Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «**ПРОГРАМНА МОДЕЛЬ КАЛЬКУЛЯТОРА ІЗ СИМЕТРИЧНОЮ ТРІЙКОВОЮ ЛОГІКОЮ**»

Спеціальність 151 –Автоматизація та комп'ютерно-інтегровані технології
Галузь знань 15 – Автоматизація та приладобудування
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

Захищено на засіданні
Екзаменаційної комісії № 46
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**

Виконав:
Студент групи КУ– 41
БУЯЛЬСЬКИЙ Дмитро
Володимирович _____

Керівник: доцент кафедри КСтаР,
к. ф.-м. н., доцент
КОТВИЦЬКИЙ Альберт Тадеушевич
_____

Рецензент: доцент кафедри загальної фізики фізичного факультету ХНУ ім. Каразіна, к.т.н., доцент
ГРЕСЬ Валерія Юріївна _____

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та п’ятьох додатків. Загальний обсяг роботи становить 80 сторінок, з яких 49 сторінок є основною частиною, яка містить 44 рисунки, 2 таблиці, 17 джерел у списку літератури та п’ять додатків.

Проблема, яка вирішується в кваліфікаційній роботі полягає у відсутності готових програмних рішень для роботи з симетричною трійковою системою числення, що обмежує дослідження та практичне застосування цієї системи.

Область застосування — комп’ютерні науки, альтернативні системи числення, освітні програми, дослідження в галузі обчислювальної техніки.

Метою кваліфікаційної роботи є розробка та програмна реалізація методів роботи з числами в симетричній трійковій системі числення.

Об'єкт дослідження – процес проектування та розробки алгоритмічних рішень для систем небінарної логіки.

Предмет дослідження – методи роботи з довільними дійсними числами у трійковій симетричній системі числення.

Ключові слова: трійкова логіка, трійкова симетрична система числення, калькулятор, C, арифметичні операції, логічні операції, програмна модель.

ABSTRACT

The explanatory note to the bachelor's qualification work consists of an introduction, three sections, conclusions, a list of references, and five appendices. The total volume of the work is 80 pages, of which 49 pages are the main part, which contains 44 figures, 2 tables, 17 sources in the list of references, and five appendices.

The problem that is being solved in the qualification work is the lack of ready-made software solutions for working with the symmetric ternary number system, which limits the research and practical application of this system.

Scope: computer science, alternative number systems, educational programs, and research in the field of computer technology.

The purpose of the qualification work is to develop and programmatically implement methods for working with numbers in the symmetric ternary number system.

The object of research is the process of designing and developing algorithmic solutions for non-binary logic systems.

The subject of research is methods of working with arbitrary real numbers in the ternary symmetric number system.

Keywords: ternary logic, ternary symmetric number system, calculator, C, arithmetic operations, logical operations, program model.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ДСЧ	–	Десяткова система числення
ТССЧ	–	Трійкова симетрична система числення
GMP	–	GNU Multiple Precision Arithmetic Library
TCA2	–	Ternary Computing Architecture 2
TNN	–	Ternary Neural Networks
TOC	–	Ternary Optical Computer
3L LCT	–	3-Level Level-Coded Ternary

ВСТУП

Сучасні цифрові технології ґрунтуються на двійковій системі числення, яка протягом десятиліть залишалась стандартом для побудови обчислювальних пристроїв. Однак із наближенням до фізичних меж масштабування транзисторів виникають серйозні обмеження, пов’язані з енергоспоживанням, тепловиділенням та ускладненням подальшого технічного прогресу. Закон Мура, який тривалий час передбачав стабільне зростання потужності комп’ютерних систем шляхом подвоєння кількості транзисторів на мікросхемі, поступово втрачає свою актуальність через технологічні та економічні виклики [1]. Це змушує дослідників звертатися до альтернативних підходів до обчислень, одним із яких є використання багатозначної логіки.

Однією з перспективних альтернатив є симетрична трійкова система числення (ТССЧ), яка забезпечує низку важливих переваг в порівнянні з двійковою: природне представлення від’ємних чисел, підвищену інформаційну щільність, ефективніше виконання арифметичних і логічних операцій, а також зменшення похибок округлення. Науковий інтерес до трійкової логіки зростає у зв’язку з розвитком таких інноваційних напрямів, як квантові обчислення, оптичні процесори, трійкові нейронні мережі та реверсивна логіка, де використання трьох логічних станів дозволяє зменшити апаратну складність, підвищити швидкодію та знизити енергоспоживання. Водночас на практиці поширенню ТССЧ перешкоджає нестача прикладних програмних рішень, які дозволяли б працювати з цією системою числення в інженерних, дослідницьких та освітніх задачах.

Новизна роботи полягає в тому, що наразі у вільному доступі відсутні готові програмні рішення для роботи з симетричною трійковою системою числення. Розробка програмної моделі калькулятора дозволить не лише досліджувати теоретичні аспекти, але й створити практичний інструмент для подальших досліджень, освітніх програм та експериментів, спрямованих на пошук нових сфер застосування трійкової логіки.

Таким чином, дослідження трійкової логіки є актуальним і необхідним для подальшого розвитку обчислювальної техніки. Впровадження симетричної трійкової системи числення може стати важливим кроком у створенні більш продуктивних, енергоефективних та економічно вигідних обчислювальних пристроїв.

Мета роботи – розробка та програмна реалізація методів роботи з числами в симетричній трійковій системі числення.

Об'єкт дослідження – процес проектування та розробки алгоритмічних рішень для систем небінарної логіки.

Предмет дослідження – методи роботи з довільними дійсними числами у трійковій симетричній системі числення.

РОЗДІЛ 1
ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО ТРІЙКОВОЇ СИМЕТРИЧНОЇ СИСТЕМИ ЧИСЛЕННЯ

1.1. Основи трійкової логіки та її переваги

Трійкова логіка є багатозначною логікою, яка є найпростішим розширенням звичайної двійкової логіки. На відміну від двійкової системи, де використовуються лише два значення (TRUE і FALSE), у трійковій логіці додається третє значення, яке найчастіше інтерпретується як UNKNOWN (невідомо). Однак у різних застосуваннях це значення може мати інші інтерпретації, такі як «невизначеність», «можливо» або «нейтральність». Це робить трійкову логіку більш гнучкою та придатною для моделювання складних реальних ситуацій, де важливо враховувати проміжні стани.

1.1.1. Об’єм даних: тріт та трайт

Однією з головних переваг трійкової логіки є здатність зберігати більше інформації в одному розряді порівняно з двійковою системою. Її базова одиниця — тріт (trit), який має три значення: -1, 0 або 1. Для порівняння, біт у двійковій системі має лише два значення. Один тріт несе приблизно 1.585 біт інформації, що забезпечує вищу щільність.

Група трітів утворює трайт (tryte). Наприклад, 6 трітів дають 729 можливих комбінацій (3^6), тоді як 8 бітів — лише 256 (2^8). Тож трійкова система дозволяє представити те саме число з меншою кількістю розрядів [2]. Наприклад, число 100 у двійковій системі займає 7 бітів (1100100), тоді як у трійковій — лише 5 трітів (10201).

1.1.2. Природність для людини

Трійкова логіка, зокрема симетрична, у багатьох випадках краще відповідає особливостям людського мислення. У повсякденному житті рішення часто приймаються не лише в категоріях «так» або «ні», а мають проміжні значення — «можливо», «не впевнений». На відміну від двійкової логіки, що змушує обирати між крайнощами, трійкова дозволяє враховувати невизначені

стани, що особливо корисно у системах прийняття рішень та штучному інтелекті, де дані можуть бути неповними чи нечіткими [3].

Обмеження двійкової логіки проявляються в її нездатності ефективно працювати з невизначеністю, що призводить до спрощених припущень і потенційних помилок. Трійкова логіка, завдяки третьому стану — «можливо», забезпечує більшу гнучкість і точність у моделюванні реальних ситуацій, особливо при обробці нечіткої або неповної інформації.

1.1.3. Менша кількість операцій

Однією з ключових переваг трійкової логіки є зменшення кількості операцій, необхідних для виконання певних задач. Замість розбиття результатів на два етапи, як у двійкових системах, трійкова система дозволяє безпосередньо закодувати три можливі результати порівняння («менше», «дорівнює» і «більше») одним трітом [4]. Це значно спрощує операцію порівняння, оскільки вся логіка виконується за один крок.

Оскільки в двійковій логіці результат порівняння не може одразу приймати три значення, потрібно реалізувати додаткову перевірку або обхідну конструкцію, що уповільнює процес (див. рис. 1.1).

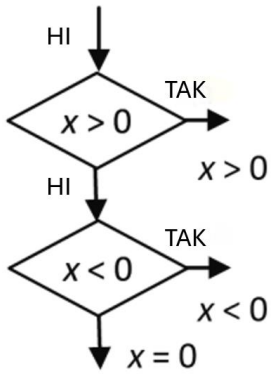

Рисунок 1.1 – Порівняння чисел у двійковій логіці.

Перевірки виконуються послідовно через дві окремі умови:

- Перша перевіряє, чи більше число x>0.
- Друга перевіряє, чи менше число x<0.

У той час як в трійковій логіці виконується одна перевірка, яка одразу визначає знак числа (негативне, позитивне чи нульове) (див. рис. 1.2).

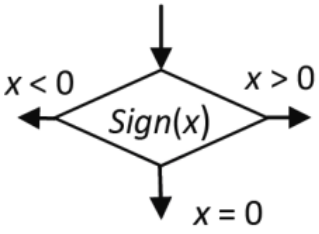

Рисунок 1.2 – Порівняння чисел у трійковій логіці.

Результат перевірки кодується трьома можливими значеннями:

- x < 0: виходить одне гілкування.
- x = 0: переходить на іншу гілку.
- x > 0: завершується третьою гілкою.

В результаті трійкова логіка дозволяє виконати операцію швидше, а результат порівняння кодується лише одним трітом, в той час як в двійковій операція кодується двома бітами.

1.1.4. Відмінності в кількості операцій

Трійкова логіка значно розширює кількість можливих операцій порівняно з двійковою логікою через наявність третього стану [5].

1. Нульарні операції — це операції, які не мають вхідних змінних і просто повертають фіксоване значення.

У двійковій логіці: $2^{2^0} = 2$; у трійковій логіці: $3^{3^0} = 3$

2. Унарні операції — це операції, які мають одну вхідну змінну.

У двійковій логіці: $2^{2^1} = 4$; у трійковій логіці: $3^{3^1} = 27$

3. Бінарні операції — це операції, які мають дві вхідні змінні.

У двійковій логіці: $2^{2^2} = 16$; у трійковій логіці: $3^{3^2} = 19\,683$

Більша кількість операцій у трійковій логіці є перевагою в тих випадках, де важлива гнучкість та ефективність представлення інформації. Однак це також може бути недоліком через збільшення складності реалізації та обмежену підтримку сучасних систем. Це робить використання трійкової логіки більш доцільним в спеціалізованих застосуваннях, таких як штучний інтелект, квантові обчислення та системи прийняття рішень.

1.2. Трійкові системи числення: особливості та переваги

Трійкова система числення є позиційною системою з основою 3, у якій кожна цифра може приймати одне з трьох значень. Вона є альтернативою двійковій системі та має ряд математичних і технічних переваг, які роблять її перспективною для використання в обчислювальній техніці. У трійковій системі числа можуть бути представлені у двох основних форматах [6]:

Несиметрична трійкова система — це стандартна позиційна система, у якій числа записуються за допомогою трьох значень (0, 1, 2). Вона працює за тими ж принципами, що й ДСЧ або двійкова система, але замість десяти або двох цифр використовує три. Значення кожної цифри залежить від її позиції в числі та визначається степенями основи, яка у цьому випадку дорівнює 3.

Симетрична трійкова система числення (ТССЧ) — це система, у якій кожне число представлено рівномірно розподіленими значеннями навколо нуля: (-1, 0, 1), (−, 0, +) або (i, 0, 1).

1.2.1. Економність коду: близькість до числа e

Трійкова система числення вважається найбільш економною з погляду кількості розрядів, необхідних для представлення чисел. Це пояснюється тим, що її основа (число 3) є найближчим цілим до математичної константи e (≈2.718), яка визначає оптимальну основу позиційної системи числення.

Оптимальний баланс досягається шляхом мінімізації функції $E(b) = b \cdot \log_b N$, мінімум якої припадає на $b = e$. Оскільки ця величина не є цілим числом, на практиці використовують найближчі цілі значення це 2 і 3. Серед них трійкова система виявляється ефективнішою, оскільки забезпечує коротший запис при меншій кількості розрядів.

Таким чином, трійкова система забезпечує вигідне поєднання мінімальної кількості розрядів і обмеженого набору символів, що робить її перспективною для спеціалізованих обчислювальних систем і подальших досліджень у сфері альтернативної логіки [7].

1.2.2. Представлення від’ємних чисел

Несиметрична система має обмеження: вона не підтримує від'ємні числа без використання додаткових символів, таких як знак мінус. Це ускладнює виконання арифметичних операцій, оскільки необхідно враховувати спеціальні правила для обробки знаку. Один із варіантів – введення додаткового знакового розряду, де «-» кодується як «1», а «+» – як «0».

На відміну від стандартної трійкової системи числення, симетрична система є врівноваженою відносно нуля, оскільки кількість додатних і від'ємних значень однакова. Це робить її більш зручною для представлення від'ємних чисел без необхідності використання додаткових символів. Для зміни знаку числа достатньо змінити ненульові цифри на симетричні, тобто поміняти місцями 1 та I [8]. Наприклад, число -4 в трійковому вигляді розраховується таким чином:

$i \cdot 3^1 + i \cdot 3^0 = -1 \cdot 3 + -1 \cdot 1 = -3 + (-1) = -4_{10}$

Така властивість робить її корисною для арифметичних операцій, оскільки вона дозволяє виконувати обчислення більш симетрично та ефективно, зменшуючи складність операцій над числами.

1.2.3. Відсутність проблем округлення

Іншою корисною властивістю ТССЧ є відсутність проблем округлення. Завдяки симетричному розташуванню цифр, коли ми відкидаємо молодші розряди числа, абсолютна величина відкинутої частини ніколи не перевищує половини значення молодшого збереженого розряду. Це означає, що відкидання молодших розрядів автоматично дає найкраще наближення для даної кількості збережених розрядів, і додаткове округлення не потрібне [8].

Приклад:

1. Двійкова система

Розглянемо число $x = 0.1011_2$ у двійковій системі. Його десяткове значення:

$$0{,}1011_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0{,}6875_{10}$$

Якщо ми відкинемо два молодші розряди, то отримаємо:

$$0{,}10_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} = 0{,}5_{10}$$

Похибка округлення:

$$|0{,}6875 - 0{,}5| = 0{,}1875$$

Ця похибка становить більше половини значення молодшого збереженого розряду: $\frac{2^{-2}}{2} = \frac{0{,}25}{2} = 0{,}125_{10}$

2. Трійкова несиметрична система

Розглянемо число x=0.1022_3 у звичайній трійковій системі. Його десяткове значення:

$$0{,}1022_3 = 1 \cdot 3^{-1} + 0 \cdot 3^{-2} + 2 \cdot 3^{-3} + 2 \cdot 3^{-4} \approx 0{,}4321_{10}$$

Якщо ми відкинемо два молодші розряди, то отримаємо:

$$0{,}10_3 = 1 \cdot 3^{-1} + 0 \cdot 3^{-2} \approx \overline{0{,}3333}_{10}$$

Похибка округлення:

$$|0{,}4321 - 0{,}3333| = 0{,}0988$$

Ця похибка становить більше половини значення молодшого збереженого розряду: $\frac{3^{-2}}{2} = \frac{0{,}1111}{2} = 0{,}0555_{10}$

3. Трійкова симетрична система

Розглянемо число x=$0.110i_3$ у трійковій симетричній системі. Його десяткове значення:

$$0{,}110i_3 = 1 \cdot 3^{-1} + 1 \cdot 3^{-2} + 0 \cdot 3^{-3} + (-1) \cdot 3^{-4} \approx 0{,}4321_{10}$$

Якщо ми відкинемо два молодші розряди (залишимо два розряди після коми), то отримаємо:

$$0{,}110i_3 = 1 \cdot 3^{-1} + 1 \cdot 3^{-2} \approx 0{,}4444_{10}$$

Похибка округлення:

$$|0{,}4321 - 0{,}4444| = 0{,}0123$$

Це не перевищує половини значення молодшого збереженого розряду: $\frac{3^{-2}}{2} = \frac{0{,}1111}{2} = 0{,}0555_{10}$

Таким чином, ТССЧ дозволяє уникнути накопичення похибок при зменшенні кількості молодших розрядів завдяки тому, що вона компенсує похибки в обидві сторони (додатну та від'ємну). Це означає, що якщо відкинуті молодші розряди мають додатне значення, воно компенсується можливістю

від'ємних значень у наступних розрядах, і навпаки. Така компенсація забезпечує більш ефективне управління похибками округлення та робить наближення більш точним у порівнянні з двійковою та трійковою несиметричною системами числення.

1.3. Реалізації трійкової симетричної системи числення

1.3.1. Практичні реалізації трійкової симетричної системи числення

ТССЧ, завдяки своїм унікальним властивостям, зуміла привернути увагу дослідників і стала основою для створення кількох практичних реалізацій комп'ютерів. Серед них найвідомішими є Сетунь, Ternac та TCA2.

