

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.021

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«__» _____ 2025 р.

Магістерська дисертація

на здобуття ступеня магістра

**за освітньо-професійною програмою «Інженерія програмного
забезпечення інформаційних систем»**

зі спеціальності 121 «Інженерія програмного забезпечення»

**на тему: «Програмний комплекс підтримки інженерно-наукових
розрахунків в галузі машинобудування»**

Виконав (-ла):

студент (-ка) II курсу, групи ІІ-341мп

Кононов Максим Анатолійович _____

Керівник:

доцент кафедри ІІІ, к.т.н., доц.,

Лісовиченко Олег Іванович _____

Рецензент:

професор каф. ІСТ, д.т.н.,

Корнага Ярослав Ігорович _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент (-ка) _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«__» _____ 2025р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Кононову Максиму Анатолійовичу

1. Тема дисертації «Програмний комплекс підтримки інженерно-наукових розрахунків в галузі машинобудування», науковий керівник дисертації Лісовиченко Олег Іванович, к.т.н., доц., затверджені наказом по університету від «06» листопада 2025 р. № 4842-с
2. Термін подання студентом дисертації «15» грудня 2025 р.
3. Об'єкт дослідження – програмне забезпечення для виконання інженерно-наукових розрахунків
4. Предмет дослідження – моделі побудови програмного забезпечення для виконання інженерно-наукових розрахунків в галузі машинобудування, алгоритми довгої арифметики та методи їх тестування
5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; розробка та тестування програмного забезпечення; дослідження ефективності розробленого програмного забезпечення.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 3-4 плакати
7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

8.Дата видачі завдання «1» вересня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Видача завдання	1.09.2025	
2	Визначення структури магістерської дисертації	5.09.2025	
3	Аналіз існуючих методів та програмних засобів	8.09.2025	
4	Проектування програмного забезпечення	26.09.2025	
5	Реалізація моделі побудови основних алгоритмів	13.10.2025	
6	Розробка програмного забезпечення	24.10.2025	
7	Виконання експериментальних досліджень	15.11.2025	
8	Оформлення пояснювальної записки	26.11.2025	
9	Подання дисертації на попередній захист	26.11.2025	
10	Подання дисертації на захист	15.12.2025	

Студент

Максим КОНОНОВ

Науковий керівник

Олег ЛІСОВИЧЕНКО

РЕФЕРАТ

Розмір пояснювальної записки – 174 аркушів, містить 13 ілюстрацій, 18 таблиць, 3 додатки, 52 посилання на джерела.

Актуальність теми. Підприємства дуже часто стикаються з багатьма обчислювальними задачами, природа яких багатогранна й різноманітна. Існуючі програмні продукти є або досить вимогливими до апаратних ресурсів комп'ютера, або вони потребують навчання персоналу і мають надмірну функціональність, або їх недостатньо для вирішення багатьох специфічних задач в машинобудуванні. Стандартних можливостей мов програмування часто не вистачає, щоб оперувати числами великої довжини. Це ускладнює задачі створення специфічних обчислювальних програмних продуктів. В свою чергу чисельні методи вимагають використання чисел великої розрядності та керованої точності обчислень. У зв'язку з цим особливої актуальності набувають рішення, які працюють з довгою арифметикою, забезпечуючи підтримку обчислень із практично необмеженою кількістю цифр. Деякі розробки не є оптимальними в плані швидкодії або мають обмежену функціональність. Виявлено потребу в створенні програмного забезпечення, яке якісно вирішує обчислювальні задачі з регульованою точністю.

Мета дослідження: покращення швидкодії, ефективності та зручності програмних засобів для виконання інженерно-наукових розрахунків у галузі машинобудування.

Об'єкт дослідження: програмне забезпечення для виконання інженерно-наукових розрахунків.

Предмет дослідження: моделі побудови програмного забезпечення для виконання інженерно-наукових розрахунків в галузі машинобудування, алгоритми довгої арифметики та методи їх тестування.

Для реалізації поставленої мети **сформульовані наступні завдання:**

- створити програмне забезпечення для вирішення деяких обчислювальних задач з машинобудування, зокрема обчислення ваги металопрокату, ваги або довжини троса, розривного зусилля та діаметра каната;
- розробити власні рішення в галузі довгої арифметики: функції для виконання основних арифметичних операцій, а також генератори випадкових наддовгих цілих та дробових чисел у заданому діапазоні або із заданою довжиною, створити рішення для генерування та обчислення випадкових математичних виразів із наддовгими числами;
- перевірити швидкодію та довести вірність результатів обчислень;
- виконати порівняльний аналіз ефективності власних рішень з популярною бібліотекою GNU Multiple Precision Arithmetic Library, використовуючи форки MPIR та MPFR;
- за допомогою сторонніх бібліотек створити більш точний аналог програмного засобу для обчислення степеня наддовгого числа;
- вирішити проблеми кросплатформеності та інтеграції нашої бібліотеки в інші програмні продукти.

Наукова новизна результатів магістерської дисертації полягає в тому, було забезпечено відсутність помилок у власній реалізації рішень на базі довгої арифметики за рахунок формалізованих методів перевірки коректності обчислень. Запропоновано метод комбінованого функціонально-продуктивнісного тестування різних бібліотек для роботи з обчислювальними інструментами, в результаті чого отримано нові експериментальні дані порівняння швидкодії. Розроблено комплексний підхід до реалізації довгої арифметики у множинних системах числення. Набула подальшого розвитку архітектура інтеграції обчислювального ядра з модулями предметно-орієнтованих розрахунків та методологія розробки галузево-орієнтованих обчислювальних систем з відкритою архітектурою.

Практичне значення отриманих результатів полягає в тому, що створений програмний продукт сприяє підвищенню продуктивності роботи інженерів та скороченню виробничих витрат, знижує потребу в більш

складних рішеннях, масштабується, містить функції регулювання точності та доведення вірності результатів обчислень.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на державному підприємстві «Суднобудівний завод імені 61 комунара» (належить до концерну «Укроборонпром»), що підтверджується відповідною довідкою про впровадження.

Ключові слова: ДОВГА АРИФМЕТИКА, GMP, ВИПАДКОВІ ЧИСЛА, СИСТЕМИ ЧИСЛЕННЯ, ІНЖЕНЕРНІ РОЗРАХУНКИ, МАШИНОБУДУВАННЯ, КАНАТ, МЕТАЛОПРОКАТ

ABSTRACT

Explanatory note size – 174 pages, contains 13 illustrations, 18 tables, 3 applications, 52 references.

Topicality. Enterprises often face many computational tasks, the nature of which is multifaceted and diverse. Existing software products are either quite demanding on computer hardware resources, or they require staff training and have excessive functionality, or they are insufficient for solving many specific tasks in mechanical engineering. The standard capabilities of programming languages are often insufficient to operate with long numbers. This complicates the task of creating specific computational software products. In turn, numerical methods require the use of large-precision numbers and controlled calculation accuracy. In this regard, solutions that work with long arithmetic, providing support for calculations with a virtually unlimited number of digits, are particularly relevant. Some tools are not optimal in terms of performance or have limited functionality. There is a need for software that can efficiently solve computational problems with adjustable precision.

The aim of the study: to improve the performance, efficiency, and usability of software tools for performing engineering and scientific calculations in the field of mechanical engineering.

The object of research: software for performing engineering and scientific calculations.

The subject of research: models for building software for performing engineering and scientific calculations in the field of mechanical engineering, long arithmetic algorithms and methods for testing them.

To achieve the stated goal, the **following tasks** have been formulated:

- to create software for solving certain computational problems in mechanical engineering, in particular, calculating the weight of rolled metal, the weight or length of a wire, the breaking force and diameter of a rope;
- develop own solutions in the field of long arithmetic: functions for performing basic arithmetic operations, as well as generators of random extra-long integers

and fractional numbers in a given range or with a given length, create solutions for generating and calculating random mathematical expressions with extra-long numbers;

- check the speed and prove the accuracy of the calculation results;
- perform a comparative analysis of the effectiveness of your own solutions with the popular GNU Multiple Precision Arithmetic Library, using MPIR and MPFR forks;
- using third-party libraries, create a more accurate analogue of a software tool for calculating the power of an extra-long number;
- solve the problems of cross-platform compatibility and integration of our library into other software products.

The scientific novelty of the results of the master's thesis lies in the fact that the absence of errors in our own implementation of solutions based on long arithmetic was ensured by formalised methods of verifying the correctness of calculations. A method of combined functional and performance testing of various libraries for working with computational tools was proposed, resulting in new experimental data on performance comparison. A comprehensive approach to the implementation of long arithmetic in multiple number systems was developed. The architecture for integrating the computational core with subject-oriented calculation modules and the methodology for developing industry-oriented computational systems with open architecture have been further developed.

The practical value of the results obtained lies in the fact that the created software product contributes to increasing the productivity of engineers and reducing production costs, it reduces the need for more complex solutions, it is scalable, and contains functions for regulating accuracy and proving the correctness of the calculation results.

Relationship with working with scientific programs, plans, topics. The work was performed at the Department of Computer Science and Software Engineering of the National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute».

Approbation. The scientific provisions of the dissertation were tested at the state-owned enterprise '61 Communards Shipbuilding Plant' (the part of the Ukroboronprom concern), as confirmed by the relevant certificate of implementation.

Keywords: LONG ARITHMETIC, GMP, RANDOM NUMBERS, NUMBER SYSTEMS, ENGINEERING CALCULATIONS, MACHINE BUILDING, ROPE, METAL ROLLING

ЗМІСТ

ВСТУП	14
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ	16
1.1 Систематичний огляд літератури	16
1.2 Огляд існуючих програмно-апаратних засобів для вирішення різноманітних задач у машинобудуванні	32
1.3 Рішення виробничих задач.....	39
1.4 Онлайн-системи інженерних розрахунків.....	42
1.5 Обґрунтування потреби в новому програмному продукті	45
Висновки до розділу	46
РОЗДІЛ 2. ПРОГРАМНІ ТА АПАРАТНІ ЗАСОБИ ДЛЯ РОЗРОБКИ І ФУНКЦІОНУВАННЯ ПЗ.....	49
2.1. Програмно-апаратні вимоги.....	49
2.2 Середовище розробки, платформа та мова програмування	51
2.3 Програми для додавання і редагування довідкових даних.....	54
Висновки до розділу	56
РОЗДІЛ 3. ПРОГРАМНИЙ КОМПЛЕКС ВИКОНАННЯ ІНЖЕНЕРНИХ РОЗРАХУНКІВ	60
3.1 Принципи роботи програми та інтерфейс користувача	60
3.2 Модуль виконання арифметичних операцій	64
3.2.1. Допоміжні функції калькулятора	66
3.2.2 Арифметичні операції додавання та віднімання.....	69

3.2.3 Арифметичні операції множення та ділення.....	73
3.2.4. Обробка арифметичних виразів.....	80
3.3 Конвертування чисел з однієї системи числення в іншу	82
3.4 Модуль розрахунку ваги металопрокату	86
3.5 Модулі розрахунків ваги, довжини діаметра та розривної сили каната або троса.....	88
Висновки до розділу	91
РОЗДІЛ 4. БІБЛІОТЕКИ ДЛЯ РОБОТИ З ДОВГОЮ АРИФМЕТИКОЮ.....	98
4.1 Загальна структура проекту для роботи з довгою арифметикою	98
4.2 Генератори випадкових наддовгих чисел.....	102
4.3 Арифметичні операції над наддовгими числами.....	105
4.3.1 Операція додавання.....	106
4.3.2 Операція віднімання	107
4.3.3 Операція множення.....	109
4.3.4 Операція ділення	112
4.4 Основні арифметичні операції бібліотеки MPIR.....	114
4.5 Обчислення степеня за допомогою бібліотек MPIR та MPFR	116
4.5.1 Порівняння розробленого рішення з іншими реалізаціями.....	118
4.6 Робота з арифметичними виразами на базі довгої арифметики.....	121
4.7 Тестування рішень для роботи з довгою арифметикою	124
4.7.1 Створення тестових пар.....	124
4.7.2 Тестування арифметичних операцій	126

4.7.2.1 Перевірка вірності результату операції ділення	128
4.7.2.2 Результати тестування основних арифметичних дій	130
4.7.3 Тестування операції піднесення до степеня	137
4.7.4 Перевірка роботи аналізатора арифметичних виразів	141
4.8 Інтеграція рішень для роботи з довгою арифметикою в інший програмний комплекс	146
4.8.1 Загальні правила міжмовної взаємодії	146
4.8.2 Використання створених рішень у вигляді динамічної бібліотеки	149
Висновки до розділу	154
ВИСНОВКИ	156
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	159
ДОДАТОК А: Довідка про впровадження	163
ДОДАТОК Б: Копії графічних матеріалів	164
ДОДАТОК В: Код програми	167

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ГЗ – глобальна змінна;

ЛЗ – локальна змінна;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

СЧ – система числення;

GMP – GNU Multiple Precision Arithmetic Library – Бібліотека арифметики довільної точності;

MPIR – Multiple Precision Integers and Rationals – Бібліотека високоточної арифметики для цілих і раціональних чисел;

MPFR – Multiple Precision Floating-Point Reliable Library – Надійна бібліотека високоточної арифметики з плаваючою комою.

ВСТУП

Планування матеріальних ресурсів та виконання високоточних інженерних розрахунків є однією з ключових передумов ефективного функціонування сучасного виробничого підприємства. Від точності та швидкості обчислень безпосередньо залежать якість готової продукції, її собівартість і конкурентоспроможність, рівень задоволеності замовників, а також загальна економічна стійкість і перспективи розвитку підприємства. Помилки або затримки на етапі розрахунків призводять до перевитрат матеріалів, збільшення тривалості виробничих циклів, фінансових збитків і, у критичних випадках, до зриву виконання виробничих завдань.

Аналіз сучасного стану проблеми показує, що на багатьох вітчизняних, зокрема державних, машинобудівних і металообробних підприємствах питання автоматизації та оптимізації інженерних розрахунків залишається недостатньо опрацьованим. Використовуване програмне забезпечення часто ґрунтується на застарілих підходах, має обмежені можливості роботи з числами великої розрядності та не забезпечує належної точності.

Водночас **світові тенденції** розвитку інженерного програмного забезпечення свідчать про активне впровадження високоточних обчислень, спеціалізованих бібліотек довгої арифметики та модульних програмно-апаратних комплексів, які можуть інтегруватися у вже наявні інформаційні системи підприємства. Провідні програмні продукти і бібліотеки на основі GMP, такі як MPIR та MPFR, широко застосовуються у наукових і промислових розрахунках, однак їх безпосереднє використання в інженерних прикладних системах часто ускладнене складністю інтеграції, відсутністю зручного інтерфейсу. Це створює розрив між наявними теоретичними та програмними напрацюваннями і реальними потребами інженерів-практиків.

На відміну від державних підприємств, малі та середні приватні виробництва, що працюють в умовах жорсткої конкуренції, змушені активно впроваджувати сучасні засоби автоматизації інженерних розрахунків. Для них швидкість, точність і надійність обчислень є критичними факторами

комерційного успіху. Проте навіть у цьому секторі часто відсутні універсальні рішення, які б поєднували зручність використання, можливість роботи з арифметикою довільної складності та інтеграцію в інші рішення.

