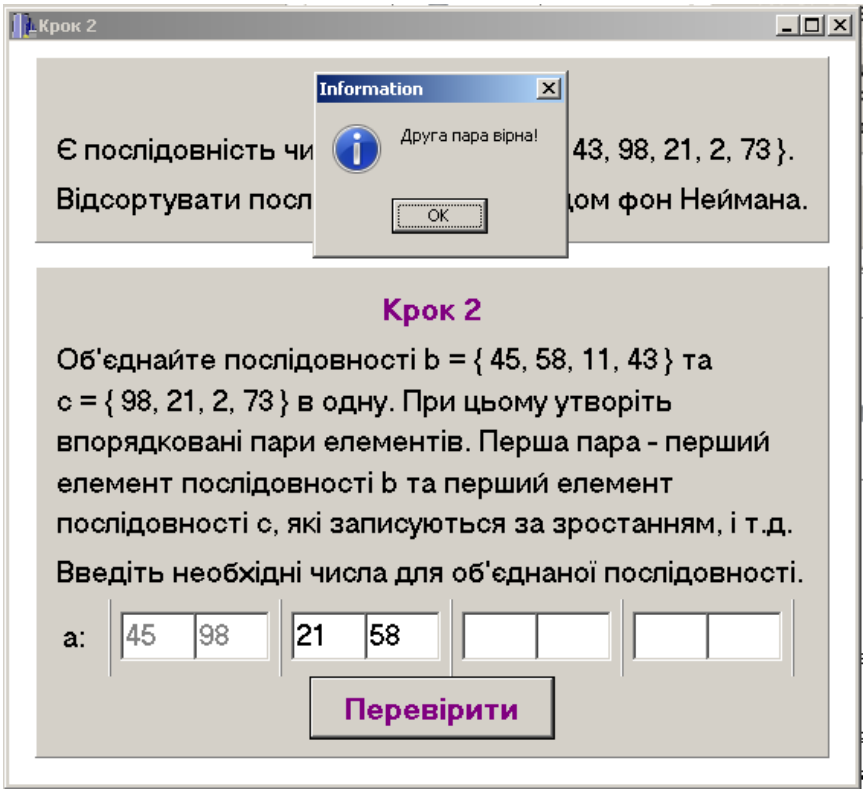

Рис. 4.13 – Підтвердження правильності другої пари на другому кроці

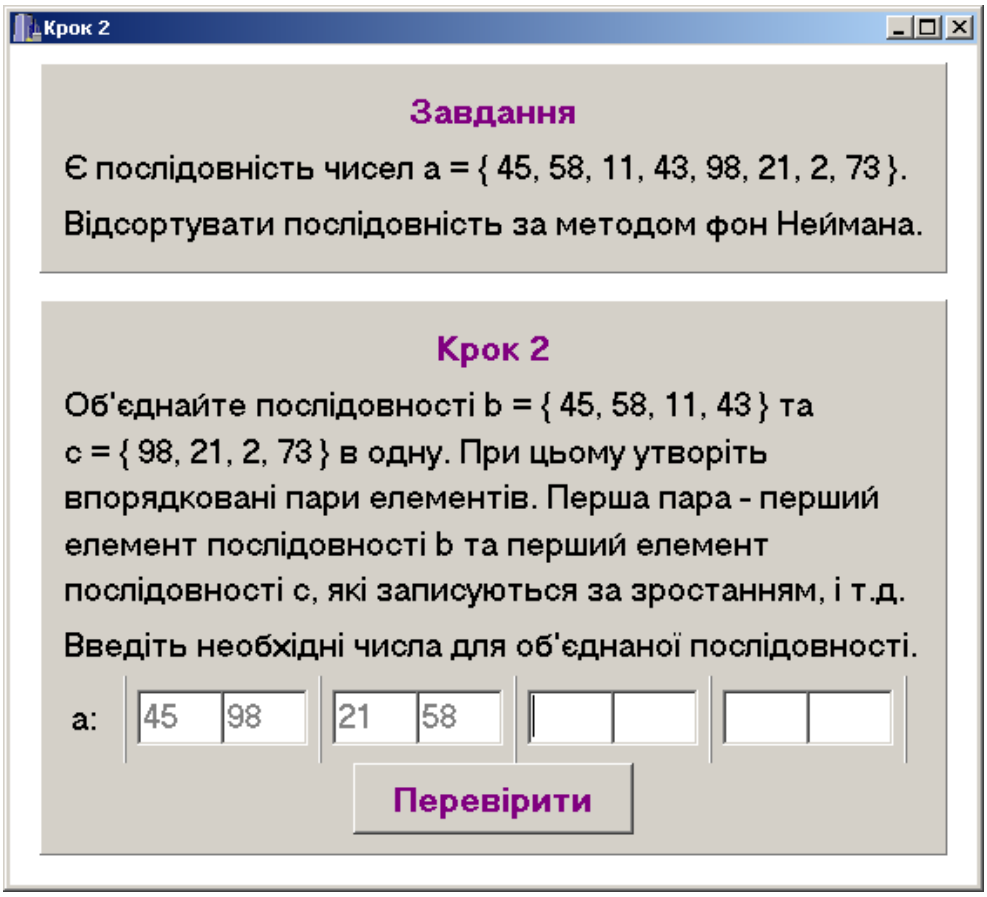

Рис. 4.14 – Перехід до заповнення третьої пари на другому кроці

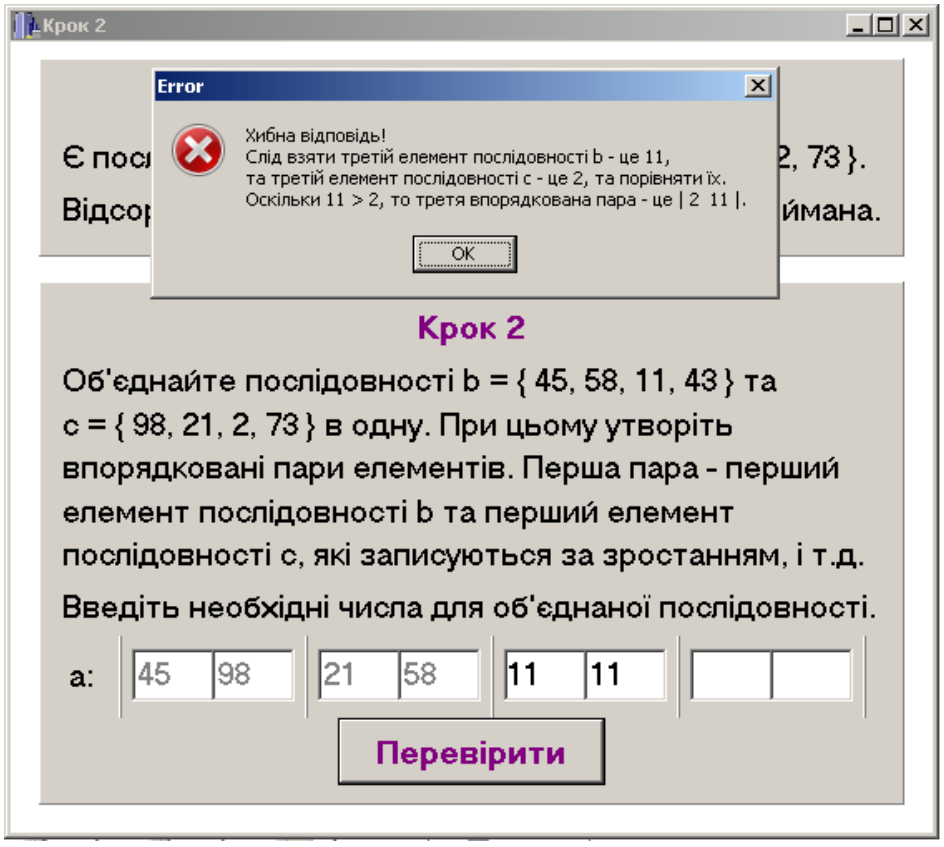

Рис. 4.15 – Помилка у третій парі на другому кроці

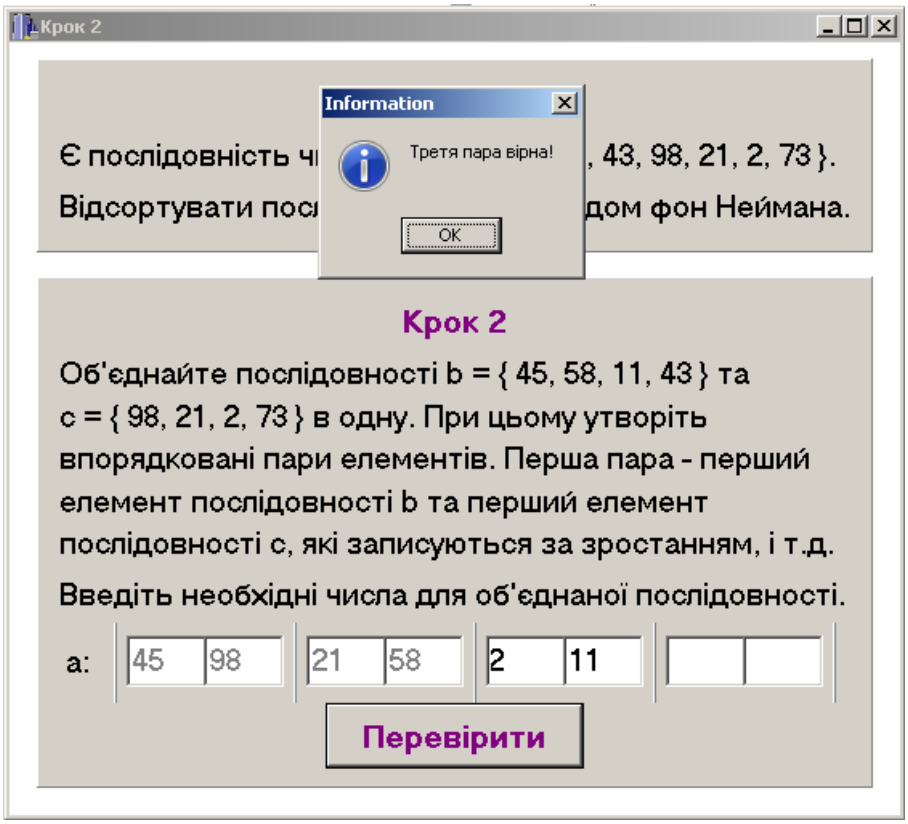

Рис. 4.16 – Підтвердження правильності третьої пари на другому кроці

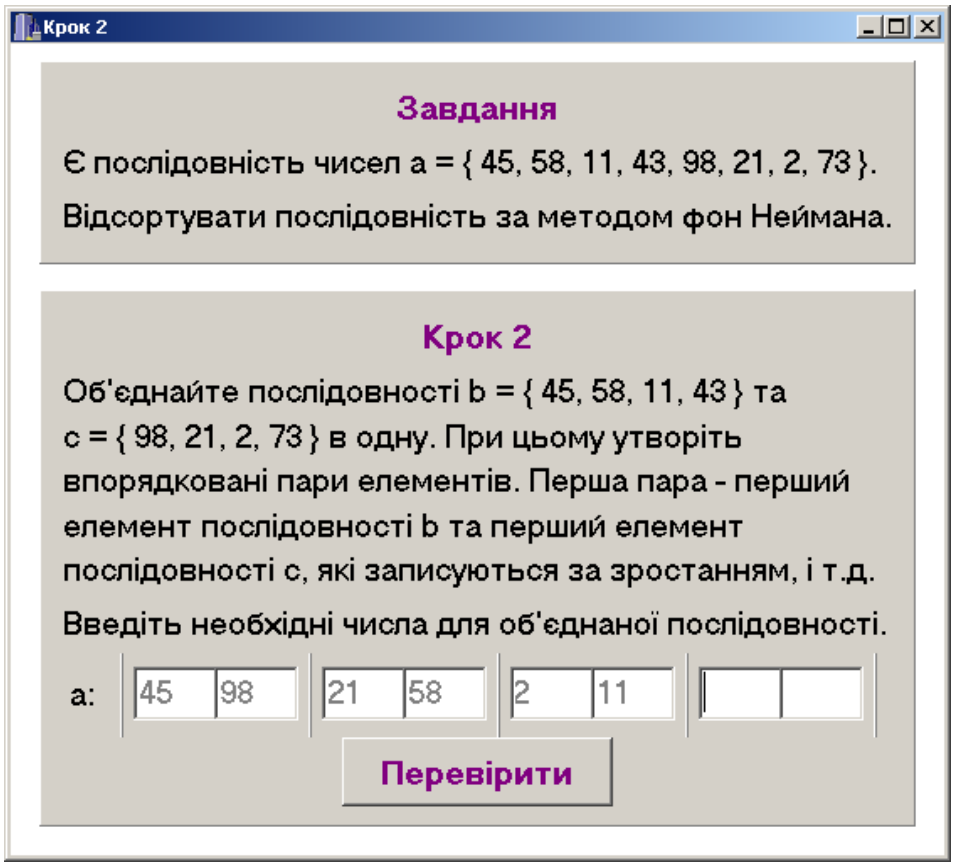

Рис. 4.17 – Перехід до заповнення четвертої пари на другому кроці

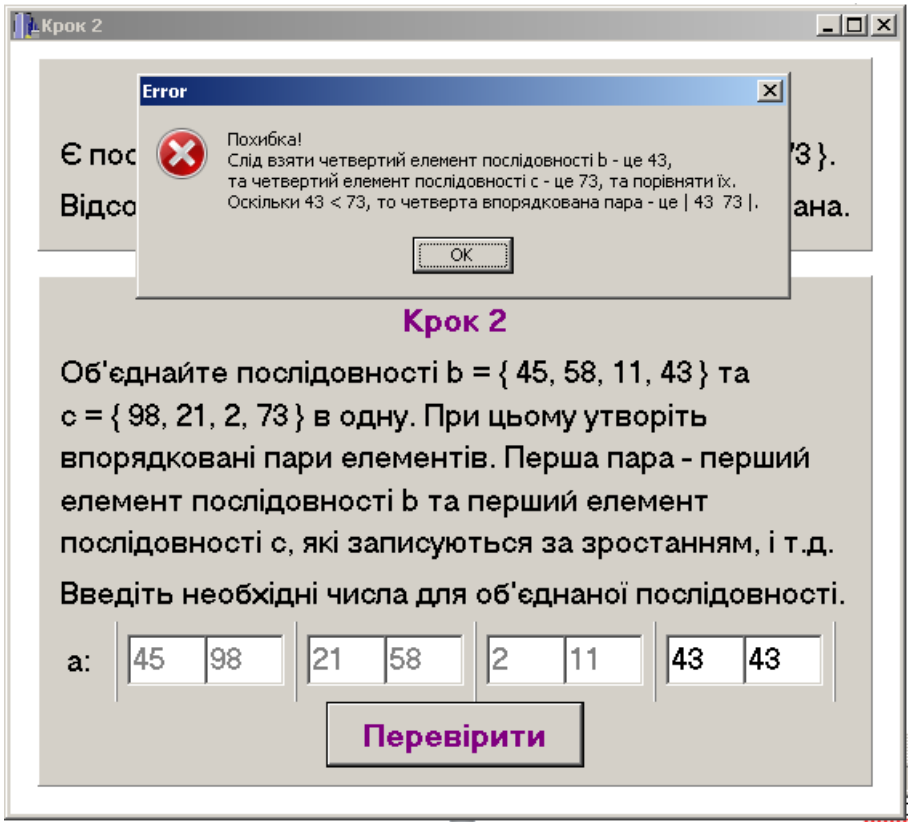

Рис. 4.18 – Помилка у четвертій парі на другому кроці

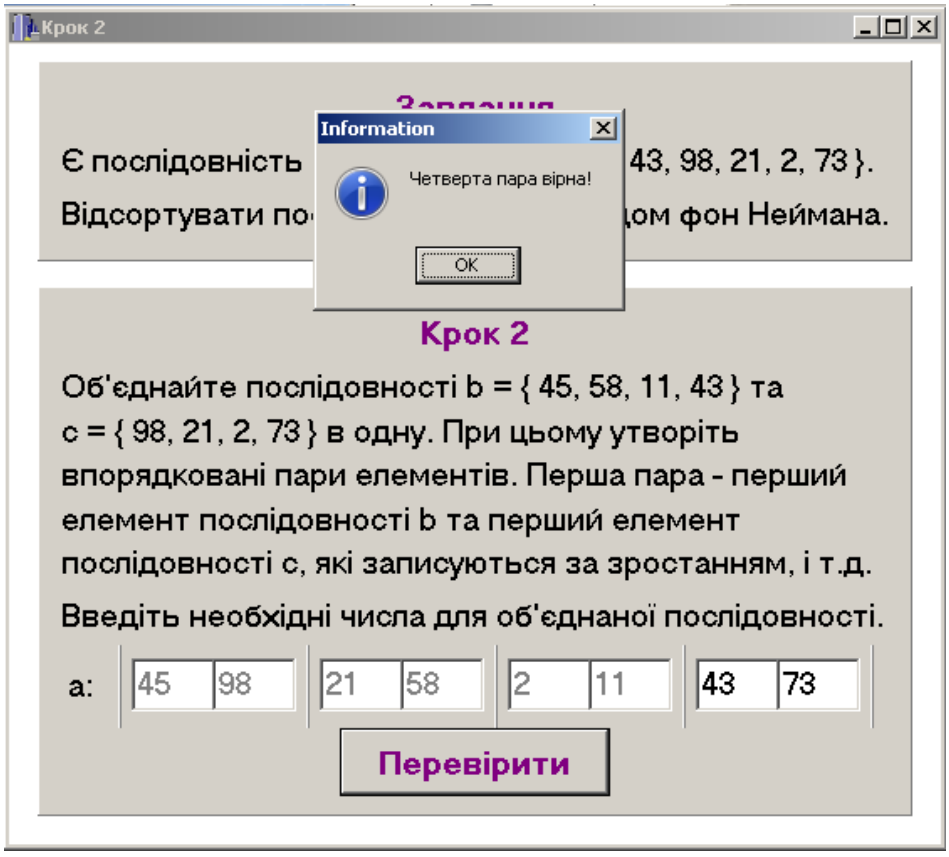

Рис. 4.19 – Підтвердження правильності четвертої пари на другому кроці

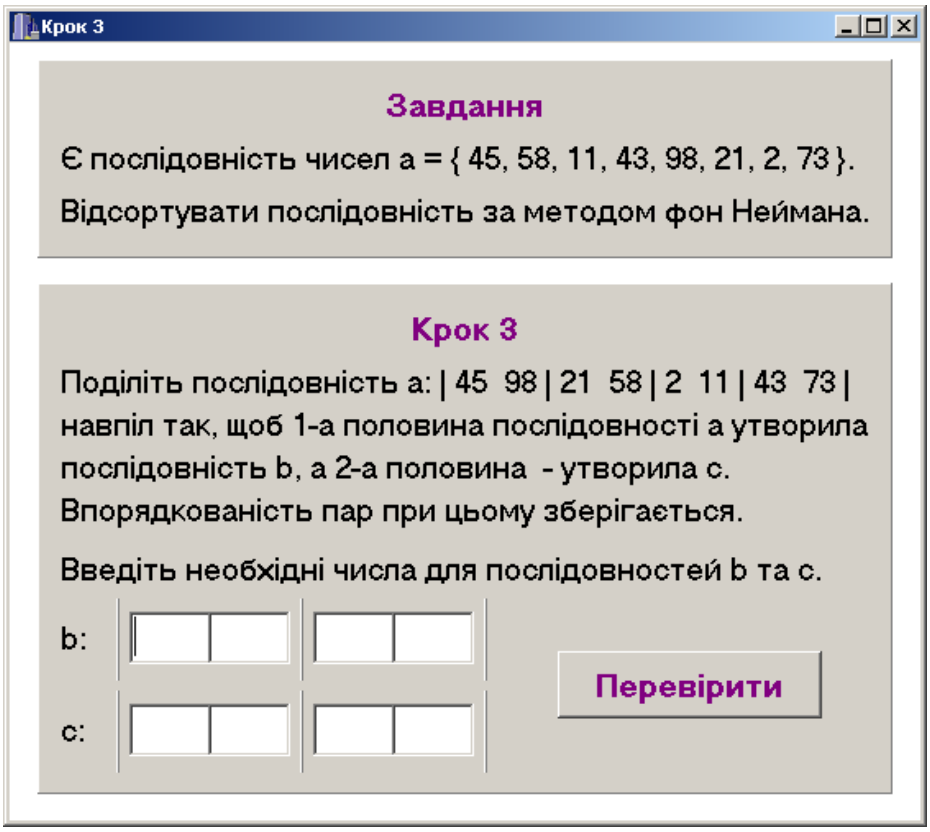

Рис. 4.20 – Третій крок

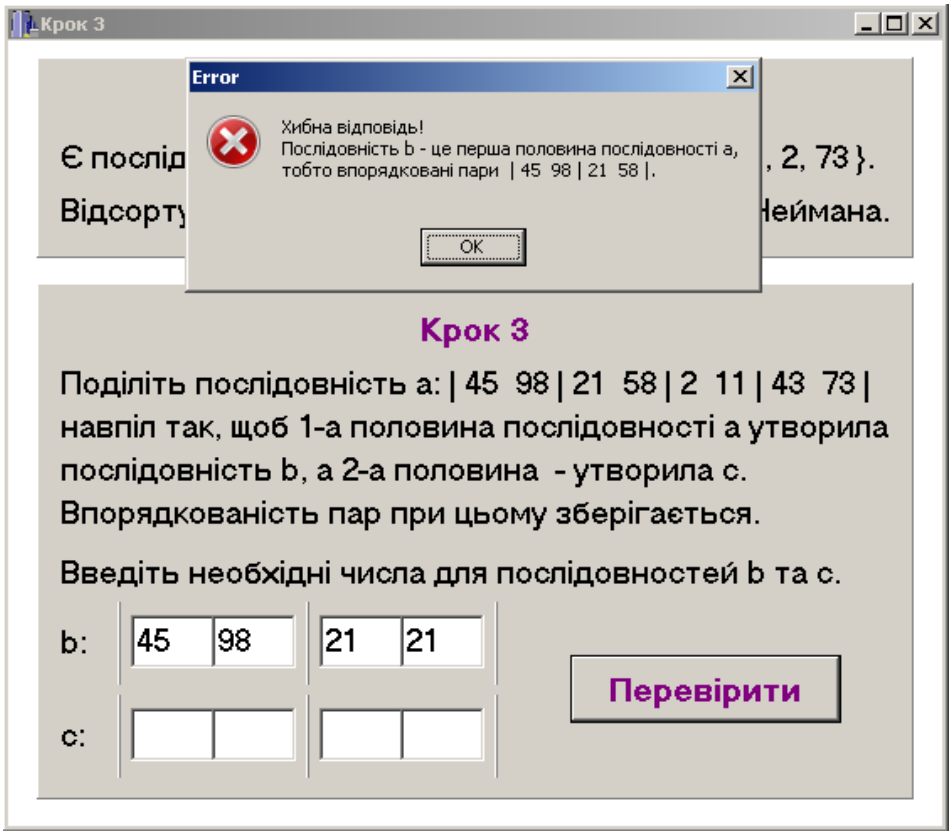

Рис. 4.21 – Помилка у послідовності *b* на третьому кроці

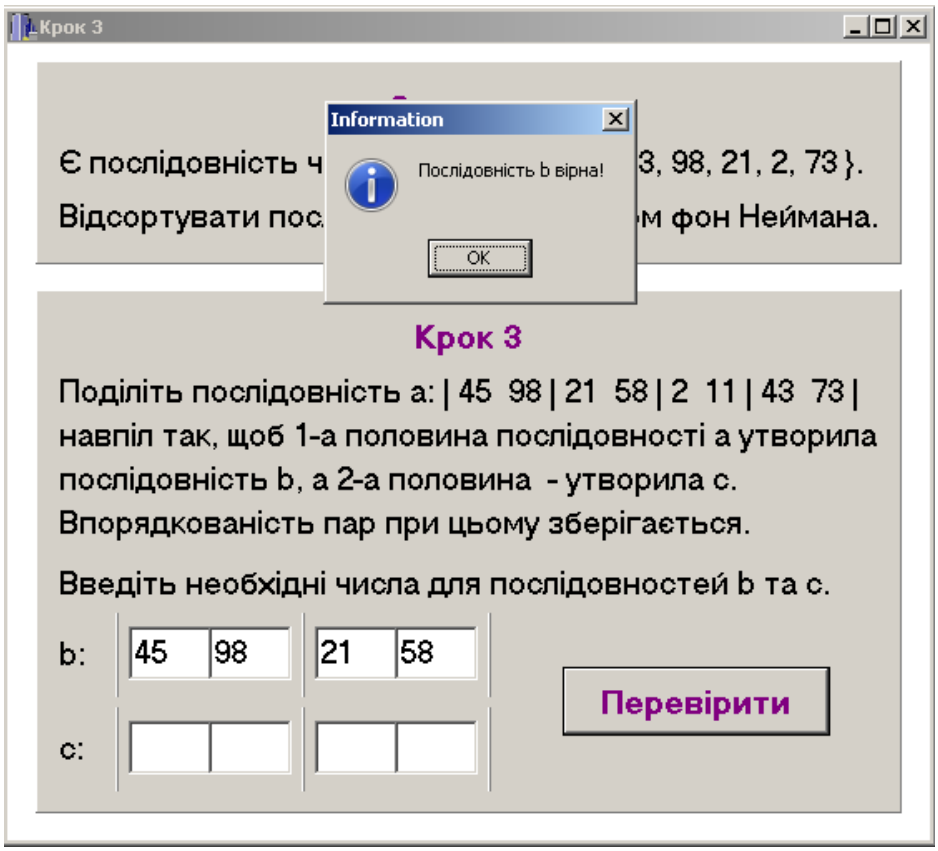

Рис. 4.22 – Підтвердження правильності послідовності *b* на третьому кроці

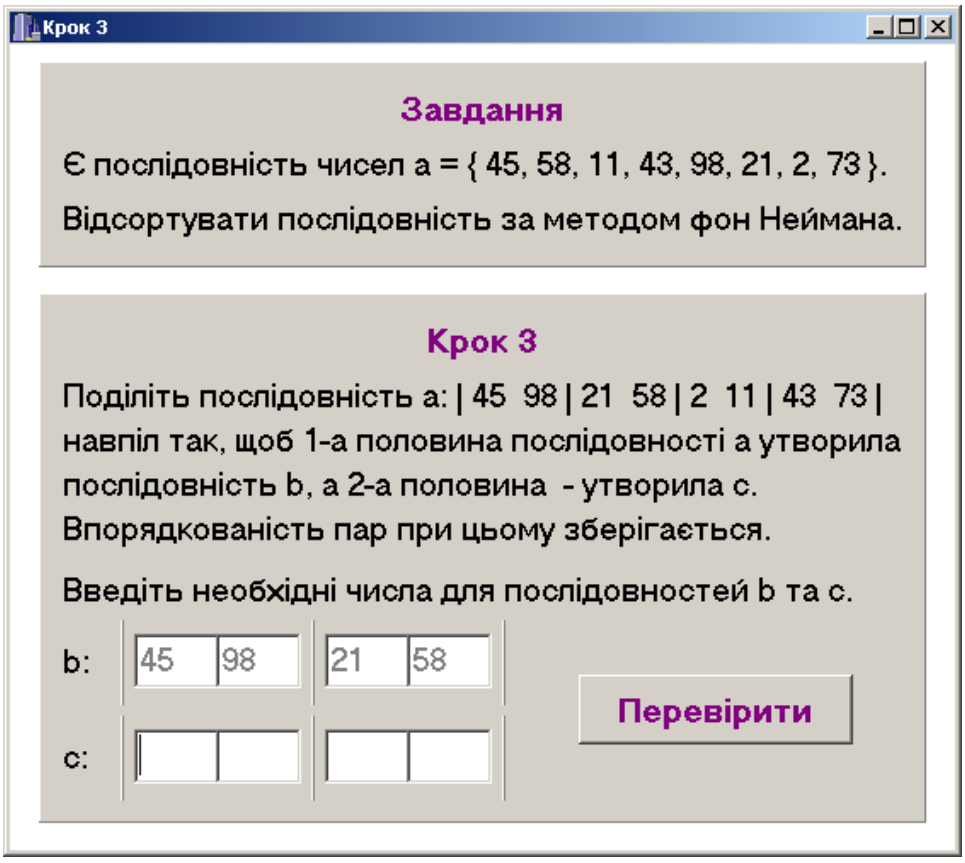

Рис. 4.23 – Перехід до заповнення послідовності *c* на третьому кроці

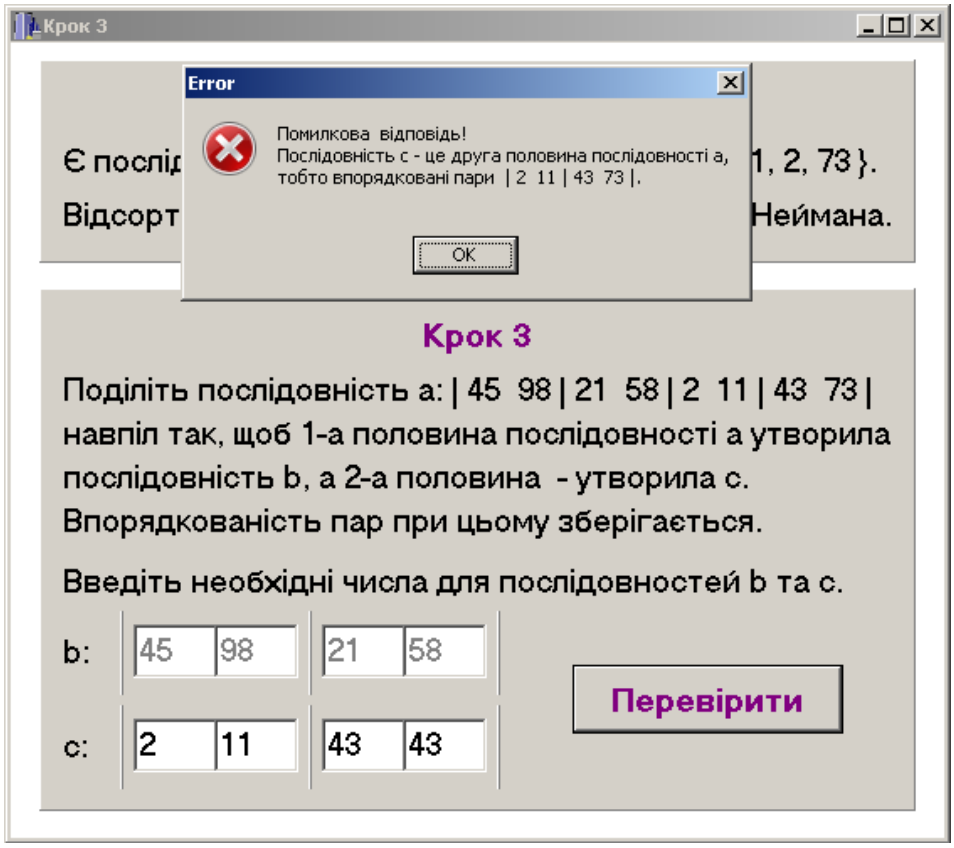

Рис. 4.24 – Помилка у послідовності *c* на третьому кроці

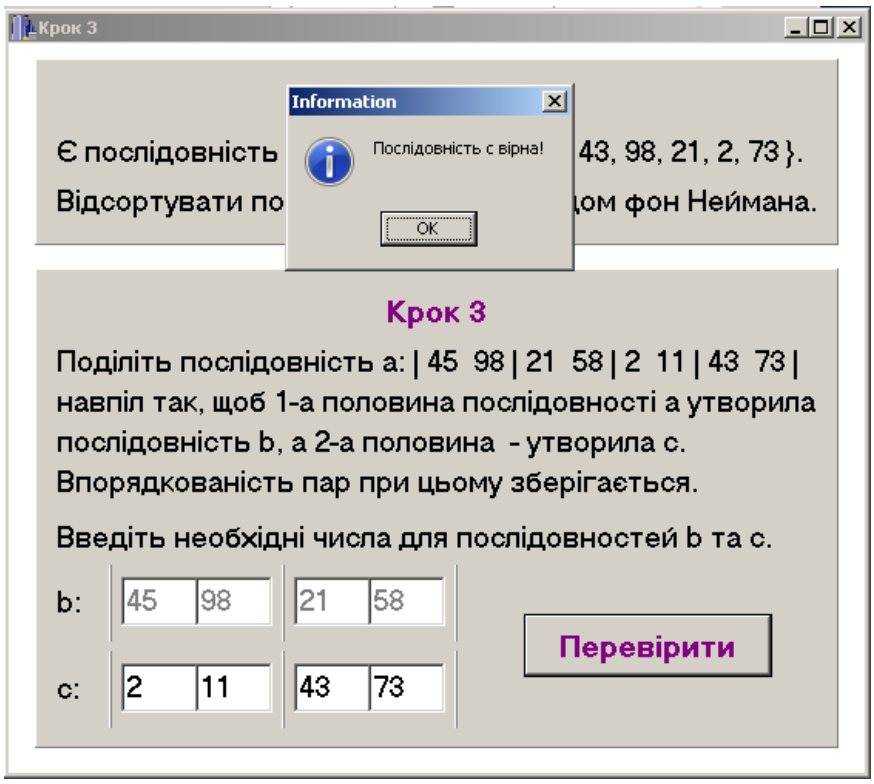

Рис. 4.25 – Підтвердження правильності послідовності *c* на третьому кроці

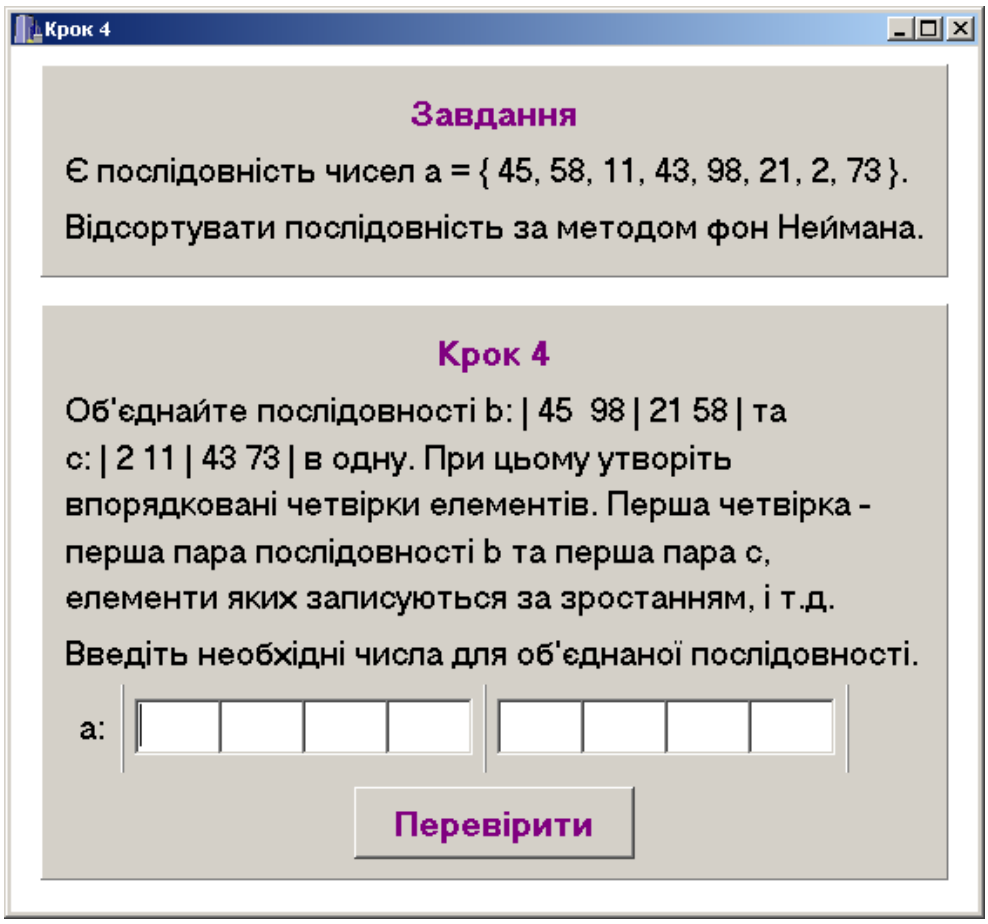