Сетунь: перший трійковий комп'ютер [9]

У 1958 році створили перший серійний комп'ютер із трійковою системою числення — Сетунь. Його процесор працював на частоті 200 кГц і виконував 4500 операцій за секунду. Оперативна пам'ять складалася з 162 дев'ятирозрядних осередків, зовнішня — магнітний барабан місткістю 3888 осередків. Використовували феррит-діодні елементи для трьох станів. Пізніше вийшла покращена версія Сетунь-70, яка додатково підтримувала двійкову систему для обміну даними з іншими машинами.

Ternac: експериментальний трійковий комп'ютер [10]

У 1973 році в Університеті штату Нью-Йорк розробили Ternac — емулятор трійкової арифметики, який працював на двійковому комп’ютері Burroughs B1700. Він підтримував числа з фіксованою (24 тріти) та плаваючою точкою (48 трітів). Цей проект показав, що трійкова система ефективна навіть на двійковому обладнанні за швидкістю і використанням пам'яті.

TCA2: сучасний проект трійкового комп'ютера [2]

Останньою відомою реалізацією трійкової симетричної системи став TCA2. Цей проект, розроблений у 2008 році в Каліфорнійському політехнічному університеті, є відносно сучасним підходом до використання трійкової логіки. Основною метою TCA2 було дослідження можливостей використання трійкової системи в енергоефективних обчисленнях. Версія TCA2 v2.0 використовує

трьохрівневу систему трійкових логічних елементів (3L LCT), яка реалізована на 1484 інтегральних транзисторах.

1.3.2. Припинення розробок трійкових систем

Незважаючи на теоретичні переваги ТССЧ, її практичне впровадження зупинилося через низку технічних, економічних та інфраструктурних обмежень. Основні причини пов'язані з домінуванням двійкової системи, відсутністю налагодженого виробництва трійкових елементів та складністю їх реалізації.

1. Домінування двійкової системи

На початку 1960-х років двійкова система числення стала стандартом у комп'ютерній індустрії. Масове виробництво двійкових компонентів із кремнію зробило їх значно дешевшими та доступнішими. У той же час трійкові комп'ютери потребували спеціалізованих елементів, які не мали широкого застосування. Це призвело до того, що трійкові системи залишилися без елементарної бази, тоді як двійкові комп'ютери отримали всебічну підтримку.

2. Відсутність налагодженого виробництва трійкових елементів

Виробництво трійкових елементів було значно складнішим і дорожчим порівняно з двійковими. Наприклад, трійкові транзистори, які могли б стабільно працювати з трьома станами, були технічно складними у виготовленні. На відміну від двійкових транзисторів, які мали лише два стани (**0** і **1**), трійкові вимагали додаткового рівня (**-1**), що ускладнювало їхню конструкцію та підвищувало вартість.

3. Економічна недоцільність

Повний перехід на трійкові системи був економічно невигідним через відсутність налагодженого виробництва трійкових елементів. Виробництво таких компонентів вимагало значних інвестицій у розробку нових технологій, тоді як двійкові системи вже були стандартом і мали велику підтримку з боку промисловості.

4. Інфраструктурні обмеження

Вся інфраструктура комп'ютерної індустрії була орієнтована на двійкові системи. Програмне забезпечення, периферійні пристрої та інші компоненти

були розроблені для двійкових комп’ютерів. Це ускладнювало впровадження трійкових систем, оскільки вони потребували сумісності з існуючими стандартами.

1.4. Потенційні області застосування трійкової симетричної логіки.

Однією з потенційних областей застосування трійкової симетричної логіки є оптичні обчислення, які демонструють значний потенціал для впровадження цієї логіки на основі трьох оптичних станів: двох взаємно перпендикулярних напрямків поляризації світла та стану його відсутності. У роботі [11] описано архітектуру трійкового оптичного комп’ютера (ТОС), де ТССЧ (-1, 0, +1) використовується як базова логічна модель. Такий підхід дозволяє реалізувати обчислювальні операції з високою швидкодією — зокрема, додавання довільної розрядності виконується за три машинні цикли — та знижує енергоспоживання. Крім того, апаратна структура процесора підтримує групову незалежну обробку бітів і функцію переконфігурування, що робить симетричну трійкову логіку привабливою для застосування в оптичних обчислювальних системах.

Квантові обчислення є ще однією перспективною сферою застосування трійкової симетричної логіки, особливо в контексті побудови зворотних (реверсивних) обчислювальних блоків. У дослідженні [12] представлено реалізацію квантово-збалансованих трійкових схем множення на основі симетричної системи числення (-1, 0, +1), придатної для зворотної логіки. Показано, що така система дозволяє ефективно зменшити структурну складність і втрати енергії. Збалансоване представлення чисел забезпечує однакову обробку додатних і від’ємних значень без необхідності додаткових перетворень, що сприяє спрощенню архітектури зворотних логічних елементів. Це створює підґрунтя для подальшого використання симетричної трійкової логіки в енергоефективних квантових обчислювальних структурах.

Нейронні мережі, зокрема трійкові нейронні мережі (TNN), також демонструють дієве застосування симетричної трійкової логіки. Як описано у праці [13], представлення ваг і активацій у форматі (-1, 0, +1) дозволяє значно скоротити споживання пам’яті й обчислювальних ресурсів. Запропонована схема

квантування поєднує асиметричні порогові значення з симетричними масштабними коефіцієнтами, що полегшує реалізацію моделі в апаратному забезпеченні. Окрім того, автори впроваджують двоетапну процедуру навчання з передачею знань, яка дає змогу підвищити точність моделі. Згідно з експериментами, запропонована модель перевершує точність існуючих методів 2-бітного квантування, що підтверджує ефективність симетричної трійкової логіки в завданнях глибокого навчання.

Висновок до розділу 1

У першому розділі було проведено аналіз сучасних підходів до ТССЧ. Розглянуто основи трійкової логіки, її переваги порівняно з двійковою, а також історичні та сучасні реалізації трійкових обчислювальних систем. Виявлено, що трійкова система числення має значні переваги над двійковою системою. Однак основним бар'єром для її широкого впровадження є відсутність готових програмних та апаратних рішень, що підтримують трійкову логіку.

Розробка програмної моделі калькулятора з використанням ТССЧ є важливим кроком у цьому напрямку. Такий калькулятор не лише дозволить досліджувати теоретичні аспекти трійкової логіки, але й стане практичним інструментом для виконання арифметичних операцій у трійковій системі.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ТРІЙКОВОГО СИМЕТРИЧНОГО КАЛЬКУЛЯТОРА

2.1. Архітектура програмної моделі

Програмну модель калькулятора із симетричною трійковою логікою було вирішено реалізувати мовою програмування C. Вибір цієї мови обумовлений високою ефективністю, низькорівневим доступом до пам’яті та широкою підтримкою арифметичних і логічних операцій, необхідних для коректної роботи з ТССЧ. Обрано стандарт C99 через його підтримку динамічних масивів, типів із фіксованою шириною, покращений синтаксис та інші сучасні можливості, що значно спрощують розробку складних алгоритмів і підвищують надійність коду [14]. Крім того, стандарт C99 є широко підтримуваним сучасними компіляторами, що забезпечує портативність розробленої моделі.

Модель складається з трьох основних модулів, кожен із яких реалізує окремий набір функціональності:

- Модуль перетворення між системами числення — перетворення чисел між десятковою та трійковою симетричною системами з обробкою цілих і дробових частин.
- Модуль арифметичних операцій — реалізація додавання, віднімання та множення з числами в ТССЧ.
- Модуль логічних операцій — виконує логічні операції AND та OR для трійкової логіки.

Користувацький інтерфейс реалізований у вигляді консольного меню, що дозволяє користувачу вводити цифру, яка відповідає потрібній операції. У функції main() відображається меню з вибором таких дій:

0. Завершення виконання програми.
1. Перетворення чисел з ДСЧ в ТССЧ.
2. Перетворення чисел з ТССЧ в ДСЧ.
3. Складання трійкових чисел.

4. Віднімання трійкових чисел.
5. Множення трійкових чисел.
6. Логічна операція AND для трійкової логіки.
7. Логічна операція OR для трійкової логіки.

Після вибору пункту меню програма виконує відповідну дію, запитуючи необхідні вхідні дані. Модульна архітектура моделі калькулятора представлена на рис. 2.1.

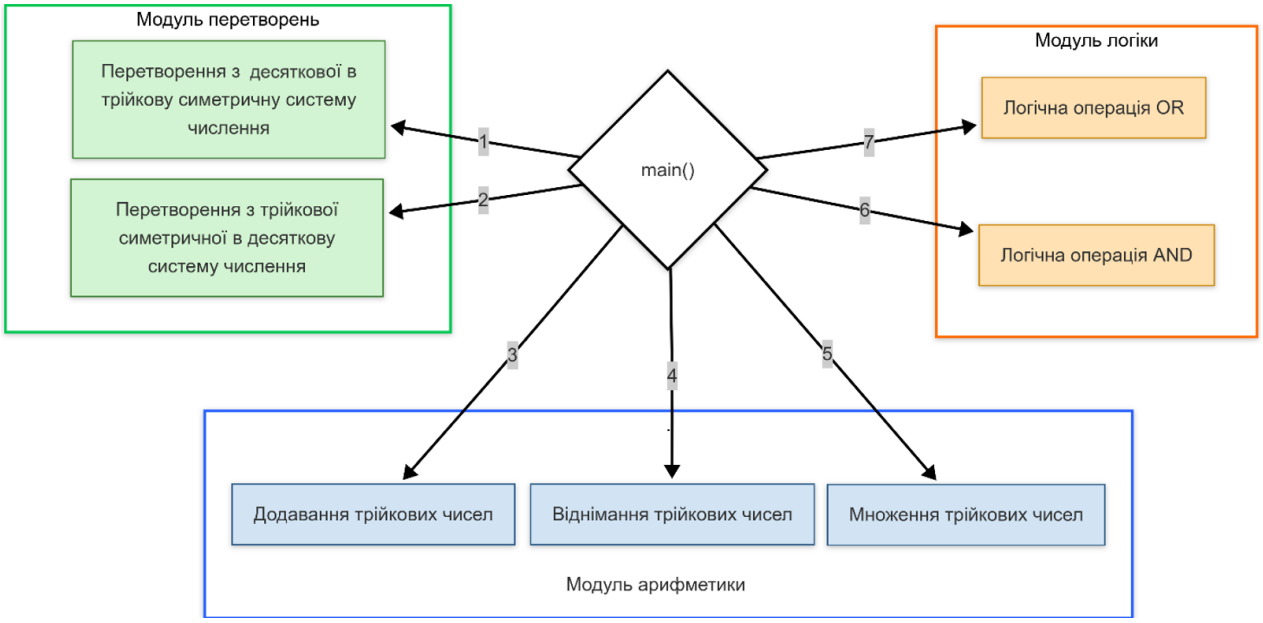

Рисунок 2.1 – Схема модульної архітектури моделі калькулятора.

Схема на рисунку 2.1 ілюструє організацію головної функції main() та її взаємодію з окремими функціональними модулями програмної моделі калькулятора: перетворення чисел, арифметичних операцій та логічних операцій.

Код програми наведено у Лістингу, що викладено у Додатку Г.

2.2. Модуль перетворення чисел між системами числення

Під час роботи з трійковими симетричними числами одним із перших і ключових завдань є перетворення цих чисел між різними системами числення. Ця задача стає основою для подальшого опрацювання таких чисел у більш складних операціях і алгоритмах, забезпечуючи надійність та точність їх виконання. Модуль перетворення чисел складається із семи функцій, наведених у табл. 2.1.

Таблиця 2.1

Функції блоку перетворення між системами числення

№	Назва	Призначення
1	read_input()	Функція зчитує вхідне число як рядок.
2	decimal_to_ternary()	Основна функція для перетворення десяткового числа (у форматі рядка) у симетричну трійкову систему.
3	whole_part()	Перетворює цілу частину десяткового числа у відповідне трійкове представлення.
4	fraction_part()	Перетворює дробову частину десяткового числа у трійковий вигляд з урахуванням точності та обмеження кількості розрядів.
5	symmetry()	Перетворює число зі звичайної трійкової системи у трійкову симетричну.
6	invert()	Інвертує знаки цифр у симетричному трійковому числі, якщо вхідне десяткове число було від'ємним.
7	ternary_to_decimal()	Основна функція для зворотного перетворення трійкового симетричного числа у десяткову форму.

Для візуального зображення логіки взаємодії між цими функціями використано блок-схему зображену на рис. 2.2.

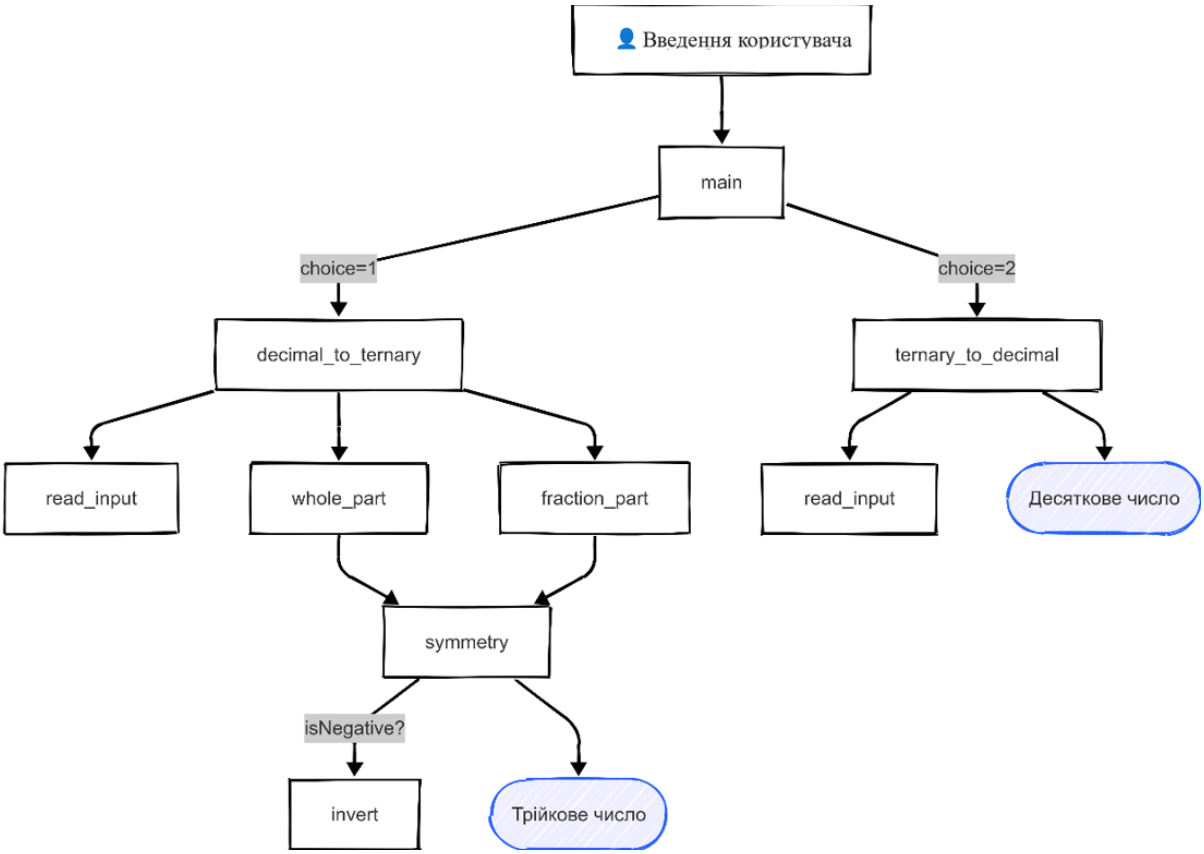

Рисунок 2.2 – Ієрархічна діаграма конвертації чисел.

Схема має два незалежні потоки обробки даних, які активізуються залежно від вибору користувача: choice=1 для перетворення з десяткової в трійкову систему та choice=2 для зворотного процесу. У першому потоці десяткове число спочатку перетворюється у звичайну трійкову систему числення, після чого спеціальна функція symmetry() балансує його, перетворюючи на симетричну. Другий потік передбачає зчитування симетричного трійкового числа та подальше відновлення десяткового значення. Ключовою особливістю є обов'язкове балансування трійкового числа через функцію symmetry(), яка забезпечує коректне представлення числа.

Розглянемо докладніше функції, що входять до складу цього блоку.

2.2.1. Функція read_input();

Першою функцією є read_input(), яка відповідає за зчитування вхідних даних, отриманих від користувача у вигляді довільного числа в ДСЧ або ТССЧ. Оскільки числа в ТССЧ будуються з використанням значень «-1», «0» та «1», для їх збереження не можна використовувати стандартні числові типи даних, адже вони не здатні коректно представити число у трійковому вигляді, наприклад, «1-10-1». Тому було прийнято рішення зберігати вхідні значення у вигляді рядка символів. Для зручності обробки введення, значення «-1» було вирішено позначати символом «i», щоб уникнути ускладнень з обробкою знаку мінус у рядку.