Актуальність даної магістерської дисертації зумовлена необхідністю подолання зазначених обмежень шляхом розробки програмного комплексу підтримки інженерних розрахунків, здатного якісно працювати з числами великої довжини, а також діяти у різних системах числення, обробляти арифметичні вирази та легко інтегруватися в інші продукти. Особлива увага в роботі приділяється використанню можливостей сучасних бібліотек довгої арифметики, розробці засобів тестування, а також міжмовній інтеграції рішень на мовах C/C++ у прикладні системи в середовищі Lazarus.

Об'єкт дослідження: програмне забезпечення для виконання інженерно-наукових розрахунків.

Предмет дослідження: моделі побудови програмного забезпечення для виконання інженерно-наукових розрахунків в галузі машинобудування, алгоритми довгої арифметики та методи їх тестування.

Метою магістерської дисертації є розробка програмного комплексу підтримки інженерних розрахунків для машинобудівних підприємств, який забезпечує виконання високоточних арифметичних операцій над наддовгими числами, автоматизацію інженерних обчислень і можливість практичного використання у реальних виробничих умовах. Галуззю застосування результатів роботи є машинобудування та суміжні галузі промисловості, де необхідні точні й надійні інженерні розрахунки.

Дослідження, проведені у даній роботі, **тісно пов'язані** з існуючими науковими та прикладними розробками у сфері довгої арифметики, чисельних методів та інженерного програмного забезпечення. Робота узагальнює та розвиває ідеї, закладені у відомих бібліотеках для високоточних обчислень, доповнюючи їх власними алгоритмічними рішеннями, засобами тестування та практичною інтеграцією у прикладний програмний комплекс, орієнтований на потреби інженерів-практиків.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

У даний момент на більшості українських металообробних підприємствах інженерні розрахунки виконуються за допомогою електронної обчислювальної техніки та документації у паперовій і електронній формах. Розвинені активні підприємства здійснюють більшість обчислень та креслень за допомогою програм автоматизованого проектування таких як Mathcad, AutoCAD, Autodesk Inventor. Проте дуже часто виникають задачі, які не зручно розв'язувати за допомогою спеціалізованих програм. Офісні пакети, такі як Microsoft Office та LibreOffice, частково вирішують дану проблему, але не надають достатньої зручності, мобільності та інтерактивності. У таких випадках розвинуті підприємства звертаються в аутсорсингові компанії або власними силами розробляють необхідне програмне забезпечення. На жаль, на деяких нестабільно працюючих заводах ситуація з автоматизацією інженерних розрахунків дещо гірша.

1.1 Систематичний огляд літератури

Першим кроком систематичного огляду літератури є визначення обсягу дослідження. Для цього необхідно сформулювати дослідницькі питання (ДП).

Предметом дослідження є методи, моделі, та засоби побудови програмного забезпечення для виконання інженерно-наукових розрахунків в машинобудуванні. Таким чином, необхідно проаналізувати існуючі засоби вирішення математичних задач у даній галузі, а також виявити особливості розробки нового програмного забезпечення для виконання обчислень в машинобудівельній галузі. Також слід визначити доцільність застосування алгоритмів довгої арифметики та генерації довгих випадкових чисел з метою тестування новостворених систем. Для цього були сформульовані дослідницькі питання, які показані в таблиці 1.1.1.

Таблиця 1.1.1 – Дослідницькі питання

ДП	Питання
ДП 1	Які існують методи та засоби виконання інженерно-наукових розрахунків?
ДП 2	У чому полягає практична цінність використання алгоритмів довгої арифметики для виконання інженерно-наукових розрахунків у машинобудуванні?
ДП 3	Які інструменти та архітектурні рішення застосовуються при розробці програмних комплексів для виконання інженерно-наукових розрахунків в машинобудуванні?
ДП 4	Яку роль відіграють алгоритми генерації довгих випадкових чисел у заданому діапазоні при розробці програмного забезпечення для виконання інженерно-наукових розрахунків?
ДП 5	Які новітні інструменти та технології можуть бути використані при розробці програмного забезпечення для виконання інженерно-наукових розрахунків?

Для дослідження публікацій було виконано електронний пошук у IEEE Xplore, Google Scholar та ACM Digital Library. Під час пошуку застосовувався розширений пошук (advanced search) з використанням булевих операторів OR та AND, а також головних термінів та синонімів. Сформовані пошукові стрічки (ПС) можна побачити в таблиці 1.1.2.

Таблиця 1.1.2 – Форми пошукових стрічок

ПС	Пошукова стрічка
ПС 1	(“engineering calculations” OR “scientific computing” OR “інженерні розрахунки”) AND (“software complex” OR “програмний комплекс”) AND (“mechanical engineering” OR “машинобудування”)

Продовження таблиці 1.1.2

ПС 2	(“arbitrary precision arithmetic” OR “multiple precision” OR “довга арифметика”) AND (“engineering software” OR “інженерне програмне забезпечення”)
ПС 3	(“numerical methods” OR “чисельні методи”) AND (“high precision” OR “висока точність”) AND (“mechanical design” OR “машинне проектування”)
ПС 4	(“software architecture” OR “архітектура програмного забезпечення”) AND (“computational engineering” OR “обчислювальна інженерія”)
ПС 5	(“BigInteger” OR “BigDecimal” OR “MPFR” OR “GMP”) AND “engineering applications”
ПС 6	(“precision arithmetic algorithms” OR “алгоритми точної арифметики”) AND (“CAD” OR “CAE” OR “FEA”)

Використання пошукових стрічок, наведених в таблиці 1.1.2, може призвести до отримання великої кількості публікацій. Для зменшення кількості знайдених публікацій були застосовані критерії включення (IC) та виключення (EC), які можна побачити в таблицях 1.1.3 та 1.1.4 відповідно.

Таблиця 1.1.3 – Критерії включення

IC	Критерій
IC 1	Тільки англомовні та україномовні дослідження
IC 2	Дослідження, що стосуються розробки програмного забезпечення для інженерно-наукових розрахунків
IC 3	Дослідження, що містять інформацію про алгоритми довгої арифметики або високоточні обчислення
IC 4	Дослідження з датою публікації від 2015 року
IC 5	Дослідження, що стосуються машинобудування або суміжних галузей

Таблиця 1.1.4 – Критерії виключення

ЕС	Критерій
ЕС 1	Повторювані дослідження (необхідно відфільтрувати дублікати)
ЕС 2	Дослідження з подібними результатами (необхідно розглядати лише найбільш повне дослідження)
ЕС 3	Дослідження, до яких не вдалося отримати доступ
ЕС 4	Дослідження, що не містять практичних результатів або експериментальних даних

Нефільтровані результати пошуку наведено в таблиці 1.1.5.

Таблиця 1.1.5 – Нефільтровані результати пошуку

Пошукова стрічка	Кількість знайдених публікацій
ПС 1	145
ПС 2	89
ПС 3	167
ПС 4	234
ПС 5	56
ПС 6	78

Після застосування критеріїв включення та виключення були обрані публікації [1-52]. Далі були виділені категорії публікацій, які можна побачити в таблиці 1.1.6.

Таблиця 1.1.6 – Категорії публікацій

Категорія	Класифікаційний опис
Validation research	Досліджені методи є новими та ще не були впроваджені на практиці, але можуть бути використані, наприклад, для експериментів в лабораторії

Продовження таблиці 1.1.6

Solution proposal	Запропоновано рішення проблеми, яке може бути або новим, або являти собою розширення існуючого методу, при цьому потенційні переваги та придатність рішення підкріплені прикладом або аргументацією
Philosophical papers	Запропоновано новий спосіб розгляду існуючих понять шляхом структурування галузі у формі таксономії або концептуальної основи
Opinion papers	Виражені особисті думки авторів про те, чи є певна методика хорошою чи поганою, або як все повинно бути зроблено

Також були виділені галузі, які можна побачити в таблиці 1.1.7.

Таблиця 1.1.7 – Галузі досліджень

Назва галузі	Опис
Методи побудови програмного забезпечення	Публікації, в яких розглядаються методології та підходи до розробки програмного забезпечення для інженерно-наукових розрахунків
Алгоритми довгої арифметики	Публікації, присвячені розробці та оптимізації алгоритмів високоточної арифметики
Архітектурні рішення	Публікації, що описують архітектурні патерни та рішення для програмних комплексів інженерних розрахунків
Засоби розробки	Публікації, в яких представлені інструменти, бібліотеки та фреймворки для створення інженерного програмного забезпечення
Застосування в машинобудуванні	Публікації, що демонструють практичне застосування програмних комплексів у галузі машинобудування

Результати пошуку за категоріями (К) наведено в таблиці 1.1.8.

Таблиця 1.1.8 – Публікації за категоріями досліджень

	Категорія	Публікації	Загальна кількість
К 1	Validation Research	38, 17, 26, 7, 13, 18, 24, 25	8
К 2	Solution Proposal	3, 8, 14, 46, 29, 36, 39, 42, 45, 23, 52, 37	12
К 3	Philosophical Papers	30, 33, 10, 41, 19, 22, 27, 31, 35, 34, 11, 15, 21, 49, 28	15
К 4	Opinion Papers	1, 4, 5, 12, 16, 43, 20, 48, 50, 51, 2, 6, 9, 40, 44, 47, 32	17

Результати пошуку по категоріям за галузями (Г) видно в таблиці 1.1.9.

Таблиця 1.1.9 – Публікації за галузями досліджень

	Назва галузі	Публікації	Загальна кількість
Г 1	Методи побудови програмного забезпечення	30, 33, 5, 10, 41, 16, 19, 22, 20, 27, 26, 35, 34	13
Г 2	Алгоритми довгої арифметики	1, 8, 14, 17, 18, 26, 29, 36, 7, 39, 13, 18	12
Г 3	Архітектурні рішення	3, 8, 43, 20, 50, 6, 11, 42, 45, 23	10
Г 4	Засоби розробки	5, 12, 51, 2, 9, 40, 15, 21, 49, 49, 28	11
Г 5	Застосування в машинобудуванні	44, 47, 32, 24, 37, 25	6

Частоту публікацій за категоріями та галузями видно в таблиці 1.1.10.

Таблиця 1.1.10 – Частота публікацій за категоріями та галузями

Категорія	Г 1	Г 2	Г 3	Г 4	Г 5
К 1	2	3	1	1	1
К 2	3	4	2	2	1
К 3	5	3	3	3	1
К 4	3	2	4	5	3

Візуалізацію результатів таблиці 1.1.10 можна побачити на рис. 1.1.1

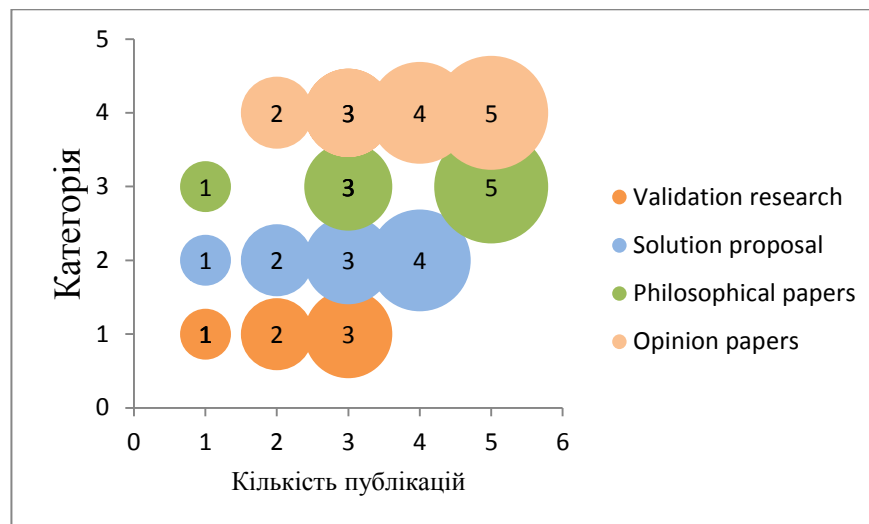


Рисунок 1.1.1 – Бульбашкова діаграма

Обговорення

Інженерно-наукові розрахунки є процесом використання математичних методів та обчислювальних алгоритмів для розв'язання складних науково-технічних задач. Особливістю таких розрахунків є потреба у високій точності обчислень, обробці великих масивів даних та використанні складних математичних моделей. Враховуючи зростаючу складність сучасних наукових та інженерних задач, а також потребу в підвищенні точності розрахунків, для розробки програмного забезпечення в цій сфері потрібні спеціалізовані підходи та технології.

Вирішення математичних задач є важливою частиною сучасних обчислювальних технологій, які використовуються для вирішення складних

задач у фізиці, інженерії, математиці та інших сферах. Вони включають в себе чисельні методи, паралельні обчислення, довгу арифметику та високопродуктивні обчислення (HPC). Довга арифметика, зокрема, відіграє ключову роль у задачах, які потребують високої точності обчислень, таких як криптографія, чисельне моделювання та обробка великих чисел. На основі аналізу наданої літератури розглянемо основні рішення, методи, моделі та інструменти розробки програмного забезпечення для вирішення цих задач, а також оцінимо їх доцільність.

Чисельні методи є основою інженерно-наукових розрахунків. Вони включають методи розв'язання систем лінійних рівнянь, диференціальних рівнянь, оптимізації та інтеграції. Проаналізувавши описану літературу, можемо виділити такі ключові рішення:

Бібліотеки чисельних методів:

1. бібліотеки BLAS і LAPACK [42]: забезпечують оптимізовані реалізації операцій лінійної алгебри, такі як множення матриць і розв'язання систем рівнянь. Вони широко використовуються в задачах чисельного моделювання та є стандартом для високопродуктивних обчислень;
2. бібліотеки ARPREC та MPFR [3, 16]: призначені для роботи з довгою арифметикою з довільною точністю. ARPREC підтримує високоточні обчислення для задач математичної фізики, тоді як MPFR забезпечує коректне округлення для чисел з плаваючою комою. Вони особливо корисні для задач, де потрібна висока точність, наприклад, у криптографії чи моделюванні динамічних систем;
3. TensorFlow [1]: платформа для машинного навчання, яка підтримує чисельні обчислення на GPU та TPU. Вона ефективна для обробки великих масивів даних і розв'язання задач, які включають нейронні мережі або оптимізацію;
4. Julia [5]: мова програмування, спеціально розроблена для вирішення математичних задач. Поєднує простоту Python з продуктивністю C++, що

робить її ідеальною для швидкого прототипування та виконання складних обчислень;

Методи чисельного аналізу:

- Методи розв'язання диференціальних рівнянь [33, 8]: включають методи скінченних елементів, скінченних різниць і методи декомпозиції доменів [34]. Ці методи використовуються для моделювання фізичних процесів, таких як теплопровідність чи гідродинаміка;
- Фізико-інформовані нейронні мережі [35]: новітній підхід, який поєднує в собі чисельні методи з машинним навчанням для розв'язання нелінійних диференціальних рівнянь.

Високопродуктивні обчислення є критично важливими для обробки великих обсягів даних і виконання складних розрахунків у реальному часі. Основні рішення на основі паралельного програмування:

- OpenMP [10]: бібліотека для паралельного програмування на багатоядерних процесорах. Вона проста у використанні і підходить для задач із спільною пам'яттю;
- MPI [18]: інтерфейс для передачі повідомлень, який використовується в розподілених системах. MPI є стандартом для кластерних обчислень і підходить для масштабних задач;
- Charm++ і AMPI [2]: фреймворки для програмування петасекундних додатків. Вони дозволяють адаптивно розподіляти обчислення між вузлами кластера, що ідеально підходить для динамічних задач.
- CUDA і GPU-програмування [23]: використання графічних процесорів для паралельних обчислень значно прискорює обробку великих масивів даних, наприклад, у машинному навчанні чи обробці зображень;
- Екзафлопсні обчислення [14]: сучасні дослідження спрямовані на створення суперкомп'ютерів, здатних виконувати 10^{18} операцій за

секунду. Це вимагає нових алгоритмів і архітектур, таких як гібридні CPU-GPU системи;

- Хмарні технології [49]: хмарні платформи, такі як AWS, Azure або Google Cloud, забезпечують доступ до потужних обчислювальних ресурсів без необхідності локального обладнання. Вони ідеально підходять для масштабування обчислень.

