ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ БІЗНЕСУ ТА СУЧАСНИХ ТЕХНОЛОГІЙ

ФОРМА НАВЧАННЯ ДЕННА
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____ О.О. Ємець
(підпис)

«_____»_____ 2021 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО БАКАЛАВРСЬКОЇ РОБОТИ

на тему

ТРЕНАЖЕР З ТЕМИ «ПОБУДОВА БЛОК-СХЕМ АЛГОРИТМІВ ЦИКЛІЧНОЇ СТРУКТУРИ НА ПРИКЛАДІ ЦИКЛУ WHILE» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ПРОГРАМУВАННЯ ІІ» ТА РОЗРОБКА ЙОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

зі спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Котенко Сергій Костянтинович _____ «___»_____2021р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Ємець Олександра Олегівна
_____ «___»_____2021р.
(підпис)

ПОЛТАВА 2021 р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

ЗАТВЕРДЖУЮ

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

«_____» _____20__ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК

ВИКОНАННЯ БАКАЛАВРСЬКОЇ РОБОТИ

Студента зі спеціальності 122 «Комп'ютерні науки»

Прізвище, ім'я, по батькові Котенко Сергій Костянтинович

1. Тема «Тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while» дистанційного навчального курсу «Програмування ІІ» та розробка його програмного забезпечення», затверджена наказом ректора № 121-Н від «1» вересня 2020 р.

Термін подання студентом бакалаврської роботи «24» травня 2021 р.

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

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

Вступ.
1. Постановка задачі.
   1.1. Постановка задачі розробки тренажера
2. Інформаційний огляд.
   2.1. Огляд робіт, де розглянуте аналогічне до теми роботи завдання.
   2.2. Позитивні аспекти розглянутих розробок.
   2.3. Негативні аспекти розглянутих розробок.
   2.4. Необхідність та актуальність теми.
3. Теоретична частина.
   3.1. Загальні відомості.
   3.2. Блок-схеми циклічної структури.
   3.3. Розробка блок-схеми, яка підлягає програмуванню.
   3.4. Обґрунтування вибору програмних засобів
4. Практична частина.
   4.1. Алгоритм роботи тренажера.
   4.2. Опис програмної реалізації.

Висновки.

4. Перелік графічного матеріалу (з точним визначенням кількості блок-схем, іншого графічного матеріалу) Блок-схема алгоритму (2 листи).

5. Консультанти розділів бакалаврської роботи

Розділ	П.І.Б., посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	Ємець Ол-ра О.	05.09.2020	05.09.2020
1. Постановка задачі	Ємець Ол-ра О.	05.09.2020	05.09.2020
2. Інформаційний огляд	Ємець Ол-ра О.	05.09.2020	05.09.2020
3. Теоретична частина	Ємець Ол-ра О.	05.09.2020	05.09.2020
4. Практична частина	Ємець Ол-ра О.	05.09.2020	05.09.2020

6. Календарний графік виконання бакалаврської роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ	04.05.21	
2. Вивчення методичних рекомендацій і стандартів та звіт керівнику	1.10.20	
3. Постановка задачі	10.10.20	
4. Інформаційний огляд джерел бібліотек та інтернету	1.11.20	
5. Теоретична частина	13.01.21	
6. Практична частина	15.04.21	
7. Закінчення оформлення	04.05.21	
8. Доповідь студента на кафедрі	25.05.21	
9. Доробка (за необхідності), рецензування	07.06.21	

Дата видачі завдання «5» вересня 2020 р.

Студент _____ Котенко С.К.
(підпис)

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

Результати захисту бакалаврської роботи

Бакалаврська 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: 45 с., 28 рис., 1 додаток (на 2 сторінках), 1 таблиця, 12 джерел.

Предмет розробки – програма-тренажер для навчання побудови блок-схем алгоритмів циклічної структури на прикладі циклу while.

Мета роботи – розробка програмного забезпечення з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while» дистанційного навчального курсу «Програмування ІІ».

Методи, які були використані для розв'язування задачі – TypeScript фреймворк Angular та фреймворк для розробки нативних десктоп додатків – Electron. Під час створення тренажера використовувалося інтегроване середовище розробки WebStorm.

Розроблено тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while».

Ключові слова: ТРЕНАЖЕР, БЛОК-СХЕМА, ЦИКЛ, ANGULAR.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ,
СКОРОЧЕНЬ І ТЕРМІНІВ 6
ВСТУП 7
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ 9
1.1 Постановка задачі розробки тренажера 9
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД 11
2.1 Огляд робіт, де розглянуте аналогічне до теми роботи завдання 11
2.2 Позитивні аспекти розглянутих розробок 11
2.3 Негативні аспекти розглянутих розробок 11
2.4 Необхідність та актуальність теми 12
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА 13
3.1 Загальні відомості 13
3.2 Блок-схеми циклічної структури 16
3.3 Розробка блок-схеми, яка підлягає програмуванню 18
3.4 Обґрунтування вибору програмних засобів 20
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА 22
4.1 Алгоритм роботи тренажера 22
4.2 Опис програмної реалізації 30
ВИСНОВКИ 40
СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ 42
ДОДАТОК А. МЕТОД КОМПІЛЯЦІЇ 44

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, символів, скорочень, термінів
Геттер	Спеціальний метод, що дозволяє отримати дані, доступ до яких безпосередньо обмежений.
Декоратор	Інструмент, який дозволяє класам (або елементам класу), додавати якісь метадані, або змінювати стандартну поведінку цих сутностей.
Препроцесор	Комп’ютерна програма, яка приймає дані на вході, обробляє їх, і віддає для використання іншій програмі (наприклад, компілятору).
Радіокнопка	Елемент інтерфейсу, який дозволяє обрати лише одну опцію із певного набору.
Фреймворк	Набір програмних рішень, для прискорення розробки програмного забезпечення. Найбільш спорідненим до фреймворку поняттям є бібліотека. Але на відміну від бібліотеки, фреймворк використовує комплексний підхід до розв’язання різних задач, і має власний синтаксис для написання програмного коду.

ВСТУП

В сучасному світі все більше навчальних закладів вводить в учбовий процес дистанційне навчання [1, 2]. Порівняно з денною формою навчання, головною перевагою дистанційної форми є перш за все її зручність: студент може самостійно вибрати час і місце для навчання, що дозволяє йому працювати або навчатися в іншому місті чи навіть країні. Крім того, заміна конспектів на електронні ресурси та новітні методи навчання, а також постійні консультації з викладачами надають цій формі самоосвіти додаткові переваги над заочною.

Одним з основних засобів дистанційного навчання є програми-тренажери [3]. Саме вони допомагають студентам значно поглибити знання з певної освітньої області та покращити їх навички в практичному застосуванні здобутих знань. Тому тренажери є актуальним та сучасним підходом до навчального процесу.

Також актуальність теми підтверджується тим, що необхідно розробити абсолютно новий засіб для учбового процесу, який буде корисним при дистанційній формі навчання.

Мета роботи – розробка програмного забезпечення з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while» дистанційного навчального курсу «Програмування ІІ» [4].

Об’єкт роботи – створення тренажера для систем дистанційного навчання.

Предмет роботи – програма-тренажер для навчання побудови блок-схем алгоритмів циклічної структури на прикладі циклу while.

Методи роботи – TypeScript фреймворк Angular [5] та фреймворк для розробки нативних десктоп додатків – Electron [6]. Під час створення тренажера використовувалося інтегроване середовище розробки WebStorm.

У теоретичній частині були розглянуті загальні відомості з теми, блок-схема роботи тренажера та обґрунтовано вибір програмних засобів. У практичній частині описаний алгоритм роботи тренажера і його програмна реалізація. За результатами роботи розроблено тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while».

Структура пояснювальної записки до бакалаврської роботи:

- титульний аркуш;

- завдання на випускову кваліфікаційну бакалаврську роботу;

- реферат, що містить предмет, мету, методи, анотацію результатів ключові слова, словосполучення;

- зміст;

- перелік умовних позначень, символів, одиниць, скорочень і термінів;

- вступ;

- суть роботи;

- висновки;

- список використаних джерел;

- додатки.

Обсяг пояснювальної записки: 45 стор., в т.ч. основна частина 31 стор., джерел – 12 назв.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі розробки тренажера

В бакалаврській роботі головною задачею є розробка тренажера з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while» дистанційного навчального курсу «Програмування ІІ».

У зв’язку з цим слід:

1. Ознайомитися з теоретичним матеріалом, пов'язаним з побудовою блок-схем алгоритмів циклічної структури.

2. Ознайомитися з тренажерами подібної тематики. Проаналізувати їх роботу, щоб уникнути негативних аспектів оглянутих програм і взяти (якщо можливо) позитивні.

3. Ознайомитися з темою «Алгоритми циклічної структури на прикладі циклу while».

Даний тренажер буде створюватися в інтегрованому середовищі розробки WebStorm з використанням TypeScript фреймворку Angular, та фреймворку для розробки нативних десктоп додатків – Electron.

Основними завданням, що випливає з задачі роботи є:

- Вибір мови програмування.
- Складання алгоритму роботи програми-тренажера.
- Розробка блок-схеми, яка підлягає програмуванню.
- Програмна реалізація тренажера.
- Перевірка та тестування програми-тренажера.

Крім того, необхідно враховувати вимоги, яким повинен відповідати тренажер.

Основні вимоги до програми:

1. Під час розв’язання задач студент повинен мати можливість прочитати підказки, які будуть пояснювати правильний розв’язок завдань.

2. Після вирішення всіх завдань, студент повинен бачити які завдання він виконав правильно, а які ні.

3. Студент повинен мати змогу отримати доступ до теоретичного матеріалу в будь-який момент не втрачаючи прогрес виконання завдань.

4. Інтерфейс тренажера має бути зрозумілим і простим у використанні.

2 ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд робіт, де розглянуте аналогічне до теми роботи завдання

В процесі написання бакалаврської роботи та розробки програми-тренажера були розглянуті наступні праці студентів, а саме тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу for» дистанційного навчального курсу «Інформатика» (Гмиза Б. Ю.) [7], тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу «do…while» дистанційного навчального курсу «Інформатика» (Недбайло Я. І.) [8] та тренажер з теми «Побудова блок-схем алгоритмів розгалуженої структури» для дистанційного навчального курсу «Програмування ІІ» (Сузанська А. О.) [9].

2.2 Позитивні аспекти розглянутих розробок

Тренажер Гмизи Б. Ю.

1. Тренажер може самостійно виправляти помилки.

2. Правильно обрана відповідь виділяється зеленим кольором, всі інші – червоним.

Тренажер Недбайло Я. І.

1. Наявність теоретичного матеріалу.

Тренажер Сузанської А. О.

1. Пронумеровані кроки.

2. Наявність пояснення неправильних відповідей.

2. Всі блок-схеми зроблені відповідно до вимог стандарту.

2.3 Негативні аспекти розглянутих розробок

Тренажер Гмизи Б. Ю.

1. Відсутність нумерації кроків.

2. Відсутність заявленого англомовного інтерфейсу.

3. Відсутність пояснення помилок.

Тренажер Недбайло Я. І.

1. Відсутність перевірки ситуації, коли користувач не обрав жодної відповіді.

2. Відсутність відомостей про розробника.

3. Немає пояснення помилок.

Тренажер Сузанської А. О.

1. В наявність є тільки україномовний інтерфейс.

2.4 Необхідність та актуальність теми

Необхідність створення тренажера полягає в тому, що створена програма допоможе студентам зрозуміти принципи побудови блок-схем алгоритмів циклічної структури, та покращить їх навички в практичному застосуванні здобутих знань.

Дана задачі є доцільною та актуальною тому, що навчальні тренажери це сучасний підхід до учбового процесу в університетах та інших закладах освіти. Також слід зазначити, що в дистанційному курсі «Програмування ІІ» і в україномовному сегменті інтернету взагалі немає тренажерів, з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while».