На вході вона отримує вказівник на змінну типу char*, через яку передається адреса для збереження зчитаного рядка. У результаті виклику read_input() у змінній *input зберігається динамічно виділений рядок із введеним користувачем значенням, готовий до подальшої обробки у програмі. Такий підхід забезпечує як безпечне зчитування, так і ефективне використання пам'яті.

2.2.2. Функція decimal_to_ternary();

Функція decimal_to_ternary() виконує перетворення десяткового числа у ТССЧ, враховуючи всі його складові: знак, цілу та дробову частини. Якщо отримане десяткове число від'ємне, то видаляється мінус, а змінна-прапорець встановлюється в значення 1. Оскільки алгоритми перетворення для цілих і

дробових чисел принципово відрізняються, функція спочатку розділяє вхідне число на дві частини за роздільником. Кожна частина обробляється окремо: ціла - функцією whole_part(), дробова - fraction_part(). Після цього отримані результати об'єднуються, і за допомогою функції symmetry() перетворюються у симетричний трійковий вигляд.

На рис. 2.3 зображена блок-схема функції.

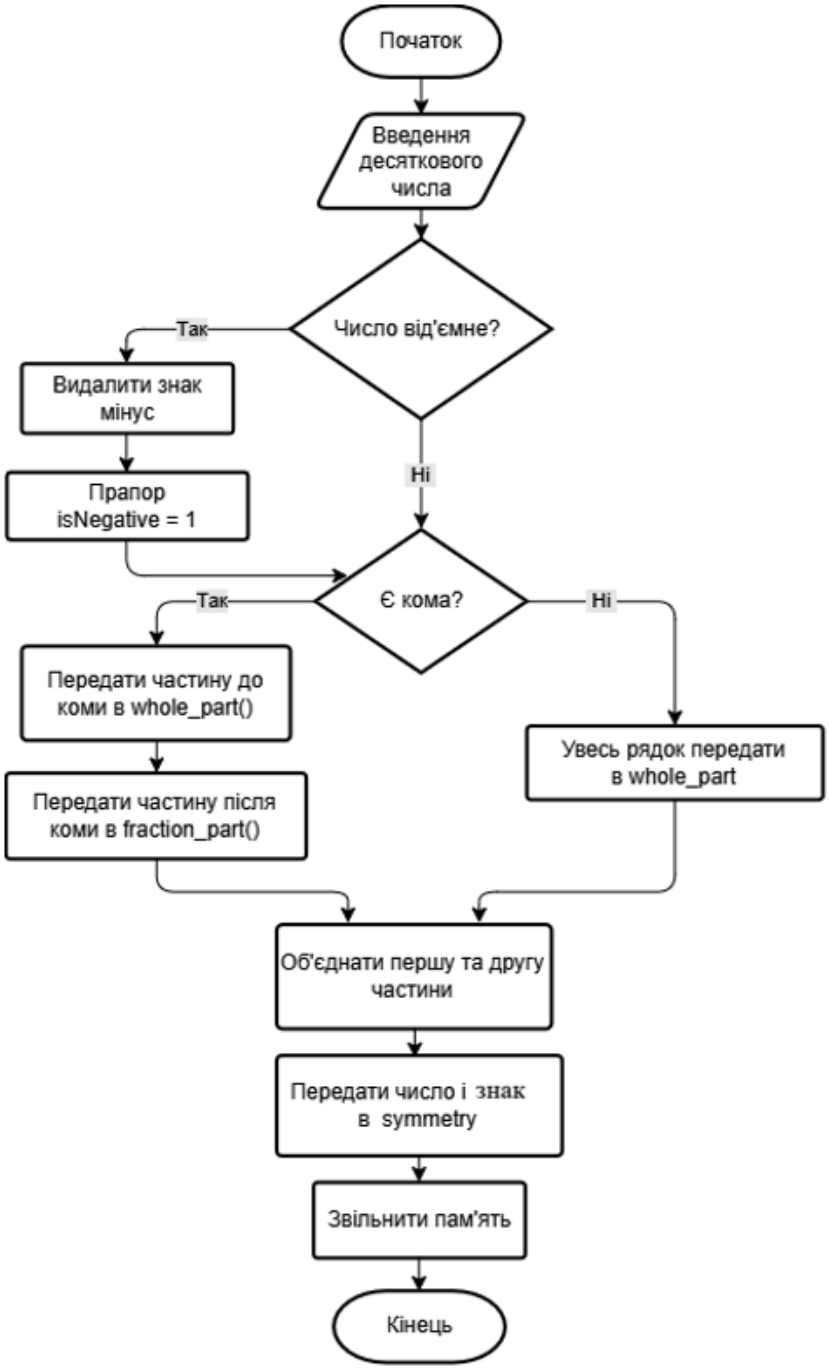

Рисунок 2.3 – Блок-схема функції decimal_to_ternary().

2.2.3. Функція whole_part();

Функція whole_part() призначена для перетворення цілої частини числа, поданої у вигляді рядка, з десяткової системи числення в несиметричну трійкову. Вона приймає як аргумент вказівник на символьний рядок char *part, що містить десяткове ціле число, та змінює його вміст, записуючи у ту саму змінну відповідне представлення в трійковій системі числення.

Перетворення здійснюється шляхом багаторазового ділення числа на основу системи числення (3) дотри, доки результат не стане нулем, при цьому на кожному кроці фіксується остача від ділення. Послідовність цих остач, розташована у зворотному порядку і становить трійкове представлення вхідного десяткового числа.

2.2.4. Фунція fraction_part();

Функція fraction_part() призначена для перетворення дробової частини десяткового числа, поданої у вигляді рядка, у трійкову систему числення. Вона приймає як аргумент вказівник на символьний рядок char *part, що містить дробову частину десяткового числа без крапки і змінює його вміст, записуючи відповідне представлення цієї дробової частини в трійковій несиметричній системі числення.

Перетворення ґрунтується на послідовному множенні дробового значення на основу системи числення (3) та відділенні цілої частини добутку — вона і є відповідною цифрою трійкового дробу. Кількість обчислених знаків після крапки обмежується константою MAX_DECIMAL_PLACES, що визначає точність обчислень і дорівнює 15.

2.2.5. Функція symmetry();

Функція symmetry() призначена для перетворення рядка, що містить число в несиметричній трійковій системі числення, у відповідне представлення в ТССЧ. Функція отримує вказівник на символьний рядок char *input, що містить число в трійковій несиметричній системі, та змінну-прапор, що вказує на знак початкового десяткового числа.

Ідея роботи полягає в тому, щоб обійти увесь рядок з кінця та обробити значення, що виходять за межі допустимого діапазону ТССЧ (від -1 до 1): цифру 2 як і (-1) із додаванням 1 до наступного старшого розряду. Можливий випадок, коли зустрічаються дві цифри 2 поспіль - тоді після обробки першої 2 на наступному розряді з'явиться 3, яка оброблятиметься як 0 із додаванням 1 до наступного старшого розряду. Механізм переносу розрядів ілюструється рисунком 2.4.

Рисунок 2.4 – Механізм переносу розрядів.

Слід зазначити, що якщо перенос відбувається під час обробки старшого розряду, то кінцеве число у ТССЧ буде довшим на 1 розряд ніж у трійковій несиметричній системі.

На рис. 2.5 наведена блок-схема функції.

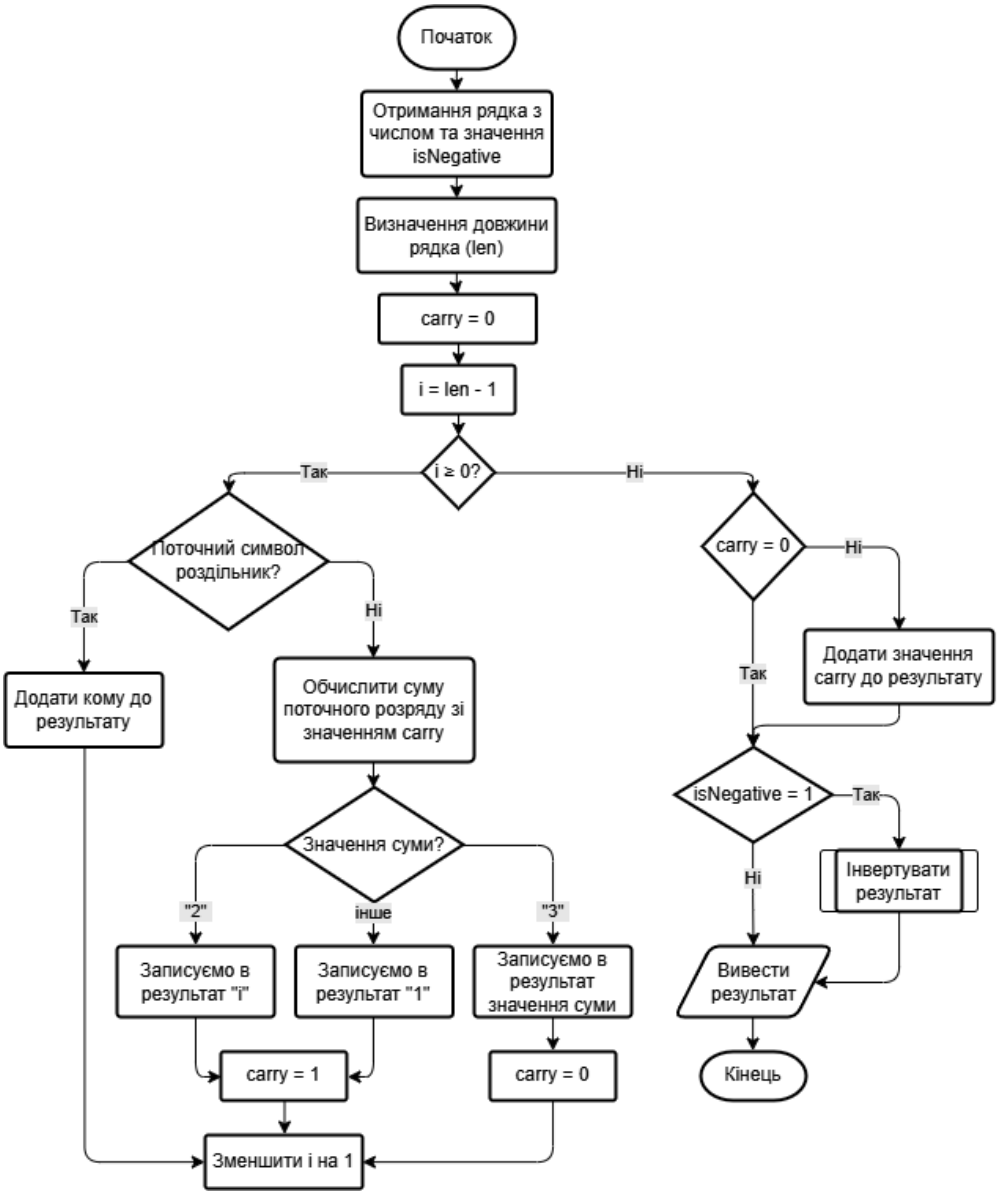

Рисунок 2.5 – Блок-схема функції symmetry().

2.2.6. Функція invert();

Функція invert() призначена для зміни знаку трійкового числа, представленого в симетричній системі числення, шляхом інверсії його розрядів. Операція інвертування в цій системі еквівалентна логічному запереченню (NOT), що, у свою чергу, відповідає арифметичній операції множення числа на -1. Таблиця істинності для операції логічного заперечення наведена на рис. 2.6 [15].

Вхід	Вихід
i	1
0	0
1	i

Рисунок 2.6 – Таблиця істинності логічного заперечення.

Дана функція є завершальним етапом перетворення числа з десяткової системи числення у трійкову симетричну, оскільки саме інверсія розрядів забезпечує коректну зміну знаку числа. Таким чином, після виконання функції вхідне число, яке було додатнім, стає від'ємним, і навпаки, що дозволяє коректно представляти числа з різними знаками в ТССЧ.

2.2.7. Функція ternary_to_decimal();

Функція ternary_to_decimal() призначена для перетворення чисел із ТССЧ у ДСЧ. Функція працює з трійковим симетричним числом представленим у вигляді рядка символів, де -1 позначається як і.

Ідея роботи функції ґрунтується на фундаментальному принципі позиційних систем числення, де кожен символ у рядку інтерпретується як цифра, яка множиться на основу системи (3) у ступені, що визначається позицією цього символу відносно роздільника. Для цілої частини, розташованої ліворуч від роздільника, ступінь основи зростає з віддаленням від нього - перший розряд має вагу 3^0, наступний 3^1 і так далі. Дробова частина, яка знаходиться праворуч від роздільника, обробляється за аналогічним принципом, але з від'ємними ступенями, де перший розряд має вагу 3^{-1}, другий - 3^{-2}, і так далі.

Максимальний ступінь визначається номером позиції роздільника мінус 1, що відповідає старшому розряду цілої частини. Потім вона послідовно обробляє кожен символ рядка зліва направо: для кожного валідного символу (крім самого роздільника) обчислюється його внесок у десяткове значення шляхом множення числового еквівалента символу на 3 у поточному ступені, після чого ступень зменшується на одиницю. Це продовжується до тих пір, поки не буде оброблено усі розряди трійкового числа.

2.3. Модуль арифметичних операцій

Модуль арифметичних операцій є основним обчислювальним ядром програми, яке відповідає за виконання базових математичних операцій у ТССЧ. Цей модуль реалізує математичні операції для симетричної трійкової системи числення, що є ключовим функціоналом програмної моделі калькулятора. Модуль складається з чотирьох функцій, наведених у табл. 2.2.

Таблиця 2.2

Функції блоку арифметичних операцій

№	Назва	Призначення
1	rationing()	Нормалізує два числа: вирівнює розряди цілої дробової частини, додає нулі за необхідності. Готує матрицю для операцій.
2	addition()	Виконує додавання/віднімання двох трійкових чисел з обробкою переносів та переповнень.
3	multiplication()	Реалізує множення трійкових чисел через послідовні зрушення та додавання.
4	remove_delimiter()	Видаляє роздільники з рядкового подання числа для спрощення обчислень.

Для візуального зображення логіки взаємодії між цими функціями використано блок-схему, наведену на рис. 2.7.

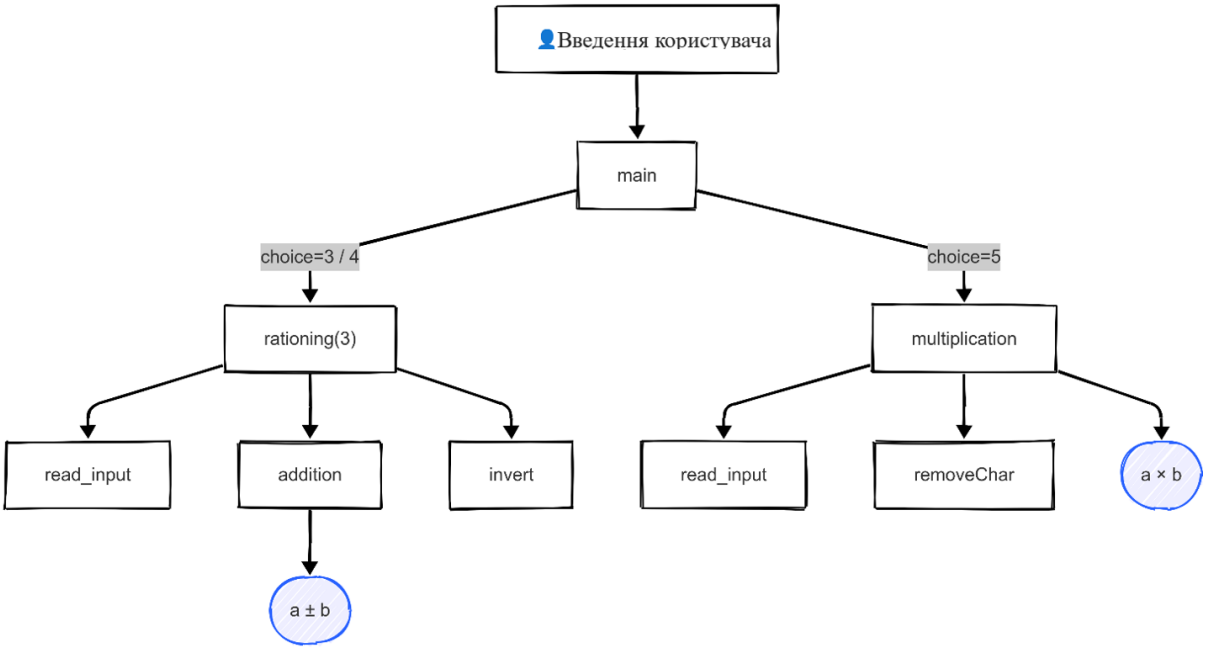

Рисунок 2.7 – Ієрархічна діаграма арифметичних операцій.

Схема має два незалежні потоки обробки, які активізуються залежно від вибору користувача: choice=3/4 для операцій додавання/віднімання та choice=5 для операції множення. У першому потоці виконуються операції додавання та віднімання, яке реалізовано як додавання інвертованого числа. Другий потік реалізує виконання операції множення. Архітектура використовує вже існуючі функцій read_input() та invert(), що були описані раніше в підрозділах 2.2.1 і 2.2.6

для мінімізації дублювання коду та забезпечення узгодженої роботи всієї системи.