Рис. 4.26 – Четвертий крок

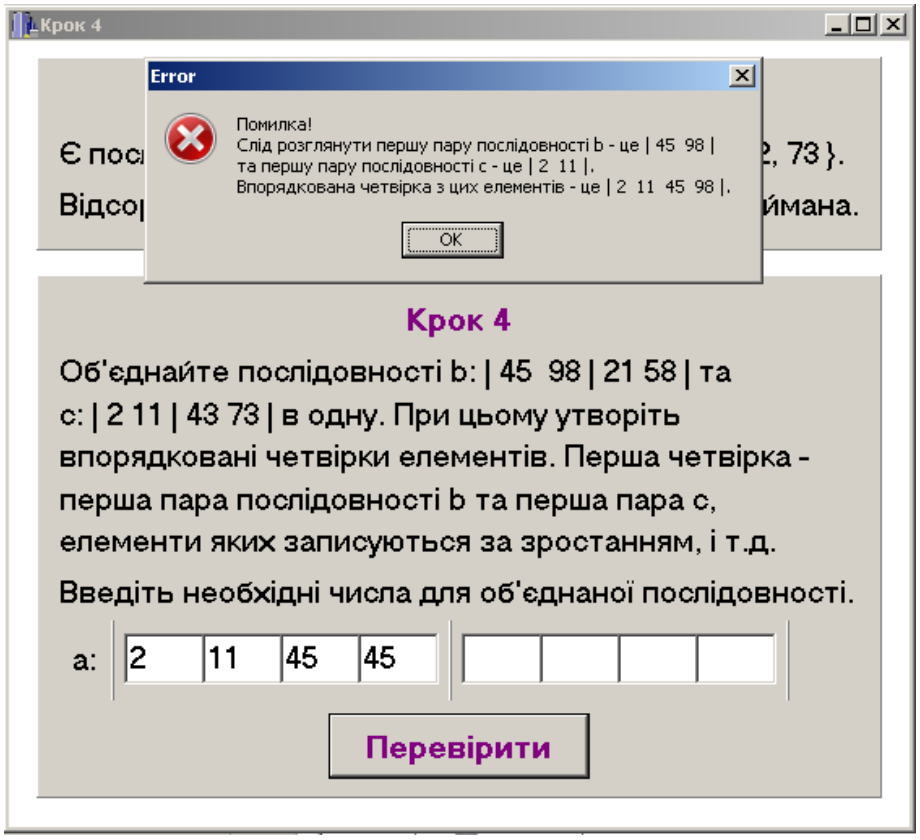

Рис. 4.27 – Помилка у першій четвірці на четвертому кроці

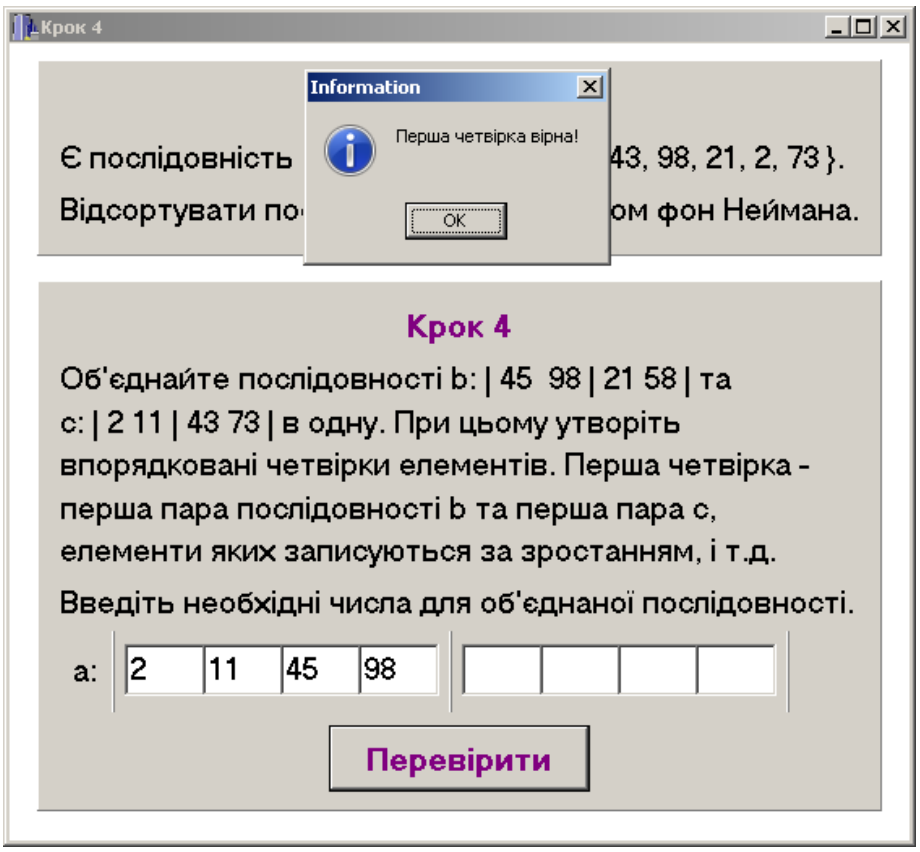

Рис. 4.28 – Підтвердження правильності першої четвірки на 4-ому кроці

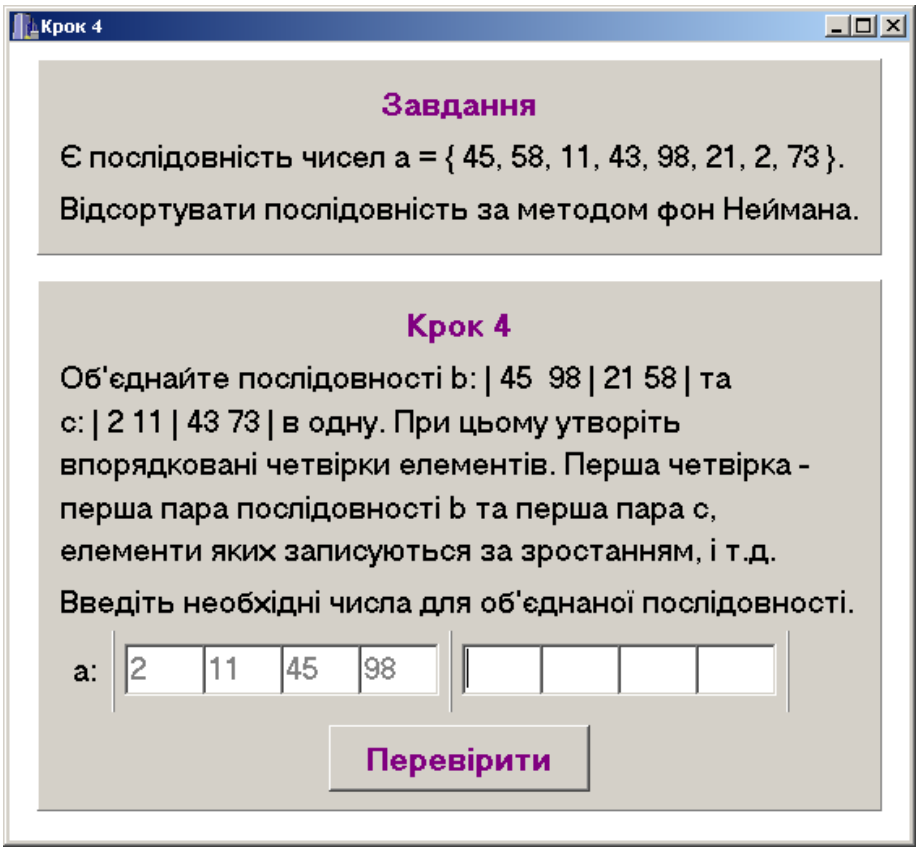

Рис. 4.29 – Перехід до заповнення другої четвірки на четвертому кроці

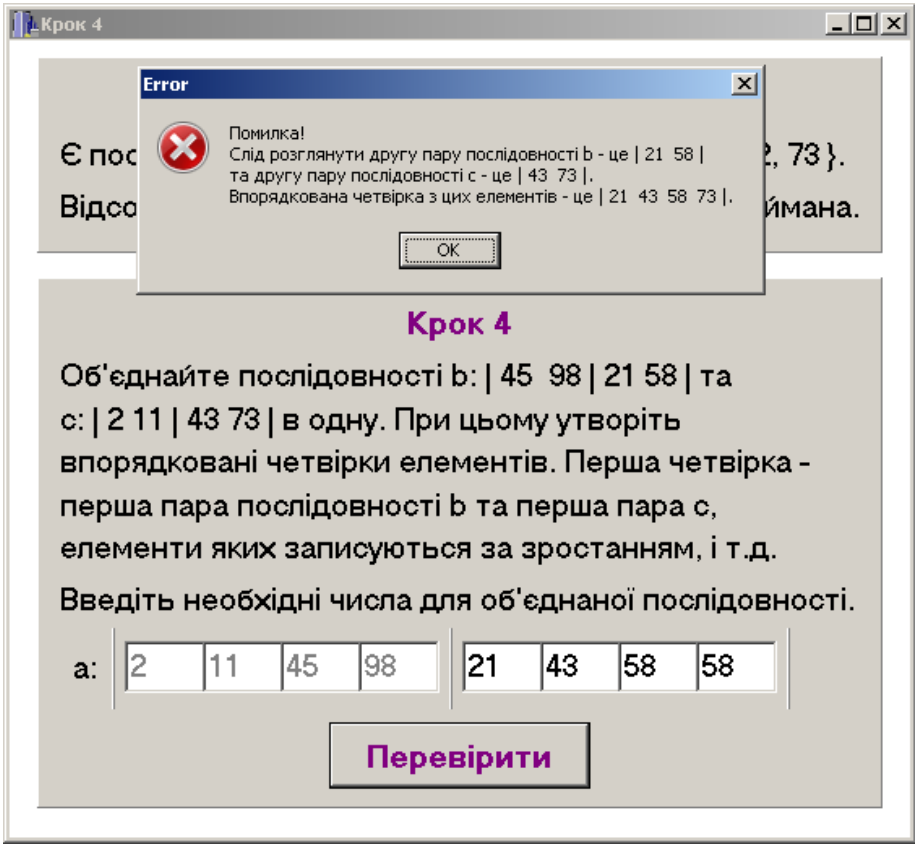

Рис. 4.30 – Помилка у другій четвірці на четвертому кроці

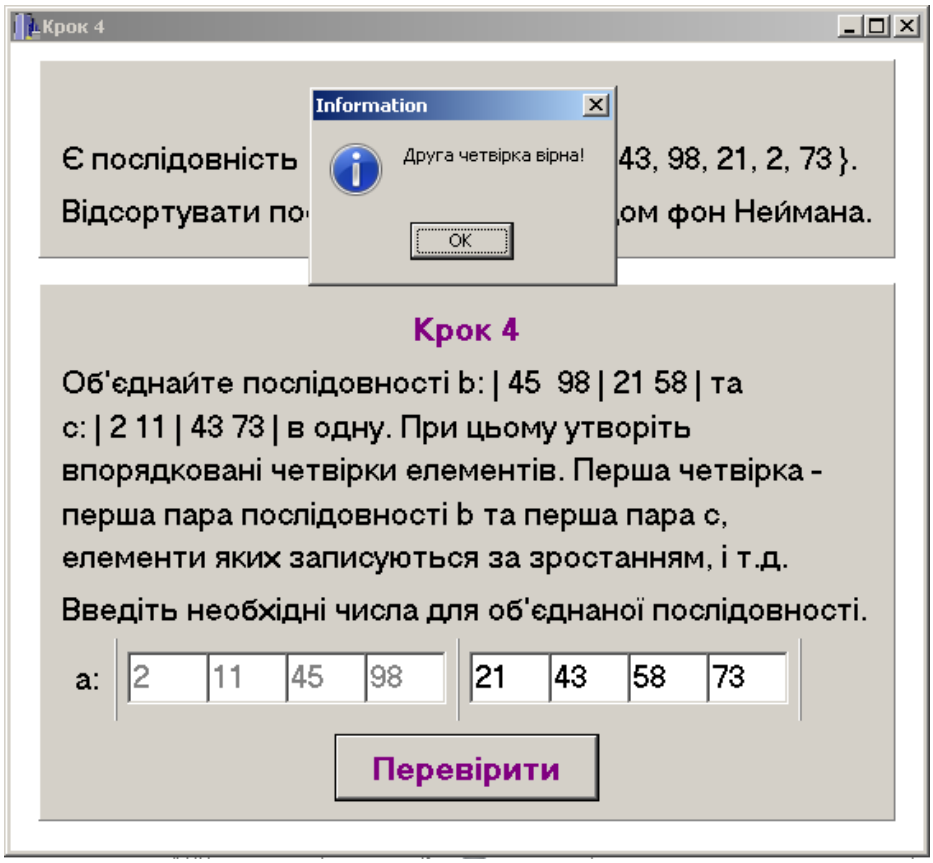

Рис. 4.31 – Підтвердження правильності другої четвірки на 4-ому кроці

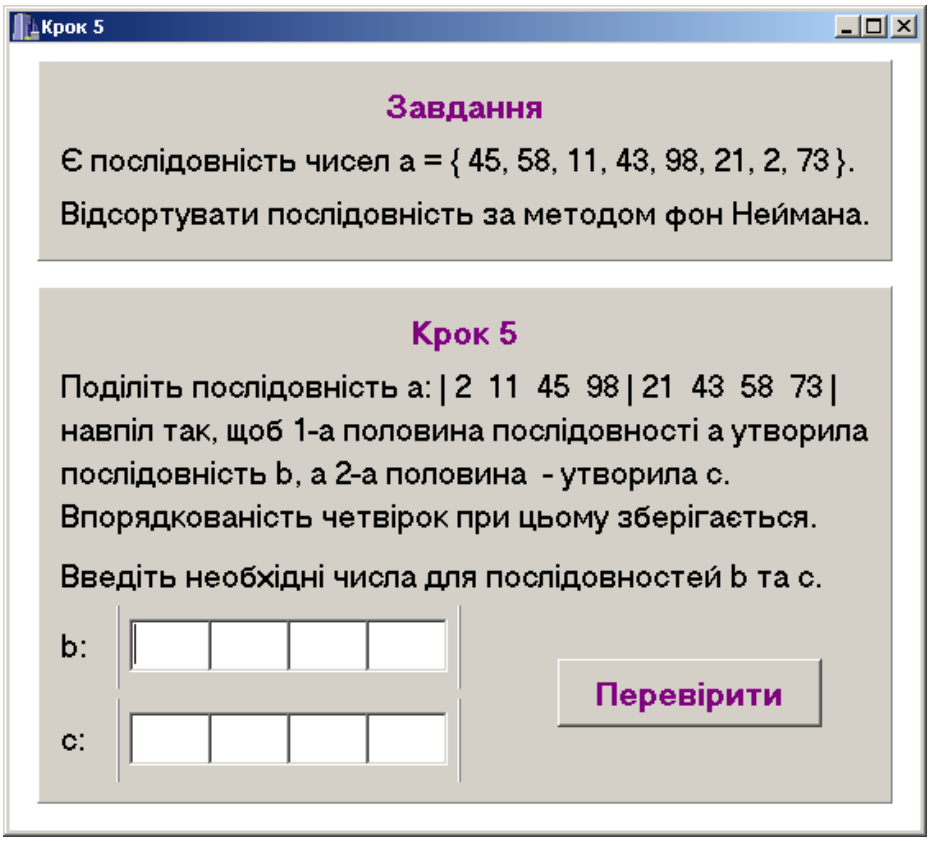

Рис. 4.32 – П’ятий крок

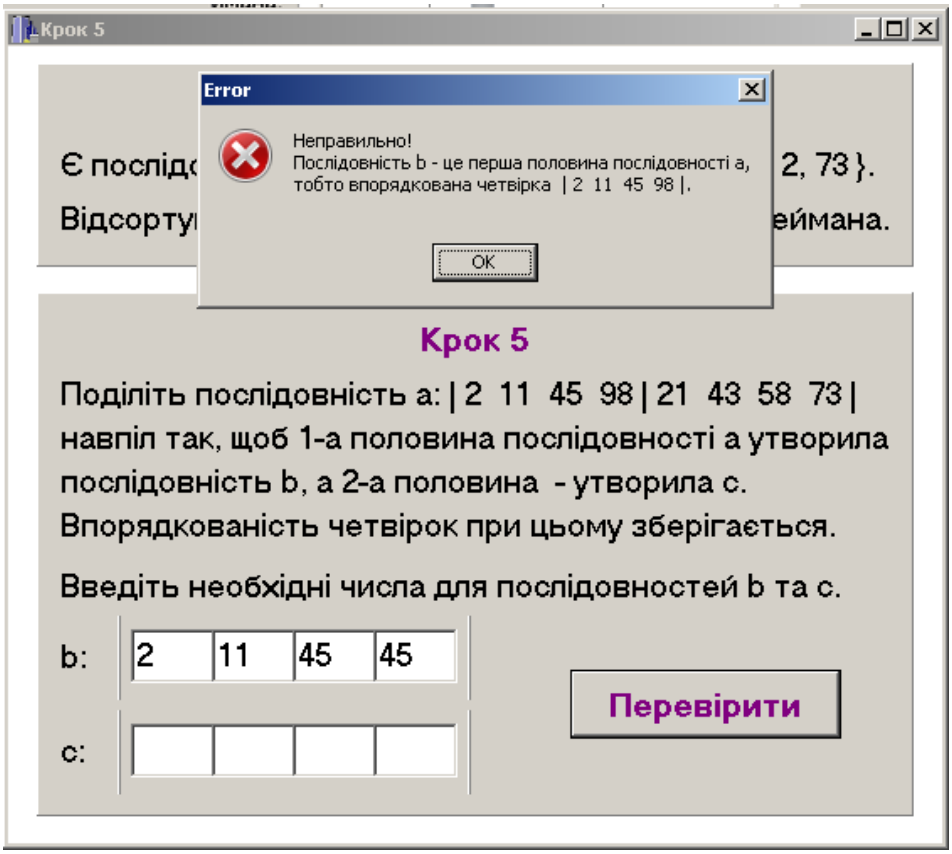

Рис. 4.33 – Помилка у послідовності *b* на п’ятому кроці

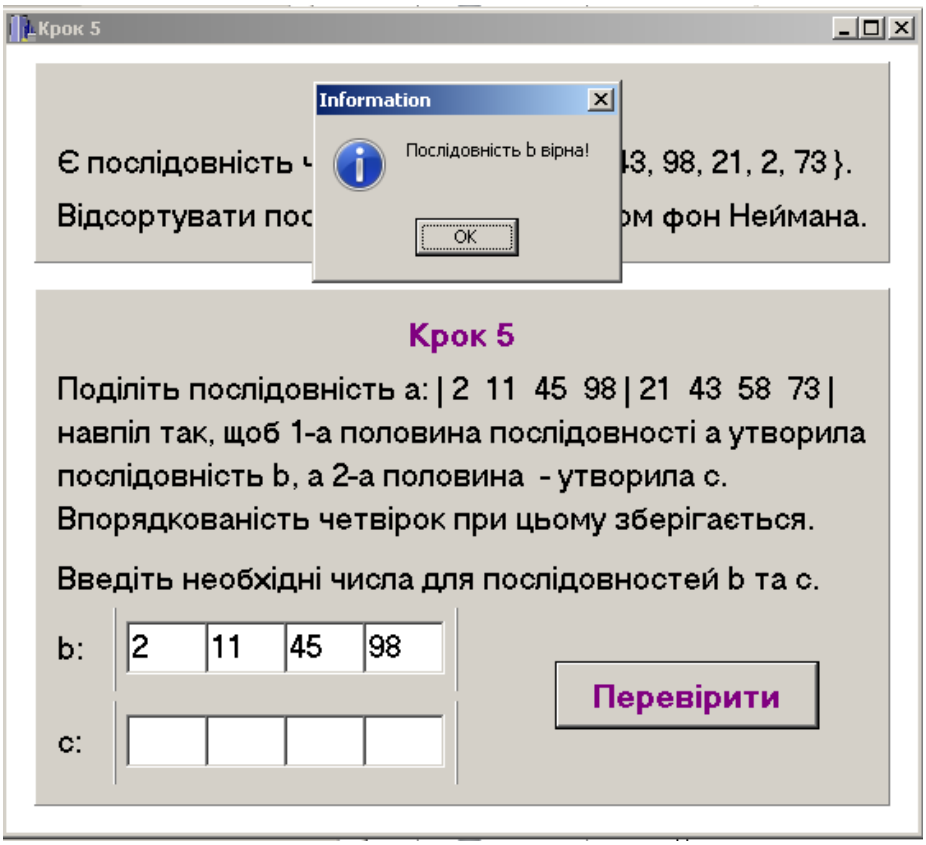

Рис. 4.34 – Підтвердження правильності послідовності *b* на п’ятому кроці

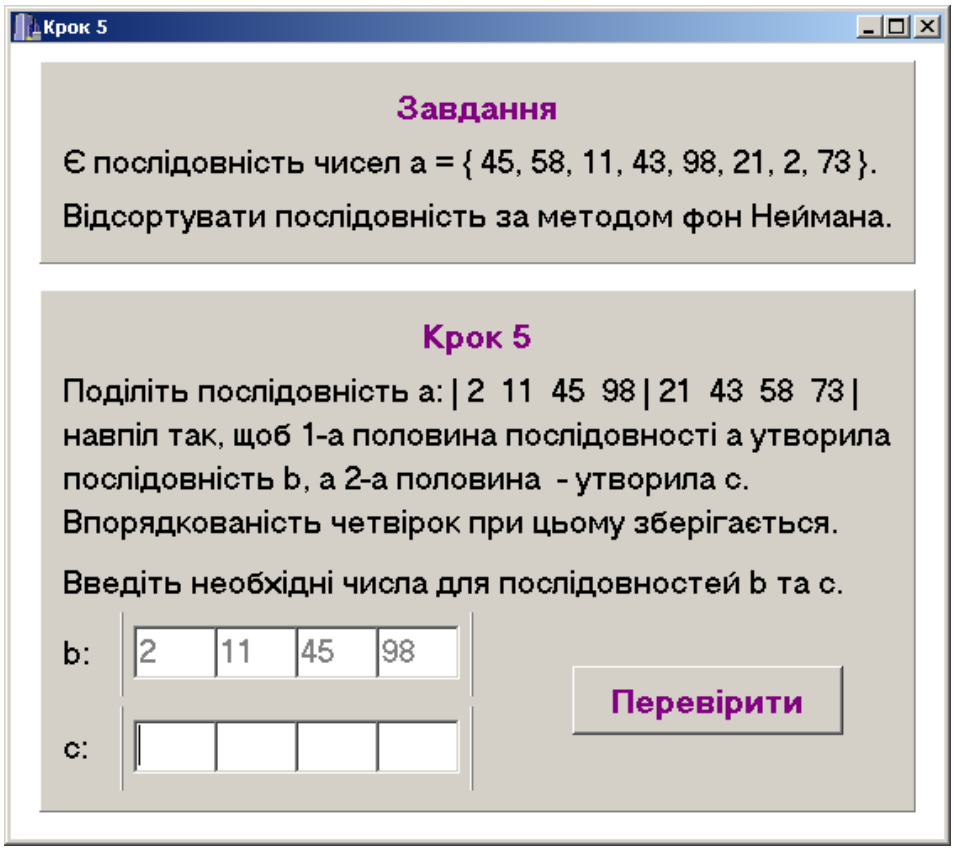

Рис. 4.35 – Перехід до заповнення послідовності *c* на п’ятому кроці

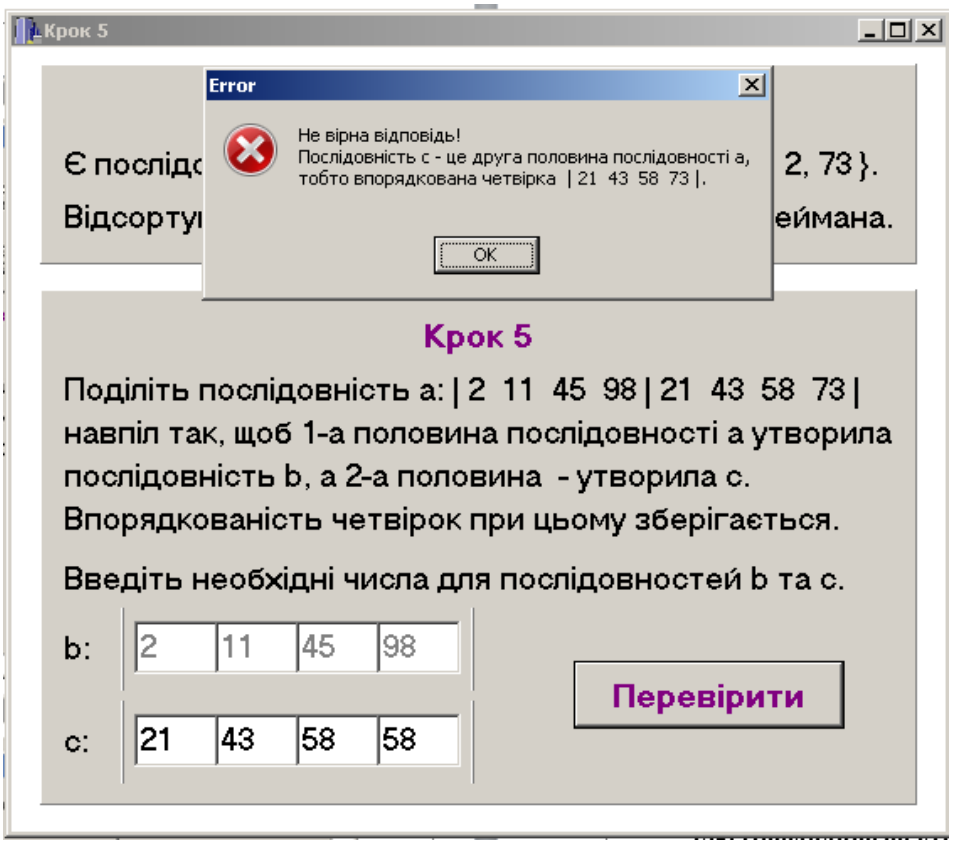

Рис. 4.36 – Помилка у послідовності *c* на п’ятому кроці

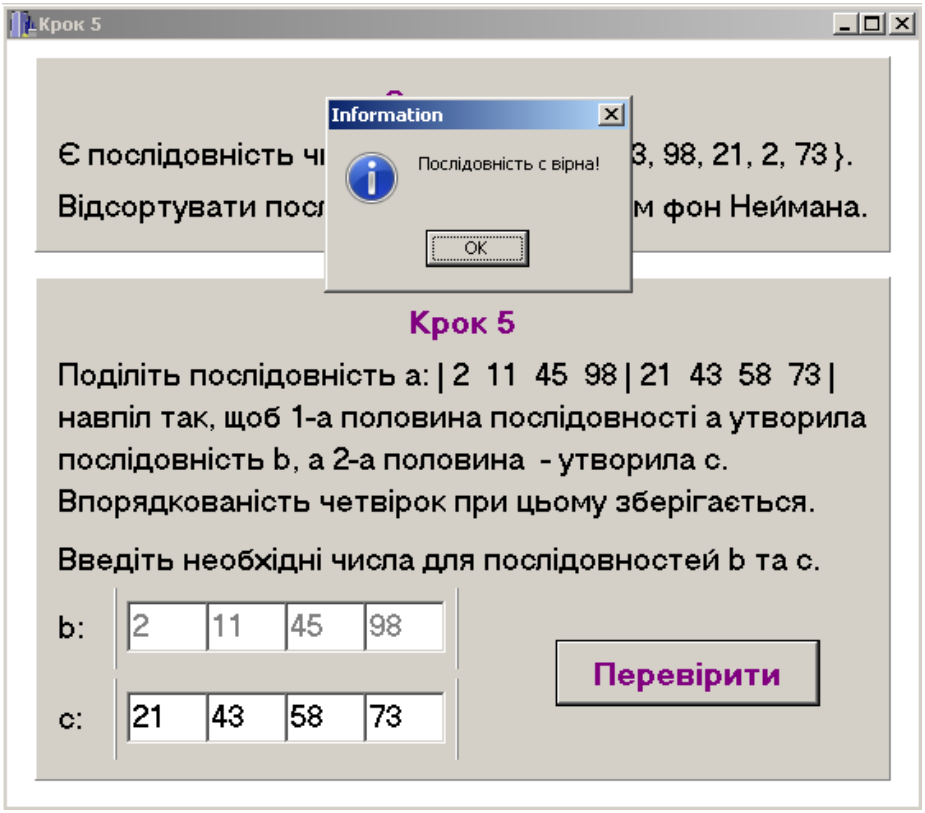

Рис. 4.37 – Підтвердження правильності послідовності *c* на п’ятому кроці

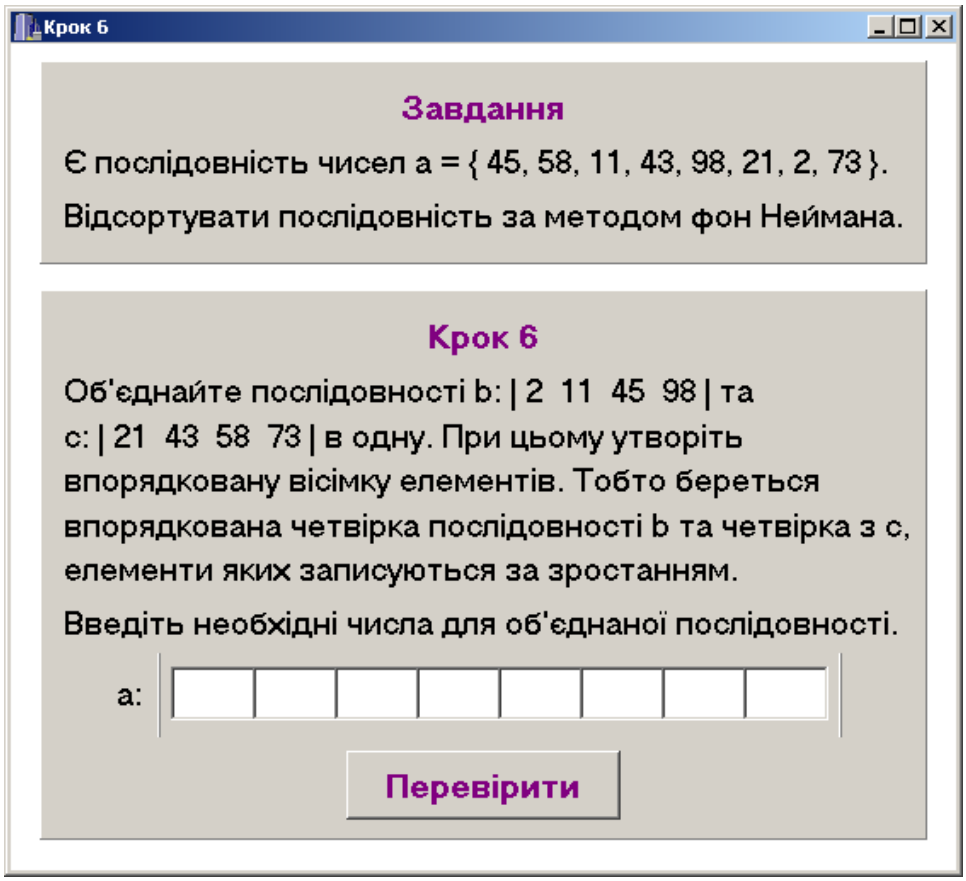

Рис. 4.38 – Шостий крок

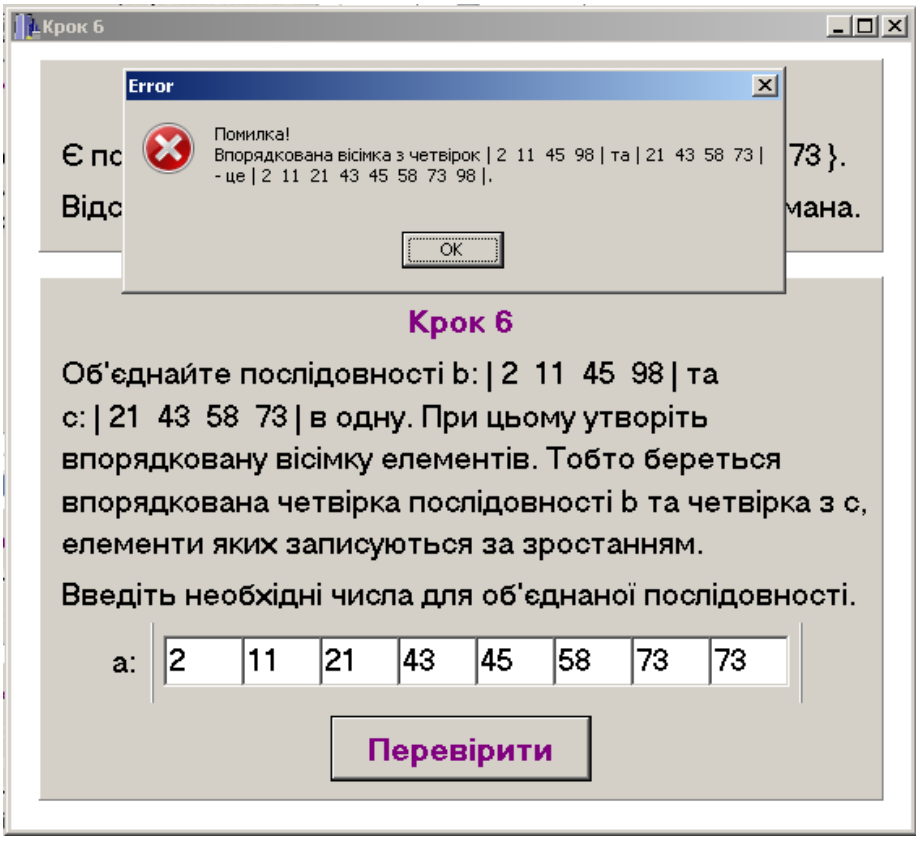

Рис. 4.39 – Помилка у впорядкованій вісімці на шостому кроці

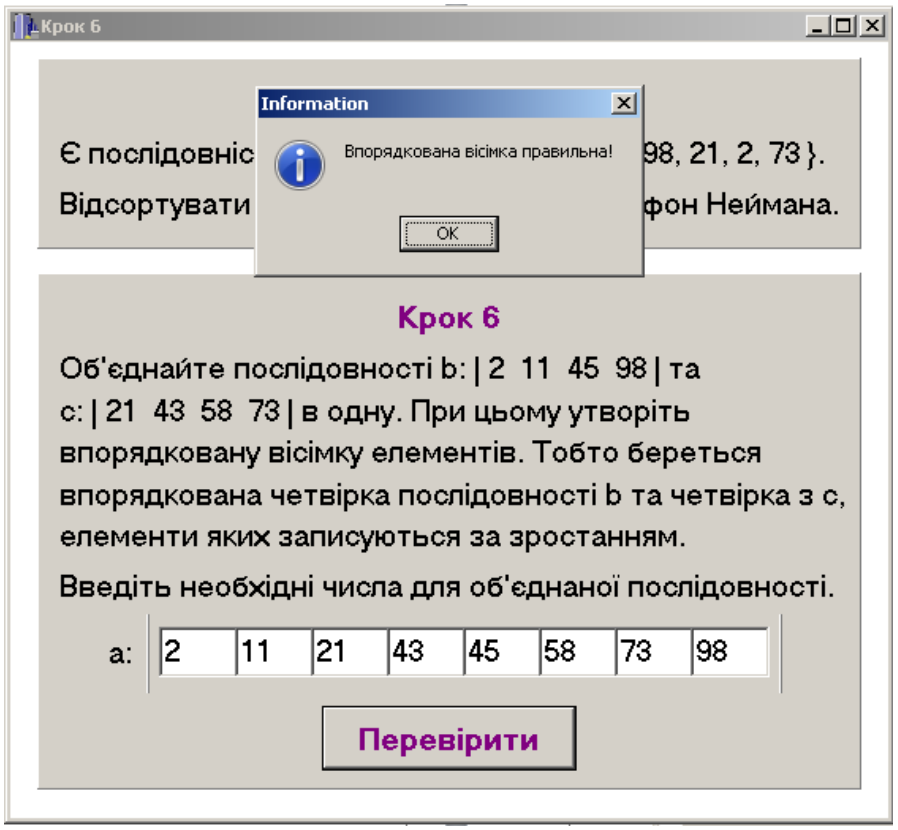