3 ТЕОРЕТИЧНА ЧАСТИНА

3.1 Загальні відомості

Блок-схема – це одним з найбільш наочних способів запису алгоритму в графічній формі, за допомогою геометричних фігур, з’єднаних між собою стрілками, які вказують порядок виконання та взаємозв'язок між блоками. Всередині елементів блок-схеми записується їх короткий зміст. Форма елемента блок-схеми визначає тип операції, а текст всередині блоку містить детальний опис певної дії. Стрілки на лініях, що з’єднують елементи блок-схеми, вказують порядок виконання команд, які передбачені алгоритмом. Блок-схеми завдяки наочності полегшують створення нових алгоритмів, розуміння роботи вже створених, і як наслідок їх оптимізацію. Чинні стандарти дозволяють легко застосовувати алгоритми, створені у вигляді блок-схем до будь-яких актуальних на сьогодні мов програмування.

Блоки у блок-схемі з’єднуються лініями зі стрілками. У кожен елемент блок-схеми може входити не менше однієї лінії (зазвичай у верхню, або ліву вершину), з блоку ж, окрім логічного, може виходити тільки одна лінія (як правило, з нижньої, або правої вершини). З логічного блоку завжди виходять хоча б дві лінії: одна у результаті виконання умови, інша – при її невиконанні. Поряд з цими лініями зведено писати результат обчислення умови. Бажано з’єднувати елементи так, щоб лінії блок-схеми не перетинались.

Якщо блок-схема занадто велика і має багато елементів, то зазвичай її розбивають на декілька блок-схем за допомогою символу з’єднувача.

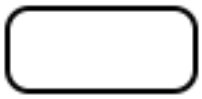
Алгоритм може бути детальним, або спрощеним (деякі зрозумілі блоки можуть не зображувати, інакше це може збільшити розмір алгоритму). Не рекомендується описувати алгоритм на рівні програмного коду.

Блок-схеми поділяють на лінійні, розгалужені, циклічні, з підпрограмами та інші.

Стандарти зображення блоків у алгоритмі, їх розміри, товщина ліній, кут нахилу ліній, написи та інше, регламентуються Державним стандартом «Схеми алгоритмів, програм, даних і систем», а саме: 19.701-90 (ISO 5807-85).

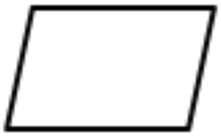
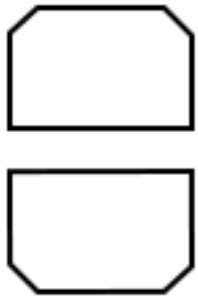
Інформація про структурні блоки, що використовуються при створенні блок-схем описана в таблиці 3.1.

Таблиця 3.1 – Структурні блоки для побудови блок-схем.

Найменування	Позначення	Функція
Термінатор		Символ прямокутника із заокругленими кутами використовується для позначення початку і кінця програми. Як правило всередині символу знаходиться слово, або фраза, яка описує початок чи кінець процесу.
Процес		Символ прямокутника використовується для позначення однієї або декількох елементарних операцій по обробці даних.
Умова		Символ ромба застосовується для позначення операції з одним входом і декількома виходами, один з яких активується після перевірки умови, яка записується всередині символу. Вхід в елемент заведено зображувати лінією, що входить у верхню вершину елементу. Якщо виходів два чи три, то як правило кожен вихід позначається лінією, яка виходить з решти вершин (двох бічних і нижньої). Якщо виходів чотири й більше, то їх слід показувати однією лінією, яка виходить з вершини елемента (зазвичай нижньої), а

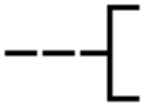
		потім розгалужується за допомогою інших ліній. Поряд з лініями біля виходів записується результат обчислення умови.

Продовження таблиці 3.1

Функція (процедура)		Символ прямокутника з подвійними вертикальними краями використовується для позначення процесу, який описаний в іншому місці програми.
Ввід/Вивід		Символ паралелограма використовується для позначення процесу введення або виведення певної інформації. Подробиці даних можуть бути вказані в коментарях. Елемент використовується, коли носій даних не визначений.
Цикл з параметром		Символ подовженого шестикутника використовується для позначення заголовка циклу з параметрами. Всередині елемента через крапку з комою записуються ім’я змінної з початковим значенням, умова виконання циклу, крок зміни параметра, який повинен впливати на змінну.
Межа циклу		Межа циклу складається з двох елементів і відповідає за початок та кінець циклу. В залежності від типу циклу умова записується або у верхньому елементі, або у нижньому. Якщо алгоритм описує цикл з передумовою, то всі дані умови необхідно записувати у верхній частині, в іншому разі, якщо

		описується цикл з післяумовою, то умову необхідно записувати в нижній частині. Інструкції, що виконуються в тілі циклу записують між цими двома елементами.

Продовження таблиці 3.1

З’єднувач		Символ кола застосовується для позначення обриву лінії та продовження її в іншому місці блок-схеми. Сполучні символи обов’язково повинні мати одне і теж унікальне позначення. Цей символ корисно використовувати в тих ситуаціях, коли вся блок-схема не поміщається на одному листі, або в ній є забагато заплутаних ліній.
Коментар		Цей елемент застосовується для внесення пояснювальних записів про різні етапи роботи алгоритму. Інформація розміщується з боку квадратної дужки та охоплюють всю її висоту. Пунктирна лінія вказує на описуваний елемент або групу елементів. Коли кількість тексту в будь-яких інших символах блок-схеми перевищує їх розмір, слід також використовувати символ коментаря.

3.2 Блок-схеми циклічної структури

Цикл – один із видів керівної конструкції, яка призначена для повторення певних команд деяку кількість разів [10, 11]. Алгоритм, який передбачає багатократне, скінчене виконання певних дій називають циклічним. Кількість ітерацій циклу має бути повністю визначена алгоритмом вирішення задачі, в

іншому випадку відбувається «зациклювання», під час якого процес розв’язання задачі не може бути остаточно завершеним. Деякі циклічні алгоритми можуть містити в собі інші цикли. Подібні структури заведено називати алгоритмами з вкладеними циклами. Циклічні алгоритми бувають з передумовою, післяумовою, безумовними і з лічильником.

В бакалаврській роботі розглядається цикл з передумовою while. Цикл while є універсальним циклом, проте програмісту потрібно сформулювати умови виходу з циклу так, щоб завершення його роботи відбувалося за певну кількість ітерацій. Домогтися цього можна, якщо в тілі циклу запрограмувати зміну значення змінної, яка впливає на умову. Найчастіше такою змінною є лічильник. Якщо необхідно збільшувати (або зменшувати) лічильник на одиницю під час кожної ітерації циклу, використовується операція префіксного або постфіксного інкремента (або декремента).

Число повторень виконання інструкцій тіла циклу while визначається в ході роботи програми та, як правило, заздалегідь невідомо.

При роботі з циклом while потрібно дотримуватися наступних рекомендацій:

• умова продовження виконання роботи циклу записується після ключового слова while;

• умова є логічним виразом, який повертає значення типу bool (примітивний тип даних, що може набувати двох можливих значень, які заведено називати true і false);

• в операторі циклу while після умови продовження виконання тіла циклу крапка з комою не ставиться;

• в тілі циклу while повинні бути присутні інструкції, які впливають на умову виконання циклу, або оператори, що можуть перервати його роботу, це необхідно для успішного завершення роботи циклу;

• цикл while – це цикл з передумовою, якщо умова з самого початку повертає хибне значення, то тіло циклу взагалі не буде виконано жодного разу;

• цикл while зазвичай застосовується в тих же ситуаціях, що і цикл for. Найзручніше використовувати його в тих випадках, коли умова циклу може не виконатися;

• у зв’язку з попередніми твердженнями цикл while вважається найбільш універсальним циклом.