Розглянемо докладніше функції, що входять до складу цього блоку.

2.3.1. Функція rationing();

Функція rationing() є найважливішою функцією для взаємодії двох трійкових чисел, вона відповідає за їх нормування. Оскільки трійкові числа записуються в рядках символів та їхня довжина може відрізнятися, то для виконання операцій між ними їх потрібно вирівняти по розрядах відносно один одного. На рис. 2.8 демонструється необхідність цього на прикладі операції додавання.

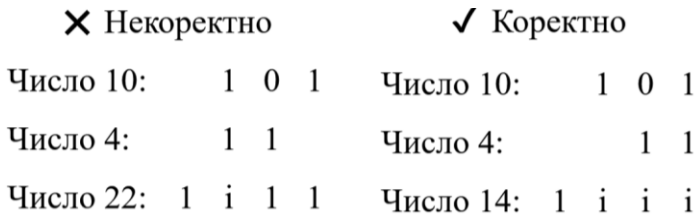

Рисунок 2.8 – Необхідність нормування.

Порозрядне виконання операцій є неможливим без попереднього вирівнювання, оскільки перші розряди чисел, насправді, мають різну вагу, що робить необхідним їх вирівнювання.

Ідея нормування полягає в тому, щоб записати числа у вигляді рядків однакової довжини. Для цього спочатку визначаються довжини цілих і дробових частин обох чисел, після чого обчислюються максимальні значення цих довжин. Довжина результуючих рядків визначається як сума найбільшої довжини цілої частини, найбільшої довжини дробової частини, плюс один символ для можливого роздільника. На основі цих даних створюється двомірний масив символів, у якому кожне число записується з відповідним зміщенням: ціла частина доповнюється нулями зліва, щоб останні цифри обох чисел опинилися на одній позиції, а якщо у чисел є дробові частини, вони також доповнюються нулями праворуч до однакової довжини. Роздільник між цілою та дробовою частинами додається для обох чисел, навіть якщо він був лише в одному з них,

що забезпечує однакову структуру обох рядків. Приклад нормування наведено на рис. 2.9.

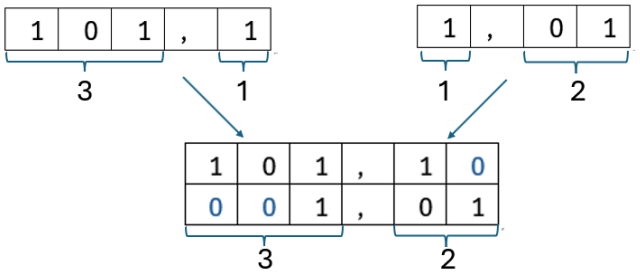

Рисунок 2.9 – Приклад нормування чисел.

Далі, на основі значення змінної choice, яке визначає вибір операції, нормовані числа у вигляді двомірного масиву передаються у відповідну функцію для виконання арифметичних або логічних дій.

На рис. 2.10 наведена блок-схема функції.

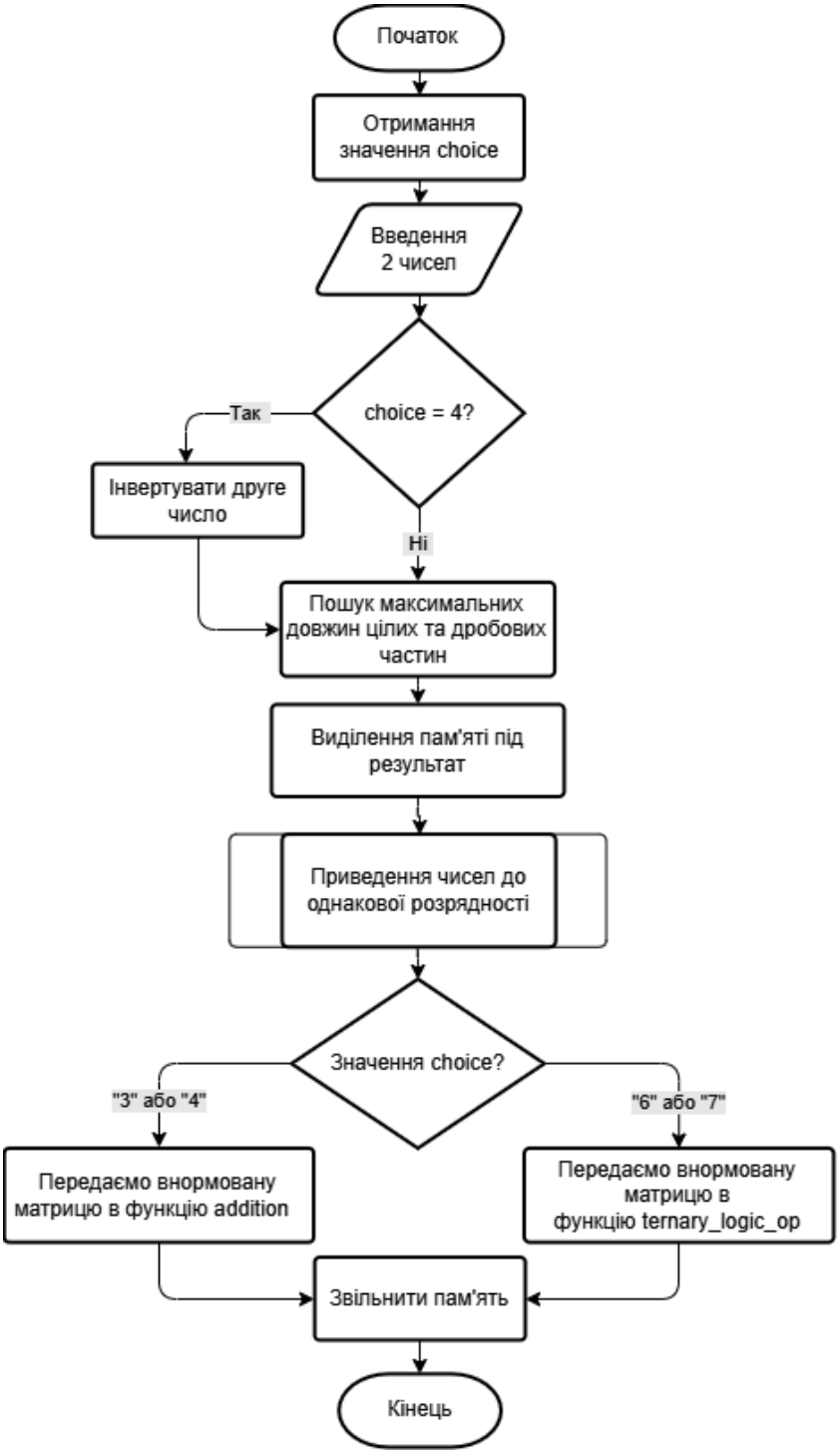

Рисунок 2.10 – Блок-схема функції rationing().

2.3.2. Функція addition();

Функція addition() реалізує операцію додавання двох чисел, представлених у ТССЧ. Вона приймає як параметр вказівник на масив рядків char **matrix, який містить два попередньо нормалізованих числа, вирівняних за допомогою функції rationing().

Для виконання операцій використовуються таблиці істинності, подані на рисунках 2.11 та 2.12, які враховують усі можливі комбінації символів і переносу.

+		A		
		i	0	1
B	i	i1	i	0
	0	i	0	1
	1	0	1	1i

Рисунок 2.11 – Таблиця істинності додавання.

-		A		
		i	0	1
B	i	0	i	1i
	0	1	0	i
	1	1i	1	0

Рисунок 2.12 – Таблиця істинності віднімання.

Процес додавання здійснюється шляхом послідовного обходу розрядів обох чисел від молодших до старших. Для кожної пари символів відповідного розряду виконується перетворення в числовий еквівалент: символ «1» відповідає значенню +1, «i» — значенню -1, а всі інші символи трактуються як 0. Сума цих значень доповнюється поточним значенням переносу, який ініціалізується нулем і змінюється залежно від результату: якщо сума перевищує +1, вона зменшується на 3, а значення переносу встановлюється рівним +1; якщо сума менша за -1, до неї додається 3, а перенесення набуває значення -1. Таким чином реалізується механізм переносу та запозичення у симетричній трійковій системі. Отримане значення після нормалізації перетворюється назад у символ і записується у відповідну позицію нового рядка результату. Якщо під час додавання зустрічається роздільник, він копіюється до результату без змін. Після завершення обробки всіх розрядів перевіряється, чи залишилося ненульове значення переносу — у такому випадку воно додається як новий старший розряд. Важливо зазначити, що у рамках операції додавання над двома числами в ТССЧ значення переносу в процесі обчислення ніколи не виходить за межі від -1 до +1, тому перевірка на інші значення не є необхідною. Наприкінці з результату видаляються зайві початкові нулі, щоб уникнути некоректного вигляду числа.

Ця функція також реалізує операцію віднімання. Для цього в функції rationing() при отриманні від користувача значення вибору, рівного 4, для другого числа виконується операція інвертування. Після цього створюється матриця з нормалізованими числами, яка передається в функцію addition(). Таким чином, замість окремої функції для віднімання, ця операція виконується як додавання від'ємного числа.

На рис. 2.13 наведена блок-схема функції.

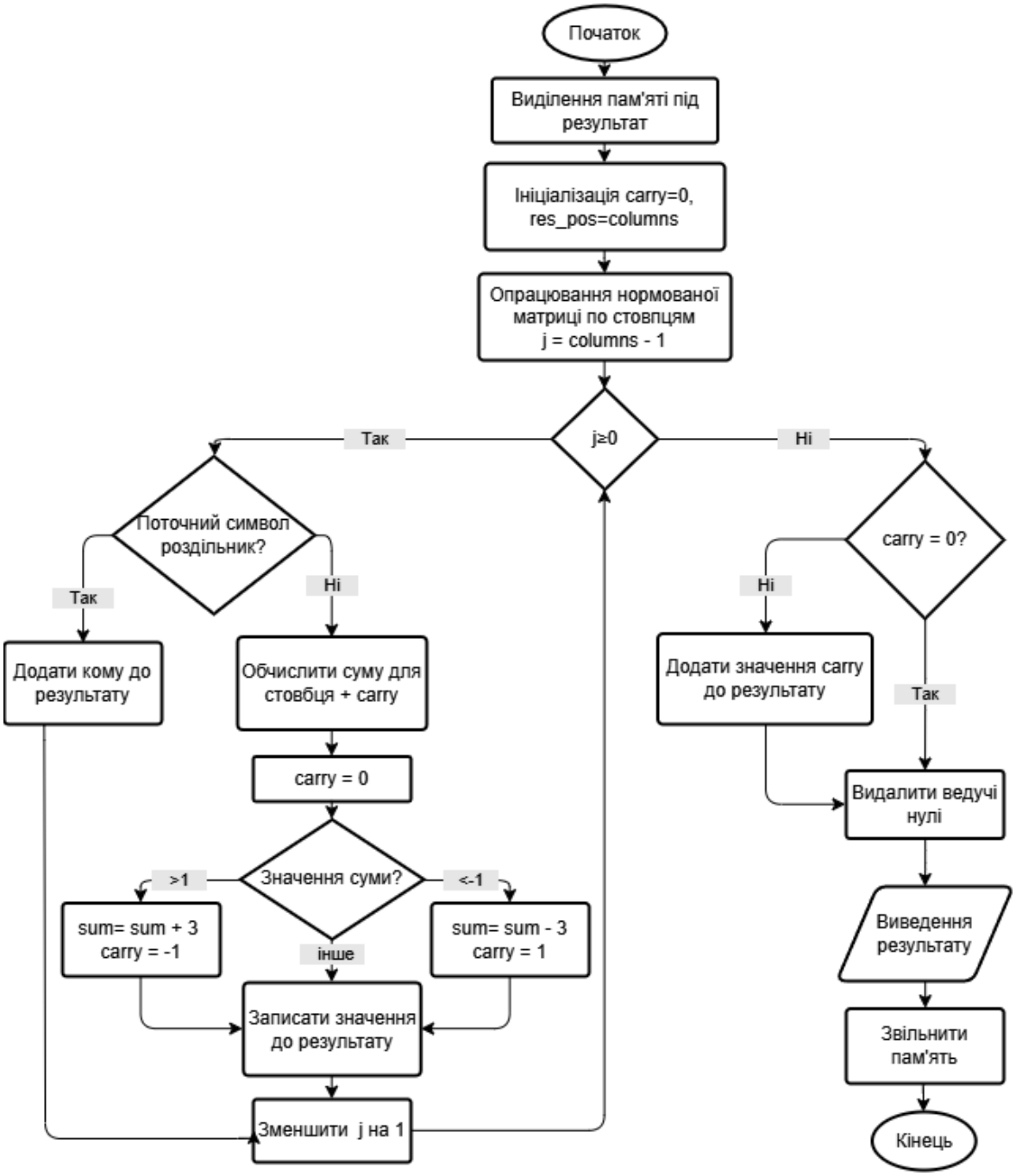

Рисунок 2.13 – Блок-схема функції addition().

2.3.3. Функція multiplication();

Функція multiplication() реалізує алгоритм множення двох чисел у ТССЧ, заснований на принципі послідовного накопичення часткових добутків.

Метод аналогічний традиційному множенню в стовпчик, однак відрізняється поетапним сумуванням: замість одночасного обчислення всіх часткових добутків з подальшим їх складанням, кожен новий частковий добуток одразу додається до накопиченого результату.

Спочатку створюється рядок-результат, заповнений нулями, довжина якого розраховується як сума довжин вхідних чисел (без урахування роздільників) плюс один додатковий розряд. Така довжина забезпечує достатньо місця для максимально можливого результату множення, включно з резервним розрядом для обробки можливих переносів та місцем для роздільника.

Алгоритм послідовно обробляє кожен розряд множника, починаючи з молодшого. Залежно від значення поточного розряду виконується одне з трьох дій:

1. Для розряду «1» — до результату додається перше число без змін.
2. Для розряду «i» — додається інвертована версія множного.
3. Для розряду «0» — операція пропускається, оскільки вважається, що ми додаємо 0.

Зсув часткових добутків реалізується через розрахунок початкової позиції складення в рядку-результаті та посимвольного запису числа справа наліво. Розрахунок початкової позиції відбувається по формулі (2.1).

$$\text{початкова позиція} = \text{довжина 1 числа} + \text{довжина 2 числа} - 1 - i \quad (2.1),$$

де і це номер розряду другого числа, на яке ми множимо перше, щоб отримати частковий добуток.

Такий підхід забезпечує правильний позиційний зсув, гарантуючи, що кожний наступний частковий додаток буде зсунутий ліворуч на один елемент.

Додавання поточного часткового додатку до результату відбувається за тим самим принципом, що й у функції addition().

Після того, як було завершено обхід розрядів другого числа для пошуку часткових добутків, виконується вставка переносу, якщо він виникнув під час складання часткових добутків. Також додається роздільник до результату так, щоб кількість розрядів дробової частини результату відповідала кількості дробових розрядів в початкових числах. Після завершення обчислень відбувається фінальна оптимізація формату числа, що передбачає видалення зайвих нулів для забезпечення коректного та стислого запису.

На рис. 2.14 наведена блок-схема функції.

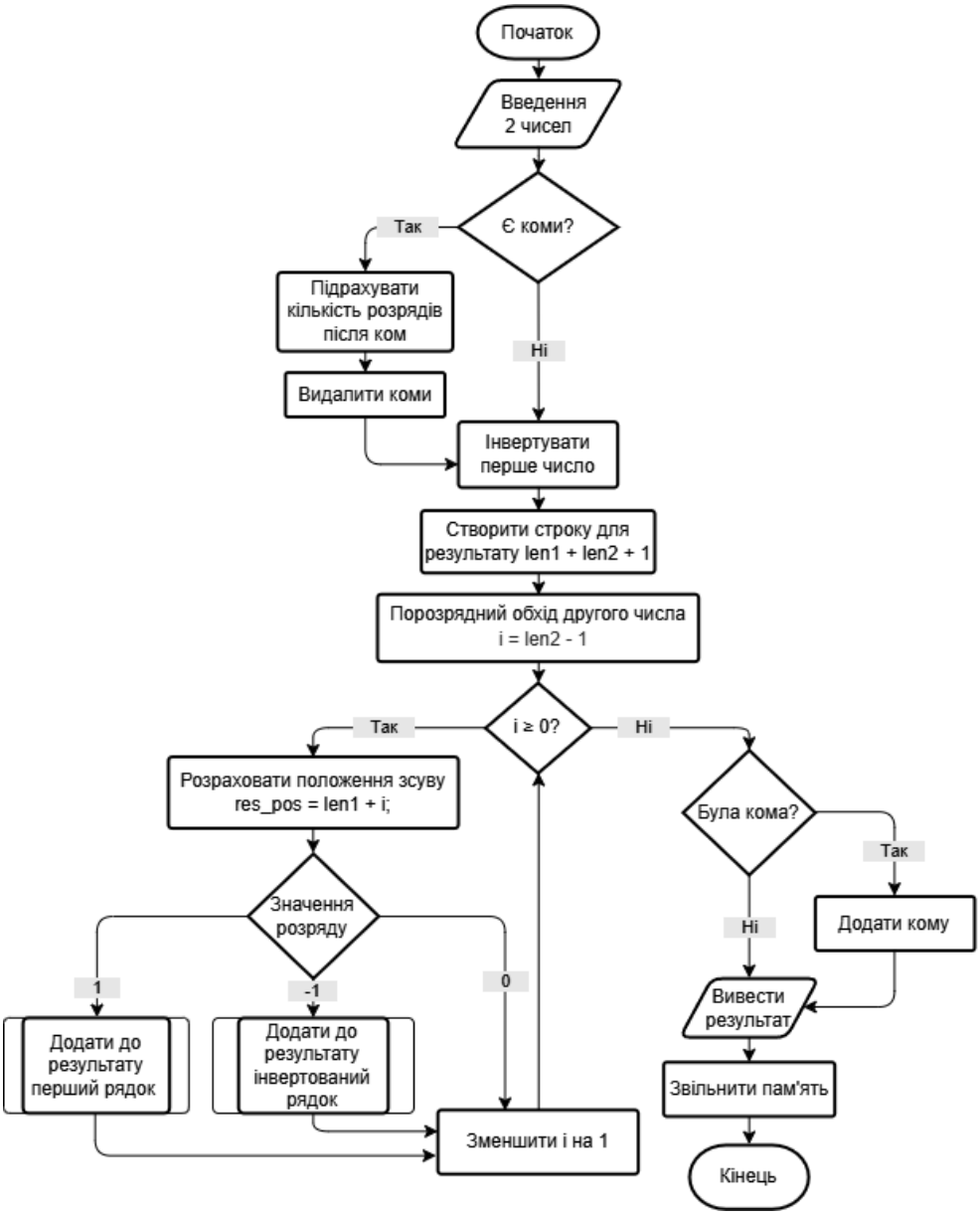

Рисунок 2.14 – Блок-схема функції multiplication().