Рис. 4.40 – Підтвердження правильності на 6-ому кроці

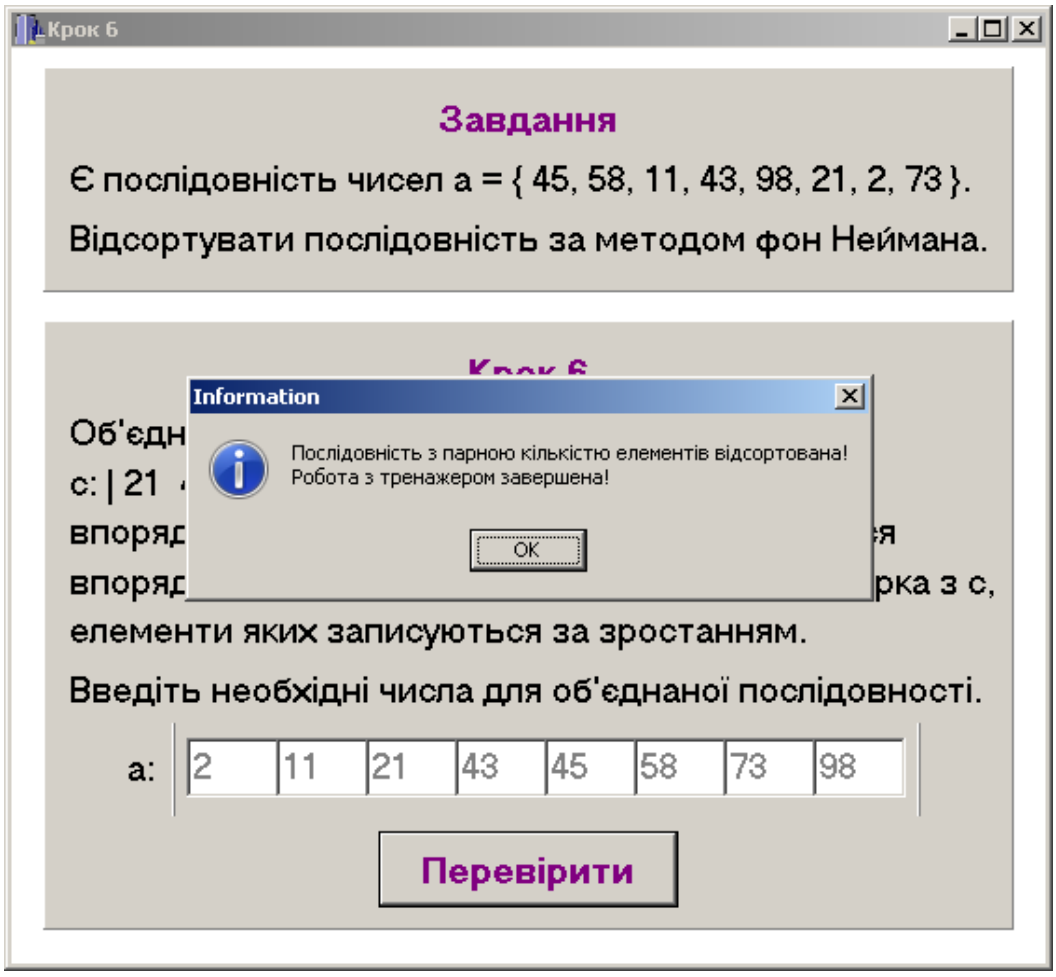

Рис. 4.41 – Кінець роботи тренажера з прикладом 1

4.2. Опис програми

Тренажер було створено на мові C++ у середовищі Borland Builder 5 [17-19].

Розглянемо, як створювався тренажер, на прикладі коду першого кроку для прикладу 1 (табл. 4.1).

Табл. 4.1 – Код першого кроку для прикладу 1

№ рядка	Програмний код
1	**void __fastcall TForm2::BitBtn1Click(TObject *Sender)** **{**
2	if (Form2->Tag==0) // перевірка послідовності b {
3	if ((StringGrid1->Cells[0][0]=="45") && (StringGrid1-> Cells[1][0]=="58") && (StringGrid1-> Cells[2][0]=="11") && (StringGrid1->Cells[3][0]=="43")) {
4	MessageDlg("Послідовність b правильна!", mtInformation, TMsgDlgButtons() << mbOK, 0);
5	Form2->Tag=1;
6	StringGrid2->Enabled=true;
7	StringGrid1->Enabled=false;
8	StringGrid2->SetFocus();
9	return; }
10	else {
11	MessageDlg("Помилка!\nПослідовність b – це перша половина послідовності a,\nтобто числа 45 58 11 43.", mtError, TMsgDlgButtons() << mbOK, 0);
12	StringGrid1->SetFocus();
13	return; } }
	… **}**

При клацанні по кнопці «Перевірити» (BitBtn1, табл. 4.1, рядок 1) відбувається перевірка спочатку послідовності *b*. Для цього використано поле Tag форми (рядок 2). Значення 0 означає, що перевіряємо послідовність *b*; значення 1 – послідовність *c*; значення 2 – перевірки закінчились.