Довга арифметика використовується для обробки чисел із великою кількістю цифр, що перевищують стандартні типи даних (int, double). Основні алгоритми та інструменти:

- Алгоритм Карацуби [21]: зменшує складність множення великих чисел з $O(n^2)$ до $O(n \log n)$. Використовується в криптографії та чисельних обчисленнях;
- Алгоритм Шьонгаге-Штрассена [37]: базується на швидкому перетворенні Фур'є (FFT) і забезпечує ще більшу ефективність для дуже великих чисел;
- Ділення з використанням множення [17]: оптимізація ділення шляхом заміни його на множення з оберненим числом.

Бібліотеки для довгої арифметики:

- GMP (GNU Multiple Precision): популярна бібліотека для високоточних обчислень, яка використовується в багатьох наукових і криптографічних додатках;
- MPFR [16]: забезпечує коректне округлення для чисел із плаваючою комою довільної точності;
- ARPREC [3]: спеціалізована бібліотека для високоточних обчислень у математичній фізиці;
- криптографічні застосування [48]: довга арифметика є основою для алгоритмів RSA, ECC та інших криптографічних систем, де потрібні

операції з великими числами.

Псевдовипадкові числа є важливими для моделювання, криптографії та чисельних методів. Основні рішення:

Mersenne Twister [30]: високоякісний генератор псевдовипадкових чисел із довгим періодом ($2^{19937-1}$). Широко використовується в наукових обчисленнях;

Методи Монте-Карло [12, 28]: використовуються для моделювання складних систем, де потрібні випадкові числа з певним розподілом;

Криптографічні генератори [15]: генератори, такі як Fortuna або Yarrow, забезпечують криптографічно стійкі псевдовипадкові числа для безпеки.

Також було проаналізовано методи, моделі та інструменти розробки ПЗ для інженерно-наукових розрахунків. Методи розробки ПЗ:

- Чисельні алгоритми [19, 40]: розробка ПЗ для чисельних розрахунків вимагає використання стабільних і точних алгоритмів. Наприклад, методи стабілізації чисельних алгоритмів [20] допомагають уникнути помилок округлення;
- Паралельні алгоритми [50]: розробка паралельних алгоритмів для НРС включає декомпозицію задач, балансування навантаження та синхронізацію потоків;
- Фізико-інформовані методи [35]: інтеграція фізичних законів у нейронні мережі для підвищення точності обчислень.

Моделі розробки ПЗ:

- Agile та DevOps [47]: гнучкі методології дозволяють швидко адаптувати ПЗ до змін у вимогах. DevOps забезпечує інтеграцію розробки, тестування та розгортання;
- Модульна архітектура: використання бібліотек, таких як BLAS, LAPACK або TensorFlow, дозволяє створювати модульні додатки, де окремі

компоненти можна замінювати або оновлювати;

- Контейнеризація [6]: Docker забезпечує відтворюваність обчислень, що є критично важливим для наукових досліджень. Jupyter Notebooks [25] дозволяють створювати інтерактивні робочі середовища для аналізу даних.

3. Інструменти розробки. Мови програмування:

- Julia [5]: надає можливість швидкого прототипування обчислень завдяки простому синтаксису та високій продуктивності;
- Python: популярний завдяки бібліотекам NumPy, SciPy та TensorFlow. Підходить для швидкої розробки та інтеграції з ML;
- C++ і Fortran: використовуються для високопродуктивних обчислень завдяки низькорівневій оптимізації.
- Rust [24]: забезпечує безпеку пам'яті та високу продуктивність, що робить його перспективним для HPC.

Інструменти візуалізації:

- Three.js [9]: бібліотека для 3D-візуалізації результатів обчислень;
- VTK [38]: інструмент для обробки та візуалізації складних наукових даних.

Інтерактивні середовища:

- Jupyter Notebooks [25]: для прототипування, аналізу даних і документування обчислень.
- MATLAB і Mathematica: традиційні інструменти для виконання обчислень, хоча вони менш гнучкі порівняно з Julia чи Python.

На рис. 1.1 можна побачити, що більше половини знайдених досліджень за галуззю теоретичних основ інженерно-наукових розрахунків

присвячені розгляду та структуризації існуючих понять, а решта виражає особисті думки авторів. Також з таблиці 1.9 можна побачити, що публікацій про алгоритми довгої арифметики за останні 5 років знайшлося лише 10, а щодо генерації довгих випадкових чисел – 8. Це свідчить про активний інтерес наукової спільноти до застосування високоточних обчислень у наукових дослідженнях.

Аналіз показує, що більшість досліджень у галузі методів та засобів виконання розрахунків (60%) належать до категорій *solution proposal* та *validation research*, що вказує на практичну спрямованість цих робіт. Водночас, дослідження сучасних технологій та інструментів представлені переважно теоретичними роботами, що може свідчити про відносну новизну цієї галузі та потребу в подальшому практичному впровадженні запропонованих рішень.

Програмні комплекси для виконання інженерно-наукових розрахунків широко застосовуються в різних галузях машинобудування:

- у проектуванні механічних систем – для розрахунку напружено-деформованого стану конструкцій [8, 15, 22];
- у гідравліці та пневматиці – для моделювання течій рідин та газів у складних системах [31, 37];
- у термодинаміці – для розрахунку теплових процесів та теплопередачі [41, 48];
- у динаміці машин – для аналізу коливань та стійкості механічних систем [52];
- у триботехніці – для розрахунку зношування та тертя в парах тертя [40, 44];
- у технології машинобудування – для оптимізації процесів обробки матеріалів [47, 51].

Аналіз літератури показав, що найбільш ефективними методами побудови програмного забезпечення для інженерно-наукових розрахунків є:

- об'єктно-орієнтовані підходи з використанням модульної архітектури, які забезпечують гнучкість та розширюваність системи – дозволяють створювати ієрархії класів для різних типів розрахунків та забезпечують повторне використання коду [1, 4, 9];
- компонентні архітектури з чітким розділенням обчислювальних ядер, інтерфейсу користувача та системи управління даними – це забезпечує можливість незалежної розробки та тестування окремих компонентів [16, 19, 26];
- мікросервісна архітектура для великих комплексів, яка дозволяє масштабувати окремі сервіси незалежно та забезпечує високу доступність системи [3, 14, 17].

Найбільш ефективними для інженерно-наукових розрахунків виявилися такі алгоритмами довгої арифметики:

- алгоритм Карацуби для множення великих чисел, який забезпечує складність $O(n^{1.585})$ замість $O(n^2)$ для звичайного множення – особливо ефективний для чисел довжиною більше 1000 розрядів [7, 12, 18];
- швидке перетворення Фур'є (FFT) для множення чисел надвеликої довжини, яке дає складність $O(n \log n \log \log n)$ – його краще застосовувати для чисел довжиною більше 10000 розрядів [25, 29, 34];
- алгоритм Ньютона-Рафсона для ділення та обчислення квадратних коренів з високою точністю – забезпечує квадратичну збіжність та стабільність обчислень [15, 22, 31].

Серед готових рішень найкращу продуктивність та стабільність показали бібліотека GMP (GNU Multiple Precision Arithmetic Library) та MPFR [2, 37, 41].

Найбільш вдалимими архітектурними рішеннями для програмних комплексів інженерно-наукових розрахунків є:

- трирівнева архітектура з розділенням на рівень представлення (GUI), бізнес-логіки (обчислювальні модулі) та даних (база даних розрахунків) – забезпечує зручність супроводження та розширення функціональності [21, 30, 35];
- plug-in архітектура з можливістю динамічного підключення нових розрахункових модулів без перекомпіляції основного додатка – особливо корисна для комплексів загального призначення [38, 42, 45];
- розподілена архітектура з можливістю виконання обчислень на кількох вузлах для складних розрахунків, які вимагають значних апаратних ресурсів [8, 49].

Найбільш зручними інструментами розробки було встановлено:

- мови програмування C++ та C# – для створення високопродуктивних обчислювальних рішень [5, 24, 27];
- Python з бібліотеками NumPy, SciPy – для прототипування та швидкого написання коду [10, 33, 36];
- Java – для створення кросплатформних рішень з хорошими можливостями інтеграції [39, 43, 46];
- Qt та WPF – для створення зручних користувацьких інтерфейсів [50].

Основними проблемами при розробці програмних комплексів інженерно-наукових розрахунків є:

- забезпечення необхідної точності обчислень при розумній продуктивності;
- інтеграція різнорідних розрахункових модулів в єдиній системі;
- валідація результатів розрахунків та контроль якості;

- зручність використання для інженерів-практиків;
- супроводження та розвиток системи.

Отже, аналіз показує, що найбільш перспективним підходом є створення модульних програмних комплексів з компонентною архітектурою, які забезпечують зручний інтерфейс користувача та використовують бібліотеки, які підтримують роботу з довгою арифметикою.

Що стосується вирішення різноманітних обчислювальних задач, Julia [5] є оптимальним вибором для швидкого прототипування та продуктивних обчислень завдяки простоті та швидкості. Вона доцільна для дослідницьких проектів і невеликих команд. Python з бібліотеками NumPy, SciPy та TensorFlow [1] підходить для широкого спектра задач, включаючи машинне навчання та обробку великих даних. C++ і Fortran залишаються найкращими для HPC, де потрібна максимальна продуктивність і низькорівнева оптимізація.

Для роботи з довгою арифметикою слід використовувати бібліотеки MPFR і ARPREC [16, 3], які є стандартом для високоточних обчислень. Вони доцільні для задач, де точність є критичною, наприклад, у криптографії чи математичній фізиці. GMP рекомендується для криптографічних застосувань завдяки швидкості та надійності.

Для високопродуктивних обчислень краще використовувати бібліотеки MPI та OpenMP [18, 10], які є доцільними для паралельних обчислень на кластерах і багатоядерних системах. MPI краще підходить для розподілених систем, тоді як OpenMP – для спільної пам'яті. CUDA [23] рекомендується для задач, що можуть бути паралелізовані на GPU, таких як обробка зображень чи машинне навчання. Charm++ [2] доцільний для динамічних і масштабованих додатків, де потрібна адаптивна декомпозиція задач.

Одними з найкращих інструментів для розробки ПЗ є Docker і Jupyter Notebooks [6, 25]: забезпечують відтворюваність і зручність у наукових дослідженнях. Вони підходять для команд, які працюють із великими

наборами даних і потребують гнучкості. Agile і DevOps [47] доцільні для проектів із частими змінами вимог, тоді як модульна архітектура дозволяє створювати масштабовані та підтримувані системи. Rust [24] є перспективним для нових проектів завдяки безпеці пам'яті та продуктивності, особливо для НРС.

Для генерації псевдовипадкових чисел доцільно використовувати рішення Mersenne Twister [30], яке є універсальним вибором для більшості наукових задач завдяки високій якості та швидкості. Криптографічні генератори [15] доцільні для застосувань, де потрібна безпека, таких як захист даних.

1.2 Огляд існуючих програмно-апаратних засобів для вирішення різноманітних задач у машинобудуванні

На вітчизняних державних машинобудівних підприємствах історично склалося так, що майже всі інженерні розрахунки виконуються так званим “гібридним” способом, тобто із використанням паперової та електронної документації, обчислювальної техніки. А мобільність, зручність та швидкість обчислень і пошуку необхідної довідкової інформації при цьому нехтується. Крім того, час, виділений працівнику на такі розрахунки, не піддавався ніякому плануванню, міг марно витрачатись на пошуки довідкової інформації та збільшення точності обчислень. Такий спосіб організації знижує видатки підприємства на розрахунки, але збільшує неекономічні видатки на матеріали та робочий час.

Певним удосконаленням засобів розрахунків, проектування і планування ресурсів стало використання такого програмного забезпечення як: 1С:Підприємство 8, Autodesk Inventor 2014, Microsoft Office 2003.

Система програм "1С:Підприємство" включає в себе платформу і прикладні рішення, розроблені для автоматизації діяльності організацій і приватних осіб. Сама платформа не є програмним продуктом для використання кінцевими користувачами, які зазвичай працюють з одним із

багатьох прикладних рішень (конфігурацій), розроблених за допомогою даного ПЗ. Такий підхід дозволяє автоматизувати різні види діяльності, у тому числі ІР, використовуючи єдину технологічну платформу.

Платформа 1С:Підприємство 8 була створена з урахуванням досвіду застосування системи програм 1С:Підприємство 7.7. З 2003 року вона показує активний розвиток. Основне завдання платформи: надання середі розробки (framework) програмістам прикладних рішень. Платформа забезпечує розробникам підвищення рівня абстракції при створенні та використанні прикладних рішень. Це дозволяє перейти від технічних і низькорівневих понять до більш змістовних і високорівневих, наблизити ці поняття до мови користувачів і фахівців в предметної області. У кінцевому підсумку це значно прискорює і уніфікує розробку прикладного рішення та його підтримку. Одночасно з цим платформа вирішує й традиційні завдання, пов'язані з продуктивністю, ергономікою, функціональністю тощо.

На машинобудівних підприємствах даний програмний комплекс використовується в таких областях:

- автоматизація виробничих і торгівельних процесів, бюджетних і фінансових відділів;
- підтримка оперативного управління підприємством;
- автоматизація організаційної та господарської діяльності;
- ведення бухгалтерського обліку з декількома планами рахунків довільних розмірів, регламентація звітності;
- широкі можливості для управлінського обліку і побудови аналітичної документації, підтримка багатовалютного обліку;
- вирішення задач планування, бюджетування і фінансового аналізу;
- розрахунок зарплати і управління персоналом.

Сучасне життя вимагає від машинобудівних підприємств великої конкурентоспроможності. Не є виключенням і підприємства, що проектують та виробляють судові машини і механізми. Проектна документація повинна розроблюватись швидко та відповідати всім нормам, а об'єкт проектування –

по можливості мати не тільки двовимірне зображення, але й, для більшої наочності, бути представленим у тривимірному вигляді. Отже, в сучасному виробництві досить широко використовуються різноманітні системи автоматизованого проектування (САПР).

Однією з таких систем є Autodesk Inventor – параметрична система тривимірного твердотілого та поверхневого проектування, що призначена для створення цифрових прототипів промислових виробів і їх проектної документації. Також у середовищі Autodesk Inventor можна виконувати інженерний аналіз виробу та перевірку його в дії. Новітній принцип проектування – це функціональне проектування, що не обмежується лише геометричним моделюванням.

Інструментальні засоби Autodesk Inventor забезпечують повний цикл конструювання та формування конструкторської документації. Дана програма допомагає машинобудівним підприємствам у таких основних напрямках:

- значне скорочення циклу розробки моделі конструкції;
- можливість спільної роботи над конструкцією всіх розробників, зокрема груп інженерів, які знаходяться на великій відстані один від одного;
- можливість вводу користувальних примітивів у параметричному вигляді з метою повторного використання;
- забезпечення доступу до тримірної моделі конструкції не тільки розробникам, а й усім групам користувачів, які беруть участь в роботі над проектом.