2.3.4. Функція remove_delimiter();

Функція remove_delimiter є допоміжною до функції multiplication() і була виділена окремо для зручності та уникнення повторення коду. Вона приймає вказівник на символьний рядок char* str, що представляє число, введене користувачем

Функція шукає роздільник та копіює всі символи після нього на одну позицію ліворуч, фактично стираючи роздільник. Таким чином, усі символи після роздільника зберігаються, що дозволяє далі виконувати множення над цілими числами без необхідності оброблювати роздільник.

2.4. Модуль логічних операцій

Модуль логічних операцій реалізує операції АБО та І з урахуванням трьох станів, які через схожість своєї структури було вирішено об’єднати в одну функцію — ternary_logic_op(). Також реалізована логічна операція заперечення, що використовувалась раніше для обробки від’ємних чисел, однак функція напряму не викликається тому була описана раніше в розділі 2.2.6.

Для візуального зображення логіки взаємодії між функціями використано блок-схему наведену на рис. 2.15.

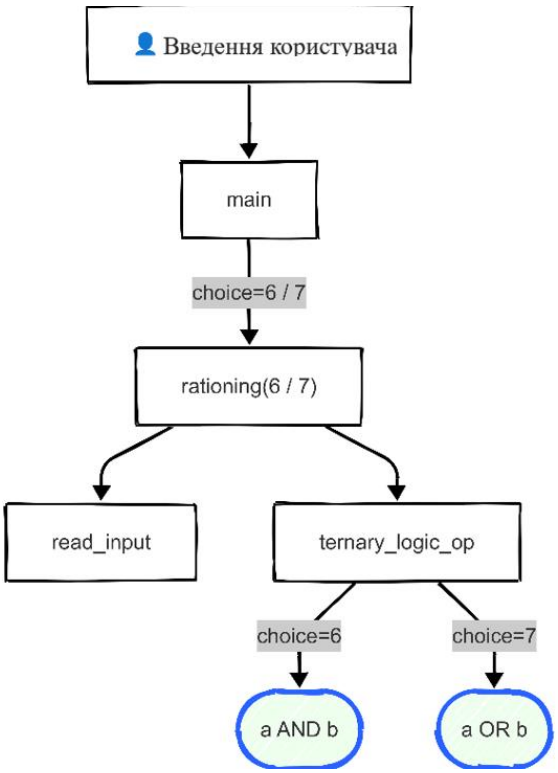

Рисунок 2.15 – Ієрархічна діаграма логічних операцій.

Схема реалізує єдиний потік обробки, де користувач обирає тип операції: choice=6 для логічного І (AND) або choice=7 для логічного АБО (OR). Після вибору система застосовує вже описану в підрозділі 2.2.3 функцію rationing() для нормалізації вхідних даних та передає їх далі для виконання логічних операцій.

2.4.1. Функція ternary_logic_op();

Функція ternary_logic_op() реалізує базові логічні операції в ТССЧ, зокрема операції логічного AND та логічного OR. У трійковій симетричній логіці, де використовуються значення «-1», «0» та «1», ці операції зберігають фундаментальні властивості аналогічних операцій у двійковій логіці. Зокрема, результат кон'юнкції відповідає мінімальному значенню серед операндів, тоді як результат диз'юнкції визначається максимальним значенням. Визначення цих операцій представлене у таблицях істинності на рисунках 2.16 та 2.17 [15].

AND		A		
		-1	0	1
B	-1	-1	-1	-1
	0	-1	0	0
	1	-1	0	1

Рисунок 2.16 – Таблиця істинності операції AND.

OR		A		
		-1	0	1
B	-1	-1	0	1
	0	0	0	1
	1	1	1	1

Рисунок 2.17 – Таблиця істинності операції OR.

Функція ternary_logic_op() приймає вказівник на масив char **matrix, який містить нормалізовані трійкові числа у симетричній системі, та параметр operation. Обробка відбувається порозрядно: для кожної пари символів на однакових позиціях функція визначає результат на основі обраної операції - для AND вибирається мінімальне значення серед операндів, для OR - максимальне. Результат формується у вигляді нового трійкового числа однакової довжини з вхідними операндами.

Висновок до розділу 2

У другому розділі було розроблено програмну модель трійкового симетричного калькулятора, який дозволяє виконувати арифметичні операції (додавання, віднімання, множення) і логічні операції (AND, OR) у ТССЧ. Особливу увагу приділено модульній архітектурі програми, що складається з трьох основних компонентів:

1. Модуль перетворення чисел, який забезпечує двосторонню конвертацію між десятковою та трійковою симетричною системами.

2. Модуль арифметичних операцій, де особливістю стало реалізація віднімання через додавання інвертованого числа, що спростило архітектуру системи. Алгоритми операцій враховують специфіку трійкової симетричної системи, включаючи механізми обробки переносів.

3. Модуль логічних операцій, який реалізує базові операції з урахуванням трьох станів трійкової логіки.

Важливим досягненням стало створення механізму нормалізації чисел, що дозволяє коректно виконувати операції з числами різної довжини представленими у вигляді рядків. Обмеженням дослідження є відсутність підтримки операції ділення, що може стати напрямком подальших розробок.

РОЗДІЛ 3
ТЕСТУВАННЯ ТА РЕКОМЕНДАЦІЇ ЩОДО ВИКОРИСТАННЯ

3.1. Модульне тестування

Модульне тестування проводиться для кожної функції окремо, без урахування її зв'язків з іншими частинами програми. Для цього функції викликаються з різними вхідними даними, результати виконання порівнюються з очікуваними значеннями. У разі розбіжності фіксується помилка або аномальне поводження.

У даному розділі будуть розглянуті результати тестування окремих функцій, які відповідають за ключові обчислення в ТССЧ: перетворення цілої та дробової частин у трійковий формат, виконання інверсії, симетризація, перетворення з трійкової в десяткову систему числення, а також арифметичних і логічних операцій.

3.1.1. Тестування функції перетворення цілої частини

Для модульного тестування функції перетворення цілої частини десяткового числа в трійкову несиметричну систему числення була розроблена тестова функція test_whole_part(). Вона передає вхідні значення безпосередньо до whole_part(), отримує результат перетворення та порівнює його з очікуваним. Тест включає перевірку трьох значень: вхідні десяткові числа «5», «107», «0»; очікувані трійкові числа «12», «10222», «0» відповідно. На рис. Д.1 зображено код цієї тестової функції.

Виконаємо перевірку через тестову функцію test_whole_part(). Результат наведено на рис. 3.1.

```
Test: whole_part()

[Test 1] whole_part("5") -> "12" (expected: "12")
[Test 2] whole_part("10") -> "10222" (expected: "10222")
[Test 3] whole_part("0") -> "0" (expected: "0")
whole_part() passed.
```

Рисунок 3.1 – Результати тестування whole_part().

Тест підтверджує коректність роботи функції whole_part.

3.1.2. Тестування функції перетворення дробової частини

Для модульного тестування функції перетворення дробової частини десяткового числа в ТССЧ була розроблена тестова функція test_fraction_part(). Вона передає вхідні значення безпосередньо до fraction_part(), отримує результат перетворення та порівнює його з очікуваним шаблоном. Тест включає перевірку трьох значень: вхідні десяткові числа «0,25» («25») , «0,5» («5»), «0,33» («33»); очікувані трійкові числа «0202…» «111111...», «022220...». На рис. Д.2 наведено код тестової функції.

Виконаємо перевірку через тестову функцію test_fraction_part(). Результат наведено на рис. 3.2.

```
Test: fraction_part()

[Test 1] fraction_part("25") -> "020202020202020" (expected: starts with ~ "020202...")
[Test 2] fraction_part("5") -> "111111111111111" (expected: starts with ~ "111111...")
[Test 3] fraction_part("33") -> "022220120101112" (expected: starts with ~ "022220...")
fraction_part() passed.
```

Рис 3.2 – Результати тестування fraction_part().

Тест підтверджує коректність роботи функції fraction_part.

3.1.3. Тестування функції балансування трійкового числа

Для модульного тестування функції перетворення несиметричного трійкового числа в симетричну форму була розроблена тестова функція test_symmetry(). Вона перевіряє коректність роботи функції symmetry() для різних вхідних даних, включаючи числа з цілою та дробовою частинами. Для перевірки було обрано 2 числа, які демонструють перенос розряду, що може виникнути під час перетворення в ТССЧ: ціле число «22» та дробове число «0,2»; очікувані значення «10i» та «1,i». На рис. Д.3 наведено код функції test_symmetry().

Виконаємо перевірку через тестову функцію test_symmetry(), щоб переконатися в правильності роботи symmetry(). Результат наведено на рис. 3.3.

```
Test: symmetry()
symmetry("22", 0): expected: "10i"
Actual:
Balanced ternary:   10i
symmetry("0,2", 1): expected: "1,i"
Actual:
Balanced ternary:   1,i
symmetry() passed.
```

Рис 3.3 – Результати тестування symmetry().

Функція symmetry() коректно виконує перетворення несиметричних трійкових чисел у симетричну форму, включаючи як цілу частину, так і дробову.

3.1.4. Тестування функції інвертування трійкового числа

Для модульного тестування функції інвертування симетричного трійкового числа (1 ↔ i) була розроблена тестова функція test_invert(). Вона перевіряє коректність роботи функції invert() для різних вхідних даних. Тест включає перевірку трьох значень: вхідні значення «1i0», «111», «0i1»; очікуваний результат «i10», «iii», «01i». На рис. Д.4 наведено код функції test_invert().

Виконаємо перевірку функції invert() через тестову функцію test_invert(). Отримані результати наведено на рис. 3.4.

```
Test: invert()
invert("1i0") -> "i10" (expected: "i10")
invert("111") -> "iii" (expected: "iii")
invert("0i1") -> "01i" (expected: "01i")
invert() passed.
```

Рис. 3.4 – Результати тестування invert ().

Отримані результати свідчать, шо реалізація функції коректна.

3.1.5. Тестування функції складання трійкових чисел

Для модульного тестування функції addition(), що відповідає за додавання чисел в ТССЧ, було розроблена тестова функція test_addition(), яка перевіряє коректність складання і віднімання, яке реалізовано як складання від’ємного числа. Функція передає однакові за довжиною числа, отримує результат та перевіряє його з очікуваним результатом. Тест включає перевірку трьох пар

значень: вхідні значення («1i» та «i1»), («11» та «11»), («ii» а «ii»); очікувані результати «0», «10i», «i01». Код функції test_addition() наведено на рис. Д.5.

Перевіримо правильність роботи функції addition() скориставшись функцією test_addition(). Результати наведено на рис. 3.5.

```
Test: addition()

[Test 1] addition(["1i", "i1"]) -> expected: 0
Result: 0

[Test 2] addition(["11", "11"]) -> expected: 10i
Result: 10i

[Test 3] addition(["ii", "ii"]) -> expected: i01
Result: i01
addition() completed.
```

Рисунок 3.5 – Результати тестування addition().

Тест підтверджує коректність роботи функції addition().

3.1.6. Тестування функції перетворення чисел в десяткову систему

Оскільки функція ternary_to_decimal() не приймає жодних параметрів, а зчитує вхідні дані безпосередньо з клавіатури за допомогою read_input(), виконати її повноцінне модульне тестування у звичному вигляді неможливо. Для перевірки правильності роботи функції було тимчасово жорстко задано значення змінної input безпосередньо всередині функції. Це дозволяє контролювати вхідні дані, уникати залежності від зовнішнього введення та перевіряти відповідність отриманого результату очікуваному значенню, розрахованому вручну. Тест включає перевірку двох значень: вхідні значення «10i1,10100i» та «i10i,1i1i1i»; очікувані результати «25,369» та «-18,75».

Перевіримо правильність виконання функції ternary_to_decimal(). Результати наведено на рис. 3.6.

```
Input ternary number: 10i1,10100i
Decimal value: 25.3689986283

Input ternary number: i10i,1i1i1i
Decimal value: -18.7503429355
```

Рисунок 3.6 – Результати тестування ternary_to_decimal().

Отримані результати співпадають з очікуваними з невеликою похибкою, що пов’язана з обмеженням точності відображення дробової частини трійкового числа, яка у програмі виводиться з шістьма знаками після коми. Також слід враховувати, що не існує дробу, який мав би скінченний вигляд одночасно в обох системах числення. Скінченні дроби в ТССЧ — це такі, що утворюються поділом на степені числа 3. Водночас у десятковій системі дроби, які містять у знаменнику число 3, є нескінченними періодичними. Наприклад, 1/3 у трійковій системі записується як «0,1», тоді як у десятковій — це періодичний дріб «0,(3)».

3.1.7. Тестування функції логічних операцій

Для модульного тестування функції логічних операцій над симетричними трійковими числами була розроблена тестова функція test_ternary_logic_op(). Вона перевіряє коректність роботи функції ternary_logic_op() для двох видів операцій — кон’юнкції (AND) та диз’юнкції (OR) — на різних вхідних даних. Код функції test_ternary_logic_op() наведено на рис. Д.6. Тест включає перевірку значень: вхідні числа «1i0» та «i10»; очікувані значення «ii0», «110».

Виконаємо перевірку функції ternary_logic_op() через тестову функцію test_ternary_logic_op(). Отримані результати наведено на рис. 3.7.

```
Test: ternary_logic_op()

[Test 1 - AND] Inputs: "1i0", "i10" -> Expected: ii0
Logic operation result: ii0
[Test 1 - OR]  Expected: 110
Logic operation result: 110
```

Рисунок 3.7 – Результати тестування ternary_logic_op().

Отримані результати свідчать, що функція правильно виконує задані логічні операції над симетричними трійковими числами.

3.2. Інтеграційне тестування

Інтеграційне тестування проводиться для груп функцій, які разом реалізують складні операції. На відміну від модульного тестування, де перевіряються окремі функції ізольовано, тут аналізується їхня взаємодія та коректність передачі даних між ними. Для цього викликаються послідовності

функцій з різними вхідними даними, а результати виконання порівнюються з очікуваними значеннями. У разі розбіжності фіксується помилка в логіці взаємодії.

У даному розділі будуть розглянуті результати тестування таких ключових операцій:

- перетворення з десяткової в трійкову систему числення (функції decimal_to_ternary, whole_part, fraction_part та symmetry)
- арифметичних операцій додавання та віднімання (функції rationing, addition та invert)
- логічних операцій AND та OR (функції rationing та ternary_logic_op)
- операції множення (функції multiplication, invert та remove_delimiter)

Тестування дозволяє переконатися, що функції коректно взаємодіють між собою та готові до подальшого використання в комплексній системі.

3.2.1. Тестування перетворення в ТССЧ

Функція decimal_to_ternary() не приймає параметрів, а тільки зчитує вхідні дані з клавіатури за допомогою read_input(), тому для перевірки правильності виконання функції decimal_to_ternary() було тимчасово жорстко задано значення змінної input безпосередньо в середині функції. Це дозволяє контролювати вхідні дані, уникати залежності від зовнішнього введення та перевіряти відповідність отриманого результату очікуваному. Для тестування були вибрані наступні значення: «25», «-37», «54,6», «-5,5»

Отримані трійкові результати наведено на рис. 3.8.

```
Input decimal number: -37
Balanced ternary:   ii0i,000000000000000

Input decimal number: 25
Balanced ternary:   10i1,000000000000000

Input decimal number: 54,6
Balanced ternary:   1i001,ii11ii11ii11ii1

Input decimal number: -5,5
Balanced ternary:   i11,iiiiiiiiiiiiiii
```

Рис 3.8 – Результати тестування decimal_to_ternary.