3.3 Розробка блок-схеми, яка підлягає програмуванню

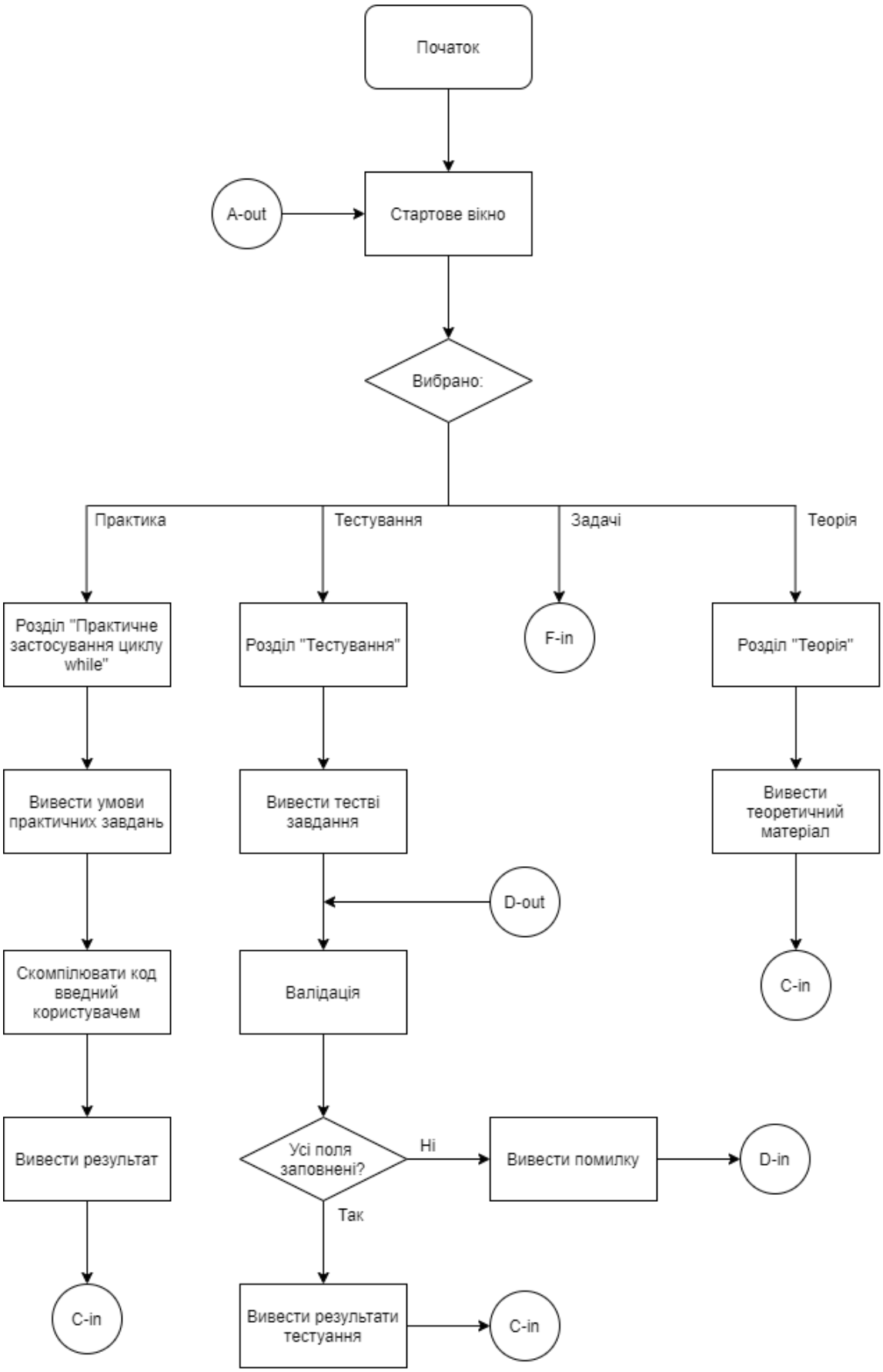


Рисунок 3.1 – Блок-схема

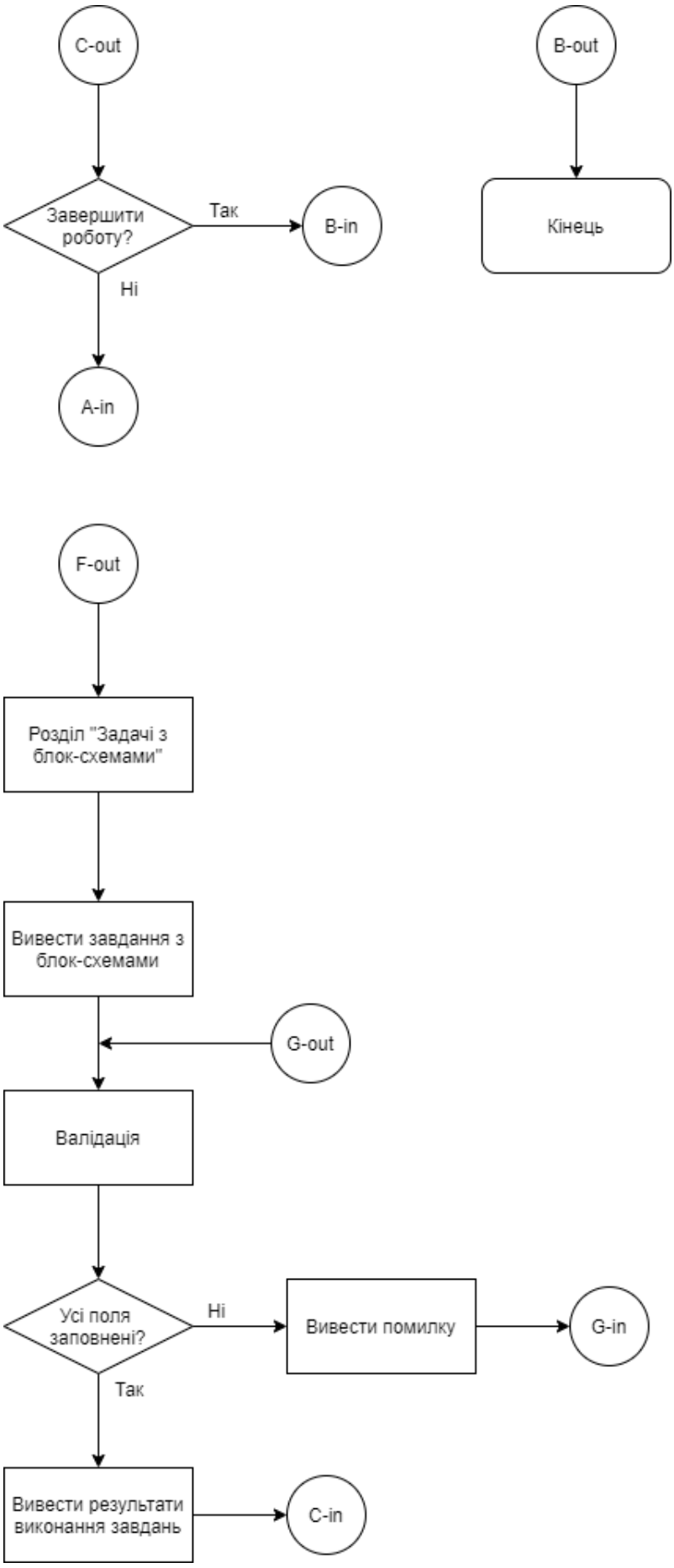


Рисунок 3.2 Продовження блок-схеми

3.4 Обґрунтування вибору програмних засобів

Для розробки тренажера було використано TypeScript фреймворк Angular. Одна з основних переваг цієї технології в тому, що програми написанні на Angular можна збирати під різні платформи будь-то ПК, смартфони на базі операційних систем Android і iOS, веб додатки і PWA додатки.

Перша версія фреймворку була розроблена у 2010 році компанією Google на мові програмування JavaScript і мала назву AngularJS. Але згодом компанія змінила стратегію розвитку цієї технології і у 2016 році випустила версію Angular 2.0, яка була написана на мові програмування TypeScript.

Програми розроблені на фреймворку Angular є SPA додатками (Single Page Application), тобто вся програма це одна сторінка, в яку в залежності від дій користувача будуть завантажуватися деякі дані і видалятися інші. Перевага такого підходу в тому, що програма стає швидше працювати, користувачеві під час навігації не потрібно чекати коли завантажаться інші скрипти, розмітка, чи стилі, бо всі данні, включаючи дані інших сторінок, будуть завантажені під час першого запуску програми й не будуть перезавантажуватися повторно.

Фреймворк Angular пропагує компонентний підхід до розробки користувацьких інтерфейсів. Це означає, що всі елементи інтерфейсу програми, які бачить користувач, є компонентами, або частинами цих компонентів. Тому коли в програмі є декілька однакових елементів інтерфейсу (або майже однакових), то це зручніше зробити одним компонентом, який буде приймати деякі параметри та показувати користувачеві однакові блоки інформації, але з різними даними. Такий підхід може скоротити загальну тривалість розробки програм і збільшити читаність коду.

Для того, щоб зібрати Angular проєкт для ПК, було використано допоміжний фреймворк Electron. Цей фреймворк був розроблений компанією GitHub у 2013 році на мовах програмування C++, JavaScript, TypeScript і Python.

Electron дозволяє розробляти, та збирати нативні десктоп програми для різних операційних систем, будь-то Windows, MacOS чи Linux.

До переваг Electron можна віднести високу якість програмного забезпечення розробленого на ньому. В каталозі додатків Electron, знаходиться понад 600 програм, серед яких доволі багато успішних, розробленими такими компаніями як Microsoft, Twitch, Google і т.д.

На базі Electron були розроблені наступні програми: Visual Studio Code, Atom, Skype, Discord, десктопний клієнт Wordpress та багато інших програм.

4 ПРАКТИЧНА ЧАСТИНА

4.1 Алгоритм роботи тренажера

Програма є portable-додатком і не потребує встановлення, для того, щоб нею користуватися достатньо відкрити відповідний exe файл.

Після запуску програми відкривається стартове вікно (див. рис. 4.1), де по центру знаходиться напис «Тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while», а під ним чотири розділи для роботи з тренажером: «Практичне застосування циклу while», «Тестування», «Завдання з блок-схемами» і «Теорія».

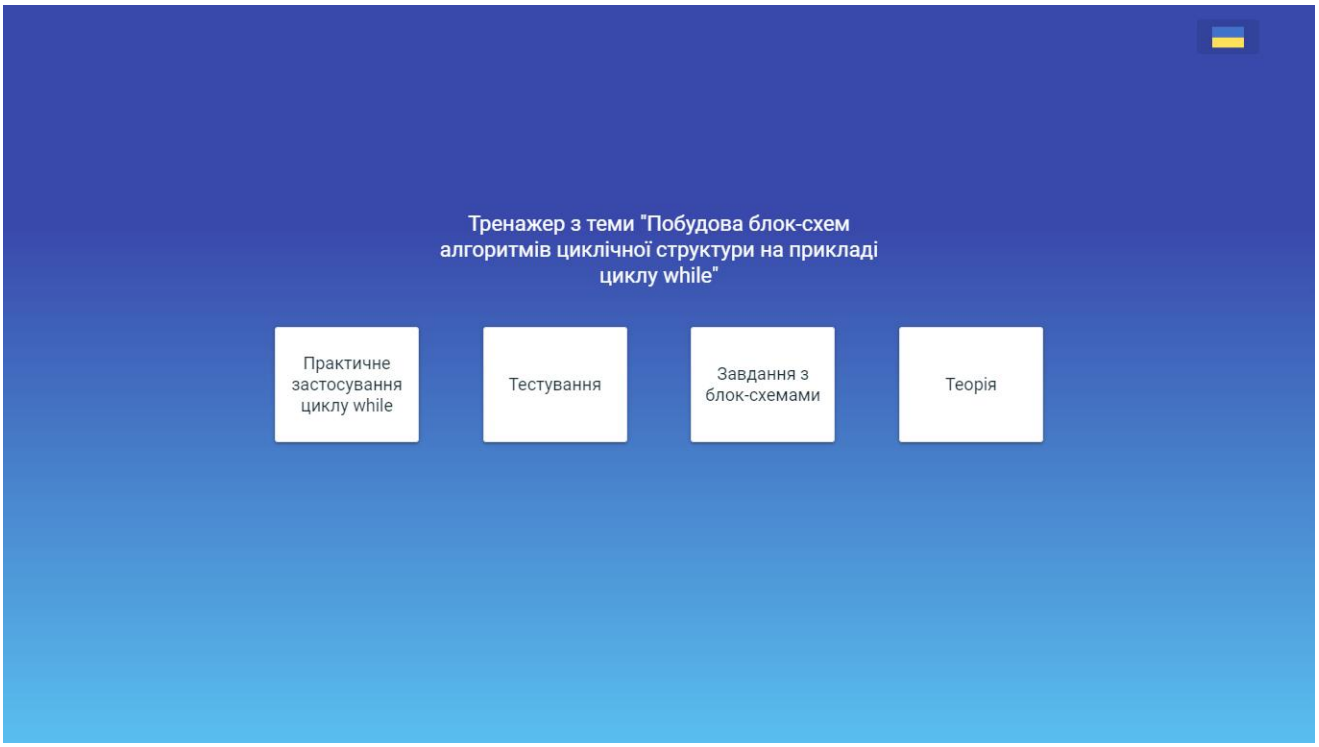


Рисунок 4.1 – Стартове вікно тренажера

У правому верхньому куті знаходиться перемикач мови, тренажер підтримує українську й англійську мови. Щоб змінити мову тренажера достатньо натиснути на перемикач і вибрати з випадного списку прапор країни, мова якої буде застосована до інтерфейсу програми (див. рис. 4.2). По замовчувані вибрана українська мова.

У розділі «Практичне застосування циклу while» користувач має змогу писати код на мові C++, компілювати його і переглядати результат роботи коду. У даному

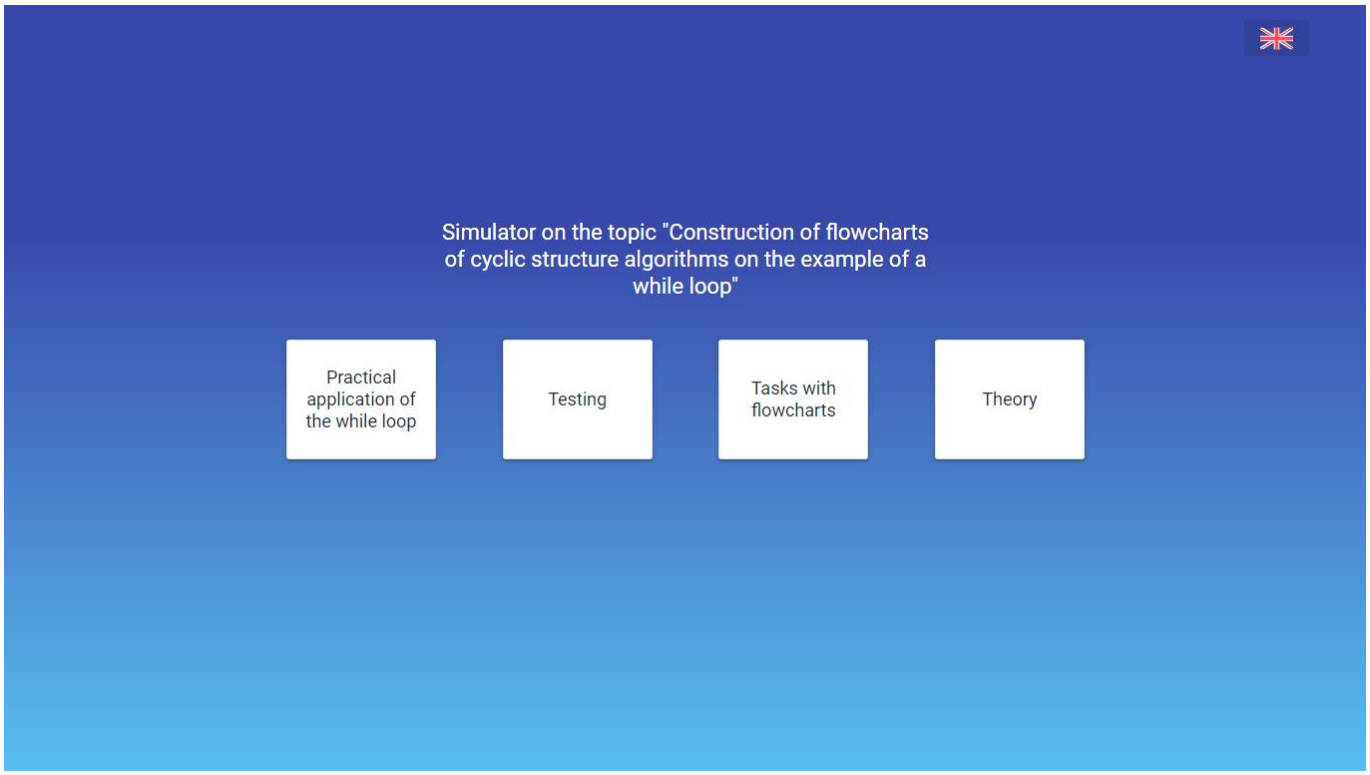


Рисунок 4.2 – Стартове вікно тренажера англійською мовою

вікні знаходяться п’ять задач, які пов’язані з циклічними алгоритмами. Кожна з задач потребує написати код, який реалізує той чи інший циклічний алгоритм з використанням циклу while (див. рис. 4.3).