Отже, в машинобудуванні використовуються такі функціональні можливості Autodesk Inventor:

- адаптивне конструювання (забезпечуються дуже високі часові й ергономічні характеристики деталей, а всі конструктивні елементи, які можуть бути використані в процесі роботи над проектом, завжди несуть в собі найостаннішу інформацію про всі операції та оновлення, що відбувалися з ними);

- адаптивне компонування (при завершенні розробки загальної конструкції моделі система дозволяє змінити форму деталі, що представлена у вигляді рисунка чи ескізу, та накласти всі взаємні зв'язки для даної деталі в конструкції таким чином, як це було передбачено розробником на початковому етапі розробки конструкції);
- вбудований конструктор елементів;
- інструментарій, який забезпечує спільну роботу над документацією;
- системи підтримки та супроводження процесу конструювання.

Сьогодні жодне підприємство не може уявити свою роботу без офісного пакету ПЗ – набору програм, призначених для роботи з електронною документацією. Склад офісного пакету, зазвичай, залежить від виробника та його спеціалізації, проте еволюція програмного забезпечення зробила типовими для офісних пакетів такі основні компоненти:

- текстовий процесор – для роботи з документів, що складаються з текстів, ілюстрацій, таблиць, графіків, діаграм тощо;
- редактор електронних таблиць – засіб обробки таблиць з даними;
- засіб створення презентацій – дозволяє створювати яскраві кольорові електронні презентації, щоб краще описати клієнтам пропозиції та роботу компанії;
- система управління базами даних в офісних пакетах, як правило, забезпечує початковий рівень функціональних можливостей створення, редагування та доповнення БД;
- графічний редактор – дозволяє редагувати зображення різних форматів.

Своє походження офісні пакети ведуть від програм обробки текстів. Перші повноцінні офісні пакети, що містять текстовий процесор, редактор електронних таблиць і різні утиліти, з'явилися лише у 80-х роках. Піонером у сфері офісних пакетів вважається компанія Lotus, яка в 1982 році випустила першу версію табличного редактора Lotus 1-2-3 (займав усього 256 КБ ОЗП). В тому ж році компанія Microsoft випустила першу версію текстового редактора Word для DOS, через деякий час до якого були додані програми

для проведення презентацій, планувальники-органайзери, а також персональні системи управління базами даних.

Основний етап еволюції офісних пакетів здійснювався з початку 80-х років до середини 90-х. Подальші удосконалення стосувалися лише покращання користувацького інтерфейсу, зручності у роботі, інтеграції в Інтернет та ін. Сучасні офісні продукти пропонують досить багато різних засобів, які часто не використовуються середньостатистичним користувачем.

Оскільки сучасні інформаційні технології впевнено рухаються в бік розвитку і втілення концепції «центральної цифрової нервової системи» підприємств, яка забезпечує доступ до будь-яких інструментів і даних, взаємозв'язок співробітників, стандартизацію і керування документообігом, то офісні пакети перетворились на офісні системи.

На більшості державних підприємств використовуються офісні продукти фірми Microsoft.

Microsoft Office являє собою інтегрований програмний пакет автоматизації офісної діяльності. Перші версії офісних програм Microsoft з'явилися в середині 80-х років. На той час вони працювали під управлінням операційної системи MS DOS. Перша Windows-версія пакету з'явилася з виходом Windows 3.0. Починаючи з 1995 року всі версії Microsoft Office є 32-бітними додатками, а сучасні включають у себе і 64-бітні реакції. З даного офісного пакету версії 2003 у машинобудуванні найчастіше використовуються такі програми:

Microsoft Office Word – головний компонент офісного пакету і безперечно один із найпопулярніших текстових процесорів. Він містить дуже широкий вибір інструментів для редагування та форматування текстів документів, а також для додання в них різноманітних об'єктів – математичних формул, растрових та векторних малюнків, графіків, діаграм і т. д. У Word 2003 включено нові функції, що спрощують створення, читання та спільне використання документів. У дану версію також включено підтримку формату XML як власного формату файлів.

Microsoft Office Excel – це потужний і простий у використанні процесор для обробки електронних таблиць. У даній версії програми збережено кращі традиції попередніх версій електронних таблиць. Excel добре працює з діаграмами і графіками та може використовувати електронні таблиці в інших поширених форматах. В Excel 2003 включено нові функції, що спрощують аналіз та спільне використання інформації.

Microsoft Office Access – система керування базами даних з широкими можливостями обробки реляційних баз даних та створення нових складних програмних елементів. Вона дозволяє створювати БД, що можуть містити в собі робочі таблиці, запити, модулі, макроси, форми, звіти та web-сторінки. Зі створеними в Access БД дуже легко працювати навіть недосвідченим користувачам. Робота в такому ПЗ спрощується завдяки виявленню помилок, які часто зустрічаються, шляхом їх відмічання та представленням варіантів їх виправлення.

Microsoft Office Power Point – це спеціалізована програма для створення комп'ютерних презентацій, які використовуються на ділових зустрічах, засіданнях фірм з навчальною метою і т. п. Вона дозволяє створювати набір слайдів, які автоматично чи в ручному режимі переглядаються на ПК, чи за допомогою цифрового проектора або розміщаються в Інтернеті. Слайди можна також роздрукувати на папері чи плівці. Файли Power Point 2003 можна легко записувати на компакт-диски, а завдяки інтеграції з плеєром Microsoft Windows Media Медіа при демонстрації слайдів можна відтворювати потокове аудіо та відео. Дана програма проста в роботі і має велику кількість інструментів, що дозволяє легко підготувати презентацію, яка в доступній формі може виразити все, що користувач хоче повідомити.

Як було зазначено вище, усі розрахунки на більшості сучасних українських металообробних підприємствах виконуються лише за допомогою або звичайного кишенькового калькулятора, або аналогічних програм на мобільному телефоні (чи смартфоні) і ПК.

Калькулятор – це електронний обчислювальний пристрій для виконання операцій над числами або алгебраїчними формулами. Він замінив механічні обчислювальні пристрої такі як абаки, рахівниці, логарифмічні лінійки, механічні або електромеханічні арифмометри, а також математичні таблиці (перш за все - таблиці логарифмів).

У залежності від можливостей та цільової сфери використання калькулятори бувають найпростішими, бухгалтерськими, інженерними (науковими), фінансовими. До окремих класів, звичайно, виділяють програмовані калькулятори, які дають можливість виконання складних обчислень за попередньо завантаженою програмою. Спеціалізовані калькулятори призначені для виконання обчислень у дуже вузьких сферах (статистичні, медичні, спеціальні фінансові розрахунки і т. п.). Подібні калькулятори часто використовуються на підприємствах, але все частіше у вигляді програм для ПК.

Найпопулярнішою програмою такого класу серед машинобудівників є стандартний калькулятор Windows, який входить до складу операційної системи.

Програма Калькулятор повністю імітує роботу звичайного “апаратного” калькулятора, його кнопки можна натискати мишкою, також можливий ввід за допомогою цифрової клавіатури, вставка математичних виразів із буфера обміну. Вона дозволяє працювати в одній з двох форм: звичайній (вид «Обычный») або інженерному (вид «Инженерный»), тобто, фактично, викликати один з двох калькуляторів. Для зміни виду калькулятора потрібно в головному меню «Вид» вибрати відповідний пункт.

Звичайний калькулятор дозволяє виконувати 4 арифметичні дії (додавання, віднімання, множення, ділення), добувати квадратний корінь, знаходити число, обернене до даного, виконувати деякі дії з процентами, а також запам’ятовувати одне число, – проміжний результат обчислень.

У інженерному режимі, який є найбільш вживаним робітниками підприємств, доступні такі додаткові функції:

- тригонометричні та гіперболічні функції, натуральний та десятковий логарифми, піднесення до степеня (для квадратів і кубів виділені окремі кнопки), добування кореня;
- перевід частин градуса в хвилини та секунди, обчислення факторіалів (для нецілого аргументу замість факторіала обчислюється гамма-функція $\Gamma(x+1)$);
- групування операцій (кнопки з дужками, є індикатор рівня вкладеності), переключення режимів відображення (фіксована або плаваюча точка), обчислення залишку від ділення.

1.3 Рішення виробничих задач

Більшість виробничих задач розв'язуються на підприємствах за допомогою довідкових даних. Усі вони необхідні для інженерних розрахунків і знаходяться в нормативних документах ГОСТ і ДСТУ. На підприємствах використовуються як паперові, так і електронні версії цих документів. Спробуємо висвітлити сутність кожного з видів цих актів.

Державний стандарт (рос. Государственный стандарт, ГОСТ; можливий український переклад аббревіатури: ДЕСТ) – одна з основних категорій стандартів у СРСР, сьогодні міждержавного стандарту в СНД. Приймається Міждержавною радою зі стандартизації, метрології і сертифікації (МГС).

У радянські часи всі ГОСТи були обов'язковими для застосування в тих областях, які визначалися преамбулою самого стандарту.

Багато з українських національних стандартів ДСТУ, що існують нині, на сировину, продукцію, а також на методи виробництва виникли ще в радянські часи. Нові ГОСТи, головним чином загального характеру, з одного боку, ставлять галузь в жорсткі рамки – вони вказують з якої сировини і за якою технологією виробляти ті або інші продукти, а з іншого боку, дозволяють виробникам зберігати свою індивідуальність.

До ухвалення відповідних технічних регламентів закон передбачає

обов'язкове виконання вимог стандартів у частині, що стосується цілей захисту життя або здоров'я громадян, майна фізичних або юридичних осіб, державного або муніципального майна; охорони довкілля, життя або здоров'я тварин і рослин; запобігання діям, що вводять в оману споживача.

В Україні на виконання Державної програми стандартизації на 2006-2010 роки розроблено Програму перегляду чинних в Україні міждержавних стандартів (ГОСТ), розроблених до 1992 року, та приведення їх у відповідність до Угоди про технічні бар'єри в торгівлі Світової організації торгівлі.

Основними напрямками виконання даної програми є:

- перевірка міждержавних стандартів на відповідність законодавству, інтересам держави, потребам споживачів, рівню розвитку науки і техніки, вимогам міжнародних та регіональних стандартів, положенням Угоди про технічні бар'єри у торгівлі;
- перегляд міждержавних стандартів із внесенням змін до них, заміну їх на відповідні міжнародні або національні стандарти;
- скасування міждержавних стандартів, які втратили актуальність, не використовуються і не відповідають вимогам чинного законодавства.

Переліки міждержавних стандартів (ГОСТ), розроблених до 1992 року, чинність яких в Україні може припинитись, розміщуються на веб-порталі ДП «УкрНДНЦ» (<http://www.ukrndnc.org.ua/>) та публікуються в щомісячному інформаційному покажчику «Стандарти».

Міністерство економічного розвитку в грудні 2015 року скасувало 12 776 ГОСТів, які залишаються частково дійсними на перехідний період до 2018 року. Підприємства за бажанням зможуть застосовувати їх добровільно. Реформа системи технічного регулювання передбачає перехід від системи обов'язкових держстандартів до європейської моделі технічного регулювання, заснованої на застосуванні технічних регламентів і добровільному використанні стандартів.

Другим типом нормативних актів, з яких може бути отримана чинна

інформація для інженерних розрахунків, є Державні стандарти України (ДСТУ) – стандарти, розроблені відповідно до чинного законодавства України, що встановлюють правила загального і багаторазового застосування виробів, їх загальні принципи або характеристики, які стосуються діяльності чи її результатів, з метою досягнення оптимального ступеня впорядкованості. Вони розроблені на основі консенсусу та затверджені уповноваженим органом. Стандарти ДСТУ існують з 1993 року.

Станом на жовтень 2014 фонд національних стандартів в Україні становить 27,5 тис. документів, 7489 з яких гармонізовані з міжнародними та європейськими. Протягом 2014 прийнято 175 національних нормативних документів, з яких 115 гармонізовані з міжнародними та європейськими, підготовлено до затвердження 350 проектів стандартів.

Державні стандарти України розробляються на організаційно-методичні та загальнотехнічні об'єкти, а саме:

- організація проведення робіт із стандартизації, науково-технічна термінологія, класифікація і кодування техніко-економічної та соціальної інформації, технічна документація, інформаційні технології, організація робіт з метрології, достовірні довідкові дані про властивості матеріалів і речовин;
- вироби загальномашинобудівного застосування (підшипники, інструмент, деталі кріплення тощо);
- складові елементи народногосподарських об'єктів державного значення (банківсько-фінансова система, транспорт, зв'язок, енергосистема, охорона навколишнього природного середовища, оборона тощо);
- продукцію міжгалузевого призначення;
- продукцію для населення та народного господарства;
- методи випробувань.

Опишемо спосіб використання довідкової документації при вирішенні реальної виробничої задачі на прикладі обчислення ваги сортового металопрокату «арматура» діаметром 80 мм довжиною 15 погонних метрів.

Із проектної документації дізнаємось номер ГОСТу або ДСТУ. У нашому випадку це ГОСТ 5781-82, після цього знайти даний документ у паперовому вигляді шляхом перегляду змісту або за допомогою пошукових систем в електронному вигляді не складає ніяких труднощів. У таблиці ГОСТу знаходимо теоретичну масу 1 м профілю для даного діаметра – 39,46 кг. Виконанням арифметичної операції $15 * 39,46$ дізнаємось, що вага 15 м арматури діаметром 80 мм становить 591,9 кг.

Наступний приклад виробничої задачі: потрібно дізнатись довжину 25 кг троса (ГОСТ 483-55) окружністю 65 мм. Для цього у вказаному ГОСТі знаходимо вагу 1 метра троса заданої окружності і формулу обчислення довжини: $25/0,32=78,125$ м. У іншому випадку, при відомій довжині та необхідності дізнатись вагу, розрахунок був би іншим: $25*0,32=8$ кг.

Приклад розв'язання іншої задачі за допомогою документації: на підприємстві є сталевий канат (6х36, 216 дротів, ГОСТ 7668-55) кількох різних діаметрів достатньої довжини; потрібно дізнатись чи зможе він витримати навантаження 5000 кг і якого діаметра повинен для цього бути, якщо буде експлуатуватись як у ручному режимі, так і в режимі підйому людей. Для вирішення такої задачі потрібно дізнатись коефіцієнт запасу міцності каната. Із документації дізнаємось, що для ручного режиму він становить 4,5, а для режиму підйому людей – 9. Оскільки канат буде працювати у двох режимах, то нам слід вибрати найбільший коефіцієнт. Помножуючи 5000 на 9 отримуємо вагу, при дії якої канат розірветься – 45000 кг. Із вказаного ГОСТу дізнаємось, що діаметр каната для підйому такої ваги має складати 28,5 мм.

Отримані результати подібних розрахунків використовуються при подальшому плануванні виробничого процесу.

1.4 Онлайн-системи інженерних розрахунків

Окрім програм, які є частиною операційної системи або створені іншими розробниками, для інженерних розрахунків можуть

використовуватись різноманітні онлайн-сервіси. Щоб ними скористатись, потрібен лише доступ в Інтернет та програма-браузер. Працюючи в інтерактивному режимі, такі обчислювальні системи не потребують встановлення на комп'ютер або смартфон. До недоліків можна віднести постійну необхідність інтернет-з'єднання, інколи більш високе навантаження на компоненти комп'ютера у порівнянні зі звичайними програмами. Легко отримати до них доступ можна, увівши необхідний запит у пошуковій системі (наприклад, «калькулятор ваги металу»).