Для запису послідовностей використали таблиці (для *b* – StringGrid1, для *c* – StringGrid2).

Якщо у таблиці StringGrid1 стоять правильні для *b* числа (рядок 3), то за допомогою процедури MessageDlg виводиться підтверджуюче повідомлення (рядок 4). В цьому випадку змінюється значення поля Tag на 1 (рядок 5); блокується таблиця StringGrid1 (рядок 7); розблоковується таблиця StringGrid2 (рядок 6); ставиться курсор в другу таблицю (рядок 8); здійснюється примусовий вихід з функції-обробника події (рядок 9).

Якщо у таблиці StringGrid1 стоять помилкові для *b* числа (рядки 10-13), то за допомогою процедури MessageDlg виводиться повідомлення про помилку з поясненням помилки (рядок 11). Курсор знову ставиться в першу таблицю, в яку ввели хибні числа (рядок 12); здійснюється примусовий вихід з функції-обробника події (рядок 13).

Перевірка послідовності *c* аналогічна (табл. 4.2).

Табл. 4.2 – Код першого кроку для прикладу 1 (продовження)

№ рядка	Програмний код
1	**void __fastcall TForm2::BitBtn1Click(TObject *Sender)**
	{
	…
2	if (Form2->Tag==1) // перевірка послідовності c
	{
3	if ((StringGrid2->Cells[0][0]=="98") && (StringGrid2-> Cells[1][0]=="21") && (StringGrid2-> Cells[2][0]=="2") && (StringGrid2->Cells[3][0]=="73"))
	{
4	MessageDlg("Послідовність c правильна!", mtInformation, TMsgDlgButtons() << mbOK, 0);
5	Form2->Tag=2;
6	StringGrid2->Enabled=false;
7	Form3->Show(); // перехід на наступний крок
8	Form3->Edit1->SetFocus();
9	Form2->Hide();
10	return;
	}
11	else
	{
12	MessageDlg("Не вірно!\n Послідовність c - це друга половина послідовності a,\nтобто числа 98 21 2 73.", mtError, TMsgDlgButtons() << mbOK, 0);
13	StringGrid2->SetFocus();
14	return;
	}
	}
	}

Різниця полягає лише в тому, що при правильній відповіді для *c* відбувається перехід на наступний крок (табл. 4.2, рядки 7-9).

Розглянемо, як створювався крок 2 прикладу 1 (табл. 4.3).

Оскільки, на другому кроці потрібно відобразити впорядковані пари, які слід відділити вертикальною рискою, то компонент таблиця (StringGrid) вже не підходить. Тому на цьому кроці для пар були використані компоненти Edit, а для вертикальних рисок – компонент Bevel. Для цього компонента властивість Shape (форма) була налаштована як bsLeftLine, тобто – вертикальна риска, а властивість Style (стиль) була налаштована як bsRaised.

Табл. 4.3 – Код другого кроку для прикладу 1

№ рядка	Програмний код
1	**void __fastcall TForm3::BitBtn1Click(TObject *Sender)**
	{
2	if (Form3->Tag==0) // перша пара
	{
3	if ((Edit1->Text=="45") && (Edit2->Text=="98"))
	{
4	MessageDlg("Перша пара вірна!", mtInformation, TMsgDlgButtons() << mbOK, 0);
5	Form3->Tag=1;
6	Edit3->Enabled=true;
7	Edit4->Enabled=true;
8	Edit1->Enabled=false;
9	Edit2->Enabled=false;
10	Edit3->SetFocus();
11	return;
	}
12	else
	{
13	MessageDlg("Помилка!\n" "Слід взяти перший елемент послідовності b - це 45,\n" "та перший елемент послідовності c - це 98, та порівняти їх.\n" "Оскільки 45 < 98, то перша впорядкована пара - це | 45 98 |.", mtError, TMsgDlgButtons() << mbOK, 0);
14	Edit1->SetFocus();
15	return;
	}
	}
	…
	}

При клацанні по кнопці «Перевірити» (BitBtn1, табл. 4.3, рядок 1) відбувається перевірка спочатку першої пари. Для цього використано поле Tag форми (рядок 2). Значення 0 означає, що перевіряємо першу пару; значення 1 – другу; значення 2 – третю; значення 3 – четверту; значення 5 – кінець перевіркам.

Для першої пари використали компонент Edit1 та Edit2. Якщо в них стоять правильні для першої пари числа (рядок 3), то за допомогою MessageDlg виводиться підтверджуюче повідомлення (рядок 4). В цьому випадку змінюється значення поля Tag на 1 (рядок 5); блокуються компоненти Edit (рядки 8-9); розблоковуються компоненти Edit для другої пари (рядки 6-7); ставиться курсор в перший компонент другої пари (рядок 10); здійснюється примусовий вихід з функції (рядок 11).

Якщо для першої пари стоять помилкові числа (рядки 12-15), то за допомогою MessageDlg виводиться повідомлення про помилку з поясненням (рядок 13). Курсор знову ставиться в перший компонент першої пари (рядок 14); здійснюється примусовий вихід з функції (рядок 15).

Перевірка послідовності наступних пар аналогічна. Різниця є лише для останньої пари (табл. 4.4), оскільки, при правильній відповіді відбувається перехід на наступний крок (табл. 4.4, рядки 8-10).

Табл. 4.4 – Код другого кроку для прикладу 1 (продовження)

№ рядка	Програмний код
1	**void __fastcall TForm3::BitBtn1Click(TObject *Sender)** **{** …
2	if (Form3->Tag==3) // четверта пара {
3	if ((Edit7->Text=="43") && (Edit8->Text=="73")) {
4	MessageDlg("Четверта пара вірна!", mtInformation, TMsgDlgButtons() << mbOK, 0);
5	Form3->Tag=4;
6	Edit7->Enabled=false;
7	Edit8->Enabled=false;
8	Form4->Show(); // перехід на наступний крок
9	Form4->Edit1->SetFocus();
10	Form3->Hide();

Продовження табл. 4.4 – Код другого кроку для прикладу 1 (продовження)

№ рядка	Програмний код

11	return;
	}
	else
	{

12	
13	MessageDlg("Похибка!\n" "Слід взяти четвертий елемент послідовності b - це 43,\n" "та четвертий послідовності c - це 73, та порівняти їх.\n" "Оскільки 43 < 73, то четверта пара послідовності - це | 43 73 |.", mtError, TMsgDlgButtons() << mbOK, 0);
14	Edit7->SetFocus();
15	return; } } }

Повний код програми (для прикладу 1) викладено у додатку Б.

ВИСНОВКИ

В рамках магістерської роботи було вивчено тему «Сортування методом фон Неймана». Було вивчено різні інформаційні джерела, в тому числі і матеріал дистанційного курсу «Алгоритми і структури даних» ПУЕТ.

Було оглянуто ряд тренажерів схожої тематики.

Для розробки тренажеру (для прикладу 1) було використано ілюстративний приклад з дистанційного курсу, де кількість елементів, що сортуються, парна. На базі прикладу було розроблено алгоритм тренажеру прикладу 1. Була створена блок-схема типових кроків алгоритму, а саме: блок-схема для першого та другого кроків прикладу 1.

Сортування набору чисел з непарною кількістю елементів для прикладу 2 було зроблено самостійно. На базі сортування цього набору чисел було розроблено алгоритм тренажеру прикладу 2.

Алгоритми прикладів 1 та 2 було реалізовано на мові C++ у середовищі Borland Builder 5. При цьому були використані візуальні компоненти. Дизайн та принцип роботи програми для обох прикладі подібний. Це зроблено спеціально для зручності користувачів програми.

Програма видає підтверджуючі повідомлення при правильних відповідях та надає пояснення у випадку помилок. Помилки автоматично не виправляються, їх повинен виправляти користувач.

Програма протестована на предмет помилок та зручності використання.

Розроблено інструкцію по використанню програми. Описано, як створювалась програма.

Тренажер передано до відділу автоматизації та роботи в ЄДБО ПУЕТ з метою впровадження в дистанційний курс «Алгоритми і структури даних» ПУЕТ, що підтверджує акт впровадження.

Результати роботи було представлені на науково-практичному семінарі «Комп’ютерні науки і прикладна математика» [21].

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Кильник В. В. Алгоритм тренажера з теми «Сортування методом перемішування» дистанційного курсу «Алгоритми та структури даних» / В. В. Кильник // VIII Всеукр. наук.-практ. конф. за міжнародною участю «Інформатика та системні науки» (м. Полтава, 16-18 березня 2017 р.) / за ред. О. О. Ємця. – С. 135-137. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/5480.

2. Олексійчук Ю. Ф. Програмна реалізація елементів тренажеру з теми «Сортування бульбашками» дисципліни «Аналіз алгоритмів» / Ю. Ф. Олексійчук, В. О. Голубенко // Комп'ютерні науки і прикладна математика (КНіПМ-2018) / за ред. О. О. Ємця. – Полтава, ПУЕТ, 2018. – С.11-16. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/6481.

3. Русін В. С. Програмна реалізація елементів тренажеру з теми «Аналіз алгоритму сортування вставками» дисципліни «Аналіз алгоритмів» / В. С. Русін, Ю. Ф. Олексійчук // VIII Всеукр. наук.-практ. конф. за міжнародною участю «Інформатика та системні науки» (м. Полтава, 16-18 березня 2017 р.) / за ред. О. О. Ємця. – С. 236-237. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/5654.

4. Щербак О. В. Розробка алгоритму тренажера з теми «Сортування вибором та сортування обміном» дистанційного курсу «Алгоритми та структури даних» / О. В. Щербак // VIII Всеукр. наук.-практ. конф. за міжнародною участю «Інформатика та системні науки» (м. Полтава, 16-18 березня 2017 р.) / за ред. О. О. Ємця. – С. 297-299. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/5479.

5. Ємець Ол-ра О. Про тренажер «Обчислення коефіцієнтів конкордації з урахуванням зв'язаних рангів» / Ол-ра О. Ємець // Інформатика та системні науки (ІСН-2015): матеріали VI Всеукр. наук.-практ. конф. (м. Полтава, 19-21 березня 2015

р.) / за ред. Ємця О. О. – Полтава: ПУЕТ, 2015. – С. 161-171. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/2492.

6. Чуб О. І. Тренажер «Рекурсивні алгоритми» / О. І. Чуб, О.О. Ємець // Комп'ютерні науки і прикладна математика (КНіПМ-2019): матеріали наук.-практ. семінару. Випуск 4. / За ред. Ємця О. О. – Полтава: Кафедра ММСІ ПУЕТ, 2019. – С. 16-19. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/7456.

7. Григор`єв В. В. Тренажер «Побудова математичної моделі однієї лінійної задачі» / В. В. Григор`єв, О.О. Ємець // Комп'ютерні науки і прикладна математика (КНіПМ-2019): матеріали наук.-практ. семінару. Випуск 4. / За ред. Ємця О. О. – Полтава: Кафедра ММСІ ПУЕТ, 2019. – С. 12-15. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/7455.

8. Ємець О. О. Про розробку тренажерів для дистанційних курсів кафедрою ММСІ ПУЕТ / О. О. Ємець // Інформатика та системні науки (ІСН-2015): матеріали VI Всеукр. наук.-практ. конф. за міжнародною участю (м. Полтава, 19-21 березня 2015 р.) / за ред. Ємця О. О. – Полтава: ПУЕТ, 2015. – С. 152-161. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/2488.

9. Інформатика та системні науки (ІСН-2013): Матеріали IV Всеукраїнської науково-практичної конференції (м. Полтава, 21-23 березня 2013 р.) / за ред. д.ф.-м.н., проф. Ємця О. О. – Полтава: ПУЕТ, 2013. – 323 с. – Режим доступу: http://dspace.uccu.org.ua/handle/123456789/1552. (ISBN 978-966-184-211-2, 18,8 др. арк.).

10. Інформатика та системні науки (ІСН-2014): матеріали V Всеукр. наук.-практ. конф. (м. Полтава, 13-15 березня 2014 р.) / за ред. О.О.Ємця. – Полтава: ПУЕТ, 2014. – 335 с. – Режим доступу: http://dspace.uccu.org.ua/handle/123456789/1942. (ISBN 978-966-184-152-8, 19,5 др. арк.).

11. Інформатика та системні науки (ІСН-2015): матеріали VI Всеукр. наук.-практ. конф. за міжнародною участю (м. Полтава, 19-21 березня 2015 р.) / за ред. Ємця О.О. – Полтава: ПУЕТ, 2015. – 402 с. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/2616.

12. Інформатика та системні науки (ІСН-2016): матеріали VII Всеукр. наук.-практ. конф. за міжнародною участю (м. Полтава, 10-12 березня 2016 р.) / за ред. О.О. Ємця. – Полтава: ПУЕТ, 2016. – 362 с. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/3676.

13. Інформатика та системні науки (ІСН-2017): матеріали VIII Всеукр. наук.-практ. конф. за міжнародною участю (м. Полтава, 16-18 березня 2017 р.) / за ред. Ємця О.О. – Полтава: ПУЕТ, 2017. – 333 с. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/5798.

14. Ємець Ол-ра О. Дистанційний курс Полтавського університету економіки і торгівлі «Алгоритми і структури даних» / Ол-ра О. Ємець. – [Електронний ресурс].

15. Соколов О. Ю. Інформатика для інженерів / О. Ю. Соколов, І. Т. Зарецька, Г. М. Жолткевич, О. В. Ярова. – Харків: Факт, 2006. – 424 с.

16. Схемы алгоритмов, программ данных и систем. Условные обозначения и правила выполнения: ГОСТ 19.701-90. [Введен 1992-01-01]. – М.: Изд-во стандартов, 1990. – 25 с.

17. Дейтел Х. М. Как программировать на С++ / Х. М. Дейтел, П. Дж. Дейтел. – М.: Бином, 2001. – 1152 с.

18. Культин Н. Б. С++ Builder в задачах и примерах / Н. Б. Культин. – СПб.: БХВ-Петербург, 2005. – 336 с.

19. Седжвик Р. Фундаментальные алгоритмы на С++: Части 1-4: Анализ. Структуры данных. Сортировка. Поиск / Р. Седжвик. – К.: Из-во «Диасофт», 2001. – 688 с.

20. Ємець О. О. Методичні рекомендації до виконання бакалаврської роботи для студентів спеціальності 122 «Комп`ютерні науки та інформаційні технології» освітня програма «Комп`ютерні науки» галузь знань – 12 «Інформаційні технології» / О. О. Ємець. – Полтава: ПУЕТ, 2017. – 71 с.

21. Козодуб В. С. Тренажер «Сортування фон Неймана» / В. С. Козодуб, О.О. Ємець // Комп’ютерні науки і прикладна математика (КНіПМ-2020): матеріали наук.-практ. семінару. Випуск 5. / За ред. Ємця О. О. – Полтава: Кафедра ММСІ ПУЕТ, 2020. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/8267.