Для перевірки правильності виконання переводу чисел із десяткової у ТССЧ було використано функцію ternary_to_decimal, що пройшла модульне тестування в розділі 3.1.6, щоб порівняти початкове число з отриманим. Значення після зворотного перетворення наведено на рис. 3.9.

```
Input ternary number: 10i1
Decimal value: 25.0000000000

Input ternary number: ii0i
Decimal value: -37.0000000000

Input ternary number: 1i001,ii11ii11ii11ii1
Decimal value: 54.5999999861

Input ternary number: 1i001,ii11ii11ii11ii1
Decimal value: 54.5999999861
```

Рисунок 3.9 – Результати зворотного перетворення.

Отримані результати свідчать про коректність перетворення довільних десяткових чисел у ТССЧ з невеликою похибкою, детально описаною у розділі 3.1.6.

3.2.2. Тестування складання та віднімання довільних чисел

Інтегральне тестування складання та віднімання довільних дійсних чисел проводиться шляхом перевірки взаємодії функцій rationing() та addition(). Оскільки функція rationing() не приймає параметрів і зчитує дані безпосередньо з клавіатури, для забезпечення контрольованого тестування значення трійкових чисел було жорстко задано всередині функції.

Для тестування було обрано пара чисел різної довжини та з наявністю або відсутністю дробової частини, щоб перевірити коректність нормування: «110i1i» і «1110i,1i10i1i», яким в ДСЧ відповідають значення «317» і «116.256»

Результати виконання операції складання після нормалізації наведено на рис. 3.10.

```
Choice (operation): 3
Enter first number: 110i1i
Enter second number: 1110i,1i10i1i

Result: 1ii1001,1i10i1i
```

Рисунок 3.10 – Результати складання довільних чисел.

Для перевірки коректності проведено зворотне перетворення результатів з ТССЧ у ДСЧ. Отримані значення наведено на рис. 3.11.

```
Enter a ternary number: 1ii1001,1i10i1i
Decimal value: 433.2560585277
```

Рисунок 3.11 - Результати зворотного перетворення.

Отриманий результат повністю збігається з очікуваним «433,256».

Результати виконання операції віднімання після нормалізації наведено на рис. 3.12.

```
Choice (operation): 4
Enter first number: 110i1i
Enter second number: 1110i,1i10i1i

Result: 1i1110,i1i01i1
```

Рисунок 3.12 – Результати віднімання довільних чисел.

Перевіримо отримані числа перетворивши результати з ТССЧ в ДСЧ та порівняємо їх з очікуваними. Значення після зворотного перетворення наведено на рис. 3.13.

```
Enter a ternary number: 1i1110,i1i01i1
Decimal value: 200.7439414723
```

Рисунок 3.13 - Результати зворотного перетворення.

Отриманий результат збігається з очікуваним «200,744».

3.2.3. Тестування логічних операцій для довільних чисел

Інтегральне проводиться шляхом перевірки взаємодії функцій rationing() та ternary_logic_op(). Під час тестування було реалізовано виклик rationing() з жорстко заданими значеннями двох трійкових чисел, що дозволяє уникнути зчитування з клавіатури та забезпечити контрольований сценарій.

Для перевірки коректності нормалізації та виконання логічних операцій було обрано два числа різної довжини «10i10» і «111». Очікувані результати: «00i10» для AND та «10111» для OR.

Результати виконання операцій наведено на рис. 3.14.

```
Choice (operation): 6
Enter first number: 10i10
Enter second number: 111

Logic operation result: 00i10

Choice (operation): 7
Enter first number: 10i10
Enter second number: 111

Logic operation result: 10111
```

Рисунок 3.14 – Результати виконання логічних операцій.

Тест підтверджує коректність роботи функції ternary_logic_op().

3.2.4. Тестування множення чисел в ТССЧ

Інтегральне тестування виконується шляхом перевірки взаємодії функції multiplication() з підфункціями remove_delimiter() і invert(). Під час тестування було реалізовано виклик multiplication() з жорстко заданими значеннями двох трійкових чисел, що дозволяє уникнути зчитування з клавіатури та забезпечити контрольований сценарій.

Для тестування були обрані два числа з дробовою частиною: «i1,i» та «11i,1i», які відповідають десятковим значенням «−2,(3)» і «11,(2)». Результат множення цих чисел наведено на рис. 3.15.

```
First ternary number (input1): i1,i
Second ternary number (input2): 11i,1i

Multiplication result: i001,i11
```

Рисунок 3.15 – Результат виконання множення.

Для перевірки коректності проведено зворотне перетворення результатів з ТССЧ у ДСЧ. Отримані значення наведено на рис. 3.16.

```
Enter a ternary number: i001,i11
Decimal value: -26.1851851852
```

Рисунок 3.16 - Результати зворотного перетворення.

Отримане значення збігається з очікуваним результатом, що свідчить про правильність реалізації функції множення.

3.3. Системне тестування

Системне тестування перевіряє комплексну роботу всіх модулів, що дозволяє виявити потенційні проблеми взаємодії між окремими компонентами системи. Тестування проводиться шляхом послідовного виконання операцій через користувацький інтерфейс програми (Рисунок 3.17-3.23):

1. Перетворення чисел «5,125», «3,15», «2,0», «1,75», «-1,2» в ТССЧ.
2. Складання чисел «5,125» і «3,15»
3. Множення результату на «2».
4. Віднімання від результату «1,75»
5. Віднімання від результату «-1,2»
6. Перетворення кінцевого результату в ДСЧ.

Очікуваний результат: $(5{,}125 + 3{,}15) \cdot 2.0 - 1{,}75 - (-1{,}2) = 16$

```
Enter your choice: 1

Enter a decimal number: 5,125
Balanced ternary:   1ii,010101010101010
Enter a decimal number: 3,15
Balanced ternary:   10,011001100110011
Enter a decimal number: 2
Balanced ternary:   1i,000000000000000
Enter a decimal number: 1,75
Balanced ternary:   1i,i1i1i1i1i1i1i1i
Enter a decimal number: -1,2
Balanced ternary:   i,i11ii11ii11ii11
```

Рисунок 3.17 – Перетворення чисел в ТССЧ.

```
Enter your choice: 3
Enter first number: 1ii,010101010101010
Enter second number: 10,011001100110011

Result: 10i,1i111i111i111i1
```

Рисунок 3.18 – Складання двох чисел.

```
Enter your choice: 5
Enter first number: 10i,1i111i111i111i1
Enter second number: 1i

Multiplication result: 1i0i,ii00ii00ii00iii
```

Рисунок 3.19 – Множення двох чисел.

```
Enter your choice: 4
Enter first number: 1i0i,ii00ii00ii00iii
Enter second number: 1i,i1i1i1i1i1i1i1i

Result: 1ii0,i11ii11ii11ii10
```

Рисунок 3.20 – Різниця між двома числами.

```
Enter your choice: 4
Enter first number: 1ii0,i11ii11ii11ii10
Enter second number: i,i11ii11ii11ii11

Result: 1ii1,000000000000000i
```

Рисунок 3.21 – Різниця з від'ємним числом.

```
Enter your choice: 2

Enter a ternary number: 1ii1,000000000000000i
Decimal value: 15.9999999303
```

Рисунок 3.22 – Отриманий результат.

```
Enter your choice: 1

Enter a decimal number: 16
Balanced ternary:   1ii1,000000000000000
```

Рисунок 3.23 – Очікуваний результат в ТССЧ.

Кінцевий результат обчислень у ТССЧ практично збігається з очікуваним. Незначна розбіжність у молодшому розряді дробової частини пояснюється особливостями подання чисел у ТССЧ та вже розглядалася у розділі 3.1.6. Отже, можна зробити висновок, що програмна модель працює правильно.

3.4. Недоліки та обмеження

Недоліком розробленої програмної моделі є відсутність перевірки коректності введених користувачем чисел. У разі введення невалідних символів вони автоматично інтерпретуються як нулі без будь-яких попереджень. Наприклад, введення рядка «10ig71» буде сприйнято як «10i001», як зображено на рис. 3.24.

```
Enter a ternary number: 10ig71
Decimal value: 217.0000000000

Enter a ternary number: 10i001
Decimal value: 217.0000000000
```

Рисунок 3.24 – Мовчазне виправлення.

Таким чином, хоча програма технічно коректно обробляє вхідні дані згідно з заданою логікою, користувач може не помітити помилку у введенні, що потенційно призведе до небажаного результату.

У розробленій програмній моделі існує обмеження на довжину введеного десяткового числа, що не повинна перевищувати 15 розрядів. Це пов’язано з використанням типу даних double, який відповідає стандарту IEEE 754 (binary64) і забезпечує точність приблизно до 15 значущих десяткових цифр [16]. У випадку перевищення цієї межі, значення автоматично округлюється на етапі зберігання, що призводить до втрати точності. На рис. 3.25 показано, як під час перетворення числа довжиною 17 розрядів у ТССЧ, а потім назад у ДСЧ, результат змінюється через внутрішню похибку зберігання.

```
Enter your choice: 1

Enter a decimal number: 12345678901234567
Balanced ternary:   1i1i000i00i11i10101111100 1i1111101
Enter your choice: 2

Enter a ternary number: 1i1i000i00i11i10101111100 1i1111101
Decimal value: 12345678901234556.0000000000
```

Рисунок 3.25 – Втрата точності при 17 розрядах.

Щоб уникнути таких неточностей, у програмі було обмежено довжину введеного десяткового числа. Як потенційне рішення цієї проблеми можна використати бібліотеку GMP [17], яка дозволяє виконувати арифметичні операції з довільною точністю.

Висновок до розділу 3

У третьому розділі було проведено комплексне тестування програмної моделі калькулятора з симетричною трійковою логікою. Модульне тестування підтвердило коректність роботи окремих функцій. Інтеграційне тестування показало, що функції взаємодіють між собою правильно, забезпечуючи точність обчислень у ТССЧ. Системне тестування продемонструвало, що програма

стабільно працює при виконанні послідовних операцій, а результати відповідають очікуваним значенням.

Проте є деякі недоліки та обмеження:

- Відсутність перевірки валідності вхідних даних, що може призводити до неочікуваних результатів.
- Обмеження на довжину введених чисел через використання типу double, що може спричинити втрату точності.

Ці недоліки можуть бути виправлені у подальших роботах шляхом впровадження додаткових перевірок та використання бібліотек для роботи з числами довільної точності, таких як GMP.

ВИСНОВКИ

У даній роботі було проаналізовано основні переваги та особливості ТССЧ у порівнянні з традиційною двійковою системою. Розглянуто існуючі підходи до реалізації трійкової логіки, її теоретичні основи, історичні та потенційні області застосування.

На основі проведеного аналізу реалізовано програмну модель калькулятора, що підтримує перетворення чисел між десятковою та ТССЧ, а також базові арифметичні (додавання, віднімання, множення) і логічні операції (AND, OR). Усі алгоритми реалізовано вручну без використання сторонніх бібліотек, з обробкою дробової частини, інверсією знаку та балансуванням розрядів. Програму написано мовою C (стандарт C99) з використанням динамічної пам’яті та роботи з рядками.

Для перевірки працездатності моделі було проведено модульне, інтеграційне та системне тестування. Отримані результати підтвердили коректність реалізації: обчислення виконуються точно, перетворення є правильними, а логіка функцій зберігається в різних сценаріях. Системні тести показали стабільність роботи навіть за умов багатокрокових обчислень.

Надалі модель може бути доповнена підтримкою ділення, розширенням точності обчислень, а також графічним інтерфейсом. Отримані результати можуть бути використані в освітніх цілях, для демонстрації принципів альтернативних систем числення або як основа для подальших досліджень у цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhu F., Xu P., Zong J. Moore's Law: The potential, limits, and breakthroughs // Матеріали 2023 International Conference on Mechatronics and Smart Systems. — 2023. — С. 307–315. — DOI: 10.54254/2755-2721/10/20230038. [Електронний ресурс]. — Режим доступу: https://www.researchgate.net/publication/374208634_Moore's_Law_The_potential_limits_and_breakthroughs. (дата звернення: 12.01.2025).
2. Connelly J. Ternary Computing Testbed: 3-Trit Computer Architecture. California Polytechnic State University (Cal Poly), San Luis Obispo, 2008. 174 с. [Електронний ресурс]. – Режим доступу: URL: https://www.scribd.com/document/78370674/Ternary-Computing-Testbed-3-Trit-Computer-Architecture (дата звернення: 15.01.2025).
3. Ciucci D., Dubois D. Three-Valued Logics, Uncertainty Management and Rough Sets // Transactions on Rough Sets XVII. Berlin; Heidelberg: Springer, 2014. С. 1–32. DOI: 10.1007/978-3-642-54756-0_1. [Електронний ресурс]. – Режим доступу: URL: https://www.researchgate.net/publication/260622998_Three-Valued_Logics_Uncertainty_Management_and_Rough_Sets (дата звернення: 16.01.2025).
4. Ornes S. How Base 3 Computing Beats Binary // Quanta Magazine. 2024. August 9. [Електронний ресурс]. – Режим доступу: URL: https://www.quantamagazine.org/how-base-3-computing-beats-binary-20240809/ (дата звернення: 17.01.2025).
5. Rine D. C. (ed.) Computer Science and Multiple-Valued Logic. Theory and Applications. Amsterdam: Elsevier, 1977. 548 с. ISBN 978-0-7204-0406-7.
6. Jones D. W. Ternary Number Systems [Електронний ресурс]. – Режим доступу: URL: http://homepage.cs.uiowa.edu/~jones/ternary/numbers.shtml (дата звернення: 21.01.2025).

7. Hayes B. Third Base // American Scientist. 2001. Vol. 89, No. 6. C. 490–494. [Електронний ресурс]. – Режим доступу: URL: http://bit-player.org/wp-content/extras/bph-publications/AmSci-2001-11-Hayes-ternary.pdf (дата звернення: 22.01.2025).
8. Knuth D. E. The Art of Computer Programming. Vol. 2: Seminumerical Algorithms. 3rd ed. Boston: Addison Wesley Longman, 1997. 762 с. ISBN 0-201-89684-2. [Електронний ресурс]. – Режим доступу: URL: https://seriouscomputerist.atariverse.com/media/pdf/book/Art%20of%20Computer%20Programming%20-%20Volume%202%20(Seminumerical%20Algorithms).pdf (дата звернення: 25.01.2025).
9. History of Ternary Computers [Електронний ресурс]. – Режим доступу: URL: https://mason.gmu.edu/~drine/History-of-Ternary-Computers.htm (дата звернення: 24.01.2025).
10. Ternac // Wikipedia: The Free Encyclopedia [Електронний ресурс]. – Режим доступу: URL: https://en.wikipedia.org/wiki/Ternac (дата звернення: 25.01.2025).
11. Jia Y., Li M. A Survey of Ternary Optical Computer Research // International Journal of Advanced Network, Monitoring and Controls. 2022. Vol. 07, No. 04. C. 47–58. [Електронний ресурс]. – Режим доступу: URL: https://www.researchgate.net/publication/371056214_A_Survey_of_Ternary_Optical_Computer_Research (дата звернення: 27.01.2025).
12. Faghih E., Taheri M., Navi K., Bagherzadeh N. Efficient realization of quantum balanced ternary reversible multiplier building blocks: A great step towards sustainable computing // Sustainable Computing: Informatics and Systems. 2023. Vol. 40. Article 100908. [Електронний ресурс]. – Режим доступу: URL: https://doi.org/10.1016/j.suscom.2023.100908 (дата звернення: 27.01.2025).
13. Anonymous authors. Revisiting Ternary Neural Networks towards Asymmetric Thresholds and Uniform Distribution // ICLR 2024. 2023. [Електронний

ресурс]. – Режим доступу:
URL: https://openreview.net/pdf?id=mWf3RGc6HG (дата звернення: 28.01.2025).