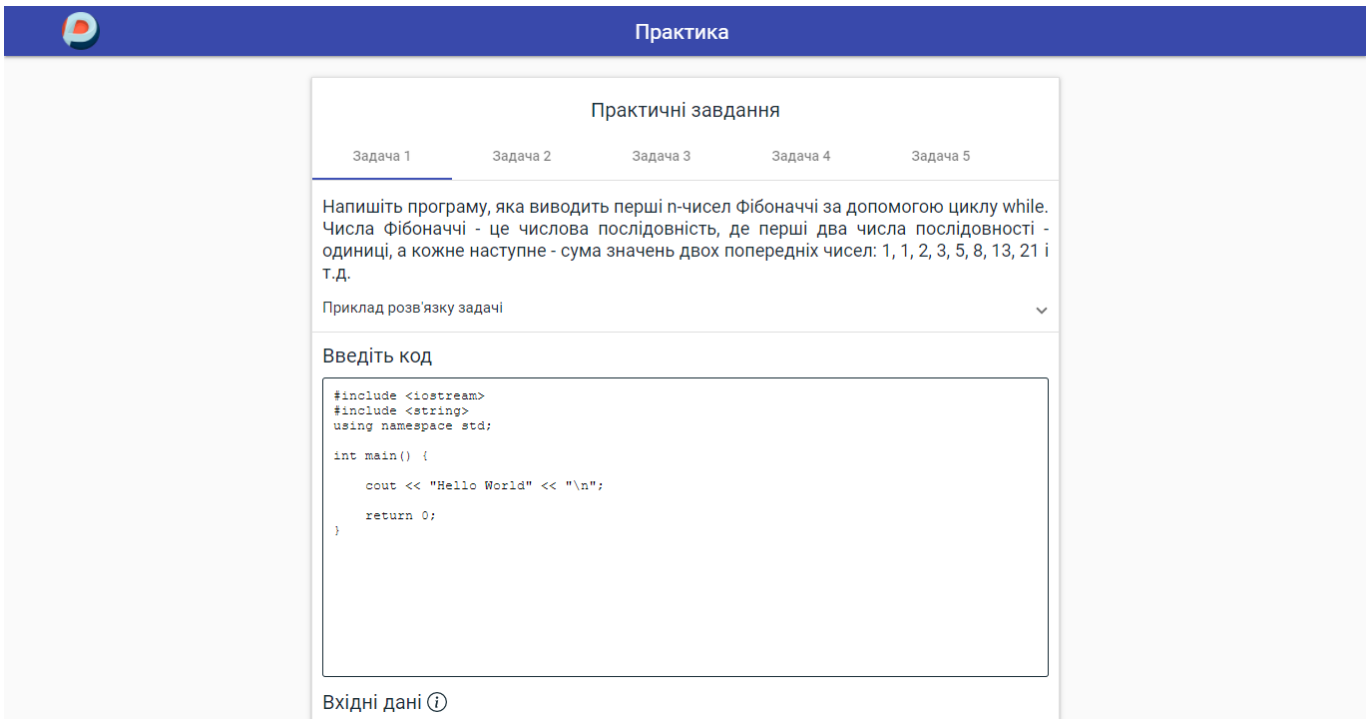


Рисунок 4.3 – Розділ «Практичне застосування циклу while»

Під умовою кожного завдання є напис «Приклад розв’язку задачі», якщо на нього натиснути, то відкриється блок коду в якому показана реалізація розв’язку завдання на мові C++ (див. рис. 4.4).

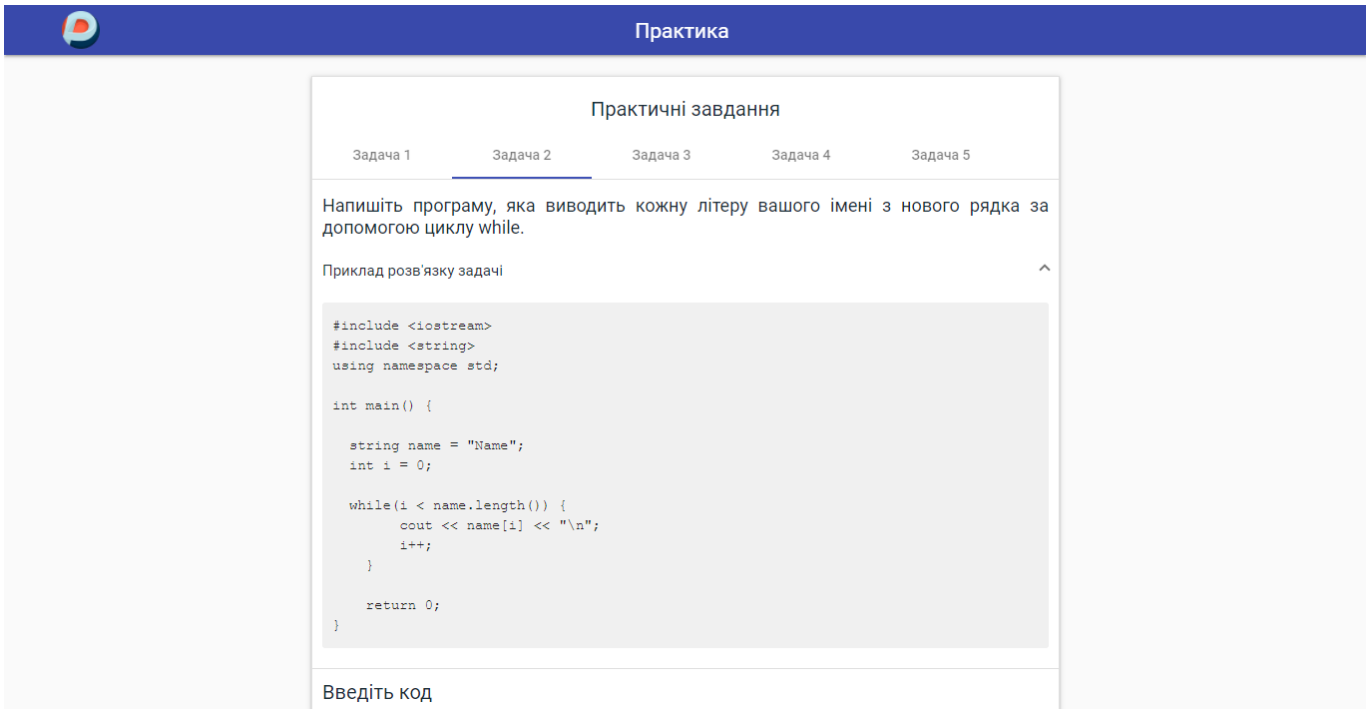


Рисунок 4.4 – Приклад розв’язку задачі

У текстовому полі «Введіть код» знаходиться базовий код, який виводить напис «Hello World». Саме в цьому полі користувач може писати власний код. Щоб скомпілювати його потрібно натиснути на кнопку «Компіляція» і в текстовому полі «Результат» виведеться результат роботу коду (див. рис. 4.5). У текстове поле «Вхідні дані» за потреби вводиться інформація, яка в коді зчитується за допомогою оператора cin. Якщо в програмі декілька операторів cin, то дані вводяться послідовно з нового рядка.

У розділі «Тестування» знаходиться список тестових завдань у кожному з яких п’ять варіантів відповіді й лише одна правильна (див. рис. 4.6). Тематика тестів пов’язана з циклічними алгоритмами, циклом while і блок-схемами. Якщо користувач не завершить тестування, повернеться до стартового вікна і вибере якийсь інший розділ, то весь прогрес тестування збережеться, і користувач зможе продовжити його після роботи з іншими розділами тренажера. Прогрес виконання

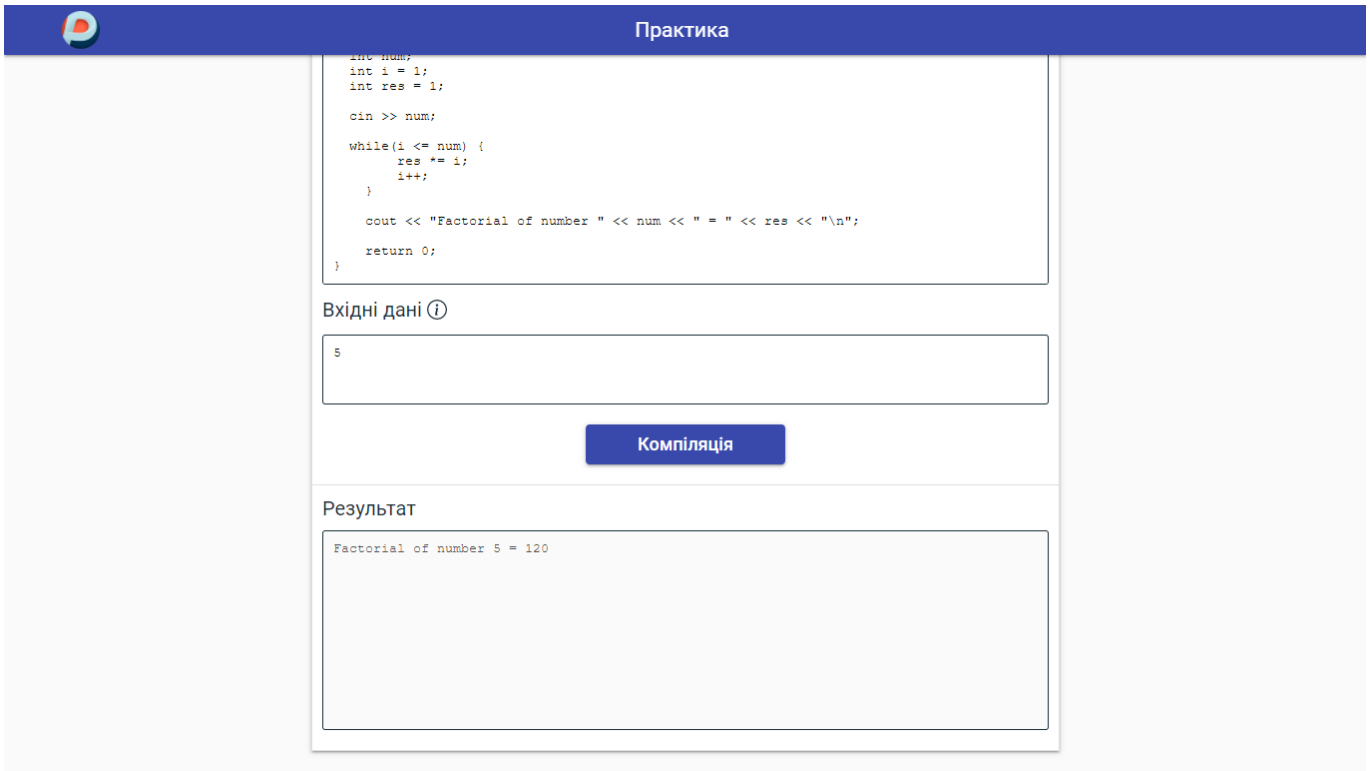


Рисунок 4.5 – Результат компіляції коду введеного користувачем

тестових завдань зберігається навіть у тому випадку, якщо користувач вирішить змінити мову тренажера. Під час кожного нового проходження тесту, усі завдання