Оглянувши велику кількість інтернет-ресурсів, які надають можливості інтерактивного виконання різних видів розрахунків, ми дійшли висновку, що їх можна умовно поділити на два види: загальноматематичні та специфічно інженерні.

До першої групи відноситься сайт ua.onlinemschool.com, створений для всіх, хто має справу з математикою. Він містить велику кількість калькуляторів для вирішення різноманітних математичних задач, які можуть використовуватись як школярами і вчителями, так і деякими інженерами на металообробних підприємствах. Сайт надає такі основні можливості:

- конвертування одиниць величин (маси і ваги, відстані та довжини, площі, об'єму, часу, швидкості, температури тощо);
- виконання будь-яких арифметичних операцій із позначенням виконаних дій;
- розрахунок відсотків;
- розв'язання рівнянь (квадратних, біквадратних, систем рівнянь методами Гауса, Крамера);
- обчислення n-того члена арифметичної прогресії та суми арифметичної прогресії;
- обчислення границі функції та похідної функції;
- розв'язання інтегралів;
- розв'язання задач з теорії ймовірності, комбінаторики та аналітичної геометрії

- робота з векторами та матрицями;
- обчислення площі, об'єму та периметра різноманітних геометричних фігур.

Проаналізувавши перелік даних можливостей, ми дізнались, що корисними для інженерів на цьому сайті є лише конвертери різноманітних величин, інтерактивні засоби розв'язання інтегралів, деяких геометричних задач та задач з теорії ймовірності. У багатьох випадках можливостей подібних сайтів недостатньо для автоматизації вирішення більшості інженерних задач.

Проте Інтернет містить надзвичайно велику кількість ресурсів для розрахунку ваги, довжини, площі та інших величин певних матеріалів, а також для виконання специфічних інженерних обчислень (таких як розрахунок кількості зварювальних матеріалів та газів для виконання ремонтних і виробничих робіт, необхідних розмірів для виготовлення втулки Гудрича (гумово-металевих підшипників) і т. п.). Одним із таких ресурсів є tda-spb.ru/calk-metall.html, який дозволяє дізнатись вагу будь-якого металу за вказаними розмірами. Проте найбільшу функціональність, на наш погляд, має сайт 1metal.com/calc-info.html, який надає користувачу більш гнучкий вибір параметрів для розрахунку. На відміну від попереднього ресурсу, він окрім виду металу дозволяє вибрати такі важливі для інженерів і планувальників параметри як вид прокату, марка сталі, густина тощо (довідкові дані для цього, напевно, були взяті з нормативних актів ГОСТ і ДСТУ).

Інколи перед інженерами виникає необхідність отримання результатів обчислень у різних системах числення з ціллю їх шифрування. Існує велика кількість онлайн-конверторів для подібних перетворень. Проте найбільше нам сподобався сайт numsys.ru, який дозволяє виконувати арифметичні операції додавання, віднімання, множення та ділення над цілими та дробовими числами в різних СЧ, а також перетворювати їх з однієї СЧ в іншу. До переваг можна віднести високу швидкість отримання точних

результатів, до недоліків – можливість роботи лише з додатними числами та найбільша довжина чисел – лише 50 символів. Саме цей сайт використовувався нами при розробці програмного комплексу з подібними можливостями як еталон результатів.

Ми дійшли висновку, що при наявності відповідних технічних можливостей, здійснення розрахунків за допомогою онлайн-ресурсів сприяє більш швидкому та зручному вирішенню інженерних задач, хоча й існує проблема достовірності довідкових даних, які використовуються в подібних обчислювальних системах.

1.5 Обґрунтування потреби в новому програмному продукті

У даному розділі ми проаналізували існуючі способи виконання інженерних розрахунків на сучасних українських державних металообробних підприємствах. Нами було розглянуті усі класи ПЗ, які використовуються на підприємствах: як засоби для проектування та ведення бухгалтерського обліку, так і обчислювальні програми і системи пошуку довідкових даних, необхідних для здійснення розрахунків, описані апаратні аналоги програмних рішень та різноманітні онлайн-сервіси з подібним функціоналом.

Було виявлено, що у сферах інженерного проектування та планування ресурсів використовується достатньо якісне ПЗ від наших та закордонних виробників, яке надається на комерційній основі разом із технічною підтримкою, не існує ніяких проблем і з оформленням різноманітної документації. Проте часто перед інженерами та планувальниками ресурсів виникають задачі, рішення яких не можуть бути достатньо автоматизованими за допомогою існуючих стаціонарних програмних комплексів. Високий рівень інтерактивності та зручності розрахунків і отримання довідкових даних надають лише деякі онлайн-сервіси. Проте до ряду недоліків таких систем відносяться постійна потреба в інтернет-з'єднанні, яке присутнє не у всіх підрозділах виробничих комплексів, проблема достовірності довідкових даних і формул розрахунків, які використовуються в інтерактивному ПЗ. Як

ми знаємо, помилки при розрахунках ваги, розмірів, навантажувальної здатності матеріалів і т. п. можуть привезти до великих фінансових збитків та пошкодження здоров'я робітників. Тому більшість інженерів та планувальників не довіряють покладені на них розрахунки автоматизованим онлайн-системам і віддають перевагу ручному пошуку необхідної інформації для обчислень, що сприяє неекономічному використанню робочого часу. Крім того, такі сервіси не надають можливості працювати з числами довжиною 100-250 символів, що є необхідним в дуже рідких випадках.

Помічено, що великі машинобудівельні підприємства, які не є достатньо завантаженими роботою, взагалі не мають доступу до більшості сучасних технологій автоматизації розрахунків. На них до сих пір використовуються апаратні калькулятори та інколи навіть механічні обчислювальні пристрої.

Тому перед нами була поставлена задача розробити програмний комплекс, який усував би велику кількість недоліків існуючих аналогів, більшість з яких доступна тільки в онлайн-режимі. Нове програмне забезпечення для інженерних розрахунків повинно здійснювати обчислення високої точності (вміти працювати з числами великої довжини), мати високий рівень інтерактивності, надавати швидкий доступ до необхідних довідкових даних і можливості їх зручного поповнення та редагування, показувати всі виконані дії для їх контролю працівниками підприємства та бути доступним на комп'ютері з операційною системою Windows будь-якої версії без потреби в інтернет-з'єднанні.

Висновки до розділу

Проаналізувавши сучасний стан автоматизації інженерних розрахунків на українських металообробних підприємствах, можна дійти висновку про наявність суттєвих недоліків у існуючих підходах до організації обчислювальних процесів. Дослідження показало, що більшість вітчизняних державних машинобудівних підприємств використовують так званий

гібридний спосіб виконання розрахунків, який поєднує паперову та електронну документацію з обчислювальною технікою, однак при цьому нехтує важливими параметрами мобільності, зручності та швидкості обчислень. Така організація робочого процесу призводить до неекономічного використання робочого часу, оскільки значна його частина витрачається на пошук довідкової інформації та підвищення точності обчислень, що в кінцевому результаті збільшує загальні витрати підприємства на матеріали та трудові ресурси.

Аналіз існуючих програмно-апаратних засобів виявив, що на підприємствах використовується різноманітне програмне забезпечення, зокрема Autodesk Inventor для тривимірного проектування та створення конструкторської документації, а також офісний пакет Microsoft Office 2003 для роботи з електронною документацією. Однак ці програмні комплекси, незважаючи на свою функціональність у відповідних сферах застосування, не забезпечують достатнього рівня автоматизації саме інженерних розрахунків. Для виконання обчислень працівники змушені використовувати прості інструменти, такі як стандартний калькулятор Windows або апаратні калькулятори, які не надають можливості зберігати історію обчислень, працювати з довідковими даними та автоматизувати типові розрахунки.

Розглянуті методи вирішення виробничих задач демонструють залежність інженерів від нормативної документації ГОСТ та ДСТУ, яка містить необхідні довідкові дані для розрахунків параметрів металопрокату, канатів, тросів та інших матеріалів. Процес отримання потрібної інформації з цих документів є трудомістким і потребує значних часових витрат на пошук відповідних таблиць та виконання подальших обчислень.

Огляд онлайн-систем інженерних розрахунків показав наявність різноманітних інтернет-ресурсів, які пропонують інтерактивне виконання математичних та специфічних інженерних обчислень. До таких ресурсів належать загальноматематичні калькулятори та спеціалізовані сервіси для розрахунку параметрів металопрокату. Незважаючи на високий рівень

інтерактивності та зручності користування, ці системи мають суттєві недоліки, серед яких необхідність постійного інтернет-з'єднання, проблема достовірності довідкових даних, обмеження по довжині чисел та відсутність можливості контролю виконаних операцій. Особливо критичною є проблема достовірності інформації, оскільки помилки в розрахунках ваги, розмірів або навантажувальної здатності матеріалів можуть призвести до значних фінансових збитків та створити загрозу здоров'ю робітників.

Проведений аналіз дозволив обґрунтувати необхідність розробки нового програмного продукту, який би усував виявлені недоліки існуючих рішень. Було встановлено, що у сферах інженерного проектування та планування ресурсів використовується якісне комерційне програмне забезпечення, однак для автоматизації безпосередньо інженерних розрахунків та роботи з довідковими даними відсутні ефективні інструменти. Більшість інженерів та планувальників не довіряють онлайн-системам через проблеми з достовірністю даних і віддають перевагу ручному пошуку інформації, що призводить до неефективного використання робочого часу. Особливо гостро ця проблема стоїть на великих машинобудівних підприємствах, які не є достатньо навантаженими роботою і не мають доступу до сучасних технологій автоматизації розрахунків.

Таким чином, виникає потреба у створенні спеціалізованого програмного комплексу для інженерних розрахунків, який повинен забезпечувати високу точність обчислень, високий рівень інтерактивності, швидкий доступ до перевірених довідкових даних з можливістю їх зручного поповнення та редагування, відображення історії виконаних дій для контролю правильності розрахунків, а також функціонування на комп'ютерах з операційною системою Windows будь-якої версії без потреби в інтернет-з'єднанні. Розробка такого програмного забезпечення дозволить підвищити ефективність роботи інженерів та планувальників, зменшити витрати робочого часу на виконання рутинних обчислень, забезпечити достовірність результатів розрахунків та сприятиме підвищенню продуктивності.

РОЗДІЛ 2. ПРОГРАМНІ ТА АПАРАТНІ ЗАСОБИ ДЛЯ РОЗРОБКИ І ФУНКЦІОНУВАННЯ ПЗ

2.1. Програмно-апаратні вимоги

Звичайно, як і для всіх програм, для функціонування розробленого нами програмного комплексу підтримки інженерних розрахунків необхідний апаратний пристрій персональний комп'ютер (ПК) – персональна електронно-обчислювальна машина, яка має експлуатаційні характеристики побутового пристрою та універсальні функціональні можливості. Він використовується як засіб масової автоматизації в основному для створення автоматизованих робочих місць у соціальній та виробничій сферах діяльності, народному господарстві тощо. Комп'ютер експлуатується користувачами, які можуть не володіти спеціальними знаннями в галузі обчислювальної техніки та програмування.

Спочатку ПК був створений як обчислювальна машина, але пізніше почав використовуватись у інших цілях – як засіб доступу до інформаційної мережі, як мультимедійна та ігрова платформа. Сучасний ПК може використовуватися декількома людьми одночасно. Хоча з розвитком можливості такого пристрою розширюються, ПК все рівно буде вважатись персональним, не залежно від того, якими б обчислювальними можливостями він не володів. Якщо не так давно лише IBM-сумісні пристрої (однотипні комп'ютери) вважались ПК, то зараз будь-який комп'ютер, який використовується людиною за власними примхами, може розглядатись як персональний. Конструктивність на сьогодні не найважливіший критерій ПК: практично будь-який комп'ютер може стати персональним у залежності від необхідності та можливостей його використання. Все більше користувачів використовують у якості ПК в основному його стаціонарні або портативні версії, різниця між якими полягає у швидкодії, габаритах та мобільності.

Отже, для розробленого нами програмного продукту потрібен звичайний ПК, який задовольняв би хоча б мінімальним вимогам операційної системи (ОС) – програмного комплексу взаємозв'язаних програм,

призначених для управління ресурсами комп'ютера і організації взаємодії з користувачем. У логічній структурі типової обчислювальної системи ОС займає положення між пристроями з їх мікроархітектурою, машинною мовою та, можливо, власними (вбудованими) мікропрограмами (драйверами) – з однієї сторони – і прикладними програмами з іншої. Операційна система дозволяє розробникам ПЗ абстрагуватись від деталей реалізації та функціонування пристроїв, надаючи мінімальний необхідний набір функцій. У більшості обчислювальних систем ОС являється основною, найбільш важливою (а іноді єдиною) частиною сімейств ПЗ. З 1990-х років найбільш вживаними ОС являються системи сімейств Windows, UNIX и UNIX-подібні системи.

Попередником операційних систем слід вважати службові програми (завантажники і монітори), а також бібліотеки часто використовуваних підпрограм, які почали розроблюватись з появою універсальних комп'ютерів першого покоління (кінець 1940-х років). Службові програми мінімізували фізичні маніпуляції оператора з пристроями, а бібліотеки дозволили запобігти багатократного програмування одних і тих самих дій (здійснення операцій вводу-виводу, обчислення математичних функцій і т. п.).

У якості операційної системи нами обрано сімейство Microsoft Windows. Даний вибір обумовлений тим, що сьогодні в світі близько 80% персональних комп'ютерів працює під керуванням тих чи інших версій Windows. Це призводить до поширення різноманітних зручних програм та середовищ розробки саме для платформи Microsoft. Не є виключенням і підприємство-замовник нашого програмного продукту: усі комп'ютери в ньому знаходяться під керуванням різних версій від Windows 98 до Windows 7. Така популярність пояснюється зручністю інтерфейсу користувача, підтримкою різноманітних сучасних технологій, більшості апаратних засобів комп'ютера – як внутрішніх, так і периферійних, а також багатьма іншими властивостями.

Для функціонування розробленого нами ПЗ, основною функцією якого

є підтримка інженерних розрахунків на машинобудівному підприємстві, потрібен ПК, який задовольняв би наступним вимогам ОС Windows: процесор Intel Pentium з частотою 233 МГц (рекомендується не менше 300 МГц) або інший аналогічний від фірми AMD, не менше 64 МБ оперативної пам'яті (рекомендується 128 МБ), жорсткий диск з вільним місцем не менше 20 МБ для роботи програми або 60 МБ для редагування існуючої програми за допомогою середовища розробки. Розроблена програма не потребує додаткового ПЗ, такого як .NET Framework або Microsoft Visual C++ Redistributable. Отже, для роботи та підтримки нашого продукту достатньо навіть дуже старого ПК з будь-якою версією ОС Windows.

2.2 Середовище розробки, платформа та мова програмування

Для створення програмного продукту було обрано інтегроване середовище розробки Lazarus останньої версії та мову програмування Object Pascal. Даний вибір був обумовлений вимогами підприємства у зв'язку з наявністю на ньому спеціалістів, які в майбутньому зможуть підтримувати та коригувати створену програму, наявністю готового програмного продукту, який був швидко та легко вдосконалений до необхідного рівня зручного комплексу для виконання інженерних розрахунків, а також особистим добрим володінням даної мови програмування.