14. ISO/IEC 9899:1999 Programming languages — C. International Organization for Standardization, 1999. 554 с. [Електронний ресурс]. – Режим доступу: URL: https://www.dii.uchile.cl/~daespino/files/Iso_C_1999_definition.pdf (дата звернення: 09.02.2025).
15. Malinowski G. Kleene Logic and Inference // Bulletin of the Section of Logic. 2014. Vol. 43, № 1/2. С. 43–52. [Електронний ресурс]. – Режим доступу: URL: https://www.researchgate.net/publication/267133321_Kleene_logic_and_inference (дата звернення: 12.02.2025).
16. IEEE Standard for Floating-Point Arithmetic (IEEE Std 754™-2008) // IEEE Computer Society. [Електронний ресурс]. – Режим доступу: URL: http://www.dsc.ufcg.edu.br/~cnum/modulos/Modulo2/IEEE754_2008.pdf (дата звернення: 03.03.2025).
17. GNU Multiple Precision Arithmetic Library (GMP) Manual. Free Software Foundation, 2023. [Електронний ресурс]. – Режим доступу: URL: https://gmplib.org/gmp-man-6.3.0.pdf (дата звернення: 05.03.2025).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 15 – Автоматизація та приладобудування
Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Буяльського Дмитра Володимировича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **ПРОГРАМНА МОДЕЛЬ КАЛЬКУЛЯТОРА ІЗ СИМЕТРИЧНОЮ ТРІЙКОВОЮ ЛОГІКОЮ**

керівник роботи Котвицький Альберт Тадеушевич, к. ф.–м. н., доц. кафедри КСтаР
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***16 квітня 2025 року № 4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити.
1. Аналіз переваг трійкової логіки та системи числення в порівнянні з двійковими.
2. Розробити алгоритми перетворення чисел між десятковою та трійковою симетричною системами числення.
3. Реалізувати алгоритми для виконання арифметичних операцій (додавання, віднімання) у трійковій симетричній системі числення.
4. Розробити алгоритми логічних операції (заперечення, логічного І та логічного АБО) у трійковій симетричній системі числення.
5. Створити програмну модель калькулятора мовою програмування С для роботи з трійковою симетричною системою числення.
6. Протестувати коректність переведення чисел та виконання арифметичних і логічних операцій у програмній моделі.
7. Проаналізувати результати тестування, визначити помилки, запропонувати шляхи їх виправлення та скласти рекомендації щодо використання програмної моделі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми кваліфікаційної роботи.	02.10.2024-09.11.2024
2	Огляд актуальності теми.	10.112024-22.11.2024
3	Аналіз літератури з трійкової логіки та трійкової системи числення.	23.11.2024-17.12.2024
4	Огляд існуючих методів роботи з числами в трійковій симетричній системі числення.	18.12.2024-26.12.2024
5	Розробка програмної моделі	27.12.2024-03.01.2025
6	Програмна реалізація алгоритму переведення довільних дійсних чисел між десятковою та трійковою симетричною системами числення.	04.01.2025-11.01.2025
7	Програмна реалізація арифметичних та логічних операцій з числами в трійковій симетричній системі числення.	12.01.2025-13.02.2025
8	Тестування розроблених програмних реалізацій.	14.02.2025-18.02.2025
9	Аналіз результатів тестування.	19.02.2025-22.02.2025
10	Підготовка та оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	23.02.2025-19.04.2025
11	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів НДР	20.04.2025-30.05.2025
12	Представлення кваліфікаційної роботи керівнику та рецензенту	30.05.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент Д. В. Буяльський _____
ініціали, прізвище підпис

Керівник роботи А. Т. Котвицький _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання

на розробку програмного виробу «Програмна модель калькулятора із симетричною трійковою логікою»

Назва розділу	Назва та зміст підрозділу
1. Вступ	1.1. Назва програмного виробу «Програмна модель калькулятора із симетричною трійковою логікою» 1.2. Галузь застосування: різні системи числення і правила роботи в них, освіта, теоретична інформатика та обчислювальна математика
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація, комп'ютерно-інтегровані технології та Робототехніка 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/962 від 16.04.2025. (представити як Додаток А до пояснювальної записки до дипломної роботи).
3.Призначення розробки	3.1. Мета розробки програмного виробу: розробка та програмна реалізація методів роботи з числами в симетричній трійковій системі числення. 3.2. Призначення програмного виробу: виконання арифметичних (додавання, віднімання, множення) та логічних (AND, OR) операцій у симетричній трійковій системі числення (-1, 0, 1) з двостороннім перетворенням між трійковим та десятковим представленнями. 3.3. Вхідні дані для розробки: Вхідними даними для програмного калькулятора є довільні дійсні числа, представлені у десятковій системі числення або у симетричній трійковій системі з використанням символів і (-1), 0 та 1. Числа можуть містити цілу та дробову частини, розділені крапкою або комою. Десяткові числа можуть бути представлені як від'ємні з використанням знаку мінус. 3.4. Вихідні дані для розробки: На виході програма повертає результати у двох основних форматах. Для операцій перетворення між системами числення калькулятор повертає еквівалентне представлення числа у цільовій системі - десяткове число з фіксованою точністю до 10 знаків після коми або трійкове число у симетричній формі. При виконанні арифметичних операцій (додавання, віднімання, множення) та логічних операцій (AND, OR) результат завжди подається

	у трійковій симетричній системі числення, що дозволяє наочно демонструвати особливості роботи з тризначною логікою.	
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: Програмний продукт повинен забезпечувати перетворення між десятковою та симетричною трійковою системами числення, виконання арифметичних (додавання, віднімання, множення) та логічних операцій (AND, OR). 4.2. Вимоги до надійності: Програма повинна коректно обробляти вхідні дані та видавати точні результати без збоїв при тривалому використанні. 4.3. Вимоги до умов експлуатації: Програма призначена для роботи на ПК з операційною системою Windows та потребує встановленого програмного забезпечення для виконання додатків С. 4.4. Вимоги до складу параметрів технічних засобів: Необхідний комп'ютер з мінімум 2 ГБ ОЗУ та 100 МБ вільного місця на диску. 4.5. Вимоги до інформаційної та програмної сумісності: Програма повинна бути сумісна з Windows. Компілятор, що підтримує стандарт С99 або новіший. 4.6. Вимоги до маркіровки та упаковки: Не потребує. 4.7. Вимоги до транспортування та зберігання: Не потребує. 4.8. Спеціальні вимоги: Немає	
5. Вимоги до програмної документації.	Програмною документацією до виробу «Програмна модель калькулятора із симетричною трійковою логікою» вважати: 1) Справжнє Технічне завдання на розробку програмного виробу (зазначити у вигляді Додатку Б пояснювальної записки кваліфікаційної роботи) 2) Програму та методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки кваліфікаційної роботи) 3) Опис програмного виробу (представити у вигляді Розділу 2 до пояснювальної записки кваліфікаційної роботи). 4) Текст програми (представити у вигляді Додатку Г пояснювальної записки до кваліфікаційної роботи).	
6.Техніко-економічні показники	Оцінку ефективності та економіко-технічних показників виконувати не потрібно	
7 Стадії та етапи розробки	Дата	Назва етапу
	02.10.2024-09.11.2024 10.112024-22.11.2024	Затвердження теми. Огляд актуальності теми.

	23.11.2024-17.12.2024	Аналіз літератури з трійкової логіки та трійкової системи числення.
	27.12.2024-03.01.2025	Розробка програмної моделі
	04.01.2025-11.01.2025	Програмна реалізація алгоритму переведення чисел між системами числення.
	12.01.2025-13.02.2025	Програмна реалізація арифметичних та логічних операцій з числами в трійковій симетричній системі числення.
	14.02.2025-18.02.2025	Тестування розроблених програмних реалізацій
	19.02.2025-22.02.2025	Аналіз результатів тестування.
	23.02.2025-19.04.2025	Підготовка та оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури.
	20.04.2025-30.05.2025	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів НДР
	30.05.2025	Представлення кваліфікаційної роботи керівнику та рецензенту
8. Порядок контролю та приймання	1) Перевірку ходу розроблення заданих проектних документів керівнику курсової роботи здійснювати 1 раз на 2 тижні. 2) Розроблені проектні документи подати в електронному вигляді та на паперових носіях в одному примірнику. 3) Захист та прийом дипломної роботи приймається на засіданні Атестаційної комісії	

Виконавець
Студент групи КУ-41
Буяльський Д. В.
_____

Замовник
К.ф.-м.н., доцент кафедри КСтаР
Котвицький А. Т.
_____

Додаток В

Затверджую

_____

«_____»_____2025 р.

ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ
програмного виробу «Програмна модель калькулятора із симетричною трійковою логікою»

1. Об'єкт випробувань

Об’єктом випробувань є програмна модель калькулятора із симетричною трійковою логікою

2. Мета випробувань

Перевірка відповідності функціональних можливостей розробленого програмного засобу вимогам, зазначеним у технічному завданні (Додаток Б до пояснювальної записки).

3. Загальні положення

3.1 Підстави щодо випробувань

Підставою для проведення випробувань є затверджене завдання на виконання бакалаврської кваліфікаційної роботи.

3.2 Місце та тривалість випробувань

Приймальні випробування програмного забезпечення проводяться на базі комп’ютерного класу кафедри у період роботи атестаційної комісії з використанням стандартного обладнання та програмного середовища зазначених у технічних вимогах.

3.3. Обсяг випробувань

Приймальні випробування програмного засобу проводяться в обсязі, визначеному цією Програмою та методикою випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або під час засідання) за участю Замовника, Виконавця та інших осіб, призначених для прийняття бакалаврської кваліфікаційної роботи.

4. Вимоги до програмного виробу

1. Забезпечувати перетворення чисел між десятковою та симетричною трійковою системами числення в обох напрямках.

2. Виконувати арифметичні операції (додавання, віднімання, множення) у трійковій симетричній системі числення.

3. Реалізовувати логічні операції (AND, OR) для трійкових чисел.

4. Обробляти довільні дійсні числа (додатні, від’ємні, цілі, дробові числа).

5. Мати консольний інтерфейс з меню для вибору операцій та зрозумілим виведенням результатів.

6. Бути сумісною з Windows та стандартними компіляторами C.

5. Вимоги до програмної документації

Програмною документацією до виробу «Програмна модель калькулятора із симетричною трійковою логікою» вважати:

1) Справжнє Технічне завдання на розробку програмного виробу (зазначити у вигляді Додатку Б пояснювальної записки кваліфікаційної роботи)

2) Програму та методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки кваліфікаційної роботи)

3) Опис програмного виробу (представити у вигляді Розділу 2 до пояснювальної записки кваліфікаційної роботи).

4) Текст програми (представити у вигляді Додатку Г пояснювальної записки до кваліфікаційної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань

Для тестування програмної моделі калькулятора із симетричною трійковою логікою використовується стандартне обладнання: персональний комп'ютер або ноутбук з операційною системою Windows, Linux чи macOS.

Мінімальні технічні вимоги включають: процесор з тактовою частотою від 1 ГГц, оперативну пам'ять об'ємом 2 ГБ та вільний дисковий простір не менше 100 МБ.

Для компіляції та виконання програми необхідний компілятор мови C (GCC, Clang тощо), який підтримує стандарт C99 або новіші версії. Як основне середовище розробки рекомендується використовувати Visual Studio Code з відповідними розширеннями для роботи з C-кодом.

6.2 Порядок проведення випробувань

Зазвичай випробовування проводяться у два етапи:

- 1-й етап ознайомчий;
- 2-й етап випробовування програмного виробу;

Перелік перевірок, що проводяться на 1-му етапі, включає наступні операції:

1. Перевірку комплектності програмної документації.

2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

3. Перевірку комплектності складу технічних і програмних засобів.

4. Методику проведення перевірок на 1 етапі випробувань.

5. Перевірка якості програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

На другому етапі заплановано п’ять тестів, які перевірятимуть відповідність програмного моделі функціональним вимогам.

Тест 1: Конвертація між десятковою та трійковою системами.

У цьому тесті перевіряється правильність перетворення чисел між десятковою та симетричною трійковою системами числення, зворотність конверсії та збереження точності.

1. Вибирається пункт меню «1. Convert to balanced ternary».
2. Вводиться десяткове число та отримується трійкове число.
3. Вибирається пункт меню «2. Convert to decimal»

4. Вводиться отриманий результат в трійковій симетричній системі числення.

5. Порівнюються початкове та фінальне десяткові значення.

```
Enter your choice: 1

Enter a decimal number: -27.5
Balanced ternary:   i000,iiiiiiiiiiiiiiii

Enter your choice: 2

Enter a ternary number: i000,iiiiiiiiiiiiiiii
Decimal value: -27.4999999961
```

Рисунок В.1 – Конвертація між десятковою та трійковою системами.

Тест вважається виконаним, оскільки початкове та відновлене десяткові числа збігаються з допустимою похибкою.

Тест 2: Перевірка операції додавання та віднімання в симетричній трійковій системі.

У цьому тесті виконується перевірка правильності операції додавання та віднімання на основі порівняння отриманого результату зі значенням другого числа.

1. Вибирається пункт меню «3. Addition».
2. Вводиться перше трійкове число.
3. Вводиться друге трійкове число.
4. Отримується результат складання.
5. Вибирається пункт меню «4. Subtraction»
6. Вводиться отриманий результат
7. Вводиться перше число
8. Порівнюються отриманий результат з другим числом

```
Enter your choice: 3
Enter first number: 101i
Enter second number: 1i1

Result: 1100

Enter your choice: 4
Enter first number: 1100
Enter second number: 101i

Result: 1i1
```

Рисунок В.2 – Перевірка операції додавання та віднімання.

Тест вважається пройденим, оскільки результат додавання збігається з очікуваним значенням.

Тест 3: Перевірка операції множення в симетричній трійковій системі

У цьому тесті виконується перевірка правильності операції множення в симетричній трійковій системі числення на основі порівняння з результатами у десятковій формі.

1. Вибирається пункт меню «5. Multiplication».
2. Вводиться перше число («101» = «10»).
3. Вводиться друге число («1i» = «2»).
4. Отримується результат у симетричній трійковій системі.
5. Перевіряється правильність, порівнявши з розрахунком у десятковій системі або вручну.

```
Enter your choice: 5
Enter first number: 101
Enter second number: 1i

Multiplication result: 1i1i
```

Рисунок В.3 – Перевірка операції множення.

```
Enter a ternary number: 1i1i
Decimal value: 20.0000000000
```

Рисунок В.4 – Перевірка отриманого результату.

Тест вважається пройденим, оскільки результат множення збігається з очікуваним значенням.

Тест 4: Перевірка логічної операції AND

У цьому тесті потрібно перевірити правильність виконання логічної операції AND у симетричній трійковій системі.

1. Вибирається пункт меню «6. AND»
2. Вводиться перше трійкове число
3. Вводиться друге трійкове число
4. Отримується результат операції AND у трійковій системі
5. Перевіряється з очікуваним результатом з таблиці істинності

```
Enter your choice: 6
Enter first number: 111000iii
Enter second number: 10i10i10i

Logic operation result: 10i00iiii
```

Рисунок В.5 – Виконання операції AND.

Тест вважається пройденим, оскільки в результаті операції AND кожен розряд дорівнює найменшому з двох значень, які стоять на відповідних позиціях у вхідних числах (**i < 0 < 1**).

Тест 5: Перевірка логічної операції OR

У цьому тесті перевіряється правильність виконання логічної операції OR у симетричній трійковій системі.

1. Вибирається пункт меню «7. **OR**».
2. Вводиться перше трійкове число.
3. Вводиться друге трійкове число.
4. Отримується результат операції OR у трійковій системі.
5. Перевіряється з очікуваним результатом з таблиці істинності.

```
Enter your choice: 7
Enter first number: 111000iii
Enter second number: 10i10i10i

Logic operation result: 11110010i
```

Рисунок В.6 – Виконання операції OR.

Тест вважається пройденим, оскільки в результаті операції OR кожен розряд дорівнює найбільшому з двох значень, які стоять на відповідних позиціях у вхідних числах (**i < 0 < 1**).

На підставі проведених випробувань програмна модель визнана такою, що пройшла всі тести без помилок.

Виконавець

студент групи КУ-41

Буяльський Д. В.