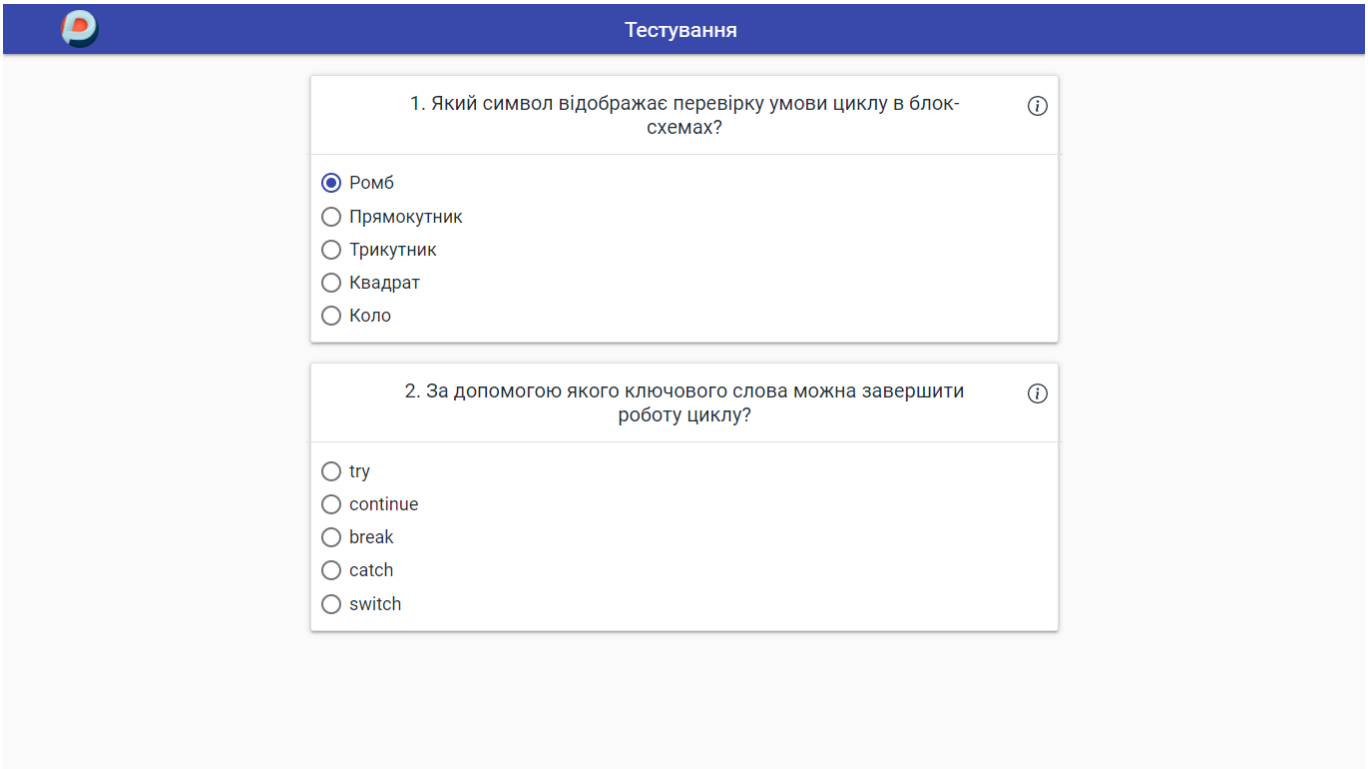


Рисунок 4.6 – Розділ «Тестування»

та відповіді рандомно перемішуються. Якщо під час проходження тестування користувач не вибере відповідь в одному чи декількох завданнях, то з’явиться вікно помилки яке повідомляє, що користувач не завершив тестування (див. рис. 4.7). Щоб закрити повідомлення потрібно натиснути на відповідний символ у правому верхньому куті вікна.

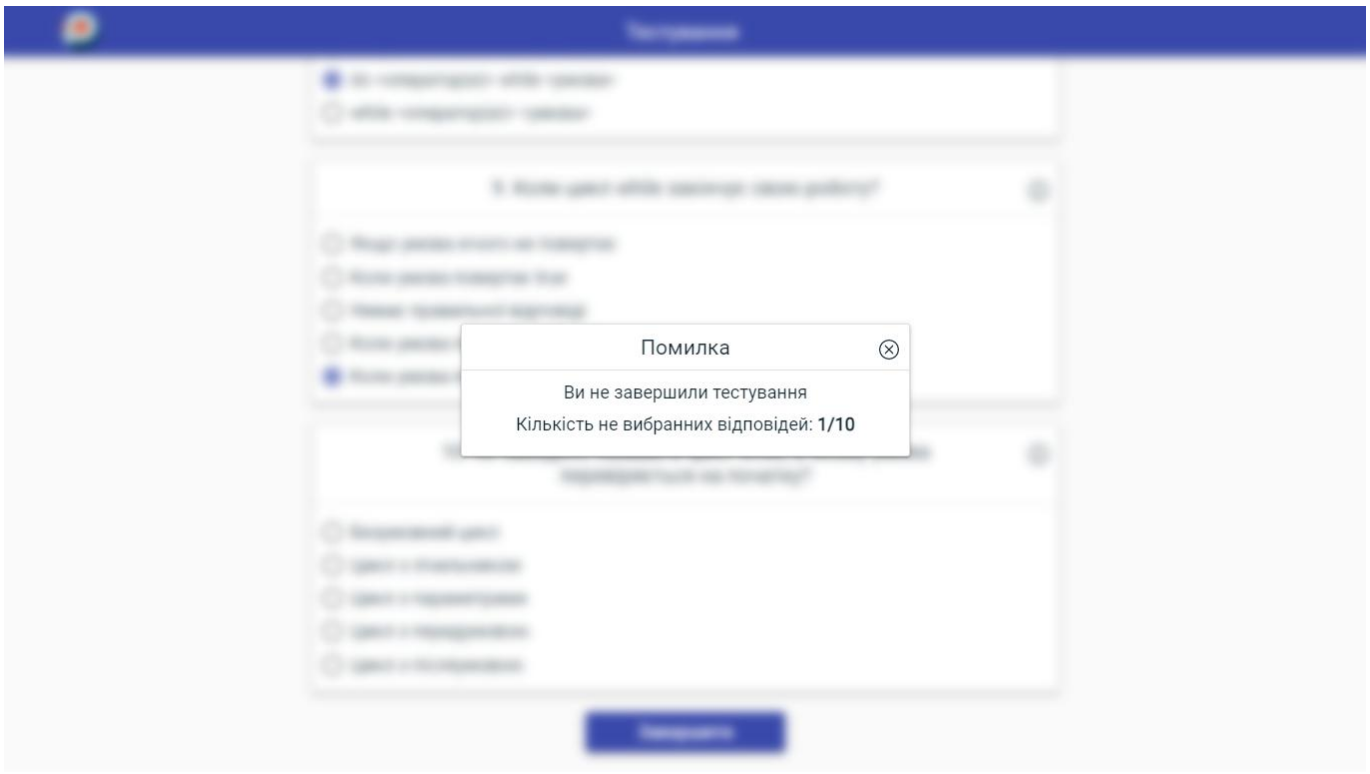


Рисунок 4.7 – Вікно помилки

В іншому разі, якщо користувач виконав усі завдання, то з’явиться вікно в якому будуть написані результати тестування у кількісному та відсотковому співвідношеннях (див. рис. 4.8).

Після завершення тестування заголовки питань на які користувач відповів правильну будуть виділені зеленим кольором, всі інші червоним (див. рис. 4.9). Радіокнопки та текст варіантів відповіді будуть заблоковані й виділені сірим кольором. Внизу сторінки кнопка «Завершити» заміниться кнопкою «Спочатку». Якщо натиснути на цю кнопку, то результати попереднього тестування видаляться, екран автоматично переміститься у верх сторінки, а всі завдання та варіанти відповіді рандомно перемішаються.

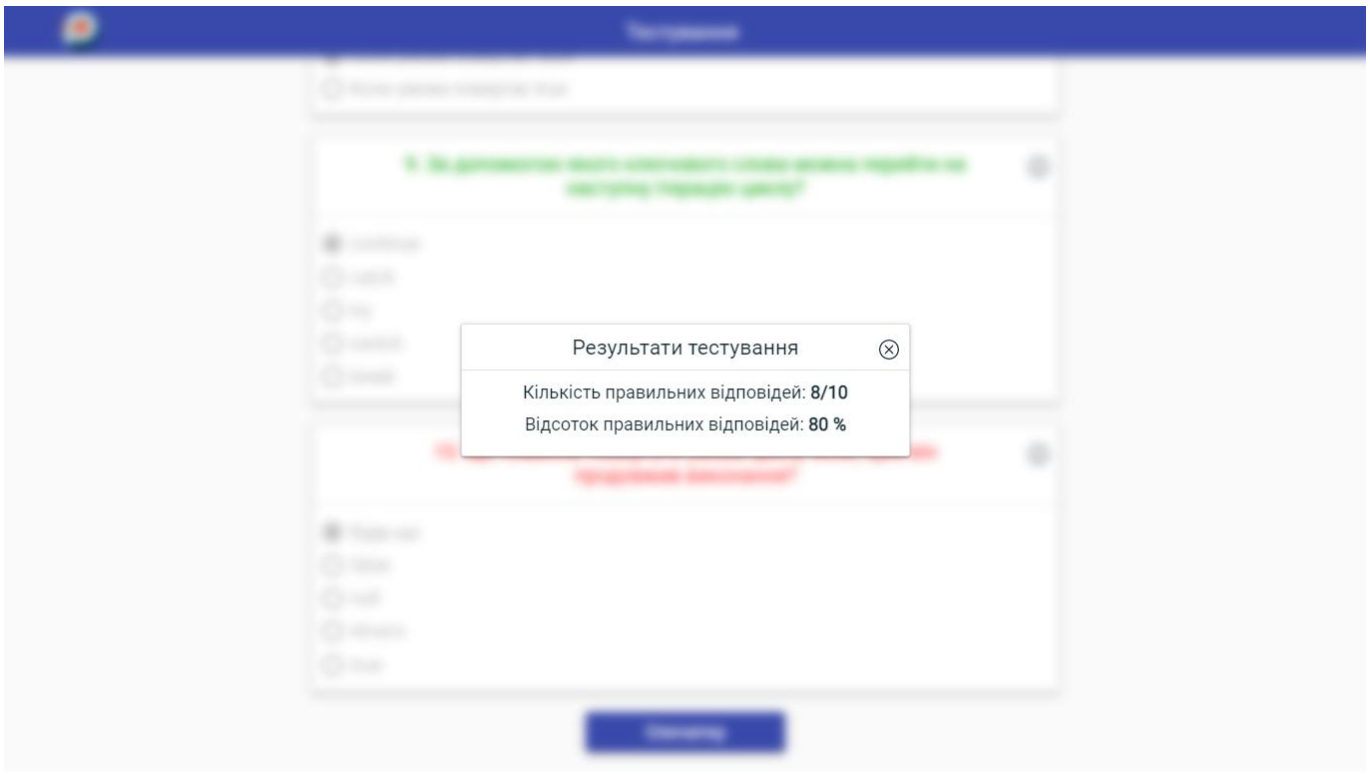


Рисунок 4.8 – Вікно результатів тестування

Для того, щоб перезапустити тестування не обов’язково натискати на кнопку «Спочатку». Користувачеві достатньо покинути розділ «Тестування», зайти в нього