ДОДАТОК А

ПРИКЛАД 2

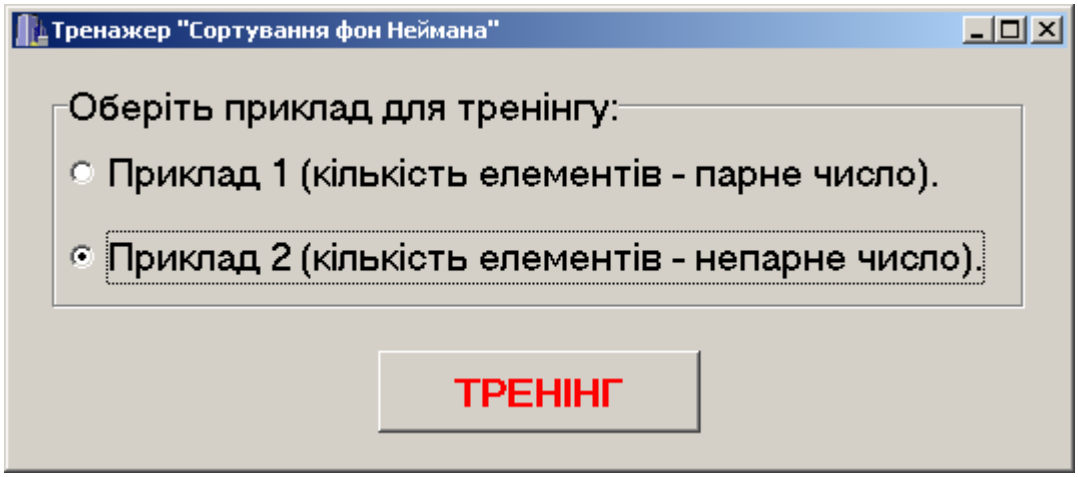

Рис. А.1 – Друге вікно програми з обраним прикладом 2

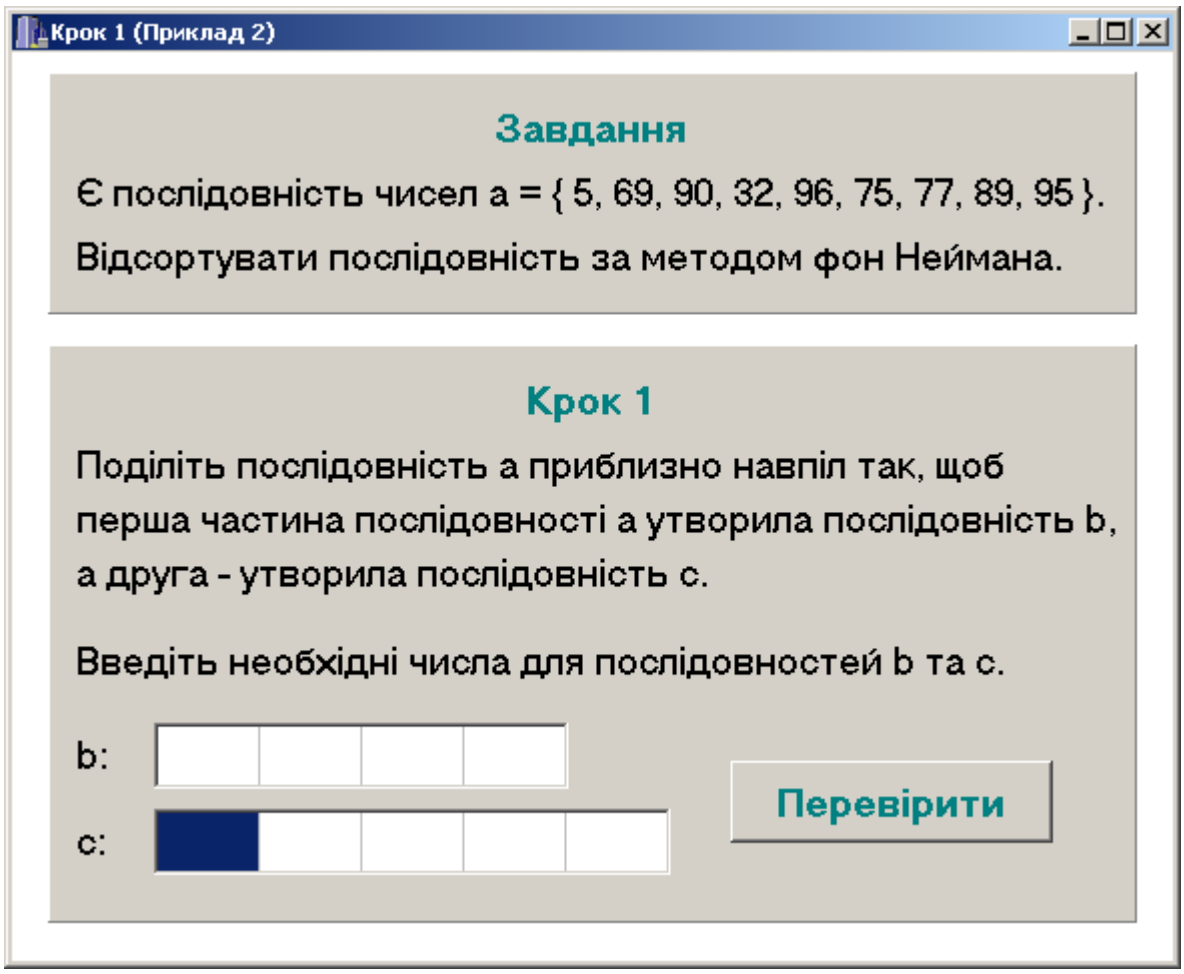

Рис. А.2 – Перший крок

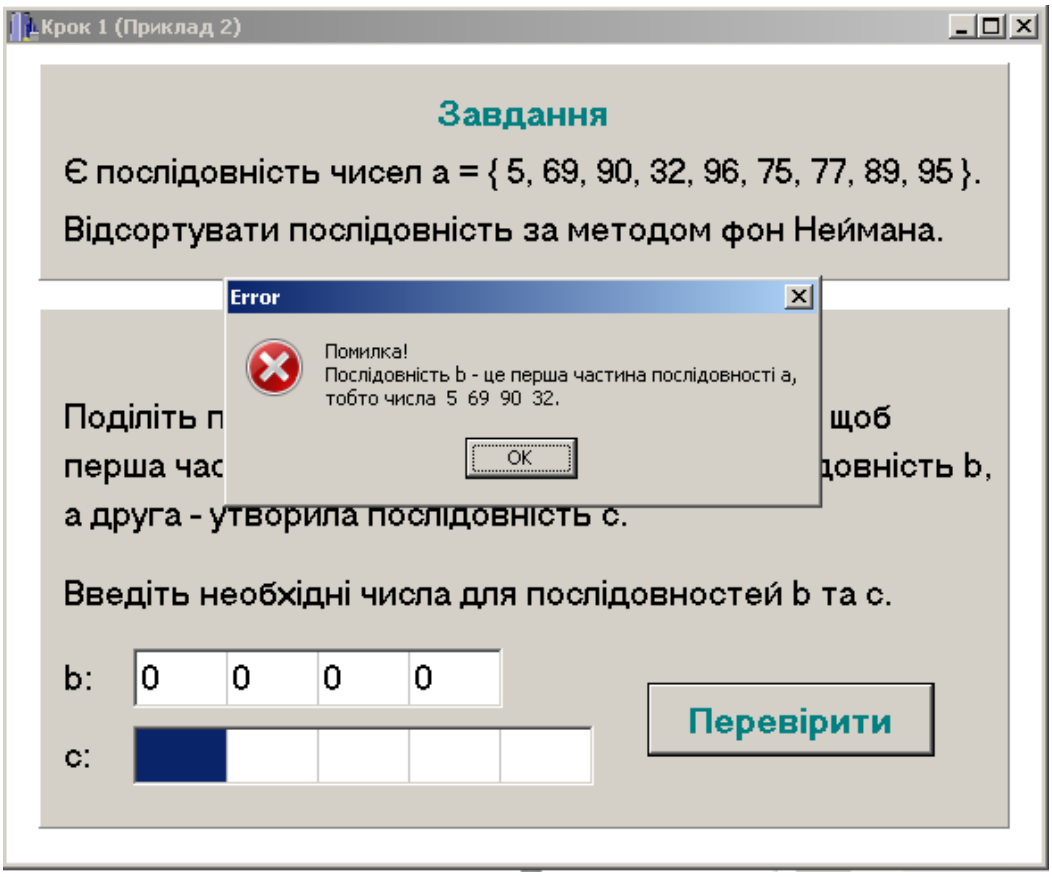

Рис. А.3 – Помилка у послідовності *b* на першому кроці

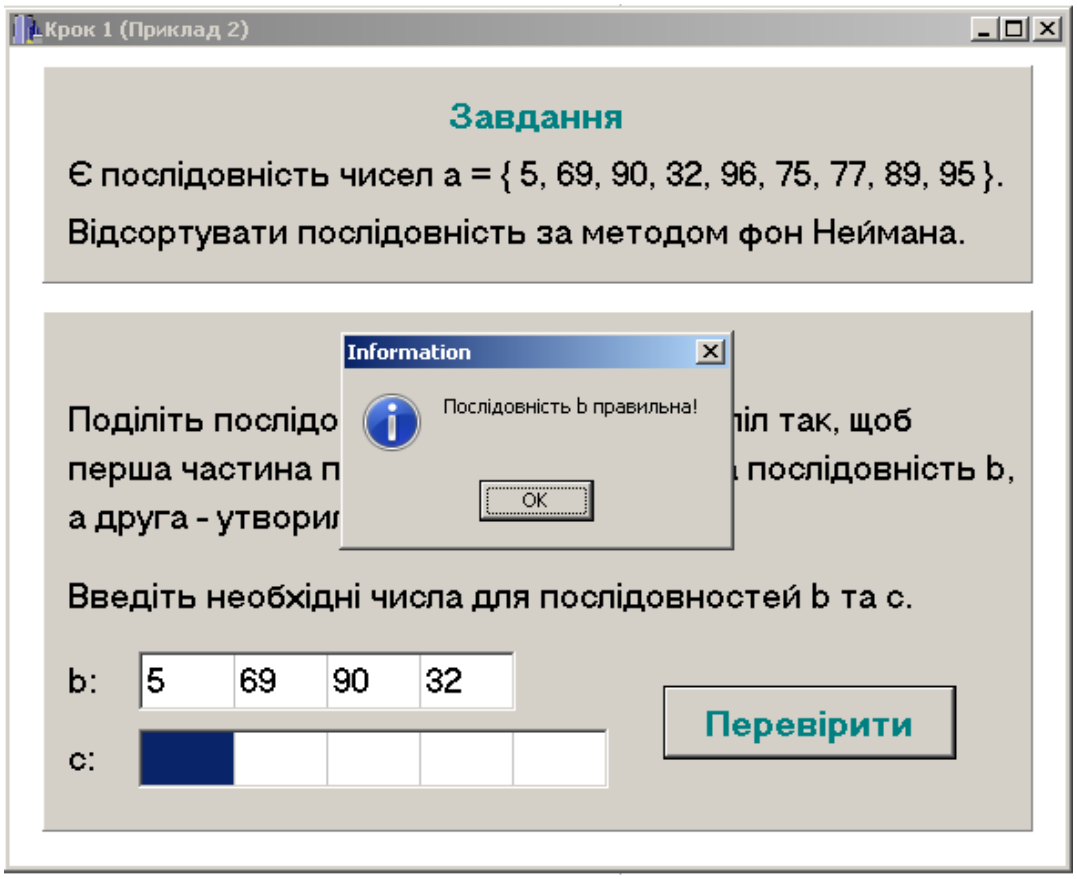

Рис. А.4 – Підтвердження правильності послідовності *b* на першому кроці

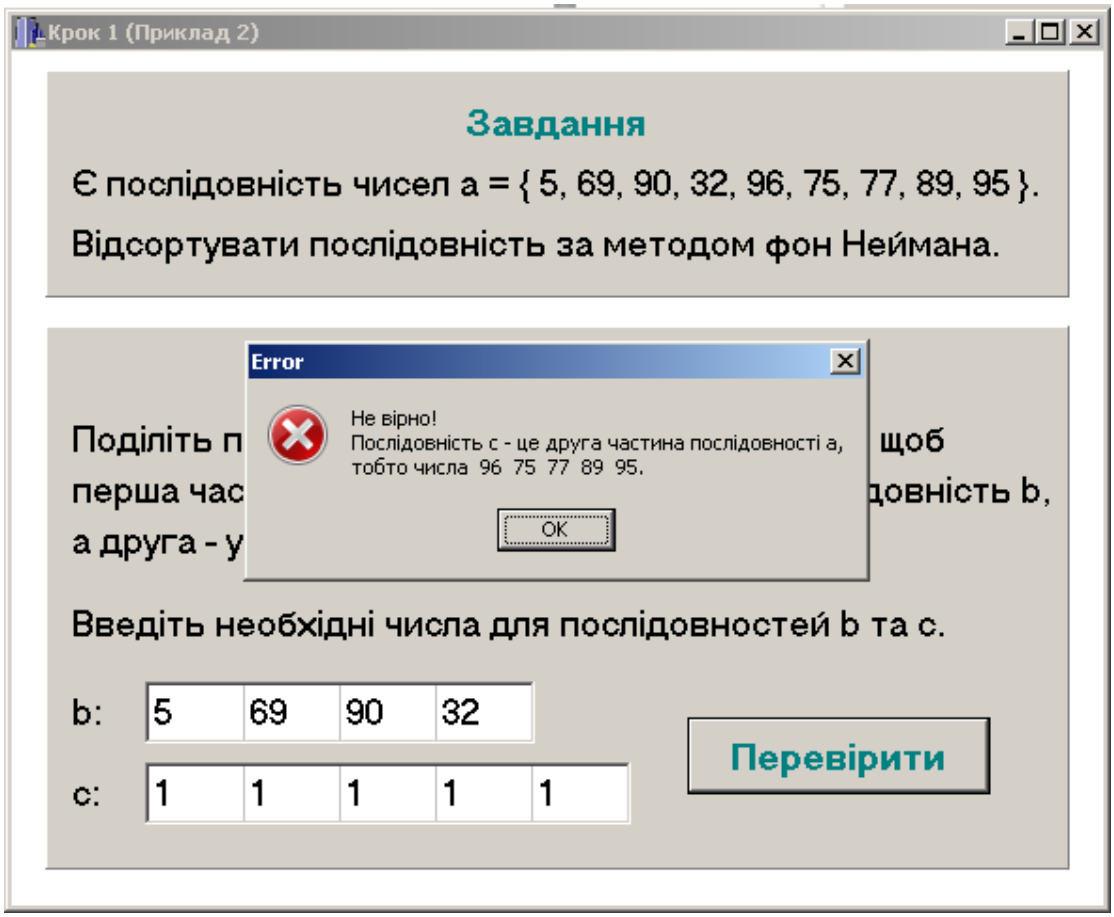

Рис. А.5 – Помилка у послідовності *c* на першому кроці

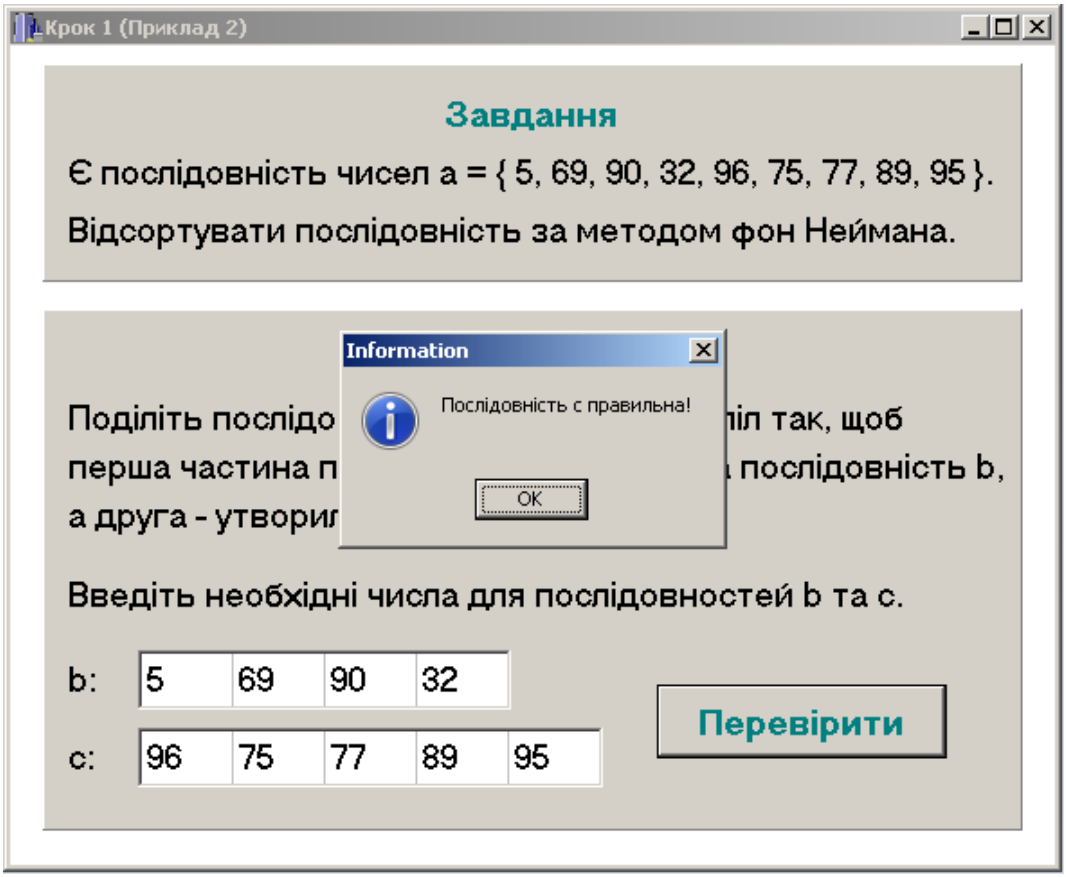

Рис. А.6 – Підтвердження правильності послідовності *c* на першому кроці

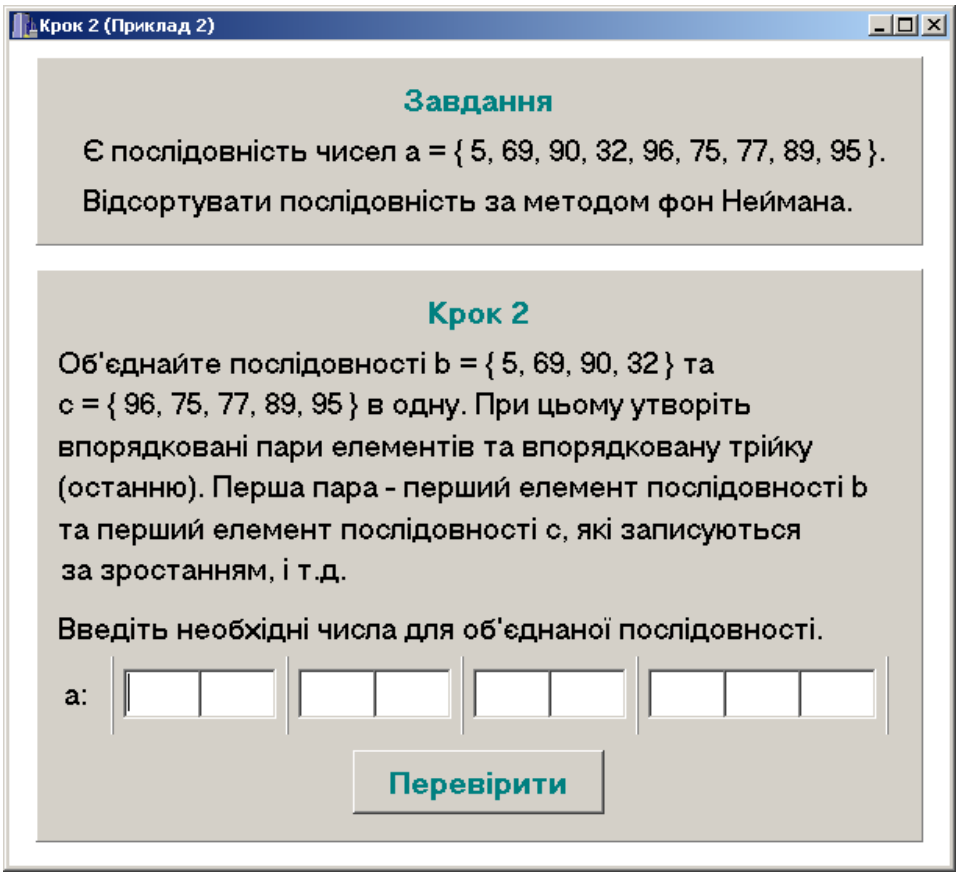

Рис. А.7 – Другий крок

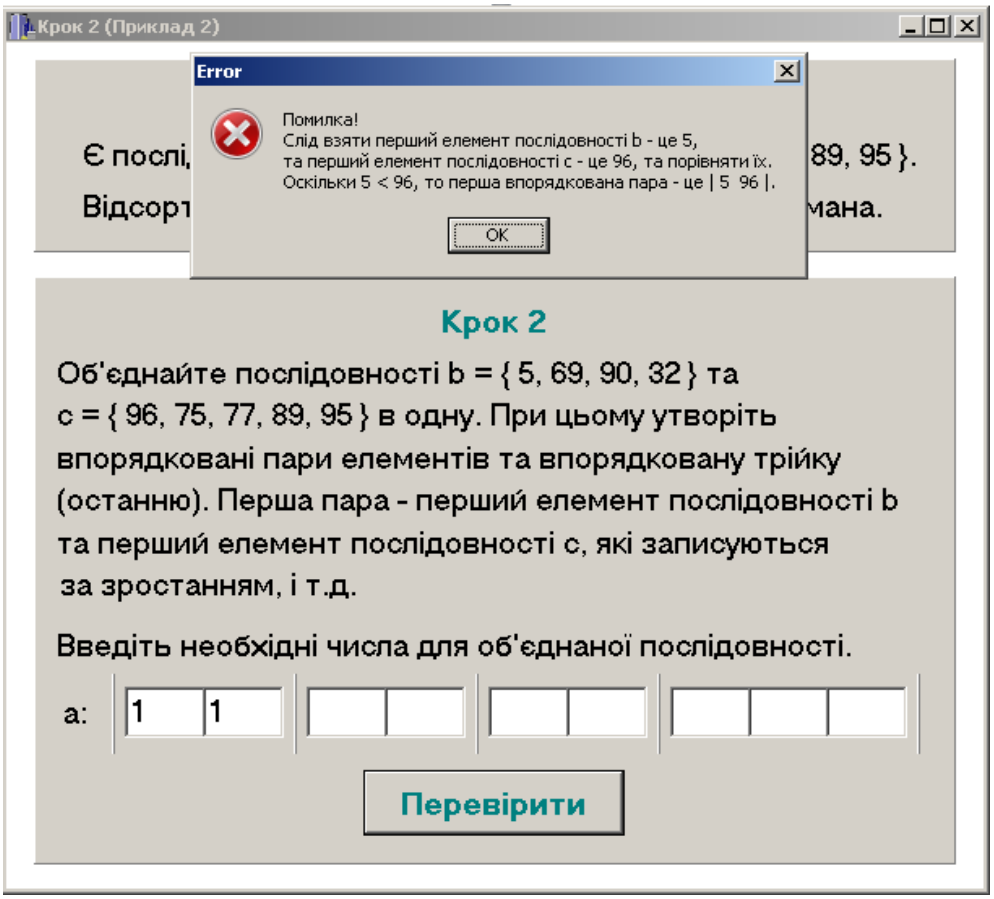

Рис. А.8 – Помилка у першій парі на другому кроці

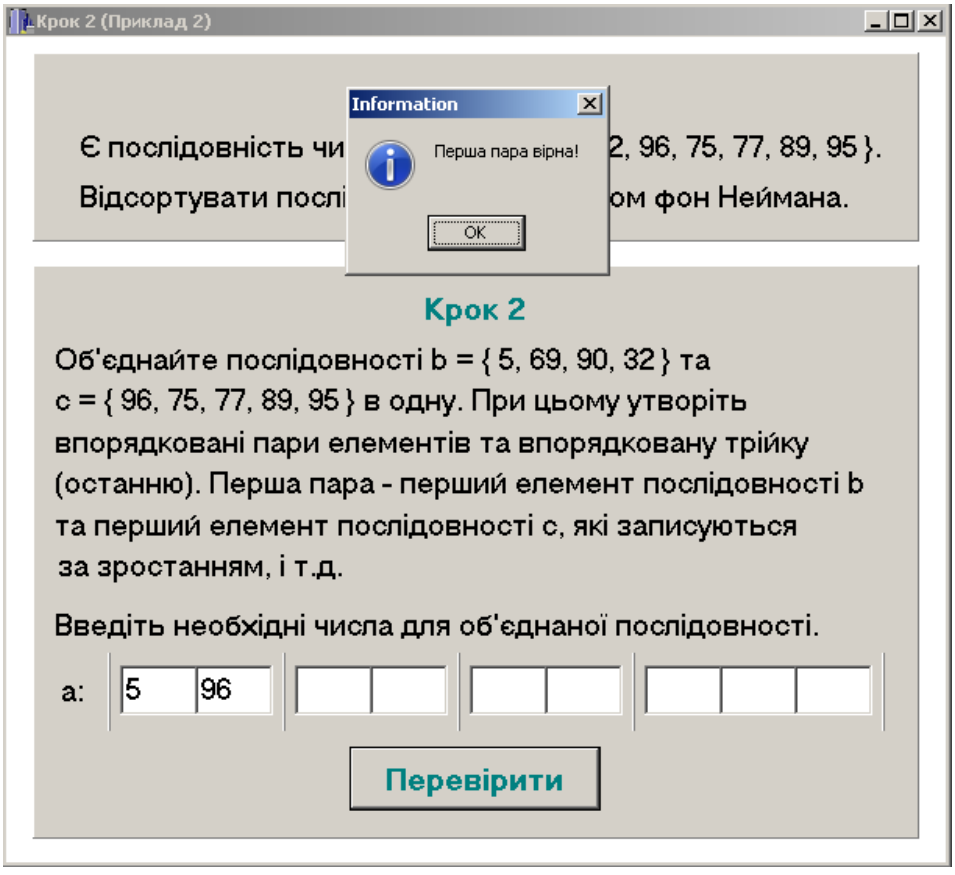

Рис. А.9 – Підтвердження правильності першої пари на другому кроці

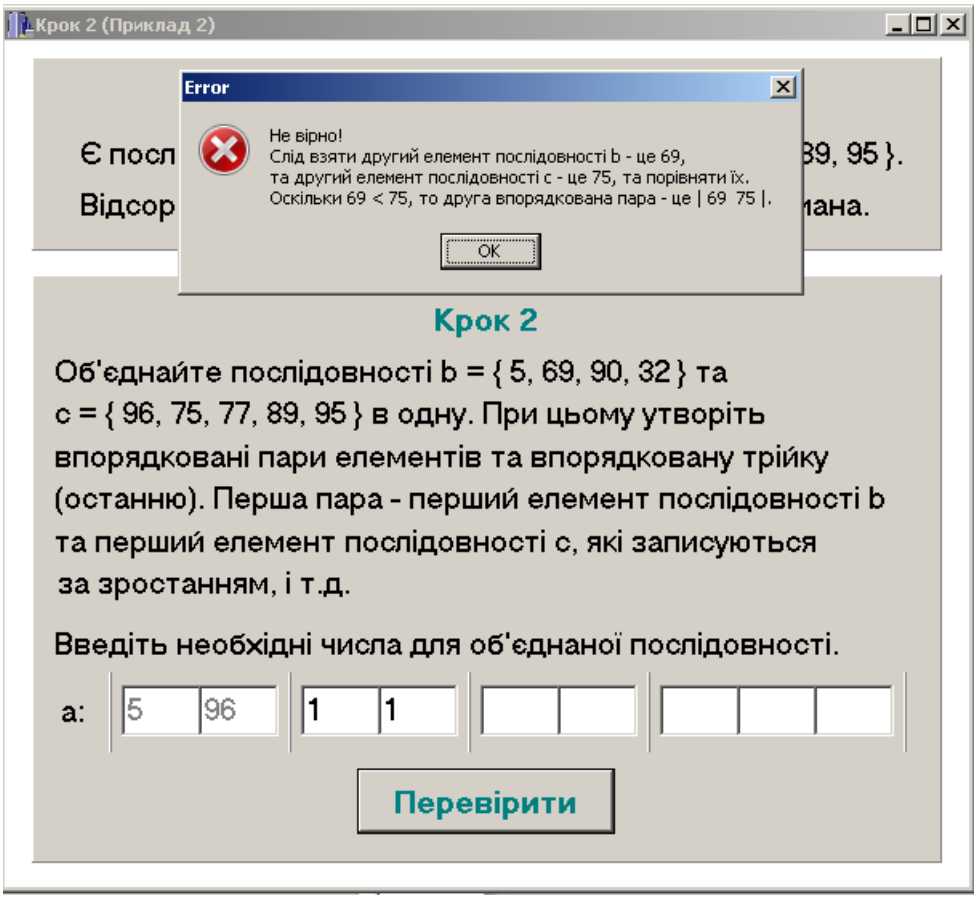

Рис. А.10 – Помилка у другій парі на другому кроці

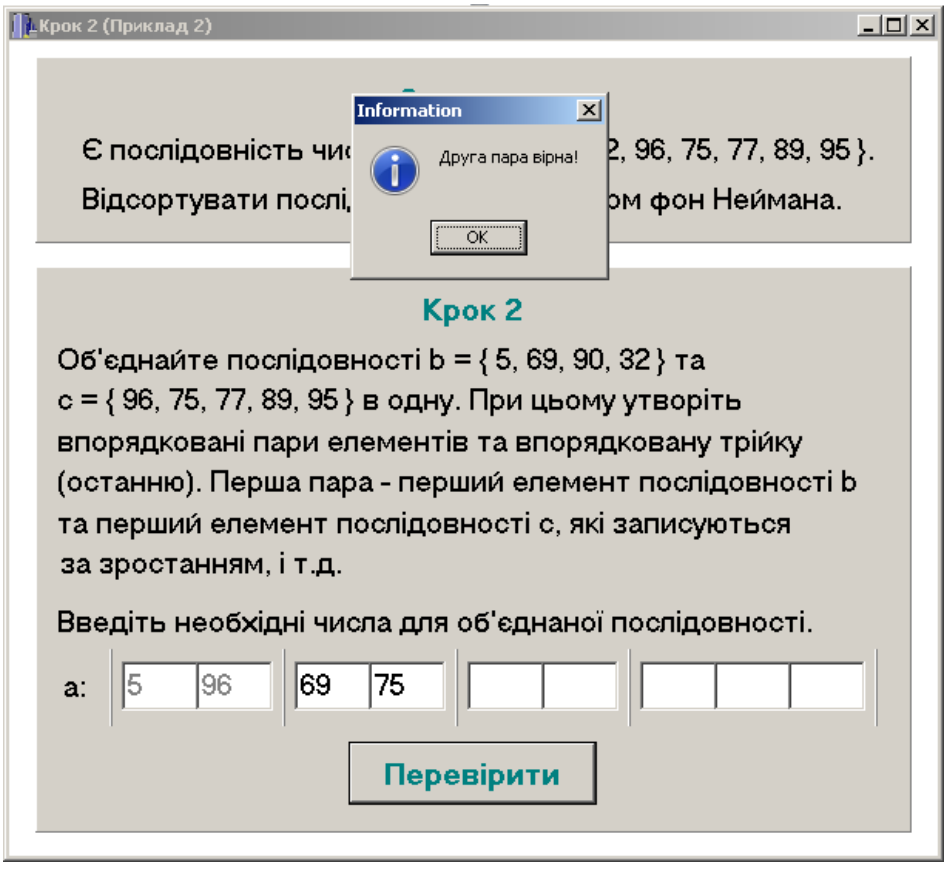

Рис. А.11 – Підтвердження правильності другої пари на другому кроці

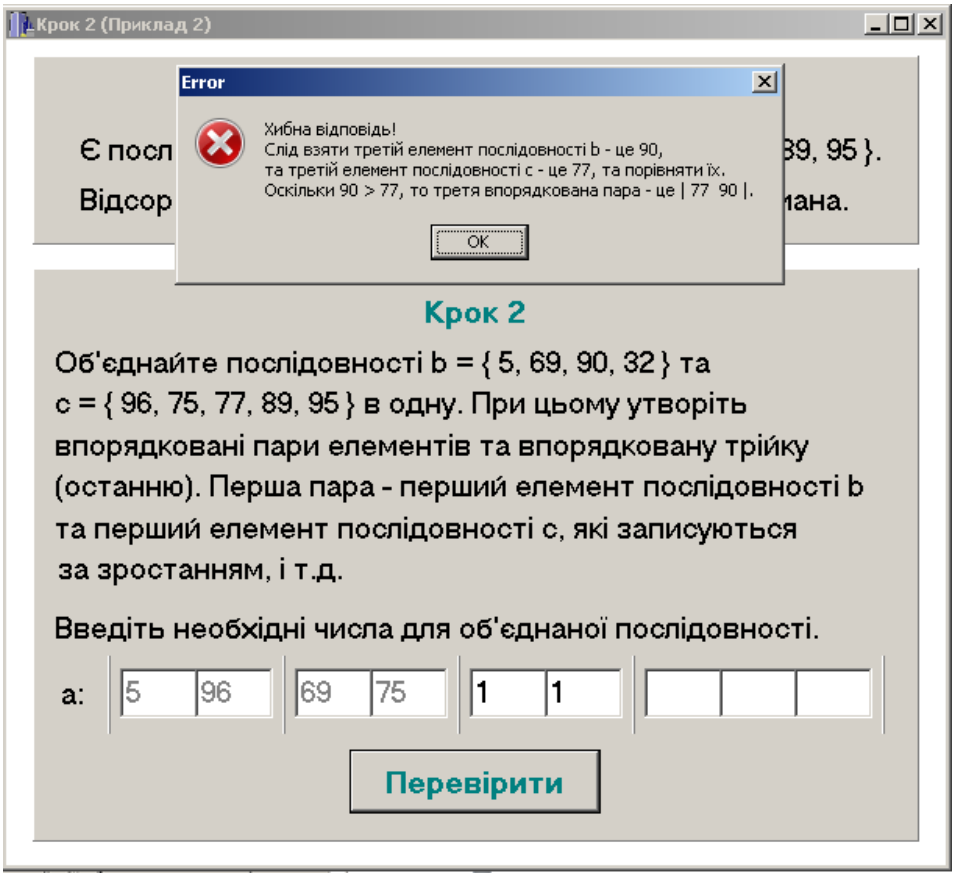

Рис. А.12 – Помилка у третій парі на другому кроці

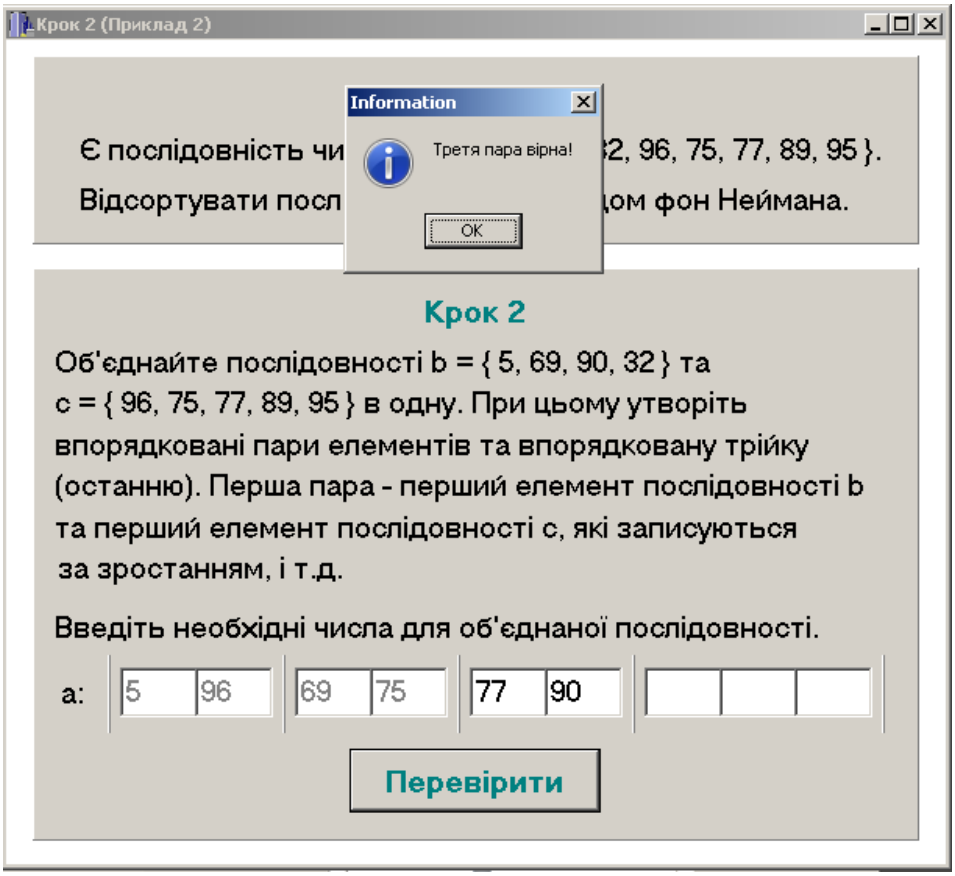

Рис. А.13 – Підтвердження правильності третьої пари на другому кроці

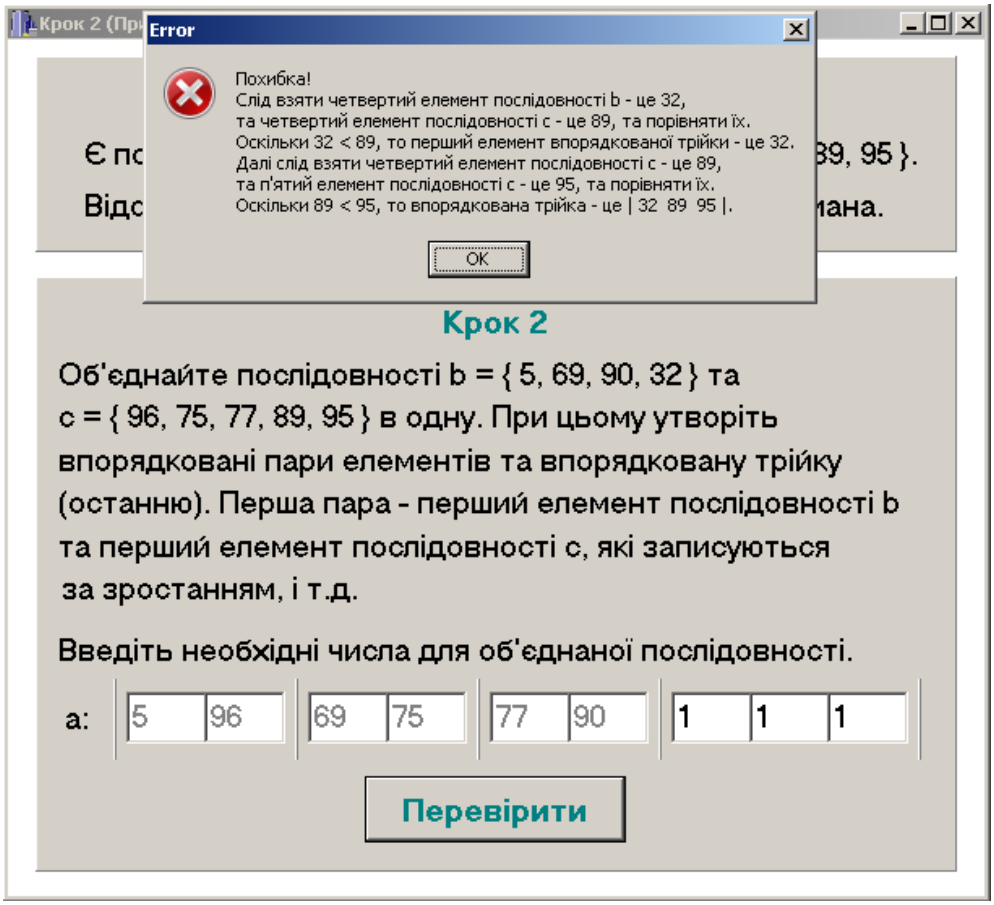

Рис. А.14 – Помилка у четвертій парі на другому кроці

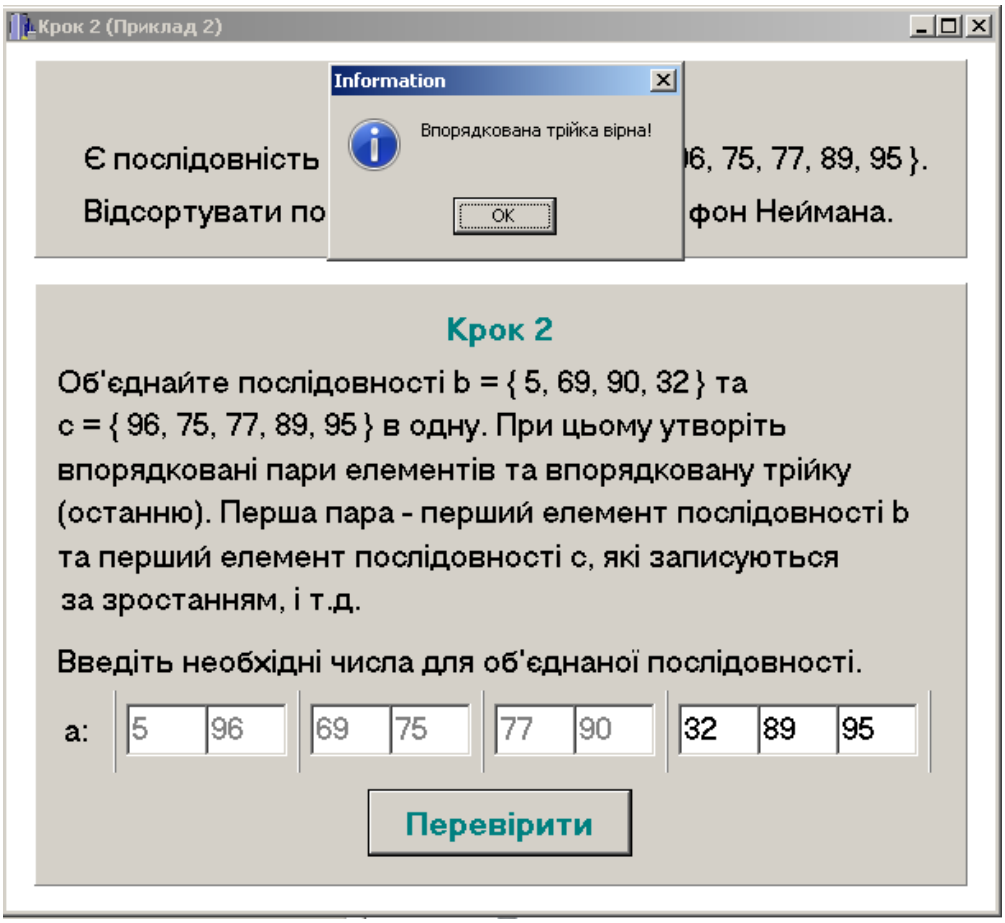

Рис. А.15 – Підтвердження правильності трійки на другому кроці

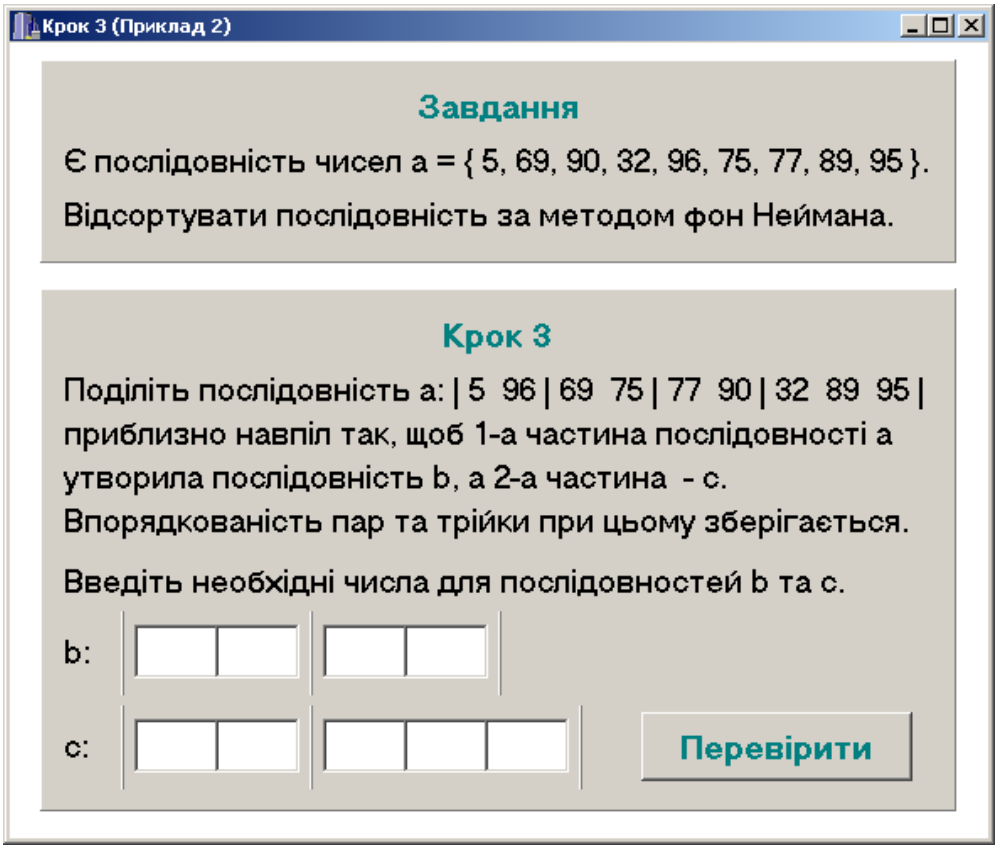

Рис. А.16 – Третій крок

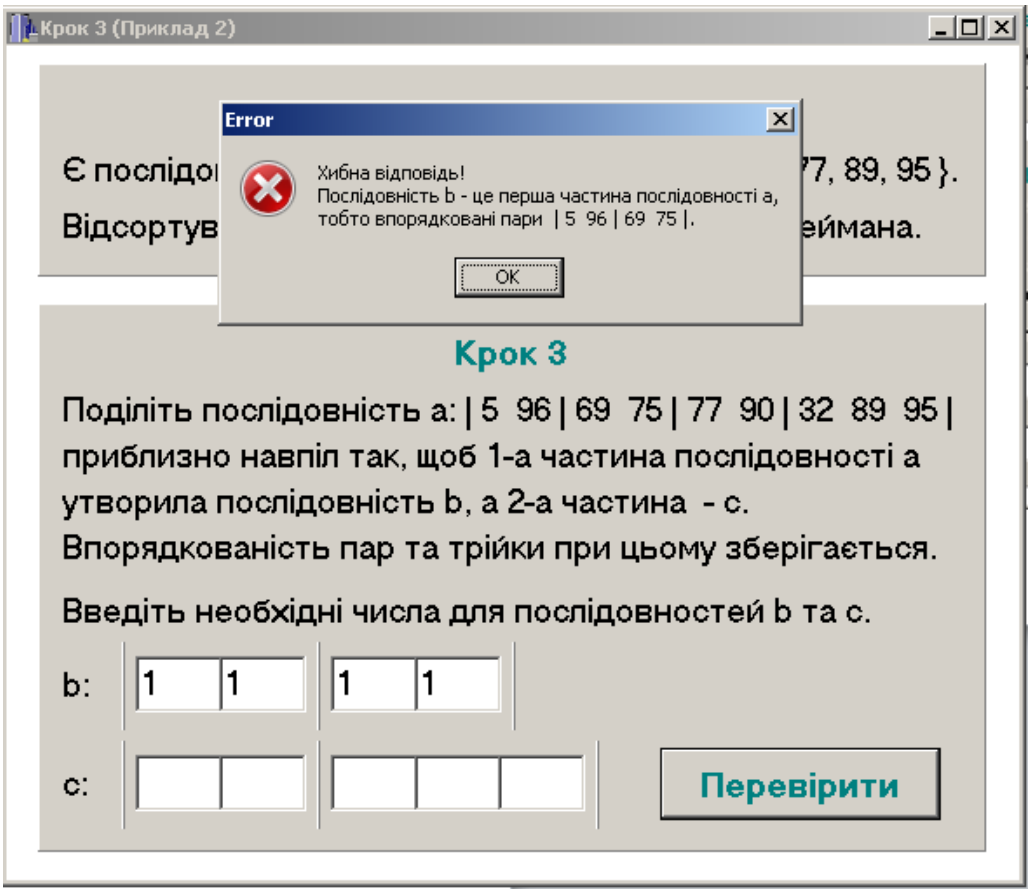

Рис. А.17 – Помилка у послідовності *b* на третьому кроці

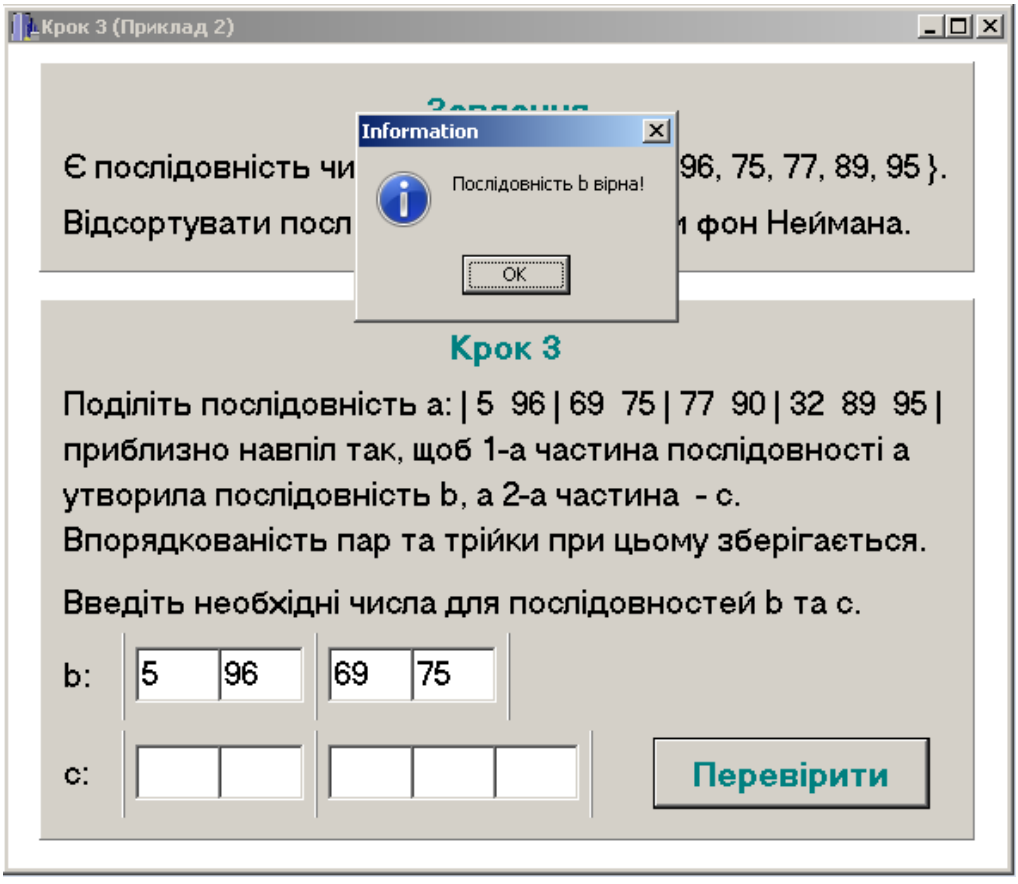

Рис. А.18 – Підтвердження правильності послідовності *b* на третьому кроці

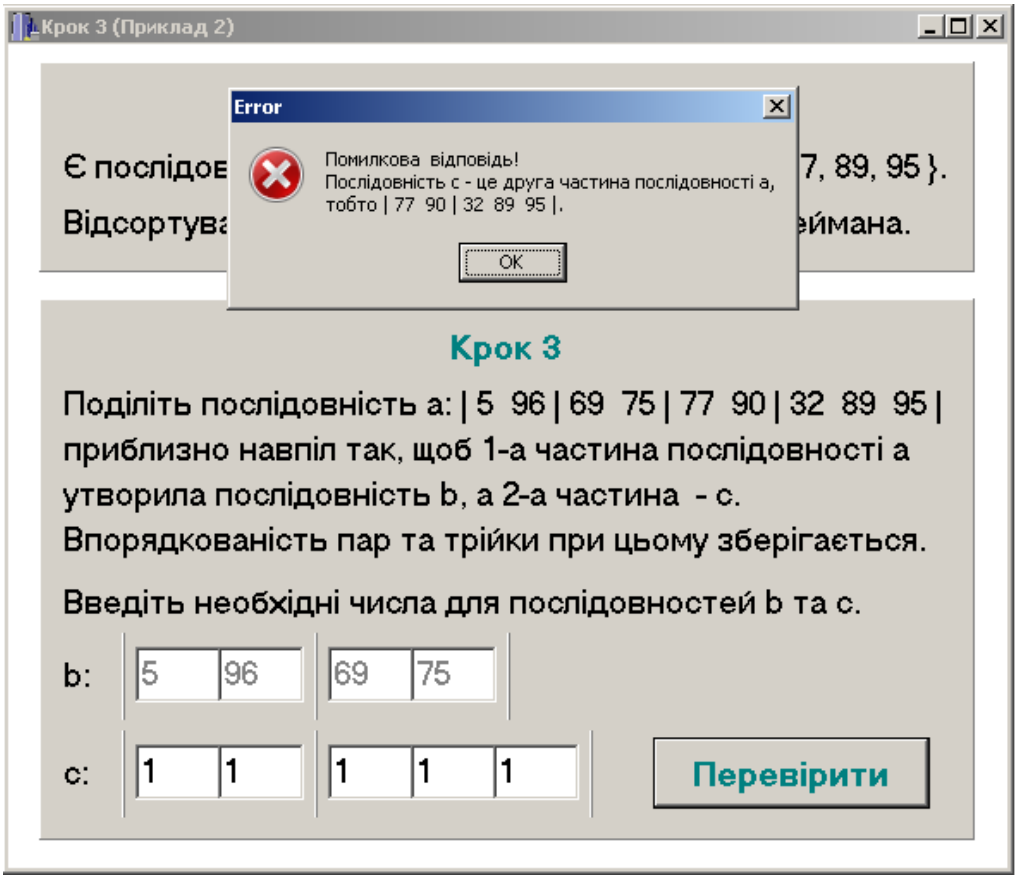

Рис. А.19 – Помилка у послідовності *c* на третьому кроці

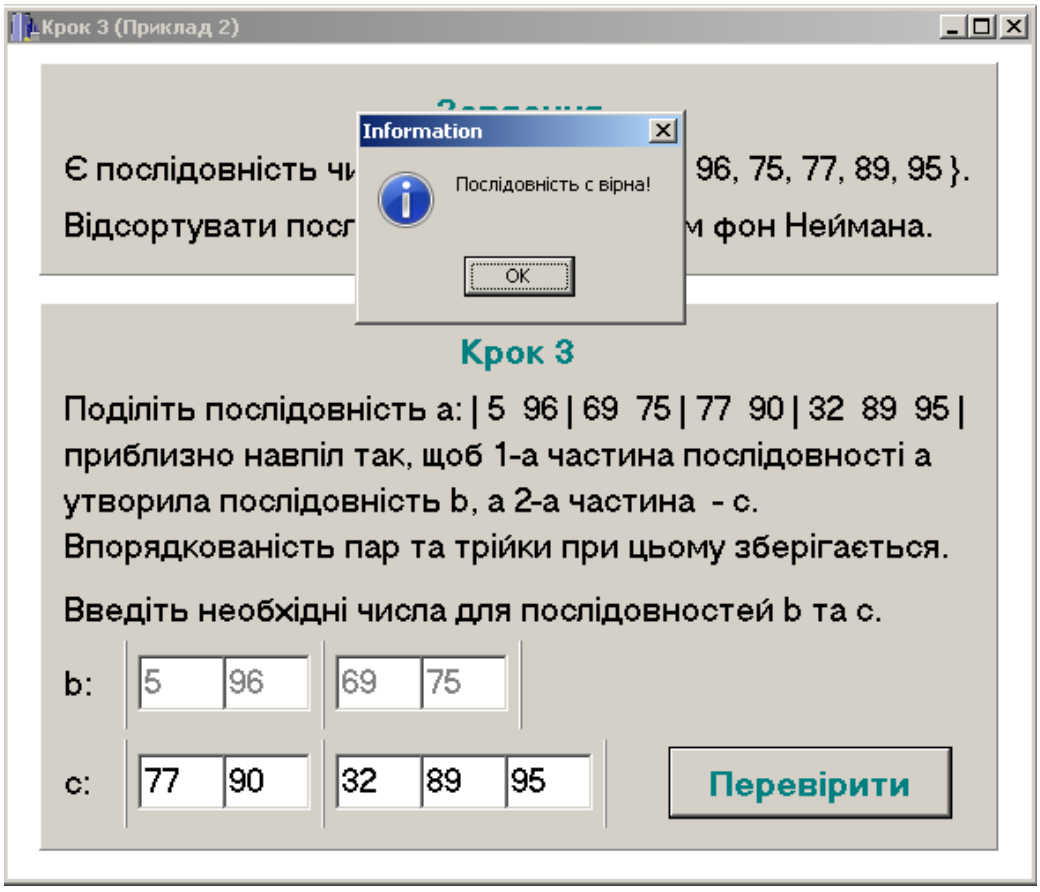

Рис. А.20 – Підтвердження правильності послідовності *c* на третьому кроці

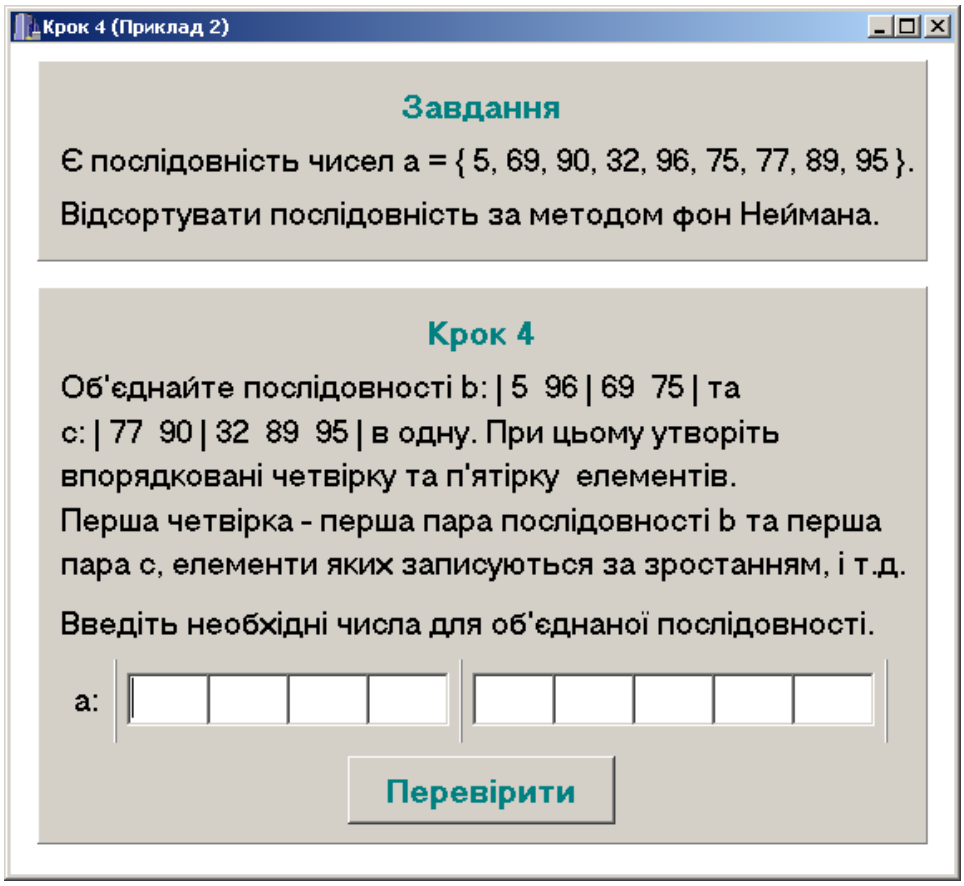

Рис. А.21 – Четвертий крок

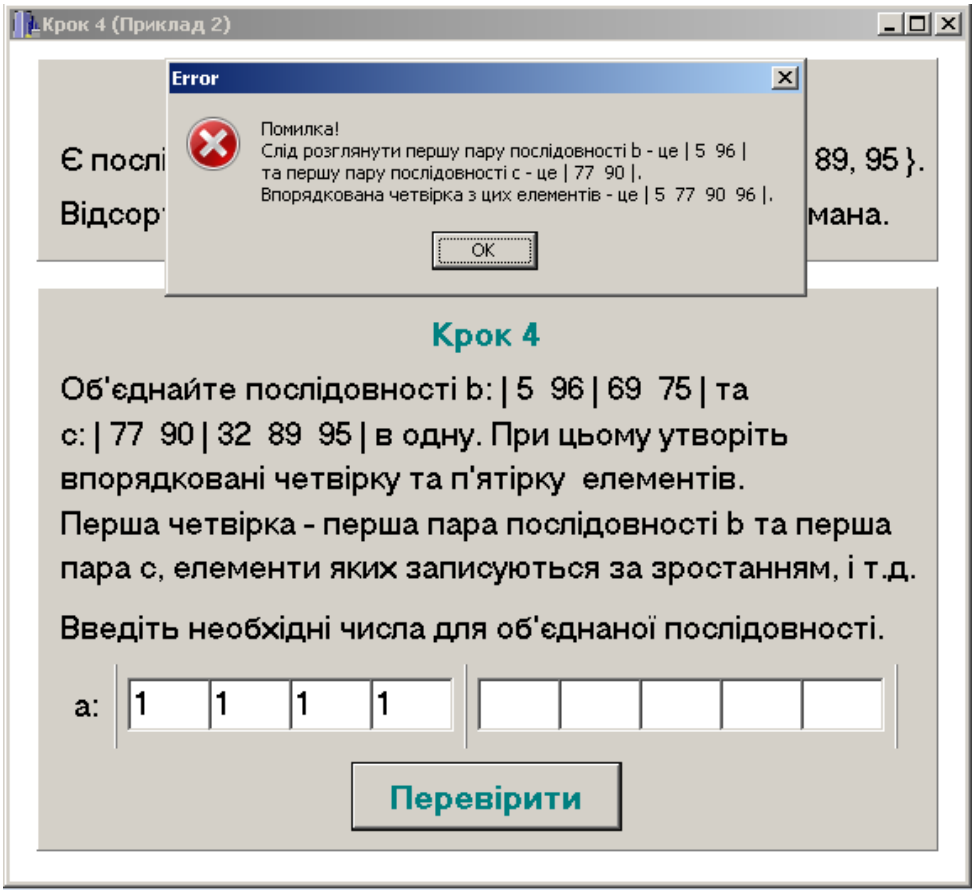

Рис. А.22 – Помилка у першій четвірці на четвертому кроці

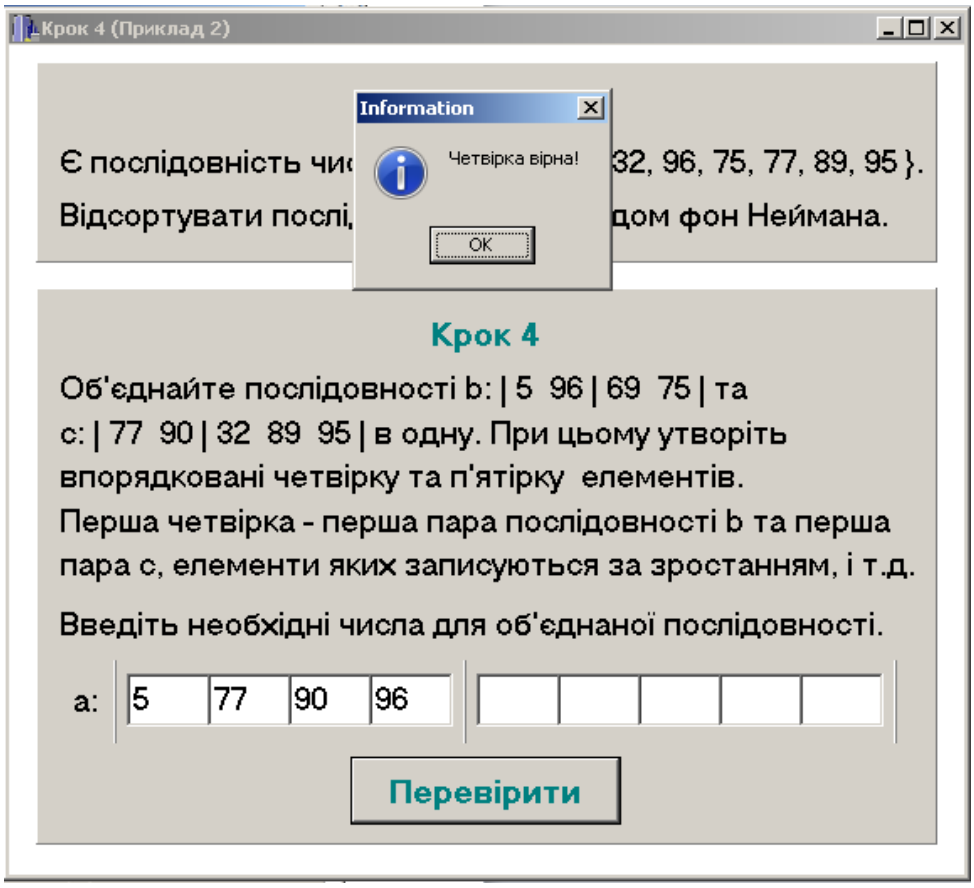

Рис. А.23 – Підтвердження правильності четвірки на 4-ому кроці

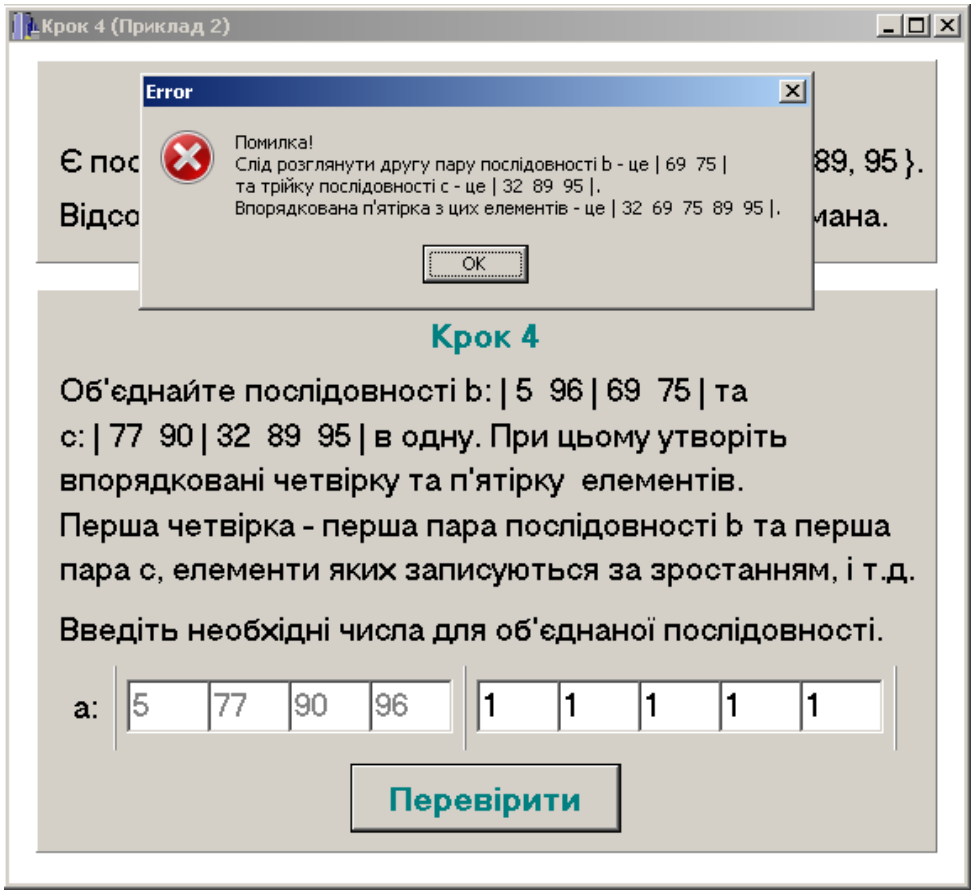

Рис. А.24 – Помилка у п’ятірці на четвертому кроці

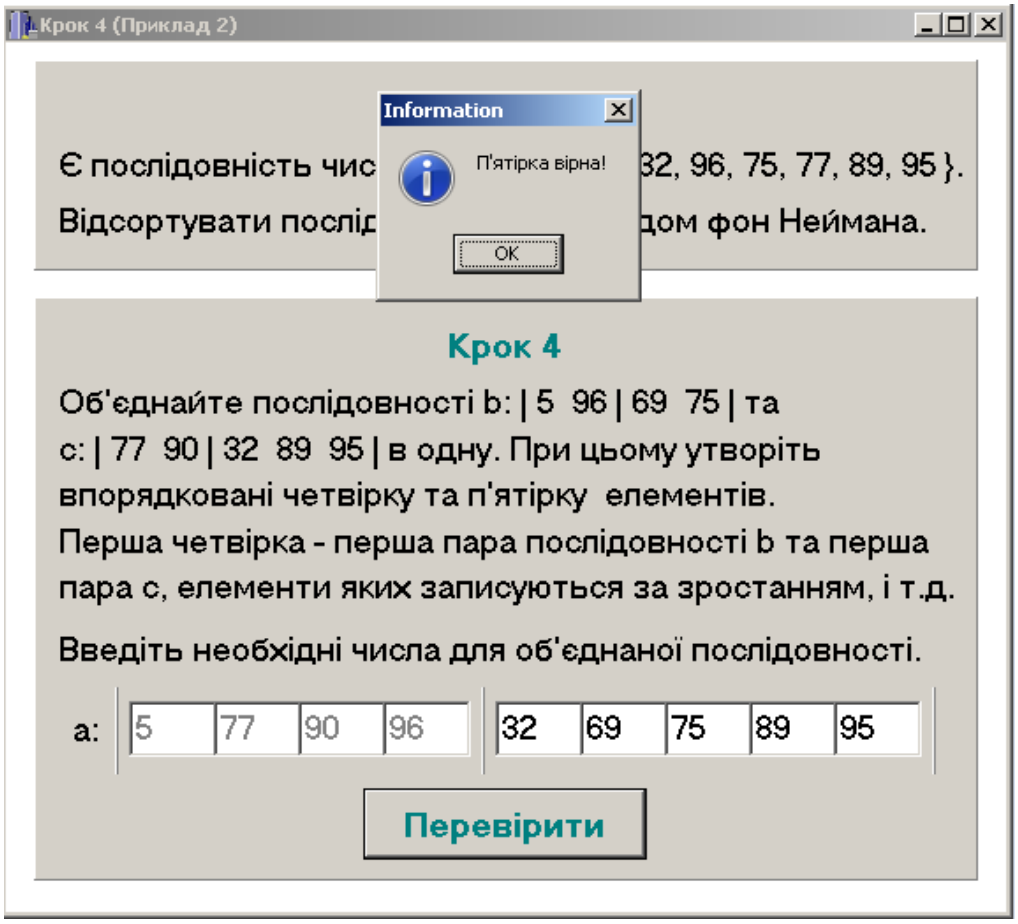

Рис. А.25 – Підтвердження правильності п’ятірки на 4-ому кроці

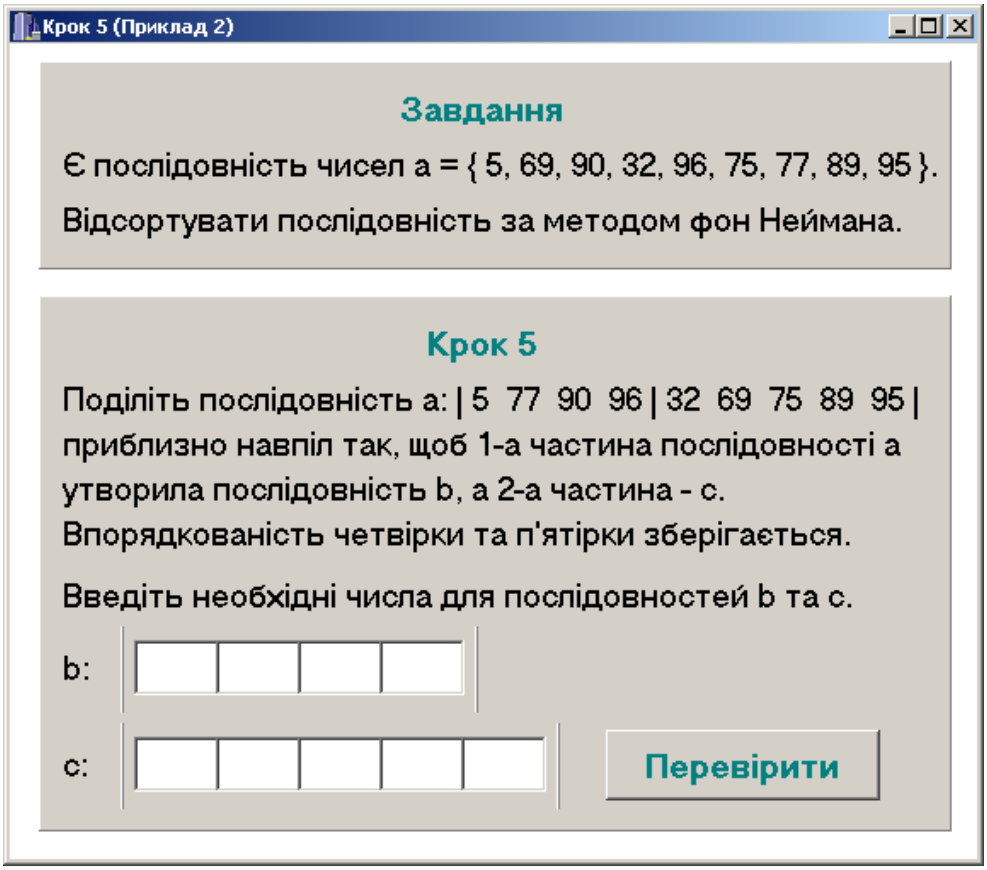

Рис. А.26 – П’ятий крок

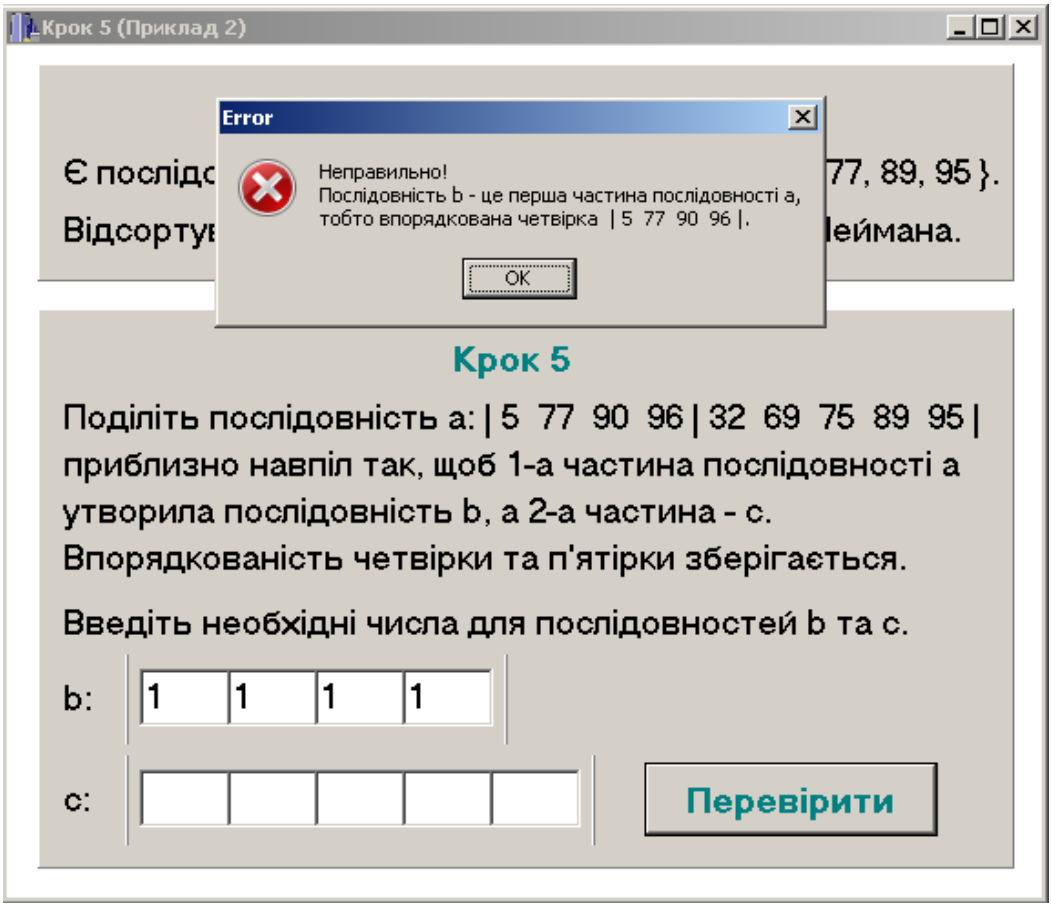

Рис. А.27 – Помилка у послідовності *b* на п’ятому кроці

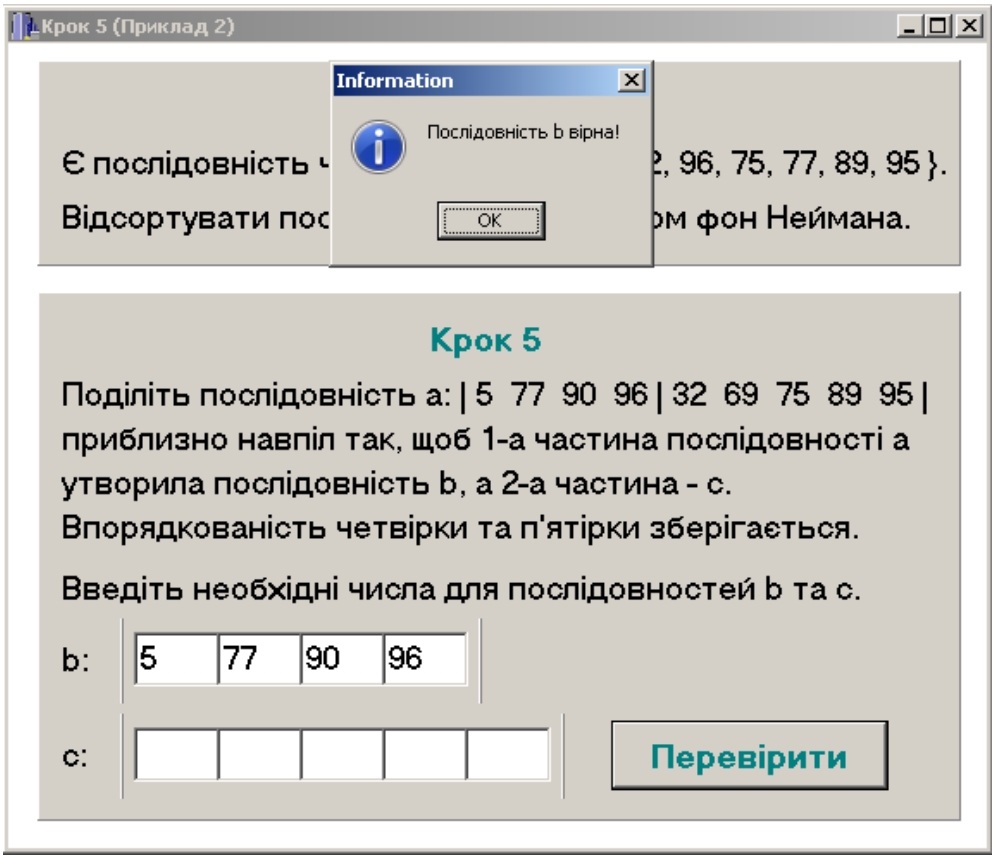

Рис. А.28 – Підтвердження правильності послідовності *b* на п’ятому кроці

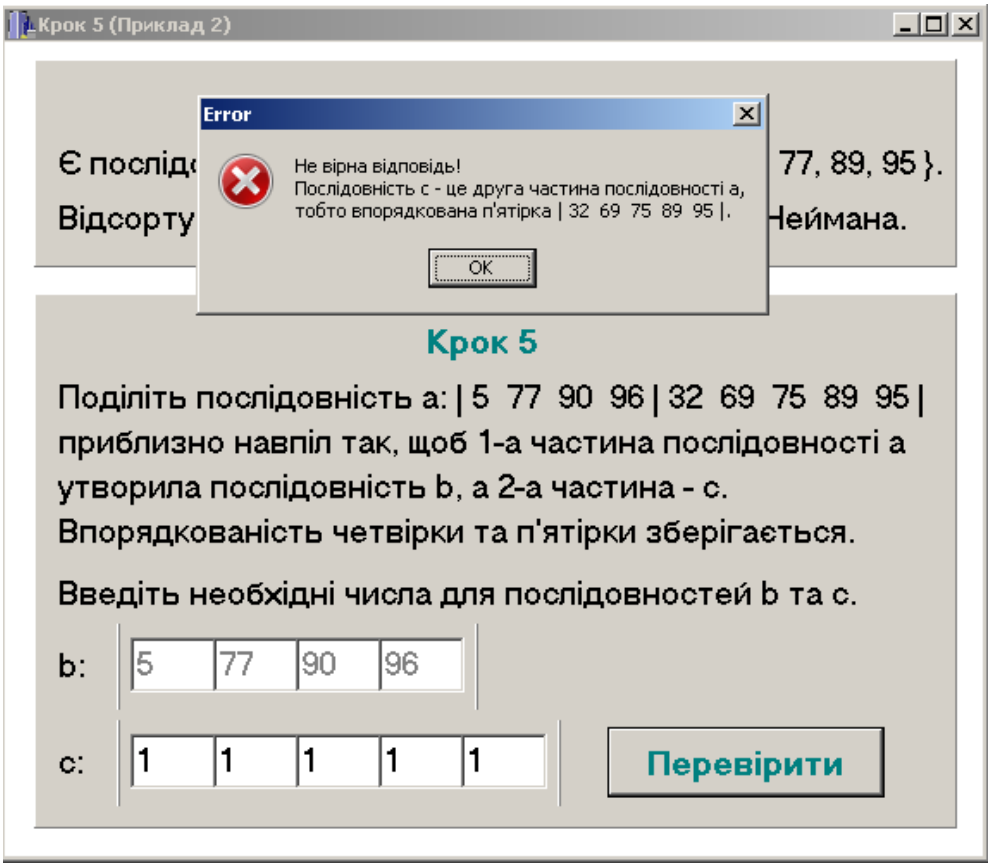

Рис. А.29 – Помилка у послідовності *c* на п’ятому кроці

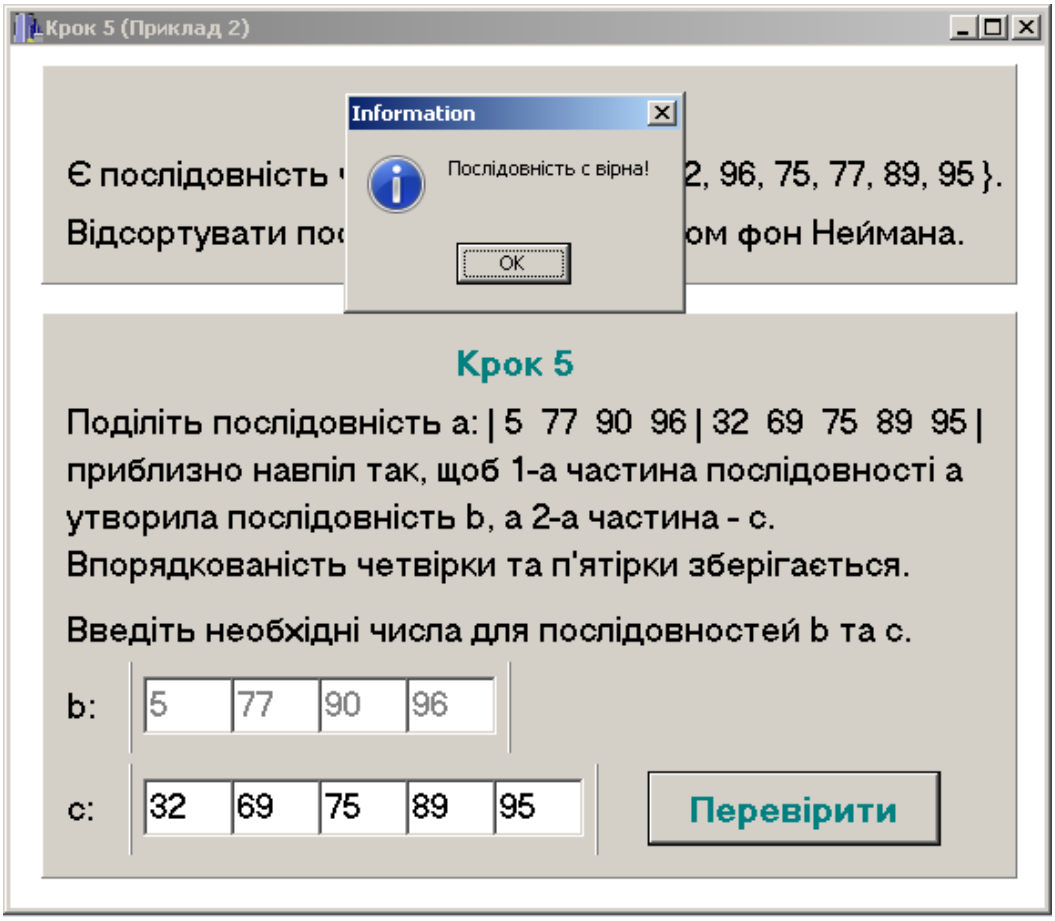

Рис. А.30 – Підтвердження правильності послідовності *c* на п’ятому кроці

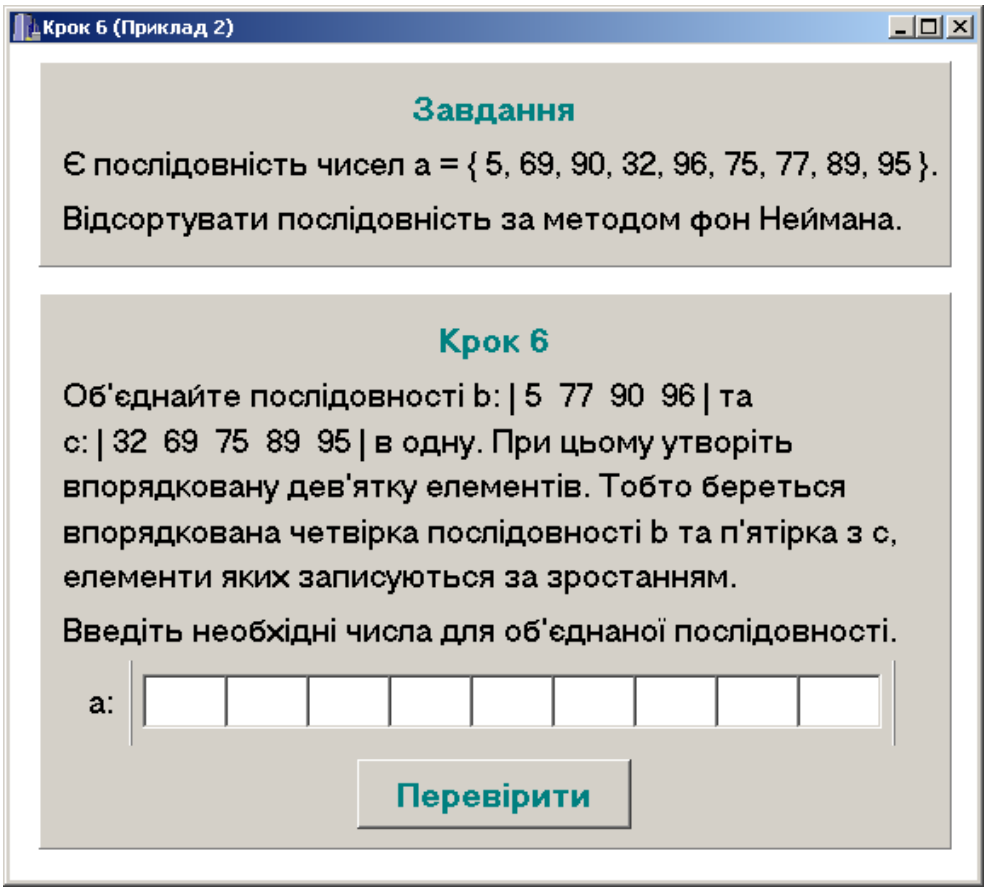

Рис. А.31 – Шостий крок

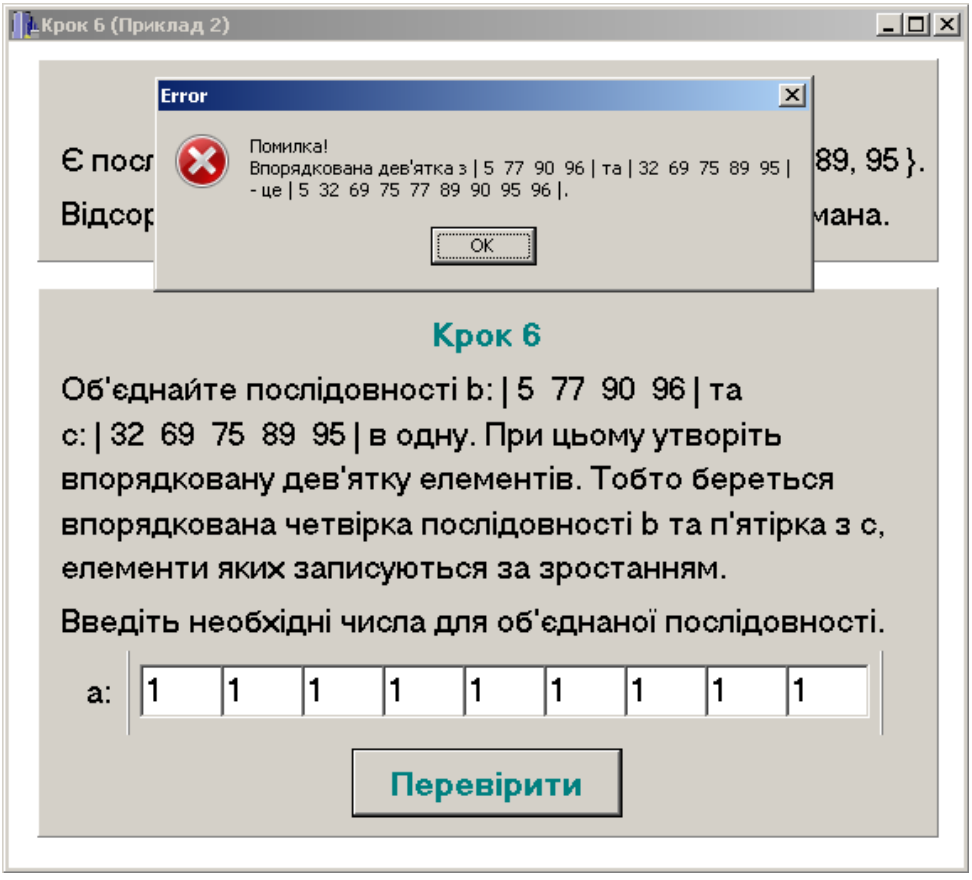

Рис. А.32 – Помилка на шостому кроці

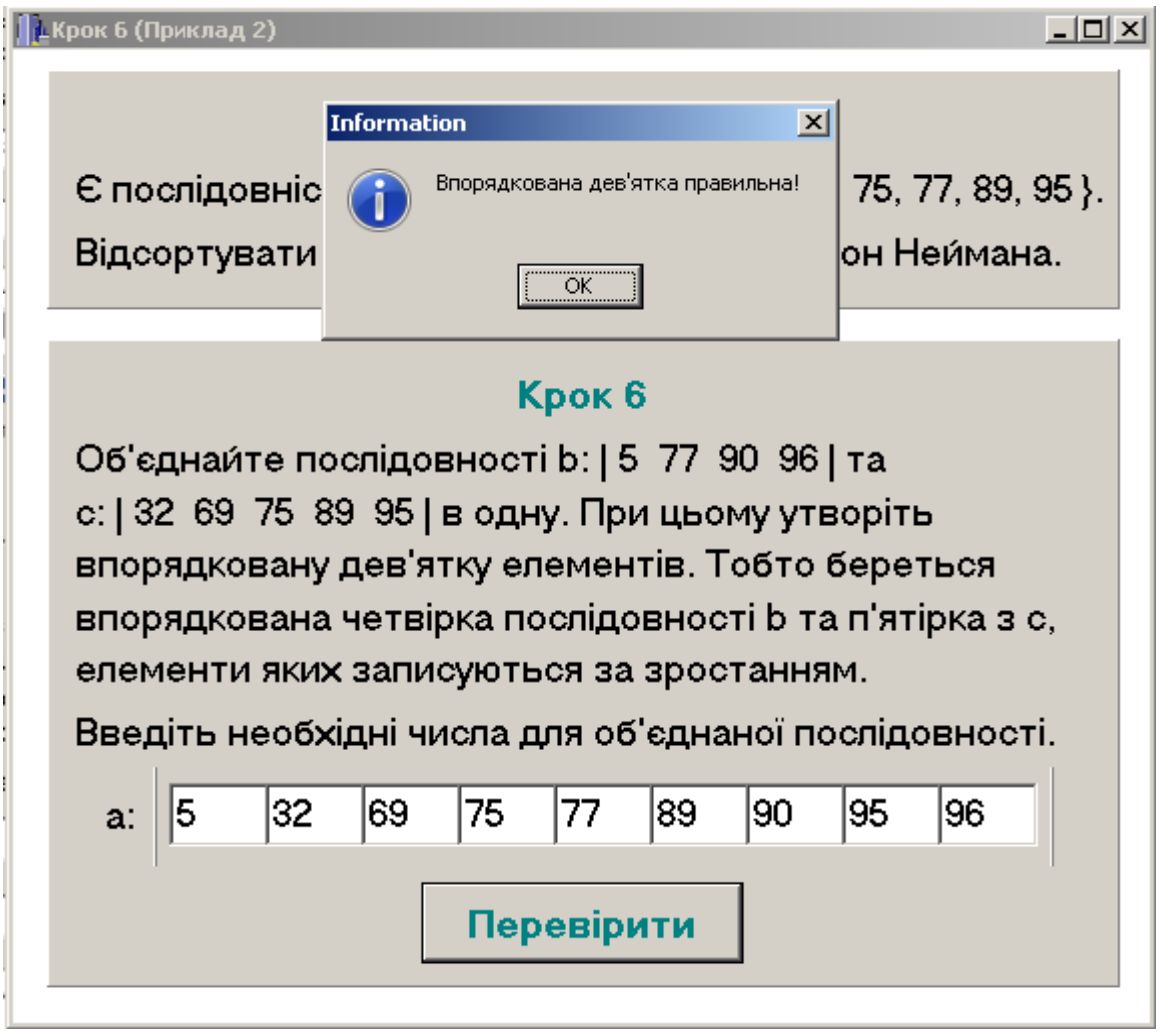

Рис. А.33 – Підтвердження правильності на 6-ому кроці

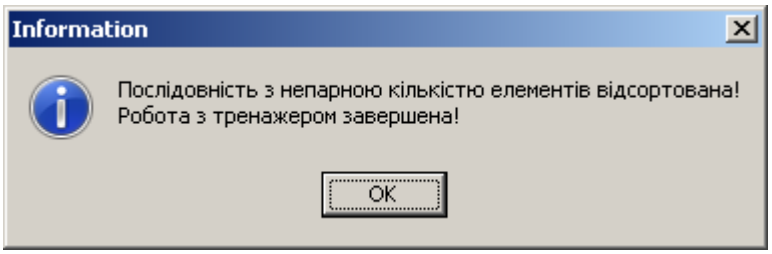

Рис. А.34– Кінець роботи тренажера з прикладом 2

ДОДАТОК Б
КОД ПРОГРАМИ (ПРИКЛАДУ 1)

Unit1.cpp

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{

  Form8->Show();

  Form1->Hide();

}
```

Unit8.cpp

```
#include <vcl.h>
#pragma hdrstop

#include "Unit8.h"
#include "Unit2.h"
#include "Unit1.h"
#include "Unit9.h"
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm8 *Form8;
__fastcall TForm8::TForm8(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm8::FormClose(TObject *Sender,
TCloseAction &Action)
{
  Form1->Close();
}


void __fastcall TForm8::BitBtn1Click(TObject *Sender)
{
  // приклад 1
  if (RadioGroup1->ItemIndex==0)
  {
    Form2->Show();
    Form2->StringGrid1->SetFocus();
    Form8->Hide();
  }

  // приклад 2
  if (RadioGroup1->ItemIndex==1)
  {
    Form9->Show();
    Form9->StringGrid1->SetFocus();
    Form8->Hide();
  }
}
```

Unit2.cpp

```
#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#include "Unit1.h"
#include "Unit3.h"
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm2::FormClose(TObject *Sender,
TCloseAction &Action)
{
  Form1->Close();
}


void __fastcall TForm2::BitBtn1Click(TObject *Sender)
{