Lazarus – це вільне (безкоштовне) середовище швидкої розробки ПЗ для компілятора Free Pascal, аналогічне Delphi. Даний проект базується на оригінальній багатоплатформній бібліотеці візуальних компонентів Lazarus Component Library (LCL), яка працює на багатьох операційних системах та апаратних платформах.

Free Pascal – це компілятор мов Pascal і Object Pascal, що функціонує під Windows, Linux, Mac OS X, FreeBSD, та іншими операційними системами. Таким чином, розроблені додатки можуть виконуватись практично на будь-якому комп'ютері під управлінням будь-якої системи.

Все що користувач бачить на екрані під час роботи різних програм, всі

елементи (кнопки, бігунки, меню і т. п.) можна реалізувати в Lazarus, оскільки в даному середовищі використовується технологія візуального програмування. Для створення графічного інтерфейсу програми розробник використовує готові компоненти, значки яких знаходяться на відповідній панелі. Для новостворених елементів програмний код генерується автоматично. Вручну програмуються тільки специфічні дії, які виконуватиме доданий елемент.

Перелічимо основні можливості середовища розробки Lazarus:

- підтримує перетворення проектів Delphi;
- реалізований основний набір елементів управління;
- редактор форм та інспектор об'єктів максимально наближені до середовища Delphi;
- вбудований налагоджувач;
- простий перехід для Delphi-програмістів завдяки подібності LCL до VCL;
- повністю юнікодний (UTF-8) інтерфейс і редактор, а тому відсутні проблеми з портуванням (переносом на іншу платформу) коду, що містить національні символи;
- потужний зручний редактор коду, що включає систему підказок, гіпертекстову навігацію по вихідних текстах, автоматичне завершення коду і рефакторинг;
- форматування коду «з коробки», використовуючи механізми Jedi Code Format;
- підтримка двох стилів асемблера: Intel і AT&T (підтримуються з боку компілятора);
- підтримка безлічі типів синтаксису Pascal: Object Pascal, Turbo Pascal, Mac Pascal, Delphi;
- має власний формат управління пакунками;
- автоматична збірка самого себе (під нову бібліотеку віджетів) натисненням однієї кнопки;

- підтримувані для компіляції ОС: Linux, Microsoft Windows (Win32, Win64), Mac OS X, FreeBSD, WinCE, OS/2.

До недоліків даного середовища розробки можна віднести:

- відсутність повної сумісності з Delphi (на відміну від Delphi, Lazarus надає можливість створювати багатоплатформні програми);
- великий розмір скомпільованого файлу при стандартних налаштуваннях середовища (виправляється шляхом відключення можливостей зберігати у новоствореному файлі налагоджувальну інформацію);
- відсутність повноцінної документації (хоча документація на сам компілятор Free Pascal доступна онлайн, або в PDF- та HTML-документах, а документація на Lazarus доступна у вигляді Wiki-підручників, які можуть редагувати самі користувачі);
- відсутність повноцінної підтримки COM (реалізована тільки підтримка методів), оскільки сфера інтересів розробників Lazarus лежить в області багатоплатформного програмування, а не в області взаємодії лише з Windows-програмами;
- неможливість перегляду значення властивостей об'єктів під час налагодження, відображення тільки змінних і полів об'єктів.

При розробці віконної версії програмного комплексу підтримки інженерних розрахунків ми використовували такі основні можливості середовища Lazarus:

- редактор форм з можливістю додавання різноманітних елементів інтерфейсу користувача. Використовувались такі компоненти і можливості редагування їхніх властивостей: TLabel (текстова мітка довжиною в один рядок), TComboBox (список рядків з можливістю вибору рядку із випадного списку з постійним відображенням вибраного), поля вводу тексту TEdit та TMemo (TEdit дозволяє вводити та відображати лише один рядок, TMemo – декілька), прапорець TCheckBox (перемикач для візуалізації станів увімкнено / вимкнено), TButton (кнопки для запуску виконання певних дій);

- зручний текстовий редактор з можливістю підсвічування елементів програмного коду, ведення його «історії»;
- вбудований компілятор мови Free Pascal разом із здатністю візуального позначення помилок при збірці та виконанні програми.

Отже, при розробці програмного комплексу ми використовували лише незначну частину компонентів середовища Lazarus. Можливості автоматичного доповнення та рефакторингу коду, відображення спливаючих підказок назв процедур та функцій при написанні команд, збереження налагоджувальної інформації у скомпільованому файлі були вимкнуті.

2.3 Програми для додавання і редагування довідкових даних

Усі довідкові дані, які використовуються нашою програмою для розрахунків, зберігаються в текстовому форматі у вигляді txt-файлів, які можуть бути відредаговані та доповнені користувачами програми за допомогою будь-якого текстового редактора.

Текстовий редактор – це самостійна комп’ютерна програма або компонент програмного комплексу (наприклад, редактор вихідного коду інтегрованого середовища розробки або вікно введення в браузері), призначена для створення та зміни текстових даних взагалі та текстових файлів в окремому вигляді.

Текстові редактори призначені для роботи з текстовими файлами в інтерактивному режимі. Вони дозволяють продивлятися зміст текстових файлів та виконувати над ними різноманітні дії – вставку, видалення і копіювання тексту, контекстний пошук і заміну, сортування рядків, перегляд кодів символів і конвертацію кодувань, друк і т. п. Часто інтерактивні текстові редактори містять у собі додаткову функціональність, призначену автоматизувати дії по редагуванню (від записаних послідовностей натиснутих клавіш до повноцінних вбудованих мов програмування), або відображають текстові дані спеціальним чином (наприклад, з підсвіченням синтаксису).

Велика кількість текстових редакторів є редакторами вихідного коду, тобто вони орієнтовані на роботу з текстами програм на тих чи інших комп'ютерних мовах програмування. Види текстових редакторів:

- відрядковий (рядковий) (англ. line editor) - працює з текстом як з послідовністю пронумерованих рядків, виконуючи операції над текстом у вказаних рядках. Прикладом такого редактора є edlin, який входить до складу MS-DOS;
- контекстний (рядковий) редактор (англ. context editor), прикладом якого може стати ECCE (англ. Edinburgh Compatible Context Editor), який виконує операції над текстом у поточній позиції.

Екранний текстовий редактор - дозволяє користувачу переміщувати курсор у тексті за допомогою клавіш або інших пристроїв введення. До таких програм можна віднести будь-який редактор, який працює в будь-якій сучасній віконній ОС.

До текстових редакторів з розширеною функціональністю відноситься текстовий процесор – це комп'ютерна програма, яка використовується для написання та модифікації документів, компоновки макету тексту і попереднього перегляду документів у тому вигляді, в якому вони будуть надрукованими. Сучасні текстові процесори, окрім форматування шрифтів і абзаців, перевірки орфографії включають можливості, які мали лише настільні видавничі системи, у тому числі створення таблиць і вставку графічних зображень. Найбільш відомими текстовими процесорами є Microsoft Word та OpenOffice Writer.

Операційна система Windows має вбудований текстовий редактор «Блокнот», який має лише базові можливості обробки тексту. Тому на підприємствах часто використовують розширений текстовий редактор Notepad++, який має більшу функціональність. Надаючи такі можливості як підсвічування синтаксису багатьох мов програмування і розмітки, нумерацію рядків, одночасна обробка декількох файлів (заміна і пошук тексту, збереження внесених змін), перегляд у різних кодуваннях, він сприяє більш

зручній та швидкій обробці текстових даних.

Отже, існує безліч програм, які можуть використовуватись для зміни та доповнення довідкових даних: від найпростішого редактора, який входить до складу ОС, до потужних текстових процесорів. Проте здійснюючи такі операції користувачу слід пам'ятати, що вся інформація коректно оброблюється програмою тільки якщо вона впорядкована єдиним зрозумілим для неї чином. Наприклад, у деяких файлах прийнято, що найменування розташовуються до символу «/» або «пробіл». При видаленні таких символів або доданні зайвих змінених текстовий файл може некоректно сприйматись нашою програмою та спричинити помилки. Або деякі компоненти програми використовують інформацію з декількох файлів, тому при її зміні або доповненні користувач повинен уважно слідкувати за відповідністю рядків, що зручно виконувати за допомогою спеціалізованих редакторів з функцією нумерації рядків. Видалення текстових файлів, які знаходяться в одному каталозі з програмою може призвести до пошкодження вже існуючих результатів розрахунків та помилок в роботі створеного нами ПЗ.

Висновки до розділу

Проведений аналіз програмних та апаратних засобів для розробки і функціонування програмного забезпечення підтримки інженерних розрахунків дозволяє зробити комплексний висновок щодо технічних аспектів реалізації проекту. Дослідження програмно-апаратних вимог показало, що для функціонування розробленого програмного комплексу необхідний персональний комп'ютер як апаратний пристрій.

Вибір операційної системи для функціонування розробленого програмного продукту обґрунтовується об'єктивними факторами поширеності та практичності. Сімейство Microsoft Windows було обрано основною платформою через його домінуюче положення на ринку персональних комп'ютерів, де близько вісімдесяти відсотків пристроїв працюють під керуванням різних версій цієї операційної системи.

Встановлені мінімальні системні вимоги для функціонування розробленого програмного забезпечення демонструють його високу оптимізацію та доступність для використання на обладнанні різного рівня технічної оснащеності. Програмний комплекс може працювати на комп'ютерах з процесором Intel Pentium з частотою від двохсот тридцяти трьох мегагерц або аналогічним процесором від AMD, оперативною пам'яттю обсягом від шістдесяти чотирьох мегабайт та вільним місцем на жорсткому диску від двадцяти мегабайт для роботи програми або шістдесяти мегабайт для можливості редагування через середовище розробки.

Вибір інструментів розробки програмного продукту базувався на комплексному аналізі технічних можливостей, вимог підприємства та практичних міркувань щодо подальшої підтримки створеного програмного забезпечення. Інтегроване середовище розробки Lazarus було обране як основний інструмент створення програми завдяки його статусу вільного безкоштовного середовища швидкої розробки для компілятора Free Pascal, що є аналогом комерційного середовища Delphi. Базування проекту Lazarus на оригінальній багатоплатформній бібліотеці візуальних компонентів Lazarus Component Library забезпечує можливість роботи на різних операційних системах та апаратних платформах, що потенційно розширює можливості майбутнього використання розробленого програмного продукту. Вибір мови програмування Object Pascal був обумовлений вимогами підприємства, пов'язаними з наявністю спеціалістів, які в майбутньому зможуть підтримувати та коригувати створену програму, а також наявністю готового програмного продукту, який міг бути швидко вдосконалений до необхідного рівня функціонального комплексу для виконання інженерних розрахунків.

Технологічні можливості середовища розробки Lazarus забезпечують ефективну реалізацію всіх необхідних функціональних компонентів програмного комплексу. Використання технології візуального програмування дозволяє створювати графічний інтерфейс програми за

допомогою готових компонентів, значки яких розміщені на відповідній панелі, при цьому програмний код для новостворених елементів генерується автоматично, а ручного програмування потребують лише специфічні дії, які виконуватимуть додані елементи. Основні можливості Lazarus включають підтримку перетворення проектів Delphi, реалізацію основного набору елементів управління, редактор форм та інспектор об'єктів максимально наближені до середовища Delphi, вбудований налагоджувач, повністю юнікодний інтерфейс та редактор, потужний зручний редактор коду з системою підказок та автоматичним завершенням коду, підтримку різних типів синтаксису Pascal та можливість компіляції для численних операційних систем.

При практичній розробці віконної версії програмного комплексу використовувались ключові компоненти середовища Lazarus, зокрема редактор форм з можливістю додавання різноманітних елементів інтерфейсу користувача, включаючи текстові мітки TLabel, випадні списки TComboBox, поля введення тексту TEdit та TMemo, прапорці TCheckBox та кнопки TButton. Також застосовувався зручний текстовий редактор з підсвіченням елементів програмного коду та веденням його історії, вбудований компілятор мови Free Pascal з можливістю візуального позначення помилок при збірці програми.

Організація зберігання та редагування довідкових даних реалізована з урахуванням принципів простоти, доступності та гнучкості обробки інформації. Всі довідкові дані, які використовуються програмою для розрахунків, зберігаються в текстовому форматі у вигляді txt-файлів, що забезпечує можливість їх редагування та доповнення користувачами за допомогою будь-якого текстового редактора без потреби у спеціалізованому програмному забезпеченні.

Для редагування довідкових даних можуть використовуватись різноманітні текстові редактори від найпростішого вбудованого редактора Блокнот операційної системи Windows до розширених редакторів з

додатковою функціональністю, таких як Notepad++, який надає можливості підсвічування синтаксису багатьох мов програмування, нумерації рядків, одночасної обробки декількох файлів та перегляду в різних кодуваннях, що сприяє більш зручній та швидкій обробці текстових даних. Однак користувачам необхідно враховувати, що вся інформація коректно обробляється програмою тільки за умови її впорядкування єдиним зрозумілим для програми чином, оскільки видалення або додавання розділових символів, порушення відповідності рядків між різними файлами або видалення текстових файлів, які знаходяться в одному каталозі з програмою, може призвести до некоректної роботи програмного забезпечення, пошкодження існуючих результатів розрахунків та виникнення помилок у функціонуванні програмного комплексу.

Таким чином, обрані програмні та апаратні засоби для розробки і функціонування програмного забезпечення підтримки інженерних розрахунків повністю відповідають поставленим вимогам щодо доступності, ефективності, простоти використання та можливості подальшої підтримки на підприємстві, що забезпечує успішну реалізацію проекту та його впровадження в виробничий процес металообробного підприємства.

РОЗДІЛ 3. ПРОГРАМНИЙ КОМПЛЕКС ВИКОНАННЯ ІНЖЕНЕРНИХ РОЗРАХУНКІВ

Ціллю існування даного програмного комплексу є:

- виконання арифметичних операцій додавання, віднімання, множення, ділення цілих та дробових знакових чисел надвеликої довжини (більше 50 символів) у всіх можливих системах числення (2-36);
- конвертування цілих та дробових чисел із однієї системи числення в будь-яку іншу;
- зміна точності операції ділення (кількість знаків після коми);
- виконання інженерних розрахунків за заданими формулами та з використанням довідкових даних, які знаходяться в окремих файлах;
- ведення журналу здійснених операцій.

3.1 Принципи роботи програми та інтерфейс користувача

Програмний комплекс був розроблений за допомогою засобів вільно розповсюджуваного середовища Lazarus та мови Free Pascal для використання у віконному режимі, ґрунтується на принципах візуального, об'єктно-орієнтованого та процедурного програмування. Усі екранні елементи інтерфейсу користувача, як і довідкові дані, названі російською мовою. Їх можна умовно поділити на три групи:

- клавіші екранного вводу символів чисел та арифметичних операцій (символи вводяться в будь-яке активне змінюване поле);
- кнопки, поля вводу, випадні списки модулів інженерних розрахунків;
- допоміжні елементи (перемикачі, кнопки очищення різних полів, текстові мітки, поля операцій, «журналу» та результату обчислення).

Усі компоненти програми розташовані на одній формі, зовнішній вигляд якої представлено на рисунку 3.1.1.