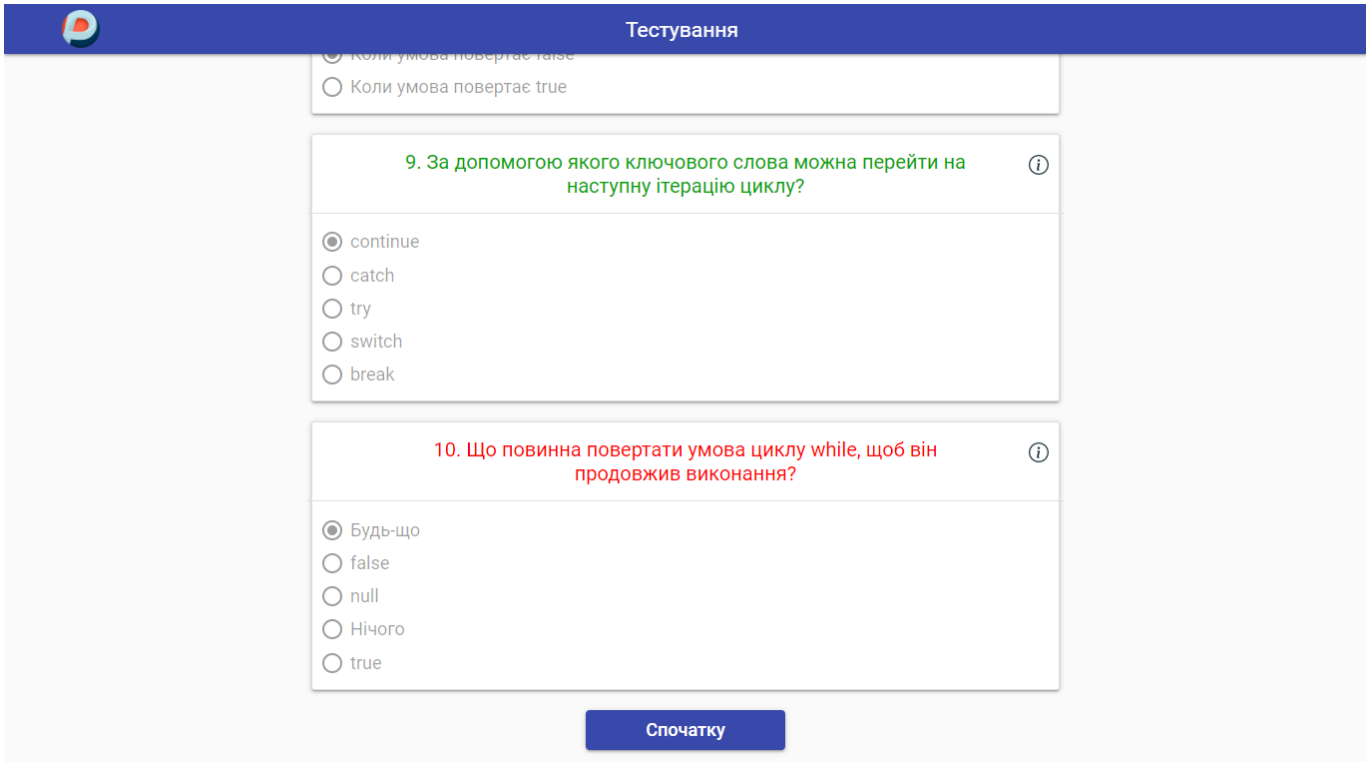


Рисунок 4.9 – Вигляд розділу після завершення тестування

знову і всі вищеперераховані дії також виконаються. Справа біля кожного заголовку є символ схожий на букву «і», якщо на нього натиснути з’явиться модальне вікно «Підказка» з поясненням правильного варіанта відповіді (див. рис. 4.10).

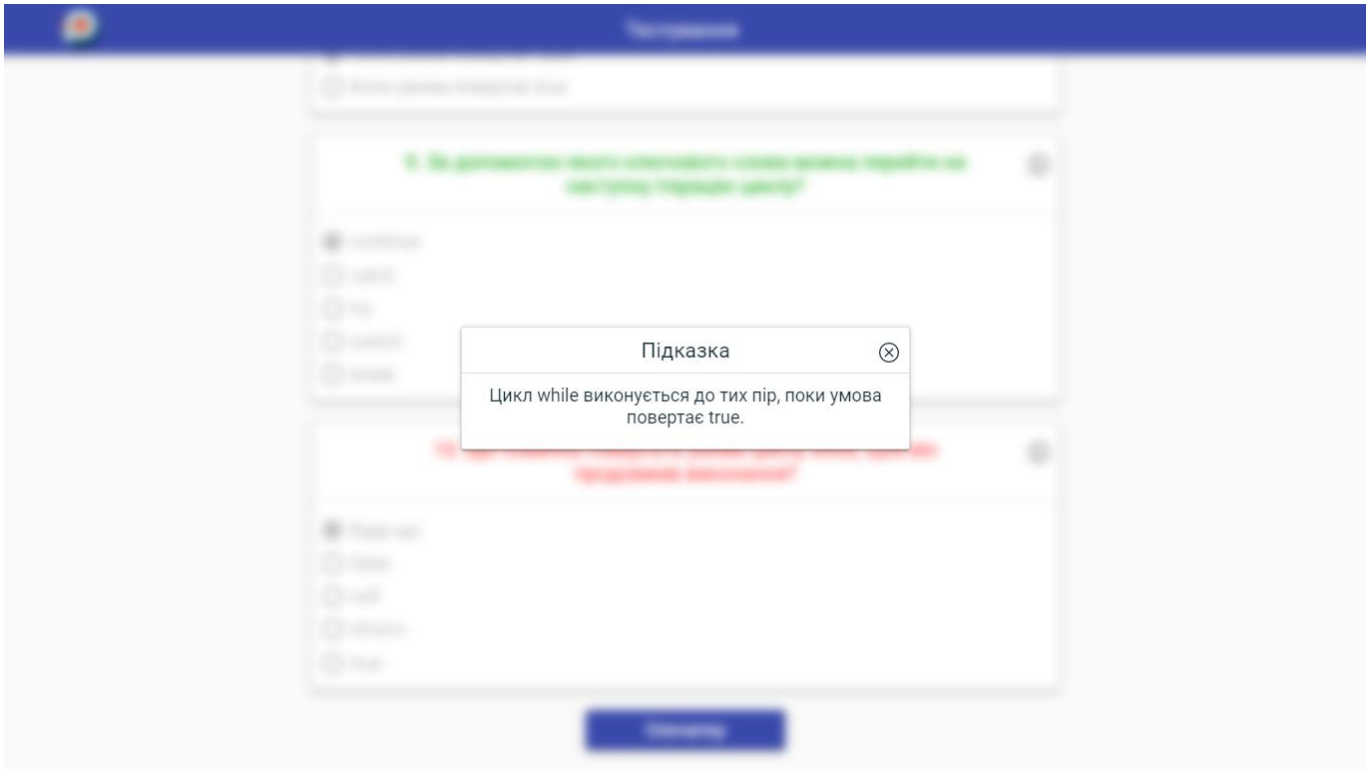


Рисунок 4.10 – Підказка

Розділ «Завдання з блок-схемами» працює аналогічним чином, як і розділ «Тестування». Єдина відмінність в тому що, в даному розділі знаходяться блок-схеми алгоритмів циклічної структури (див. рис. 4.11). До кожного завдання прикріплена блок-схема і користувач повинен проаналізувати кожну з низ, зрозуміти яким чином будуть працювати програми написані по досліджуваним блок-схемам і виконати завдання.

Якщо натиснути на якусь із блок-схем, то вона відкриється у повноекранному режимі (див. рис. 4.12).

Розділ «Теорія» містить детальний опис теоретичного матеріалу з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while». Цей розділ дає відповіді на такі питання як: «Що таке цикл», «Що таке блок-

схема», «Як будувати блок-схеми» та інше. У ньому є приклад блок-схеми, яка демонструє

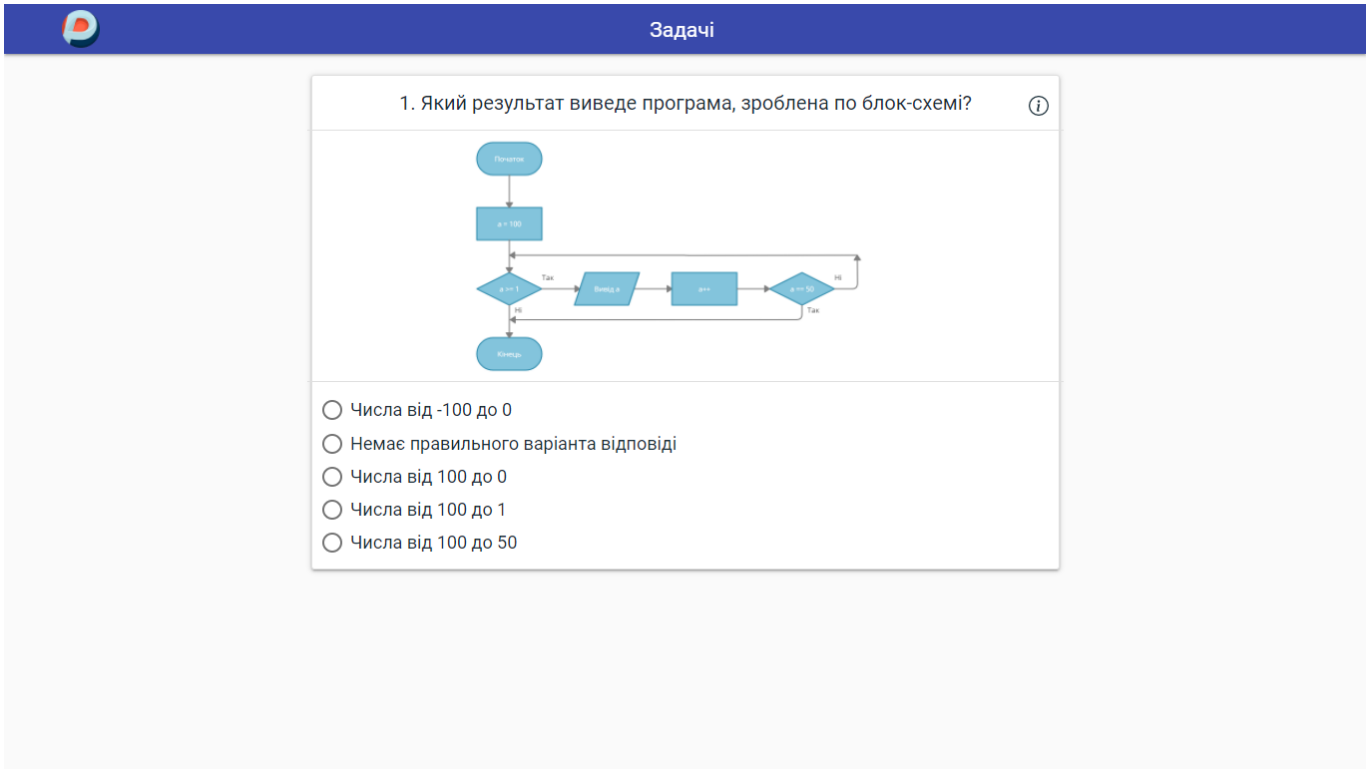


Рисунок 4.11 – Розділ «Завдання з блок-схемами»

роботу циклічного алгоритму, де детально розглядається кожен елемент блок-схеми, та аналізується результат роботи програми, яка буду зроблена по ній (див. рис. 4.13).