  if (Form2->Tag==0) // перевірка послідовності b
  {
    if ((StringGrid1->Cells[0][0]=="45") && (StringGrid1->
Cells[1][0]=="58") && (StringGrid1-> Cells[2][0]=="11") &&
(StringGrid1->Cells[3][0]=="43"))
    {
       MessageDlg("Послідовність b правильна!",
mtInformation, TMsgDlgButtons() << mbOK, 0);
       Form2->Tag=1;
       StringGrid2->Enabled=true;
       StringGrid1->Enabled=false;
       StringGrid2->SetFocus();
       return;
    }
    else
    {
       MessageDlg("Помилка!\nПослідовність b – це перша
половина послідовності a,\nтобто числа  45  58  11  43.",
mtError, TMsgDlgButtons() << mbOK, 0);
       StringGrid1->SetFocus();
       return;
    }
  }
```

```
   if (Form2->Tag==1) // перевірка послідовності  с
   {
    if ((StringGrid2->Cells[0][0]=="98") && (StringGrid2-
>Cells[1][0]=="21") && (StringGrid2-> Cells[2][0]=="2") &&
(StringGrid2->Cells[3][0]=="73"))
    {
       MessageDlg("Послідовність с правильна!",
mtInformation, TMsgDlgButtons() << mbOK, 0);
       Form2->Tag=2;
       StringGrid2->Enabled=false;
       Form3->Show(); // перехід на наступний крок
       Form3->Edit1->SetFocus();
       Form2->Hide();
       return;
    }
    else
    {
       MessageDlg("Не вірно!\nПослідовність с - це друга
половина послідовності a,\nтобто числа  98  21  2  73.",
mtError, TMsgDlgButtons() << mbOK, 0);
       StringGrid2->SetFocus();
       return;
    }
  }
}
```

Unit3.cpp

```
#include <vcl.h>
#pragma hdrstop

#include "Unit3.h"
#include "Unit1.h"
#include "Unit4.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm3 *Form3;

__fastcall TForm3::TForm3(TComponent* Owner)
        : TForm(Owner)
{
}
```

```
void __fastcall TForm3::FormClose(TObject *Sender,
TCloseAction &Action)
{
  Form1->Close();
}


void __fastcall TForm3::BitBtn1Click(TObject *Sender)
{
  if (Form3->Tag==0) // перша пара
  {
    if ((Edit1->Text=="45") && (Edit2->Text=="98"))
    {
       MessageDlg("Перша пара вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form3->Tag=1;

       Edit3->Enabled=true;
       Edit4->Enabled=true;

       Edit1->Enabled=false;
       Edit2->Enabled=false;

       Edit3->SetFocus();
       return;
    }
    else
    {
       MessageDlg("Помилка!\n" "Слід взяти перший елемент
послідовності b - це 45,\n" "та перший елемент послідовності
c - це 98, та порівняти їх.\n" "Оскільки 45 < 98, то перша
впорядкована пара - це | 45  98 |.", mtError,
TMsgDlgButtons() << mbOK, 0);
       Edit1->SetFocus();
       return;
    }
  }

  if (Form3->Tag==1) // друга пара
  {
    if ((Edit3->Text=="21") && (Edit4->Text=="58"))
    {
       MessageDlg("Друга пара вірна!",mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form3->Tag=2;
```

```
        Edit5->Enabled=true;
        Edit6->Enabled=true;

        Edit3->Enabled=false;
        Edit4->Enabled=false;

        Edit5->SetFocus();
        return;
     }
     else
     {
        MessageDlg("Не вірно!\n" "Слід взяти другий елемент
послідовності  b - це 58,\n" "та другий елемент
послідовності c - це 21, та порівняти їх.\n" "Оскільки 58 >
21, то друга впорядкована пара - це | 21  58 |.", mtError,
TMsgDlgButtons() << mbOK, 0);
        Edit3->SetFocus();
        return;
     }
   }


   if (Form3->Tag==2) // третя пара
   {
     if ((Edit5->Text=="2") && (Edit6->Text=="11"))
     {
        MessageDlg("Третя пара вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
        Form3->Tag=3;

        Edit7->Enabled=true;
        Edit8->Enabled=true;

        Edit5->Enabled=false;
        Edit6->Enabled=false;

        Edit7->SetFocus();
        return;
     }
     else
     {
        MessageDlg("Хибна відповідь!\n" "Слід взяти третій
елемент послідовності b - це 11,\n" "та третій елемент
послідовності c - це 2, та порівняти їх.\n" "Оскільки 11 >
```

```
2, то третя впорядкована пара - це | 2  11 |.", mtError,
TMsgDlgButtons() << mbOK, 0);
       Edit5->SetFocus();
       return;
    }
  }


  if (Form3->Tag==3) // четверта пара
  {
    if ((Edit7->Text=="43") && (Edit8->Text=="73"))
    {
       MessageDlg("Четверта пара вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form3->Tag=4;

       Edit7->Enabled=false;
       Edit8->Enabled=false;

       Form4->Show();  // перехід на наступний крок
       Form4->Edit1->SetFocus();
       Form3->Hide();

       return;
    }
    else
    {
       MessageDlg("Похибка!\n" "Слід взяти четвертий елемент
послідовності b - це 43,\n" "та четвертий послідовності c -