_____

Додаток Г

Лістинг 1

Програмна модель калькулятора із трійковою логікою

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>

#define MAX_DECIMAL_PLACES 15
#define MAX_INPUT_LENGTH 100

void read_input(char **input) {
    char buffer[MAX_INPUT_LENGTH];
    if (!fgets(buffer, sizeof(buffer), stdin)) {
        *input = NULL;
        return;
    }

    buffer[strcspn(buffer, "\n")] = '\0';
    *input = strdup(buffer);
}

void invert(char *result) {
    while (*result) {
        *result = (*result == '1') ? 'i' : (*result == 'i' ?
'1' : *result);
        ++result;
    }
}

void addition(char **matrix) {
    if (!matrix || !matrix[0] || !matrix[1]) {
        fprintf(stderr, "Invalid matrix input\n");
        return;
    }

    const int columns = strlen(matrix[0]);
    char *result = (char *)malloc(columns + 1); // +2 було
    if (!result) {
        fprintf(stderr, "Memory allocation failed\n");
        return;
    }

    memset(result, '0', columns + 1);
    result[columns + 1] = '\0';

    int carry = 0;
    int res_pos = columns;

    for (int j = columns - 1; j >= 0; j--) {
```

```
            if (matrix[0][j] == ',' || matrix[0][j] == '.') {
                result[res_pos--] = ',';
                continue;
            }

            int sum = carry;
            char c1 = matrix[0][j], c2 = matrix[1][j];
            if (c1 == '1') sum += 1;
            else if (c1 == 'i') sum -= 1;
            if (c2 == '1') sum += 1;
            else if (c2 == 'i') sum -= 1;

            carry = 0;
            if (sum > 1) {
                sum -= 3;
                carry = 1;
            } else if (sum < -1) {
                sum += 3;
                carry = -1;
            }
            result[res_pos--] = (sum == 1) ? '1' : (sum == -1) ?
'i' : '0';
        }

        if (carry != 0) {
            result[res_pos--] = (carry == 1) ? '1' : 'i';
        }

        int start_pos = 0;
        while (result[start_pos] == '0' && result[start_pos + 1] !=
',' && result[start_pos + 1] != '\0') {
            start_pos++;
        }
        if (start_pos > 0) {
            memmove(result, result + start_pos, strlen(result) -
start_pos + 1);
        }

        printf("\nResult: %s\n", result);
        free(result);
    }

    void ternary_logic_op(char **matrix, int operation) {
        if (!matrix || !matrix[0] || !matrix[1]) {
            fprintf(stderr, "Invalid matrix input\n");
            return;
        }

        int columns = strlen(matrix[0]);
        char *result = malloc(columns + 1);
        if (!result) {
            fprintf(stderr, "Memory allocation failed\n");
```

```
            return;
        }

        for (int j = 0; j < columns; j++) {
            char a = matrix[0][j];
            char b = matrix[1][j];

            if (a == ',' || a == '.' || b == ',' || b == '.') {
                result[j] = ',';
                continue;
            }
            if (operation == 6) {
                result[j] = (a == 'i' || b == 'i') ? 'i' :
                            (a == '0' || b == '0') ? '0' :
                            (a == '1' && b == '1') ? '1' : '0';
            } else if (operation == 7) {
                result[j] = (a == '1' || b == '1') ? '1' :
                            (a == '0' || b == '0') ? '0' :
                            (a == 'i' && b == 'i') ? 'i' : '0';
            }
        }
        result[columns] = '\0';

        printf("\nLogic operation result: %s\n", result);
        free(result);
    }

    void rationing(int choice) {
        char *input1 = NULL, *input2 = NULL;

        printf("Enter first number: ");
        read_input(&input1);
        printf("Enter second number: ");
        read_input(&input2);

        if (!input1 || !input2) {
            fprintf(stderr, "Invalid input: NULL pointer\n");
            free(input1);
            free(input2);
            return;
        }

        if (choice == 4) {
            invert(input2);
        }

        char *delim1 = strpbrk(input1, ",.");
        char *delim2 = strpbrk(input2, ",.");

        int whole_len1 = delim1 ? delim1 - input1 : strlen(input1);
        int whole_len2 = delim2 ? delim2 - input2 : strlen(input2);
        int frac_len1 = delim1 ? strlen(delim1 + 1) : 0;
```

```
        int frac_len2 = delim2 ? strlen(delim2 + 1) : 0;

        int max_whole = (whole_len1 > whole_len2) ? whole_len1 :
whole_len2;
        int max_frac = (frac_len1 > frac_len2) ? frac_len1 :
frac_len2;
        int total_len = max_whole + max_frac + (max_frac ? 1 : 0);

        char **matrix = malloc(2 * sizeof(char *));
        if (!matrix) {
            fprintf(stderr, "Memory allocation failed\n");
            free(input1);
            free(input2);
            return;
        }

        for (int i = 0; i < 2; i++) {
            matrix[i] = calloc(total_len + 1, sizeof(char));
            if (!matrix[i]) {
                for (int j = 0; j < i; j++) free(matrix[j]);
                free(matrix);
                free(input1);
                free(input2);
                fprintf(stderr, "Memory allocation failed\n");
                return;
            }
        }

        for (int i = 0; i < 2; i++) {
            char *src = (i == 0) ? input1 : input2;
            char *delim = (i == 0) ? delim1 : delim2;

            int curr_whole = delim ? delim - src : strlen(src);
            int curr_frac = delim ? strlen(delim + 1) : 0;

            int pad_whole = max_whole - curr_whole;
            memset(matrix[i], '0', pad_whole);
            strncpy(matrix[i] + pad_whole, src, curr_whole);

            if (max_frac > 0) {
                char *dest_comma = matrix[i] + max_whole;
                *dest_comma = ',';

                if (delim) {
                    strncpy(dest_comma + 1, delim + 1, curr_frac);
                    memset(dest_comma + 1 + curr_frac, '0',
max_frac - curr_frac);
                } else {
                    memset(dest_comma + 1, '0', max_frac);
                }
            }
        }
```

```
        switch (choice) {
            case 3:
            case 4:
                addition(matrix);
                break;
            case 6:
            case 7:
                ternary_logic_op(matrix, choice);
                break;
            default:
                fprintf(stderr, "Invalid operation choice\n");
                break;
        }

        for (int i = 0; i < 2; i++) free(matrix[i]);
        free(matrix);
        free(input1);
        free(input2);
    }

    void remove_delimiter(char* str) {
        char* pos = strpbrk(str, ",.");
        if (pos) {
            memmove(pos, pos + 1, strlen(pos));
        }
    }

    void multiplication() {
        char *input1 = NULL, *input2 = NULL;

        printf("Enter first number: ");
        read_input(&input1);
        printf("Enter second number: ");
        read_input(&input2);

        if (!input1 || !input2) {
            fprintf(stderr, "Invalid input\n");
            free(input1);
            free(input2);
            return;
        }

        int len1 = strlen(input1);
        int len2 = strlen(input2);

        char* comma1_pos = strpbrk(input1, ",.");
        char* comma2_pos = strpbrk(input2, ",.");

        int decimal_places1 = comma1_pos ? len1 - (comma1_pos -
input1) - 1 : 0;
```

```
        int decimal_places2 = comma2_pos ? len2 - (comma2_pos -
input2) - 1 : 0;
        int    total_decimal_places    =    decimal_places1    +
decimal_places2;

        remove_delimiter(input1);
        remove_delimiter(input2);

        len1 = strlen(input1);
        len2 = strlen(input2);

        char *inverted = strdup(input1);
        if (!inverted) goto cleanup;

        invert(inverted);

        char *result = calloc(len1 + len2 + 1, 1);
        if (!result) {
            free(inverted);
            goto cleanup;
        }
        memset(result, '0', len1 + len2);

        for (int i = len2 - 1; i >= 0; i--) {
            if (input2[i] == '0') continue;

            const char *source = (input2[i] == '1') ? input1 :
inverted;
            int carry = 0;
            int res_pos = len1 + i;

            for (int j = len1 - 1; j >= 0; j--) {
                int digit = (source[j] == '1') ? 1 : (source[j] ==
'i') ? -1 : 0;
                int current = (result[res_pos] == '1') ? 1 :
                              (result[res_pos] == 'i') ? -1 : 0;

                int sum = current + digit + carry;
                carry = 0;

                if (sum > 1) {
                    sum -= 3;
                    carry = 1;
                } else if (sum < -1) {
                    sum += 3;
                    carry = -1;
                }

                result[res_pos] = (sum == 1) ? '1' : (sum == -1) ?
'i' : '0';
                res_pos--;
            }
```

```
            if (carry != 0) {
                result[res_pos] = (carry == 1) ? '1' : 'i';
            }
        }

        int result_len = strlen(result);
        if (total_decimal_places > 0) {
            if (result_len > total_decimal_places) {
                memmove(result + result_len - total_decimal_places
+ 1,
                        result       +       result_len       -
total_decimal_places,
                        total_decimal_places + 1);
                result[result_len - total_decimal_places] = ',';
            } else {
                char *new_result = malloc(total_decimal_places +
result_len + 3);
                if (new_result) {
                    sprintf(new_result,                  "0,%0*d",
total_decimal_places - result_len, 0);
                    strcat(new_result, result);
                    free(result);
                    result = new_result;
                }
            }
        }

        while (result[0] == '0' && result[1] != ',' && result[1]
!= '\0') {
            memmove(result, result + 1, strlen(result));
        }

        printf("\nMultiplication result: %s\n", result);

        free(inverted);
        free(result);

    cleanup:
        free(input1);
        free(input2);
    }


    void symmetry(char *input, int isNegative) {

        //printf("Non-symmetrical ternary: %s\n", input);

        int len = strlen(input);
        int y = 0;

        for (int i = len - 1; i >= 0; i--) {
```

```
        if (input[i] == ',' || input[i] == '.') {
            continue;
        }

        int x = input[i] - '0';
        int z = y + x;

        if (z <= 1) {
            input[i] = (z == 1) ? '1' : '0';
            y = 0;
        } else if (z == 2) {
            input[i] = 'i';
            y = 1;
        } else if (z == 3) {
            input[i] = '0';
            y = 1;
        }
    }

    if (y != 0) {
        memmove(input + 1, input, len + 1);
        input[0] = '1';
    }

    if (isNegative) {
        invert(input);
    }
    printf("Balanced ternary:   %s\n", input);
}

void whole_part(char *part) {
    if (strlen(part) > MAX_DECIMAL_PLACES) {
        printf("Error: Number too long\n");
        part[0] = '0';
        part[1] = '\0';
        return;
    }

    long long number = strtoll(part, NULL, 10);

    if (number == 0) {
        part[0] = '0';
        part[1] = '\0';
        return;
    }

    int i = 0;
    while (number > 0) {
        part[i++] = '0' + (number % 3);
        number /= 3;
    }
    part[i] = '\0';
```

```
        for (int left = 0, right = i - 1; left < right; left++,
right--) {
            char temp = part[left];
            part[left] = part[right];
            part[right] = temp;
        }
    }

    void fraction_part(char *part) {

        double fractionalValue = atof(part) / pow(10,
strlen(part));
        char buffer[2];
        part[0] = '\0';

        for (int i = 0; i < MAX_DECIMAL_PLACES; i++) {
            fractionalValue *= 3;
            int digit = (int)fractionalValue;
            fractionalValue -= digit;
            sprintf(buffer, "%d", digit);
            strcat(part, buffer);
        }
    }

    void ternary_to_decimal() {
        char *input = NULL;
        printf("\nEnter a ternary number: ");
        read_input(&input);
        if (!input) {
            fprintf(stderr, "Invalid input\n");
            return;
        }

        char *delimiter = strpbrk(input, ".,");
        int wholePartLength = (delimiter) ? (delimiter - input) :
strlen(input);

        double result = 0.0;
        int power = wholePartLength - 1;

        for (int i = 0; input[i] != '\0'; i++) {
            if (input[i] == '.' || input[i] == ',') {
                continue;
            }
            int digit = 0;
            if (input[i] == 'i') digit = -1;
            else if (input[i] == '1') digit = 1;
            else digit = 0;

            result += digit * pow(3.0, power);
            power--;
```

```
        }

        printf("Decimal value: %.10f\n", result);
        free(input);
    }

    void decimal_to_ternary() {
        char *input = NULL;
        printf("\nEnter a decimal number: ");
        read_input(&input);
        if (!input) {
            fprintf(stderr, "Invalid input\n");
            return;
        }

        char *firstPart = NULL;
        char *secondPart = NULL;
        char *combined = NULL;
        int isNegative = 0;

        if (input[0] == '-') {
            isNegative = 1;
            memmove(input, input + 1, strlen(input) + 1);
        }

        char *delimiter = strpbrk(input, ",.");

        size_t whole_len = delimiter ? delimiter - input :
strlen(input);
        size_t max_whole_len = whole_len * 2 + 1;
        size_t max_frac_len = MAX_DECIMAL_PLACES;
        size_t total_len = max_whole_len + max_frac_len + 2;

        firstPart = malloc(max_whole_len);
        secondPart = malloc(max_frac_len);
        combined = malloc(total_len);

        if (!firstPart || !secondPart || !combined) {
            fprintf(stderr, "Memory allocation failed\n");
            goto cleanup;
        }

        if (delimiter) {
            strncpy(firstPart, input, whole_len);
            firstPart[whole_len] = '\0';

            size_t frac_len = strlen(delimiter + 1);
            strncpy(secondPart, delimiter + 1, max_frac_len - 1);
            secondPart[frac_len] = '\0';
        } else {
            strncpy(firstPart, input, max_whole_len - 1);
            strcpy(secondPart, "0");
```

```
        }

        whole_part(firstPart);
        fraction_part(secondPart);

        snprintf(combined,    total_len,    "%s,%s",    firstPart,
secondPart);

        symmetry(combined, isNegative);

    cleanup:
        free(input);
        free(firstPart);
        free(secondPart);
        free(combined);
    }

    int main() {
        int choice;

        do {
            printf("\nMenu: \n");
            printf("1. Convert to balanced ternary\n");
            printf("2. Convert to decimal\n");
            printf("3. Addition\n");
            printf("4. Subtraction\n");
            printf("5. Multiplication\n");
            printf("6. AND\n");
            printf("7. OR\n");
            printf("0. Exit\n");
            printf("Enter your choice: ");

            if (scanf("%d", &choice) != 1) {
                fprintf(stderr, "Invalid input. Exiting.\n");
                break;
            }
            while (getchar() != '\n');

            switch (choice) {
                case 1: decimal_to_ternary(); break;
                case 2: ternary_to_decimal(); break;
                case 3:
                case 4:
                case 6:
                case 7: rationing(choice); break;
                case 5: multiplication(); break;
                case 0: printf("Exiting...\n"); break;
                default: printf("Invalid choice. Try again.\n");
            }
        } while (choice != 0);
        return 0;
    }
```

Додаток Д

Код тестових функцій

```
void test_whole_part() {
    char buf[32];
    // Test 1
    strcpy(buf, "5");
    whole_part(buf);
    printf("\n[Test 1] whole_part(\"5\") -> \"%s\" (expected: \"12\")\n", buf);
    assert(strcmp(buf, "12") == 0);

    // Test 2
    strcpy(buf, "107");
    whole_part(buf);
    printf("[Test 2] whole_part(\"10\") -> \"%s\" (expected: \"10222\")\n", buf);
    assert(strcmp(buf, "10222") == 0);

    // Test 3
    strcpy(buf, "0");
    whole_part(buf);
    printf("[Test 3] whole_part(\"0\") -> \"%s\" (expected: \"0\")\n", buf);
    assert(strcmp(buf, "0") == 0);
    printf("whole_part() passed.\n");
}
```

Рисунок Д.1 - Код тестової функції test_whole_part().

```
void test_fraction_part() {
    char frac[32];

    // Test 1
    strcpy(frac, "25");
    fraction_part(frac);
    printf("\n[Test 1] fraction_part(\"25\") -> \"%s\" (expected: starts with ~ \"020202...\")\n", frac);
    assert(strcmp(frac, "020202020202020") == 0);
    // Test 2
    strcpy(frac, "5");
    fraction_part(frac);
    printf("[Test 2] fraction_part(\"5\") -> \"%s\" (expected: starts with ~ \"111111...\")\n", frac);
    assert(strcmp(frac, "111111111111111") == 0);
    // Test 3
    strcpy(frac, "33");
    fraction_part(frac);
    printf("[Test 3] fraction_part(\"33\") -> \"%s\" (expected: starts with ~ \"022220...\")\n", frac);
    assert(strcmp(frac, "022220120101112") == 0);
    printf("fraction_part() passed.\n");
}
```

Рисунок Д.2 - Код тестової функції test_fraction_part().

```c
void test_symmetry() {
    // Test 1
    char val1[] = "22";
    printf("symmetry(\"22\", 0): expected: \"10i\"\nActual:\n");
    symmetry(val1, 0);
    assert(strcmp(val1, "10i") == 0);

    // Test 2
    char val3[] = "0,2";
    printf("symmetry(\"0,2\", 0): expected: \"1,i\"\nActual:\n");
    symmetry(val3, 0);
    assert(strcmp(val3, "1,i") == 0);
    printf("symmetry() passed.\n");
}
```

Рисунок Д.3 - Код тестової функції test_symmetry().

```c
void test_invert() {
    // Test 1
    char val1[] = "1i0";
    invert(val1);
    printf("invert(\"1i0\") -> \"%s\" (expected: \"i10\")\n", val1);
    assert(strcmp(val1, "i10") == 0);

    // Test 2
    char val2[] = "111";
    invert(val2);
    printf("invert(\"111\") -> \"%s\" (expected: \"iii\")\n", val2);
    assert(strcmp(val2, "iii") == 0);

    // Test 3
    char val3[] = "0i1";
    invert(val3);
    printf("invert(\"0i1\") -> \"%s\" (expected: \"01i\")\n", val3);
    assert(strcmp(val3, "01i") == 0);

    printf("invert() passed.\n");
}
```

Рисунок Д.4 - Код тестової функції test_invert().

```c
void test_addition() {
    char **matrix = malloc(2 * sizeof(char*));
    matrix[0] = malloc(2);
    matrix[1] = malloc(2);

    // Test 1
    strcpy(matrix[0], "1i");
    strcpy(matrix[1], "i1");
    printf("\n[Test 1] addition([\"1i\", \"i1\"]) -> expected: 0 ");
    addition(matrix);

    // Test 2
    strcpy(matrix[0], "11");
    strcpy(matrix[1], "11");
    printf("\n[Test 2] addition([\"11\", \"11\"]) -> expected: 10i ");
    addition(matrix);

    // Test 3
    strcpy(matrix[0], "ii");
    strcpy(matrix[1], "ii");
    printf("\n[Test 3] addition([\"ii\", \"ii\"]) -> expected: i01 ");
    addition(matrix);

    free(matrix[0]);
    free(matrix[1]);
    free(matrix);
    printf("addition() completed.\n");
}
```

Рисунок Д.5 - Код тестової функції test_addition().

```c
void test_ternary_logic_op() {
    char **matrix = malloc(2 * sizeof(char*));
    matrix[0] = malloc(3);
    matrix[1] = malloc(3);
    // Test 1
    strcpy(matrix[0], "1i0");
    strcpy(matrix[1], "i10");
    printf("\n[Test 1 - AND] Inputs: \"1i0\", \"i10\" -> Expected: ii0 ");
    ternary_logic_op(matrix, 6);
    printf("[Test 1 - OR]  Expected: 110");
    ternary_logic_op(matrix, 7);
    free(matrix[0]);
    free(matrix[1]);
    free(matrix);
    printf("ternary_logic_op() completed (check results visually).\n");
}
```

Рисунок Д.6 - Код тестової функції test_ternary_logic_op().