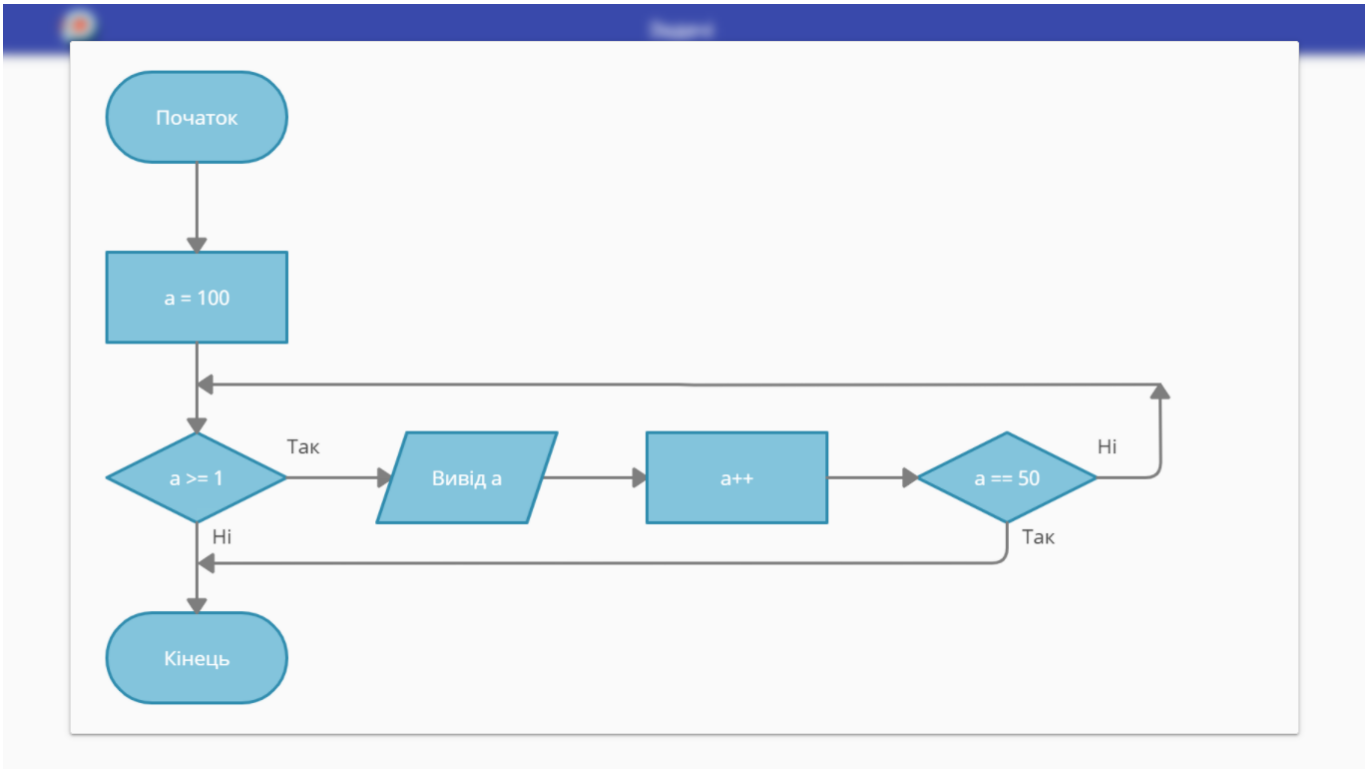


Рисунок 4.12 – Повноекранний вигляд блок-схеми

Щоб вийти з тренажера, користувач повинен натиснути на відповідну іконку в правому верхньому куті програми.

Рисунок 4.13 – Розділ «Теорія»

4.2 Опис програмної реалізації

Тренажер написаний на фреймворку Angular і оперує наступними сутностями: компоненти, сервіси, стріми, директиви.

Компонент - це клас, який використовується для відображення однієї чи декількох частин користувацького інтерфейсу. Щоб повідомити фреймворк, що клас є компонентом потрібно позначити цей клас через декоратор @Component. Зазвичай компонент в Angular складається із трьох файлів: name.component.html (файл, який містить базову розмітку компонента), name.component.scss (файл, який налаштовує зовнішній вигляд розмітки) і name.component.ts (файл, який містить функціонал компонента написаний на мові TypeScript). Стилі компонентів частіше всього розробляються на препроцесорі SASS, але Angular дає можливість використовувати інші препроцесори будь-то Less, Stylus, SCSS або звичайний CSS.

Сервіс – це клас в якому зберігається основний функціонал програми. В сервісах прийнято оперувати даними, отримувати їх з сервера, робити над ними якісь операції та інше. Щоб позначити клас як сервіс, використовується декоратор @Injectable.

Стрім – це сутність, яка дозволяє реалізувати парадигму реактивного програмування. Щоб створити стрім, потрібно оголосити спеціальну змінну за допомогою вбудованої в Angular бібліотеки RxJS. Для того, щоб отримати зі стріму якусь інформацію, на нього потрібно підписатися, такі фрагменти коду, де відбувається підписка, заведено називати підписниками. Підписники під час роботи програми постійно очікують ту чи іншу інформацію від стріму. Коли в стрім додаються дані, то всі підписники відразу ж їх отримують, незалежно від того, де в коді вони знаходяться. Після отримання інформації, підписники обробляють її і продовжують очікувати нові дані.

Директива – сутність, яка дозволяє описувати просту логіку на мові TypeScript, всередині HTML тегів. В Angular є вбудовані директиви, такі як *ngIf, *ngSwitch, *ngFor, а також кастомні, які створює сам програміст. Щоб створити власну директиву необхідно оголосити клас і позначити його через декоратор @Directive.

В головній директорії програми знаходяться два розділи: main і shared. Розділ main має однойменний компонент, який відповідає за головне вікно тренажера. Цей компонент містить розмітку головної сторінки і метод навігації по програмі (див. рис. 4.14).

```
public async navigate(path: string): Promise<void> {
  await this.router.navigate( commands: [path])
}
```

Рисунок 4.14 – Метод навігації

Розділ, який містить загальні для всієї програми сутності, прийнято називати shared, і теоретично повинен бути в кожній програмі розробленій на фреймворку Angular, хоча на практиці це не завжди так. В розділі shared знаходяться два підрозділи: components, services.

В директорії components знаходяться загальні для тренажера компоненти: header, modal, photo-viewer, quiz, sandbox, select-language, testing, theory.

Компонент header відповідає за блок користувацького інтерфейсу, який знаходиться вгорі кожного розділу тренажера. В розмітці цього компонента виводиться назва розділу з яким працює користувач і логотип тренажера. Компонент містить наступні методи: toMain, observeCurrentTitle (див. рис. 4.15). Метод toMain відповідає за повернення до головного вікна програми, а метод observeCurrentTitle містить підписку на стрім headerTitle$, який відслідковує зміну заголовка розділу і оновлює його в розмітці.

```
public async toMain(): Promise<void> {
  await this.router.navigate( commands: ['main'])
}

private observeCurrentTitle(): void {
  this.uiService.headerTitle$
    .pipe(takeUntil(this.destroy$))
    .subscribe( next: (res: string) => {
      this.headerTitle = res
    })
}
```

Рисунок 4.15 – Методи компонента header

Компонент modal відповідає за показ і закриття всіх модальних вікон, які є в тренажері. В розмітці компонента виводиться заголовок модального вікна, наприклад «Підказка», і текст повідомлення. Компонент містить наступні методи: observeOpenModal, closeModal (див. рис. 4.16). Метод observeOpenModal відстежує відкриття і закриття модальних вікон, якщо модальне вікно відкривається, то метод вимикає можливість скролити сторінку, якщо модальне вікно закривається, то скролл вмикається. Метод closeModal закриває модальні вікна.

Компонент photo-viewer відповідає за показ різних зображень у повноекранному режимі. Цей компонент використовується у розділах «Тестування», «Завдання з блок-схемами» і «Теорія».

```
private observeOpenModal(): void {
  this.modalService.openModal$
    .pipe(takeUntil(this.destroy$))
    .subscribe( next: (res: Modal) => {
      if (res?.isOpen) {
        this.renderer.setStyle(document.body,  style: 'overflow',  value: 'hidden')
      } else {
        this.renderer.setStyle(document.body,  style: 'overflow',  value: 'auto')
      }
    })
}

public closeModal(): void {
  this.modalService.openModal$.next(<Modal>{})
}
```

Рисунок 4.16 – Методи компонента modal

Компонент має наступні методи: afterOpenPhotoProcess, afterHidePhotoProcess, hidePhoto (див. рис. 4.17). Логіка роботи компонента подібна, до логіки компонента modal. Методи afterOpenPhotoProcess викликається після ініціалізації компонента і вимикає можливість скролити сторінку. Метод afterHidePhotoProcess викликається під час видалення компонента і вмикає можливість скролити сторінку. Метод hidePhoto закриває вікно повноекранного перегляду зображення.

```
private afterOpenPhotoProcess(): void {
  this.renderer.setStyle(document.body,  style: 'overflow', value: 'hidden')
}

private afterHidePhotoProcess(): void {
  this.renderer.setStyle(document.body,  style: 'overflow', value: 'auto')
}

public hidePhoto(): void {
  this.uiService.showPhotoViewer$.next(<PhotoViewer>{})
}
```

Рисунок 4.17 – Методи компонента photo-viewer

Компонент quiz відповідає за розділ «Завдання з блок-схемами». В розмітці розділу описується форма елементами якої є блоки з питаннями, підказками і варіантами відповіді. В компоненті знаходяться наступні методи: loadQuizzes, select (див. рис. 4.18). Метод loadQuizzes завантажує дані завдань з блок-схемами. Метод select спрацьовує, коли користувач вибирає якийсь із варіантів відповіді в завданнях.

```
private loadQuizzes(): void {
  const saveQuizzes: Log = this.trainerDataService.saveQuizzes
  Object.entries(saveQuizzes).forEach((el: [string, LogItem]) => {
    this.form.controls[`quiz_${el[0]}`].patchValue(el[1].answersId)
    this.select(+el[0], el[1].answersId)
  })
}

public select(quizId: number, answerId: number, radio?: MatRadioButton): void {
  if (this.disableRadioButtons) return
  if (radio && !radio.checked) radio.checked = true

  this.trainerDataService.saveQuiz(quizId, answerId)
  const item: Quiz = this.quizzes.find( predicate: (el: Quiz) => el.id === quizId)
  const res: Answer = item.answers.find( predicate: (el: Answer) => el.id === answerId)
  this.answers[quizId] = res.isRight
}
```

Рисунок 4.18 – Методи компонента quiz

Компонент sandbox відповідає за розділ «Практичне застосування циклу while». В розмітці компонента знаходяться три текстові поля. В перше вводиться код, в друге дані, які зчитуються із коду за допомогою оператора cin (якщо такі дані є), в третє виводиться результат компіляції. В компоненті знаходяться наступні методи: codeInput, dataInput (див. рис. 4.19). Метод codeInput зчитує код з першого текстового поля і зберігає його для подальшої компіляції. Метод dataInput зчитує дані які будуть необхідні для роботи з оператором cin. Також в компоненті присутній метод submit, який призначений для компіляції C++ коду за допомогою API Judge0 CE. Код методу наведений у додатку А.

Компонент select-language відповідає за зміну мови інтерфейсу тренажера. В розмітці компонента описується випадний список мов. Компонент містить наступні

```
public codeInput(event: Event): void {
  this.codeInputValue = (<HTMLTextAreaElement>event.target).value
}

public dataInput(event: Event): void {
  this.dataInputValue = (<HTMLTextAreaElement>event.target).value
}
```

Рисунок 4.19 – Методи компонента sandbox

методи: observeChangeLanguage, setCurrentLanguage, changeMenuStatus (див. рис. 4. 20). Метод observeChangeLanguage відстежує момент зміни мови в випадному списку, і викликає метод setLanguage із languagesService, який змінює мову всього інтерфейсу. Метод setCurrentLanguage викликається під час зміни мови в випадному списку мов. Метод changeMenuStatus закриває, або відкриває, випадний список мов.

```
private observeChangeLanguage(): void {
  this.languagesService.currentLanguage$
    .pipe(takeUntil(this.destroy$))
    .subscribe( next: (lang: Language) => {
      this.languagesService.setLanguage(lang)
    })
}

public setCurrentLanguage(lang: Language): void {
  this.languagesService.currentLanguage$.next(lang)
  this.trainerDataService.initServiceValues()
}

public changeMenuStatus(): void {
  this.isOpen = !this.isOpen
}
```

Рисунок 4.20 – Методи компонента select-language

Компонент testing відповідає за розділ «Тестування». В розмітці компонента міститься список тестів з варіантами відповіді і підказками. В компоненті є наступні методи: loadQuestions, select (див. рис. 4.21). Метод loadQuestions завантажує дані тестування. Метод select спрацьовує коли користувач вибирає якийсь із варіантів відповіді в тестах.

```
private loadQuestions(): void {
  const saveQuestions: Log = this.trainerDataService.saveQuestions
  Object.entries(saveQuestions).forEach((el: [string, LogItem]) => {
    this.form.controls[`test_${el[0]}`].patchValue(el[1].answersId)
    this.select(+el[0], el[1].answersId)
  })
}

public select(questionId: number, answerId: number, radio?: MatRadioButton): void {
  if (this.disableRadioButtons) return
  if (radio && !radio.checked) radio.checked = true

  this.trainerDataService.saveQuestion(questionId, answerId)
  const item: Question = this.questions.find( predicate: (el: Question) => el.id === questionId)
  const res: Answer = item.answers.find( predicate: (el: Answer) => el.id === answerId)
  this.answers[questionId] = res.isRight
}
```

Рисунок 4.21 – Методи компонента testing

Компонент theory відповідає за розділ «Теорія». В розмітці компонента знаходиться текст розділу, зображення і таблиця, яка описує елементи блок-схем. Компонент не має методів.