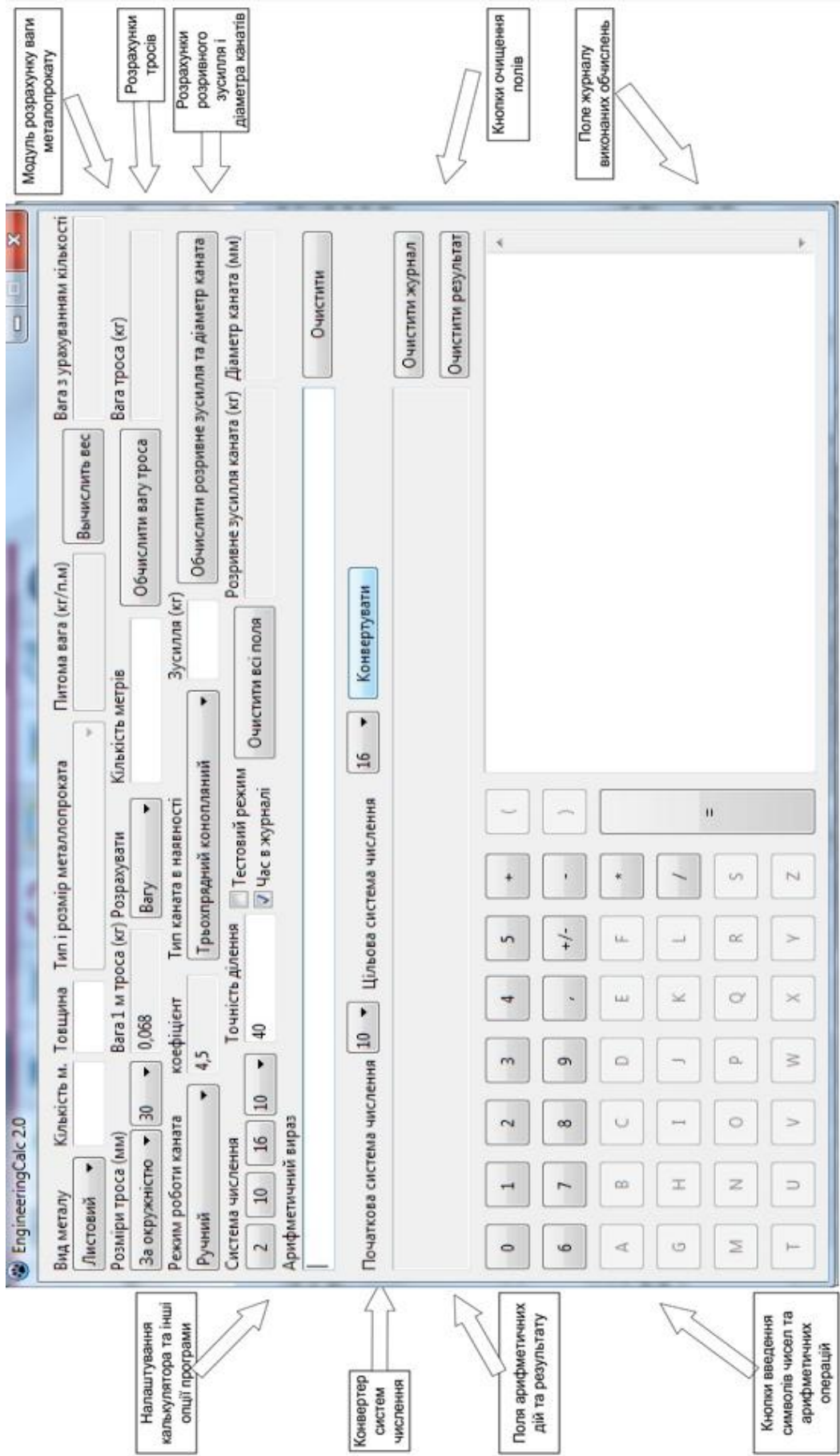


Рисунок 3.1.1 – Елементи інтерфейсу користувача

Отже, структуру даного програмного комплексу можна умовно поділити на три частини: калькулятор, конвертер систем числення та модулі інженерних розрахунків, які на даному етапі розвитку проекту містять можливості обчислення ваги металопрокату і троса, довжину троса, розривне зусилля та необхідний діаметр каната. Схему загального принципу роботи всіх компонентів програми проілюстровано на рисунку 3.1.2.

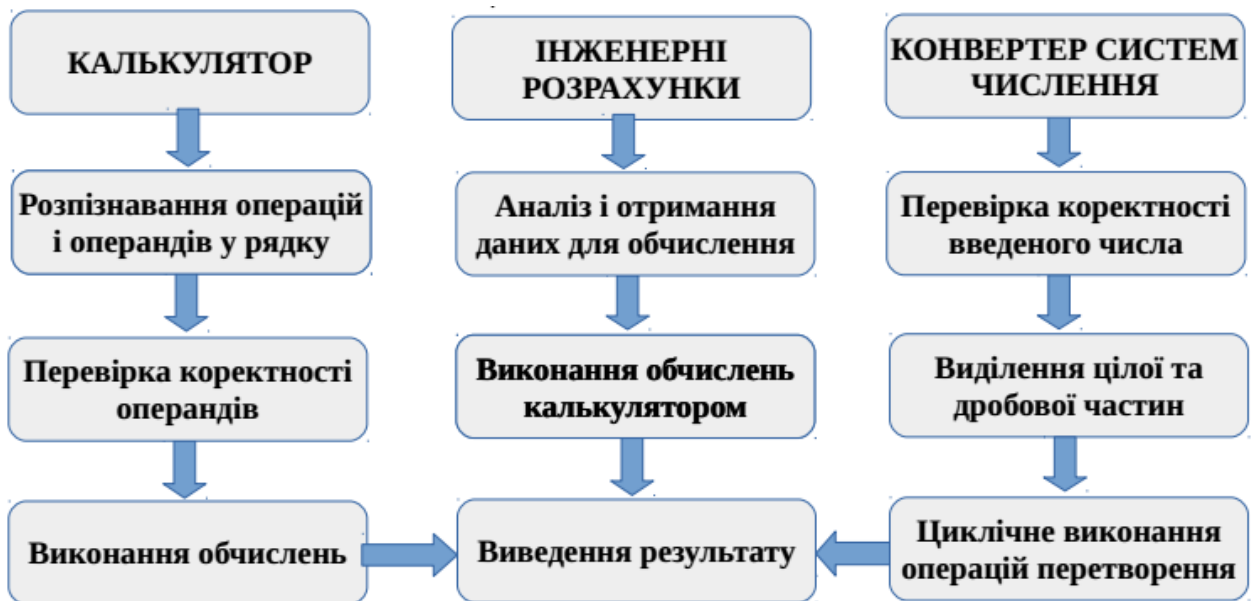


Рисунок 3.1.2 – Загальні принципи роботи ПЗ

Розглянемо особливості інтерфейсу користувача програми більш детально.

Кнопки введення числових символів активуються в залежності від того, в якій системі числення працює калькулятор. Отже, якщо калькулятор працює в шістнадцятковій системі числення, «екранна клавіатура» має вигляд, зображений на рисунку 3.1.3.

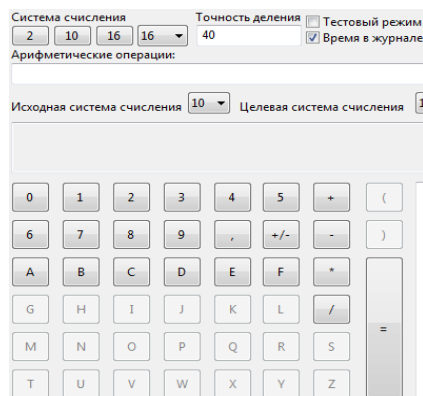


Рисунок 3.1.3 – Екранна клавіатура в режимі роботи в 16-й СЧ

Кожна кнопка екранного вводу виконує дії, закодовані в єдиній, однаковій для всіх, процедурі:

```
procedure TForm1.NumberButton(Sender: TObject);
var ch:string;
begin ch:=(Sender as TButton).hint; charinput(ch); end;
```

У властивостях кожної такої кнопки в полі Hint (підказка) прописаний символ, який відповідає тому, який має вводиться при натисканні на кнопку. Дана процедура лише зчитує символ з поля підказки натиснутої кнопки і передає його для введення у відповідне поле за допомогою іншої процедури charinput. Така реалізація є найбільш раціональною та сприяє уникненню зайвого однотипного коду. У попередній версії програми для кожної кнопки екранного вводу числових символів була створена окрема процедура, яка відрізнялась від іншої лише символом, який має вводиться при натисканні.

Принцип роботи процедури charinput ґрунтується на використанні ГЗ типу boolean, які переходять у стан true при активації певного поля форми: підпрограма перевіряє стани всіх змінних і вводить символ у кінець поля, відповідного певній boolean-змінній. Оскільки після натискання на будь-яку кнопку інтерфейсу користувача фокус залишається на натиснутій кнопці, що не дуже зручно для користувача, дана процедура ще й повертає фокус на поле, в яке був уведений символ та встановлює каретку в кінець рядка.

Подібним чином реалізована можливість активації та деактивації клавіш екранного вводу в залежності від вибраної СЧ калькулятора. Стан кнопок змінюється процедурою, в якій створюється масив елементів типу boolean та в залежності від номера активного чи неактивного елементу, активується або деактивується певна кнопка:

```
procedure tform1.buttonenable(n:byte); // n – система числення
var i:byte; a:array[0..35] of boolean;
begin
for i:=0 to 35 do a[i]:=false; // формування «списку» активних кнопок
for i:=0 to n-1 do a[i]:=true; // формування «списку» неактивних кнопок
```

```

if a[0]=true then b0.Enabled:=true else b0.Enabled:=false;
if a[1]=true then b1.Enabled:=true else b1.Enabled:=false;
if a[2]=true then b2.Enabled:=true else b2.Enabled:=false;
...
if a[35]=true then b35.Enabled:=true else b35.Enabled:=false; end;

```

Такий спосіб реалізації є набагато кращим ніж ручне формування переліку команд активації екранних клавіш для кожної СЧ.

Інтерфейс користувача також містить дві опції: «Тестовый режим» та «Время в журнале». Перша опція включає або виключає можливість виконання обчислень з використанням дужок. Оскільки ця можливість не в достатній мірі протестовані та потребує удосконалень, за замовчуванням вона вимкнена: на даному етапі розвитку проекту при виконанні пріоритетних дій у дужках можливі помилки (хоча відповідна процедура в повній мірі реалізована).

Друга опція корисна при тестуванні програми та необхідна для оцінки швидкодії обчислювальних алгоритмів: вона активує або деактивує можливість відображення кількості секунд та їх частин з точністю до трьох знаків після коми, витрачених на арифметичні операції або перетворення чисел з однієї СЧ в іншу.

До особливостей інтерфейсу користувача відноситься й автоматичне прокручування «журналу» виконаних операцій, наявність декількох кнопок очищення різних полів, відображення спливаючих підказок при наведенні курсора мишки на перемикачі.

3.2 Модуль виконання арифметичних операцій

Арифметичні операції додавання, віднімання, множення та ділення є головними функціями даного програмного комплексу для виконання інженерних розрахунків. Як було сказано раніше, наша програма здійснює їх в усіх системах числення, з двійкової по тридцяти шісткову, над цілими та дробовими числами з урахуванням їх знаку на принципах довгої арифметики.

Для збереження операндів у пам'яті та виконання різних операцій над ними використовується тип даних `string` (рядок), максимальна довжина якого у мові Pascal становить лише 255 символів. Проте використання новітнього середовища розробки Lazarus разом з директивою `{H+}` збільшує найбільшу довжину рядка до 2 Гб. Отже, фізично наш програмний комплекс здатен виконувати обчислення над числами дуже великої довжини. Вибір такого типу даних для збереження операндів головним чином обумовлений наявністю різноманітних готових зручних функцій для обробки та перетворень введеної інформації, можливістю використовувати не тільки звичні для нас традиційні символи цифр, достатньою максимальною довжиною операндів.

Модуль виконання арифметичних операцій лише умовно може називатись модулем, та становить сукупність основних і допоміжних функцій для виконання розрахунків. Нічого спільного із терміном «модуль» він не має (традиційно модулем у програмуванні називають функціонально завершений фрагмент програми, оформлений у вигляді окремого файлу з вихідним кодом, призначений для використання в інших програмах), хоча частина створених нами процедур можуть бути легко вбудовані в інший програмний продукт.

Сутність роботи даного модуля ґрунтується на використанні як глобальних так і локальних змінних. Усі функції повертають лише одне значення, проте деякі з них зберігають дані і в ГЗ (хоча директива `var` при написанні аргументів функцій нами в деяких випадках використовувалась). У майбутньому планується здійснення переробки коду програми, в ході якої будуть усунені всі ГЗ, використання яких вважається ознакою поганого стилю програмування. Усі глобальні змінні прокоментовані у коді для більш кращого розуміння. У них зберігаються рядки вхідних операндів `s1` та `s2`, залишок від ділення `od`, «вирівняні» змінені рядки операндів `se1` та `se2`, система числення `sys`, точність операції ділення `pres` (кількість знаків після коми), позиції коми в рядках операндів `z1` та `z2`, прапорці `op_plusb` та

or_minusb, за допомогою яких здійснюється фільтр дій, необхідних для виконання операцій лише множення та ділення.

3.2.1. Допоміжні функції калькулятора

Усі функції модуля виконання арифметичних операцій мають зрозумілу назву та прокоментовані у програмі для більш кращого розуміння коду. Їх можна умовно поділити на дві великі групи: основні та допоміжні. Допоміжні функції виконують дії, спільні для декількох арифметичних операцій, та використовуються головним чином для уникнення повторювальних дій. Це може частково утруднити розуміння коду програми іншими спеціалістами, але чіткі назви і наявність коментарів не дають втратити сутність змісту при розбиранні чужого коду. Перелічимо всі допоміжні функції, які використовуються для здійснення арифметичних операцій.

`function ntoch(i:byte):string` – перетворює цифру на символ (наприклад, 5 – «5», 15 – «F» і т. д.). Вхідний параметр – число типу `byte` (найменший цілий тип), повертає результат у якості рядка (`string`). Принцип дії функції ґрунтується на перевірці умови для вибору однієї з двох операцій шляхом порівняння вхідного числа: якщо число більше 9, то до нього додається 55 (оскільки коди великих літер латинського алфавіту в таблиці символів починаються з 65) і функції присвоюється результат виконання команди `chr` (отримання символу за його кодом). В іншому випадку, якщо вхідне число менше 10, функція отримує значення вхідної цифри, перетвореної на рядок;

`function chton(ch:char):byte` – функція обернена до `ntoch` (перетворює вхідний символ `ch` у ціле число типу `byte`). На початку підпрограми створюється множина символів `set1`, в яку за допомогою циклічної операції додаються символи цифр десяткової СЧ. Після цього здійснюється перевірка вхідного символу: якщо він входить у множину, то функція отримує значення його коду в таблиці символів шляхом виконання операції перетворення символу в число, в іншому випадку дана підпрограма повертає

результат виконання команди `ord(ch)-55` (отримує числовий код символу). На відміну від попередньої функції `ntoch`, від отриманого коду символу необхідно відняти число 55, оскільки великі латинські літери в таблиці символів починаються з номера 65;

`function normalize(var s1,s2:string):byte` – функція вирівнювання (або «нормалізації») вхідних операндів з комою, повертає позицію коми операнду з найбільшою цілою частиною. Результат роботи функції раніше використовувався у програмі для визначення позиції коми в результаті виконання деяких арифметичних операцій (зокрема додавання), але в ході більш детального тестування було, виявлено, що такий спосіб отримання необхідної позиції є правильним не в усіх випадках, тому він був замінений нами на інший більш надійний. На початку даної підпрограми у `ГЗ z1` та `z2` поміщуються значення позиції коми вхідних операндів (якщо операнд має лише цілу частину, то в змінну записується його довжина, збільшена на 1). Після цього здійснюється вирівнювання довжини цілої частини вхідних чисел: до числа з найкоротшою цілою частиною додається символ «0». Наступними діями є отримання довжин дробових частин операндів за допомогою виконання команди `length(s1)-pos(comma,s1)` (подібна команда виконується і для другого операнду); якщо кома відсутня, відповідній змінній присвоюється нульове значення. Результати виконання обох команд зберігаються відповідно в `z1` та в `z2`. Далі в кінець операнду з найкоротшою дробовою частиною додаються символи «0» і коми видаляються з операндів. Перетворення операндів за допомогою функції `normalize` є необхідним при виконанні операцій додавання та віднімання, а також залежних від них операцій множення та ділення, над числами з різною довжиною цілої або дробової частин;

`procedure eqsize(var s1,s2:string)` – процедура, подібна до вище описаної функції `normalize`. На відміну від попередньої, дана підпрограма здійснює «нормалізацію» лише цілих частин вхідних операндів та не повертає ніяких значень. Змінені операнди дублюються в глобальних змінних `se1` та `se2`.