це 73, та порівняти їх.\n" "Оскільки 43 < 73, то четверта
пара послідовності - це | 43  73 |.", mtError,
TMsgDlgButtons() << mbOK, 0);
       Edit7->SetFocus();
       return;
    }
  }
}
```

Unit4.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "Unit4.h"
#include "Unit1.h"
#include "Unit5.h"
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm4 *Form4;

__fastcall TForm4::TForm4(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm4::FormClose(TObject *Sender,
TCloseAction &Action)
{
   Form1->Close();
}


void __fastcall TForm4::BitBtn1Click(TObject *Sender)
{
  if (Form4->Tag==0) // перевірка послідовності b
  {
    if ((Edit1->Text=="45") && (Edit2->Text=="98")
     && (Edit3->Text=="21")&& (Edit4->Text=="58"))
    {
       MessageDlg("Послідовність b вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form4->Tag=1;

       Edit5->Enabled=true;
       Edit6->Enabled=true;
       Edit7->Enabled=true;
       Edit8->Enabled=true;

       Edit1->Enabled=false;
       Edit2->Enabled=false;
       Edit3->Enabled=false;
       Edit4->Enabled=false;

       Edit5->SetFocus();
       return;
    }
    else
    {
       MessageDlg("Хибна відповідь!\n" "Послідовність b – це
перша половина послідовності a,\n" "тобто впорядковані пари
```

```
| 45  98 | 21  58 |.\n", mtError, TMsgDlgButtons() << mbOK,
0);
       Edit1->SetFocus();
       return;
    }
  }

  if (Form4->Tag==1) // перевірка послідовності  с
  {
    if ((Edit5->Text=="2") && (Edit6->Text=="11")
     && (Edit7->Text=="43") && (Edit8->Text=="73"))
    {
       MessageDlg("Послідовність с вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form4->Tag=2;

       Edit5->Enabled=false;
       Edit6->Enabled=false;
       Edit7->Enabled=false;
       Edit8->Enabled=false;


       Form5->Show(); // перехід на наступний крок
       Form5->Edit1->SetFocus();
       Form4->Hide();

       return;
    }
    else
    {
       MessageDlg("помилкова відповідь!\n" "Послідовність с
– це друга половина послідовності a,\n" "тобто впорядковані
пари  | 2  11 | 43  73 |.\n", mtError, TMsgDlgButtons() <<
mbOK, 0);
       Edit5->SetFocus();
       return;
    }
  }
}
```

Unit5.cpp

```
#include <vcl.h>
#pragma hdrstop
```

```
#include "Unit5.h"
#include "Unit1.h"
#include "Unit6.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm5 *Form5;
__fastcall TForm5::TForm5(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm5::FormClose(TObject *Sender,
TCloseAction &Action)
{
  Form1->Close();
}


void __fastcall TForm5::BitBtn1Click(TObject *Sender)
{
  if (Form5->Tag==0) // перевірка першої четвірки
  {
    if ((Edit1->Text=="2") && (Edit2->Text=="11")
     && (Edit3->Text=="45") && (Edit4->Text=="98"))
    {
       MessageDlg("Перша четвірка вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form5->Tag=1;

       Edit5->Enabled=true;
       Edit6->Enabled=true;
       Edit7->Enabled=true;
       Edit8->Enabled=true;

       Edit1->Enabled=false;
       Edit2->Enabled=false;
       Edit3->Enabled=false;
       Edit4->Enabled=false;

       Edit5->SetFocus();
       return;
    }
```

```
    else
    {
       MessageDlg("Помилка!\n" "Слід розглянути першу пару
послідовності b - це | 45  98 |\n" "та першу пару
послідовності c - це | 2  11 |.\n" "Впорядкована четвірка з
цих елементів - це | 2  11  45  98 |.", mtError,
TMsgDlgButtons() << mbOK, 0);
       Edit1->SetFocus();
       return;
    }
  }

    if (Form5->Tag==1) // перевірка другої четвірки
  {
    if ((Edit5->Text=="21") && (Edit6->Text=="43")
     && (Edit7->Text=="58") && (Edit8->Text=="73"))
    {
       MessageDlg("Друга четвірка вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form5->Tag=2;

       Edit5->Enabled=false;
       Edit6->Enabled=false;
       Edit7->Enabled=false;
       Edit8->Enabled=false;


       Form6->Show(); // перехід на наступний крок
       Form6->Edit1->SetFocus();
       Form5->Hide();

       return;
    }
    else
    {
       MessageDlg("Помилка!\n" "Слід розглянути другу пару
послідовності b - це | 21  58 |\n" "та другу пару
послідовності c - це | 43  73 |.\n" "Впорядкована четвірка з
цих елементів - це | 21  43  58  73 |.", mtError,
TMsgDlgButtons() << mbOK, 0);
       Edit5->SetFocus();
       return;
    }
  }
}
```

Unit6.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "Unit6.h"
#include "Unit1.h"
#include "Unit7.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm6 *Form6;