В директорії services знаходяться загальні для тренажера сервіси: animation, language, trainer-data, ui, utils.

Сервіс animation відповідає за анімації тренажера. В сервісі є методи scrollTo і createTimeline, поле isInitMainPageAnimation і геттер gsap (див. рис. 4.22). Якщо поле isInitMainPageAnimation рівне значенню true, то на головній сторінці тренажера будуть відтворюватися анімації, якщо false, то анімацій не буде. Це необхідного для того, щоб анімації на головній сторінці спрацювали тільки один раз. Геттер gsap повертає головний об’єкт бібліотеки GSAP (одна із найпопулярніших бібліотек по роботі з анімаціями). За допомогою об’єкта gsap

створюються всі анімації в програмі. Метод scrollTo відтворює системний скролл. Прикладом роботи цього методу є плавний скролл вгору сторінки коли користувач натискає кнопку «Спочатку» в розділах «Тестування» і «Завдання з блок-схемами». Наступний метод

```
public isInitMainPageAnimation: boolean = false

public get gsap(): GsapType {
  return gsap
}

public scrollTo(x: number, y: number, duration: number): void {
  gsap.to(window, vars: {duration, scrollTo: {y, x}})
}

public createTimeline(params: TimelineParams): Timeline {
  return gsap.timeline(params)
}
```

Рисунок 4.22 – Тіло сервісу animation

createTimeline використовуючи бібліотеку gsap створює таймлайн. Таймлайн – це функція, яка дозволяє контролювати порядок і час виконання однієї, або декількох анімацій.

Сервіс language відповідає за зміну мови інтерфейсу. В сервісі знаходяться поля textContent і key, стрім currentLanguage$ і метод setLanguage (див. рис. 4.23). Поле textContent це об’єкт, в якому зберігаються дані всіх мов тренажера. Поле key це ключ, який містить поточну мову інтерфейсу. Стрім currentLanguage$ потрібен для відстежування зміни мови. Метод setLanguage викликається під час зміни мови тренажера.

```
public currentLanguage$: BehaviorSubject<Language> = new BehaviorSubject<Language>( _value: 'UA')

public readonly textContent: LanguageContent = {
  UA: {...},
  EN: {...}
}

public key: LanguageContentItem = this.textContent.UA

public setLanguage(lang: Language): void {
  this.key = this.textContent[lang]
}
```

Рисунок 4.23 – Тіло сервісу language

Сервіс trainer-data зберігає всі дані тестування і завдань з блок-схемами. В ньому знаходяться шість полів і один метод initServiceValues (див. рис. 4.24). В полях questionsArray, quizzesArray після ініціалізація сервісу зберігаються дані тестів і завдань з блок-схемами на поточній мові інтерфейсу за допомогою методу initServiceValues. Поля shuffleQuestions і shuffleQuizzes необхідні для збереження рандомно перемішаних даних завдань. Полях questionsLog, quizzesLog потрібні для збереження прогресу виконання завдань.

```
private questionsArray: Question[] = []
private quizzesArray: Quiz[] = []

public shuffleQuestions: Question[] = []
public shuffleQuizzes: Quiz[] = []

private questionsLog: Log = {}
private quizzesLog: Log = {}

public initServiceValues(): void {
  this.questionsArray = [...]

  this.quizzesArray = [...]
}
```

Рисунок 4.24 – Тіло сервісу modal

Сервіс ui відповідає за різні елементи користувацького інтерфейсу, такі як модальні вікна, вікно перегляду зображень, та інше. В ньому знаходяться три стріми headerTitle$, openModal$ і showPhotoViewer$ (див. рис. 4.25). Стрім headerTitle$ спрацьовує, коли змінюється заголовок розділу, openModal$ коли відкриваються модальні вікна в тренажері, а showPhotoViewer$ коли відкривається вікно перегляду зображення.

```
public headerTitle$: BehaviorSubject<string> = new BehaviorSubject<string>( _value: '')
public openModal$: BehaviorSubject<Modal> = new BehaviorSubject<Modal>(<Modal>{})
public showPhotoViewer$: BehaviorSubject<Viewer> = new BehaviorSubject<Viewer>(<Viewer>{})
```

Рисунок 4.25 – Тіло сервісу ui

В сервісі utils знаходяться допоміжні методи програми. Метод timeout створює штучну затримку, на певний проміжок часу. Метод shuffle рандомно перемішує елементи масиву (див. рис. 4.26). Саме цей метод використовується для перемішування тестових завдань і їх варіантів відповіді

```
public timeout(ref: any, time: number): void {
  const tm: number = setTimeout( callback: () => {
    ref()
    clearTimeout(tm)
  }, time)
}

public shuffle(arr: any[]): any[] {
  return arr.sort( compareFn: () => Math.random() - 0.5)
}
```

Рисунок 4.26 – Тіло сервісу utils

ВИСНОВКИ

В дипломній роботі були розглянуті загальні відомості з теми, відбулося ознайомлення з алгоритмами циклічної структури, правилами побудови блок-схем і програмними засобами, які були використані для реалізації поставленої задачі.

За результатами виконання бакалаврської роботи було:

1. Обрано інтегроване середовище розробки – WebStorm і фреймворк Angular.

2. Складено алгоритм роботи програмного забезпечення.

3. Складено блок-схеми цього алгоритму.

4. Створено тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу while» дистанційного навчального курсу «Програмування ІІ».

5. Протестована робота всіх розділів тренажера.

Перевагами розробленого програмного забезпечення є:

1. Україномовний та англомовний інтерфейси.

2. Можливість прямо в тренажері писати код на мові С++.

3. Наявність розділу з теоретичним матеріалом.

4. Наявність завдань з блок-схемами.

5. Після виконання завдань, заголовки питань, на які користувач відповів правильно, виділяються зеленим кольором, всі інші – червоним.

6. У підказках можна побачити правильний варіант розв’язку завдань з поясненнями.

7. Інтуїтивний інтерфейс.

8. Сучасні анімації.

9. Вікно помилки, яке повідомляє, що користувач не виконав всі завдання.

10. Під час кожного нового тестування, всі завдання і варіанти відповіді рандомно перемішуються.

11. Швидкість роботи програми, через те, що програми розроблені на фреймворку Angular є SPA додатками.

Мета та завдання бакалаврської роботи виконано, створено тренажер, який навчає студентів будувати блок-схеми алгоритмів циклічної структури на прикладі циклу while.

Пояснювальну записку оформлено відповідно до рекомендацій виконання бакалаврської роботи [12].

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Концепція розвитку дистанційної освіти в Україні (затверджено Постановою МОН України В. Г. Кременем 20 грудня 2000 р.).

2. Биков В. Ю. Дистанційне навчання в країнах Європи та США і перспективи для України / В. Ю. Биков // Інформаційне забезпечення навчально-виховного процесу: інноваційні засоби і технології: кол. Монографія / В.Ю. Биков, О.О. Гриценчук, Ю.О. Жук та ін. / Академія педагогічних наук України, Інститут засобів навчання. – К.: Атіка, 2015. – С. 77–140.

3. 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 міжн. 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.

4. Ємець О. О. Дистанційний курс Полтавського університету економіки та торгівлі «Програмування ІІ» для студентів спеціальностей «Комп'ютерні науки та інформаційні технології», «Комп'ютерні науки» / О. О. Ємець. – [Електронний ресурс].

5. Офіційний сайт фреймворку Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/.

6. Офіційний сайт фреймворку Electron [Електронний ресурс] – Режим доступу до ресурсу: https://www.electronjs.org/.

7. Гмиза Б. Ю. Тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу for» дистанційного навчального курсу «Інформатика» та розробка його програмного забезпечення / Б. Ю. Гмиза, О. О. Ємець // Комп'ютерні науки і прикладна математика (КніПМ-2019): матеріали наук.-практ. Семінару. Випуск 3. / За ред. Ємця О. О. – Полтава: Кафедра ММСІ ПУЕТ, 2019. – С. 38-39. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/7036.

8. Недбайло Я. І. Тренажер з теми «Побудова блок-схем алгоритмів циклічної структури на прикладі циклу «do…while» дистанційного навчального курсу «Інформатика» та розробка його програмного забезпечення / Я. І. Недбайло, Є. М. Ємець. – Полтава: Кафедра ММСІ ПУЕТ, 2020. – С. 35-37. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/9018.

9. Сузанська А. О. Тренажер з теми «Побудова блок-схем алгоритмів розгалуженої структури» для дистанційного навчального курсу Полтавського університету економіки і торгівлі «Програмування ІІ» / А. О. Сузанська, Є. М. Ємець. – Полтава: Кафедра ММСІ ПУЕТ, 2021. – С. 36-39. – Режим доступу: http://dspace.puet.edu.ua/handle/123456789/10347.

10. Оператори циклу в С++ [Електронний ресурс] – Режим доступу: http://cpp.dp.ua/operatory-tsyklu/.

11. Цикли у програмуванні [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Цикл_(програмування).

12. 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.

ДОДАТОК А. МЕТОД КОМПІЛЯЦІЇ

```
  public async submit(): Promise<void> {
    this.submitted = true
    const outputText: HTMLElement = this.output.nativeElement

    try {
      outputText.innerHTML = ''
      outputText.innerHTML += 'Creating Submission ...\n'

      const body: Submission = {
        source_code: this.codeInputValue,
        stdin: this.dataInputValue,
        language_id: this.compilerService.languageId
      }

      const jsonResponse: ResponseSubmission = await
this.compilerService.createSubmission(body)
      outputText.innerHTML += 'Submission Created ...\n'

      let jsonGetSolution: Solution = {
        status: {description: 'Queue'},
```

```
      stderr: null,
      compile_output: null
    }

    while (
      jsonGetSolution.status.description !== 'Accepted' &&
      jsonGetSolution.stderr == null &&
      jsonGetSolution.compile_output == null
      ) {
      outputText.innerHTML = `Creating Submission ... \nSubmission Created
...\nChecking Submission Status\nstatus : ${jsonGetSolution.status.description}`

      if (jsonResponse.token) {
        jsonGetSolution = await this.compilerService.getSolution(jsonResponse.token)
      }
    }

    if (jsonGetSolution['stdout']) {
      const output: string = atob(jsonGetSolution['stdout'])
      outputText.innerHTML = ''
      outputText.innerHTML += output
    } else if (jsonGetSolution.stderr) {
      const error: string = atob(jsonGetSolution.stderr)
      outputText.innerHTML = ''
      outputText.innerHTML += `\n Error: ${error}`
    } else {
      const compilation_error: string = atob(jsonGetSolution.compile_output)
      outputText.innerHTML = ''
      outputText.innerHTML += `\n Error: ${compilation_error}`
    }
```

```
    this.submitted = false
  } catch (e: Exception) {
    console.error('Submit error: ', e)
    outputText.innerHTML = ''
    outputText.innerHTML = `Error: ${JSON.stringify(e)}`
    this.submitted = false
  }
}
```