__fastcall TForm6::TForm6(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm6::BitBtn1Click(TObject *Sender)
{
  if (Form6->Tag==0) // перевірка послідовності b
  {
    if ((Edit1->Text=="2") && (Edit2->Text=="11")
     && (Edit3->Text=="45") && (Edit4->Text=="98"))
    {
       MessageDlg("Послідовність b вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form6->Tag=1;

       Edit5->Enabled=true;
       Edit6->Enabled=true;
       Edit7->Enabled=true;
       Edit8->Enabled=true;

       Edit1->Enabled=false;
       Edit2->Enabled=false;
       Edit3->Enabled=false;
       Edit4->Enabled=false;

       Edit5->SetFocus();
       return;
    }
    else
    {
       MessageDlg("Неправильно!\n" "Послідовність b – це
перша половина послідовності a,\n" "тобто впорядкована
```

```
четвірка  | 2  11  45  98 |.\n", mtError, TMsgDlgButtons()
<< mbOK, 0);
       Edit1->SetFocus();
       return;
    }
  }

  if (Form6->Tag==1) // перевірка послідовності с
  {
    if ((Edit5->Text=="21") && (Edit6->Text=="43")
     && (Edit7->Text=="58") && (Edit8->Text=="73"))
    {
       MessageDlg("Послідовність с вірна!", mtInformation,
TMsgDlgButtons() << mbOK, 0);
       Form6->Tag=2;

       Edit5->Enabled=false;
       Edit6->Enabled=false;
       Edit7->Enabled=false;
       Edit8->Enabled=false;

       Form7->Show(); // перехід на наступний рік
       Form7->Edit1->SetFocus();
       Form6->Hide();

       return;
    }
    else
    {
       MessageDlg("Не вірна відповідь!\n" "Послдідовність с
– це друга половина послідовності а,\n" "тобто впорядкована
четвірка  | 21  43  58  73 |.\n", mtError, TMsgDlgButtons()
<< mbOK, 0);
       Edit5->SetFocus();
       return;
    }
  }
}


void __fastcall TForm6::FormClose(TObject *Sender,
TCloseAction &Action)
{
  Form1->Close();
}
```

Unit7.cpp

```
#include <vcl.h>
#pragma hdrstop

#include "Unit7.h"
#include "Unit1.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm7 *Form7;

__fastcall TForm7::TForm7(TComponent* Owner)
        : TForm(Owner)
{
}


void __fastcall TForm7::FormClose(TObject *Sender,
TCloseAction &Action)
{
  Form1->Close();
}


void __fastcall TForm7::BitBtn1Click(TObject *Sender)
{
    // перевірка

    if ((Edit1->Text=="2") && (Edit2->Text=="11")
     && (Edit3->Text=="21") && (Edit4->Text=="43")
     && (Edit5->Text=="45") && (Edit6->Text=="58")
     && (Edit7->Text=="73") && (Edit8->Text=="98"))
    {
       MessageDlg("Впорядкована вісімка правильна!",
mtInformation, TMsgDlgButtons() << mbOK, 0);

       Edit1->Enabled=false;
       Edit2->Enabled=false;
       Edit3->Enabled=false;
       Edit4->Enabled=false;
       Edit5->Enabled=false;
       Edit6->Enabled=false;
       Edit7->Enabled=false;
       Edit8->Enabled=false;
```

```
       MessageDlg("Послідовність з парною кількістю
елементів відсортована!\n" "Робота з тренажером завершена!",
mtInformation, TMsgDlgButtons() << mbOK, 0);
       Form1->Close();

       return;
    }
    else
    {
       MessageDlg("Помилка!\n" "Впорядкована вісімка з
четвірок | 2  11  45  98 | та | 21  43  58  73 |\n" "- це |
2  11  21  43  45  58  73  98 |.\n", mtError,
TMsgDlgButtons() << mbOK, 0);
       Edit1->SetFocus();
       return;
    }
}
```

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

Погоджено
Керівник _____ Ємець Ол-ра О.

«____» _____ 20__ р.
МП

Погоджено
Начальник відділу управління освітньою діяльністю
_____Молчанова Н.Ю.
«____» _____ 20__ р.
МП

Довідка
про рекомендації щодо впровадження та використання результатів дослідження по дипломній роботі (проекту) в освітньому процесі вищого навчального закладу

Вищий навчальний заклад Укоопспілки «Полтавський університет економіки і торгівлі» студента спеціальності 122 «Комп'ютерні науки» групи КН-61 ПВ ступеня магістр, денної форми навчання Козодуба Владислава Сергійовича, на тему «Розробка програмного забезпечення з теми «Сортування фон Неймана» дистанційного навчального курсу «Алгоритми та структури даних», підготовлену за матеріалами кафедри математичного моделювання та соціальної інформатики

Впровадити такі рекомендації: програмний продукт, що реалізує тренажер з теми «Сортування фон Неймана» дистанційного навчального курсу «Алгоритми та структури даних».

Студент _____ (підпис) Козодуб В. С.

Науковий керівник _____ (підпис) Ємець Ол-ра О.

Завідувач кафедри ММСІ _____ (підпис) Ємець О. О.

Начальник відділу автоматизації та роботи в ЄДБО _____ (підпис) Кулібаба В.В.