`function zerodelb(s:string):string` – функція видаляє зайві символи «0» на початку вхідного рядкового числа і повертає змінений рядок. Дана підпрограма працює лише з цілими числами. Принцип її дії ґрунтується на тому, що за допомогою циклічної команди `while s[i]='0' do inc(i)` (перед цим змінній `i` присвоєно значення 1) визначається кількість нульових символів на початку рядка, після чого символи видаляються командою `delete(s,1,i-1)`. Якщо після виконання даної операції символів у рядку не залишилось, функція повертає значення «0». Виконання даної функції є необхідним для обробки результату виконання арифметичних операцій, оскільки в ході них у більшості випадків залишаються зайві нулі;

`function zerodele(s:string):string` – функція, подібна до попередньої, але видаляє зайві символи «0» в кінці рядка. Дана підпрограма виконує операції, протилежні функції `zerodelb`: спочатку змінній `i` присвоюється значення довжини вхідного рядка, після цього циклічною операцією `while s[i]='0' do dec(i)` здійснюється підрахунок символів «0» у вхідному операнді, які потім видаляються командою `delete(s,i+1,length(s)-i)`. Якщо результатами таких дій став пустий рядок, підпрограма повертає значення «0». Функція виконується лише для дробових чисел (перед її запуском здійснюється перевірка позиції коми в рядку);

`function compare(s1,s2:string):char` – функція, яка порівнює між собою строкові числа `s1` та `s2` і повертає символ «>», «<» або «=». Якщо операнди рівні між собою функція отримує значення «=», в іншому випадку виконуються наступні дії: здійснюється нормалізація операндів за допомогою функції `normalize`, циклічна команда `while s1[i]=s2[i] do inc(i)` визначає позицію останньої рівної цифри, після чого шляхом порівняння кодів символів рядка в позиції `i` визначається знак, який повертає дана функція;

`function check(s:string):boolean` – підпрограма перевірки відповідності рядкового числа заданій СЧ. Якщо операнд не правильний, функція повертає значення `false`, в іншому випадку отримує значення `true`. Спочатку

здійснюється перевірка коректності вказаної в програмі СЧ (вона не повинна бути меншою 2 та більшою 36), вхідний операнд не може бути пустим. Після цього створюється множина символів, доступних лише у вказаній СЧ: циклом `for i:=48 to 48+sys-1 do set1:=set1+[chr(i)]` додаються символи, доступні в так званих «цифрових» системах числення. Якщо СЧ більша за 10, то циклом `for i:=65 to 65+sys-9 do` додаються символи латинських літер. Далі здійснюється перевірка вхідного рядка: якщо був знайдений символ, відсутній у множині, функції присвоюється значення `false`, цикл переривається і відбувається вихід з підпрограми, в іншому випадку функція завершує свою роботу із значенням `true`;

`function sign(var s1,s2:string):boolean` – функція визначення знаку результату для операцій множення та ділення в залежності від знаків двох вхідних операндів `s1` та `s2` повертає значення `true` (результат від’ємний і до нього в майбутньому слід додати символ «-») або `false` (результат додатний і рядок результату ніяких змін не потребує). Підпрограма перевіряє перші символи обох операндів та визначає знак результату за традиційними правилами арифметики: якщо обидва операнди від’ємні, результат додатний, якщо один додатний і один від’ємний, результат від’ємний, у іншому випадку результат додатний. Символи знаків видаляються із вхідних операндів, функція не додає знак до результату, а лише перевіряє початкові операнди перед виконанням арифметичним дій.

Кожна арифметична операція, за виключенням множення, реалізована декількома функціями, які отримують в якості вхідних операндів два рядки `s1` та `s2` і повертають результат також рядкового типу. Тепер опишемо механізми здійснення всіх арифметичних операцій у різних СЧ.

3.2.2 Арифметичні операції додавання та віднімання

Операція додавання реалізована за допомогою двох функцій: `plus` та `op_plus`. Перша з них є основною і включає в себе лише дії для перевірки

коректності введених даних і визначення знаку результату. На її початку виконується перевірка коректності операндів за допомогою вже описаної нами функції `check`, при невідповідності хоча б одного з двох чисел підпрограма завершується зі значенням `Enter correct operands!`. Далі слідує операція визначення знаку результату, і в залежності від нього здійснюється вибір допоміжних функцій – `or_plus` (додавання чисел без знаку) або `or_minus` (віднімання чисел без знаку). Звичайно, перед їх викликом видаляється знак числа, якщо воно від'ємне. Вибір функцій `or_plus` або `or_minus` при виконанні операції `plus` здійснюється за такими принципами:

- якщо обидва числа від'ємні або обидва додатні, викликається підпрограма додавання без знаку `or_plus`;
- якщо одне з чисел від'ємне, а інше – додатне, операнди віднімаються функцією `or_minus`.

Функція `or_plus` виконує додавання як цілих так і дробових чисел. На її початку визначається позиція коми вихідного результату додавання шляхом визначення найдовшої дробової частини одного з двох операндів. Кількість знаків після коми першого операнду записується командою `length(s1)-pos(comma,s1)` у змінну `z1`, другого операнду – відповідно командою `length(s2)-pos(comma,s2)` у змінну `z2`. В іншому випадку, при відсутності дробових частин, відповідні `z1` та `z2` отримують нульові значення. Після цього найбільше із значень, тобто `z1` або `z2`, присвоюється локальній змінній цілого типу `sr`. Далі відбувається нормалізація операндів за допомогою вже описаної функції `normalize` та процедури `eqsize`, коми видаляються і подальші операції виконуються як над цілими числами. Механізм виконання арифметичної операції додавання наступний:

Ініціалізація: змінним `p` та `q` присвоюються довжини вхідних операндів відповідно `s1` та `s2`; у змінній `i` зберігається кількість «десятків» (вона залежить від СЧ), яку необхідно додати до результату `r` (початкові значення `i` та `r` – 0);

Виконання циклу «`while (p>0) and (q>0)`» («поки не будуть оброблені

всі символи обох рядків»), у ході якого здійснюється наступна послідовність дій: за допомогою функції `chton` отримуються числові значення символів операндів `s1` та `s2` з індексами `p` та `q` і зберігаються у змінних `c` (для рядка `s1`) та `d` (для рядка `s2`), після чого `p` та `q` зменшуються на 1; далі вони разом з десятком `i` додаються до результату `r` за допомогою звичайної стандартної операції додавання (команда `r:=c+d+i`); виконується перевірка: якщо результат `r` більший за СЧ, яка менша на 1, то від результату віднімається вказана СЧ і кількість «десятків» `i` стає 1, в іншому випадку змінна `i` отримує нульове значення; за допомогою функції `ntoch` визначається символ результату `r` і додається до кінцевого рядка `e`.

Після виконання циклу відбувається перевірка, чи є «залишок» `i` (якщо є, то на початок кінцевого результату додається символ «1», який збільшує число на один розряд) і видалення зайвих символів «0» за допомогою команди `zerodelb`. Наступні дії пов'язані лише з роботою із комою та виконуються у випадку якщо прапорець `or_plusb` має значення `false` (якщо дана функція `or_plus` викликається через підпрограму множення, а не тільки для одноразового додавання чисел, `or_plusb` отримує значення `true`). Це пояснюється тим, що кома, поставлена процедурою додавання, зовсім не потрібна у результаті для операції множення, оскільки механізми визначення позиції коми в цих операціях різні. Отже, після встановлення коми за допомогою команди `insert(comma,e,length(e)-cp+1)` в уже вище визначеній позиції `cp` відбувається доробка результату: якщо кома є першим символом, до результату додається «0», якщо останнім – кома видаляється; у випадку коли після всіх цих дій кома відсутня, здійснюється повторний виклик функції `zerodelb`, інакше – видалення зайвого символу «0» здійснюється обома командами `zerodelb` та `zerodele`. Після цього функція отримує значення рядка `e`.

Функція `minus` виконує операцію віднімання цілих та дробових знакових чисел. Принцип її дії подібний до операції додавання: на початку виконується перевірка коректності вхідних операндів, після чого

визначається знак результату і викликаються відповідні підпрограми за певних умов:

- якщо обидва числа від’ємні або обидва додатні, вони віднімаються функцією `op_minus`;
- якщо одне число від’ємне, а одне додатне, операнди додаються за допомогою підпрограми `op_plus`, причому якщо перше число від’ємне, а друге додатне, то результат від’ємний.

На початку процедури `op_minus` здійснюється ініціалізація локальних змінних: `mc` типу `boolean`, яка відповідає за знак результату, отримує значення `false`, а цілочислова змінна `cp` (позиція коми) – значення 0. Подібно до вище описаної функції додавання, операція ділення використовує дану функцію віднімання, але деякі її команди є для неї зайвими. Тому вони фільтруються за допомогою ГЗ типу `boolean` `op_minusb`: функція порівняння чисел `compare` запускається тільки якщо `op_minus` викликається не через функцію ділення (`op_minusb` у стані `false`). Таке порівняння необхідно для визначення того факту, чи не віднімаємо ми від меншого числа більше: якщо перший операнд менший, а другий більший, вони міняються місцями за допомогою локального рядка `t` і змінна `mc` отримує значення `true` (після виконання арифметичних дій результат стане від’ємним і необхідно додати до нього символ «-»).

Подальші операції здійснюються для визначення позиції коми в результаті віднімання. Спочатку в глобальну змінну `z1` записується результат роботи команди `length(s1)-pos(comma,s1)` (якщо кома відсутня, `z1` отримує нульове значення), подібною операцією визначається зміст `z2`. Якщо `z1` більша за `z2`, змінній `cp` присвоюється значення `z1`, в іншому випадку – значення `z2`. Після таких дій виконується «нормалізація» (вирівнювання) операндів за допомогою вже описаних підпрограм `normalize` та `eqsize`, у ході яких коми видаляються. Далі слідує команда ініціалізації необхідних для циклу віднімання змінних цілого типу `integer`: у `n` записується довжина рядка `s1` (означає загальну кількість ітерацій циклу), у `r` (результат віднімання

окремих розрядів) та у і («позичена одиниця» правішого розряду) – 0.

Наступний алгоритм, реалізований нами у циклі «while (n>0)», повністю ідентичний способу віднімання чисел у стовпчик:

- цілочислові змінні k та p отримують числові значення n-х символів відповідно операндів s1 та s2 за допомогою допоміжної функції chton, причому від k ще й віднімається «позичена одиниця» і та кількість ітерацій циклу n зменшується на 1;
- подальші дії залежать від числа k: якщо воно дорівнює -1, то йому присвоюється значення sys-1, де sys – СЧ роботи калькулятора; якщо k навпаки більша за p, то до неї додається вказана СЧ та змінній i присвоюється значення 1, в іншому випадку i отримує значення 0, яке означає, що позичення розрядів не виконувались;
- результату віднімання r присвоюється різниця змінних k та p (команда «r:=k-p;») і в кінець рядка кінцевого результату s додається число r, перетворене на символ допоміжною функцією ntoch.

Після завершення циклу виконується доробка результату s: запускається допоміжна функція zerodelb для видалення зайвих символів «0» на початку рядка, якщо вони є та результат має бути цілим числом. Подальші команди виконуються, коли ГЗ op_minusb знаходиться в стані false (підпрограма віднімання op_minus викликана саме користувачем, а не операцією ділення): стандартна процедура insert вставляє кому в позицію length(s)-cp+1, циклічно видаляються зайві символи «0» і коми з обох боків кінцевого рядка s, при необхідності додається невисначаючий «0» і змінюється знак результату. У кінці присвоюється значення змінної s.

3.2.3 Арифметичні операції множення та ділення

Підпрограма mult здійснює множення цілих та дробових знакових чисел у всіх допустимих СЧ. Подібно до функцій вище описаних арифметичних операцій, на її початку здійснюється перевірка коректності введених даних та ініціалізація ГЗ і ЛЗ. Глобальна змінна op_plusb отримує

значення `true`, яке означає, що дана функція викликатиме підпрограму додавання `op_plus`, яка повинна повернути суму чисел без урахування коми та знаку. Локальна змінна `ms`, «відповідальна» за знак числа, ініціалізується із значенням `false`, після чого, якщо хоча б одне з чисел від'ємне, їй присвоюється відповідне значення допоміжною вже описаною нами функцією `sign`. Далі визначається позиція коми результату множення: у глобальну цілочислову змінну `z1` заноситься результат роботи команди `length(s1)-pos(comma,s1)`, який визначає кількість знаків після коми першого операнду; аналогічна команда виконується для другого операнду: `z2:=length(s2)-pos(comma,s2)`. Ліз `n1`, в якій зберігається позиція коми результату множення, отримує значення суми `z1` та `z2`, після чого коми видаляються з операндів (при їх наявності). Якщо першим символом операнду є «0», то здійснюється видалення зайвих символів за допомогою функції `zerodelb`.

Перед самим циклом множення виконується вирівнювання операндів процедурою `eqsize` та починається ініціалізація необхідних змінних: у `n` заноситься довжина першого операнду `s1`, у `l1` – довжина другого `s2`, змінні `r` (результат звичайного множення розрядів), `i` та `il` (кількість символів «0», які необхідно додати до суми результатів відповідно першого та другого циклів) отримують значення 0; результат всієї операції множення зберігається в рядку `p2`, який перед запуском циклу приймає значення «0».

Реалізований нами алгоритм множення нагадує спосіб виконання даної арифметичної операції у стовпчик. Основний цикл перетворює усі розряди першого операнду, використовуючи результат роботи другого вкладеного циклу, який проходить по всім розрядам другого операнду. Головний цикл «`while l1 > 0`» починається з ініціалізації змінних, необхідних для роботи другого вкладеного циклу: змінна `b` типу `byte` отримує числове значення `l1`-го символу числа `s2` за допомогою функції `chton`; змінній `n` присвоюється довжина рядка `s1`; сума розрядів зберігається в рядковій змінній `p`, яка перед запуском другого циклу отримує значення «0»; цілочисловий лічильник

кількості символів «0» і, необхідний у другому вкладеному циклі також ініціалізується із значенням 0; значення кількості кроків основного циклу $l1$ зменшується на 1, після чого починає роботу другий вкладений цикл.

У ньому виконуються дії в такій послідовності:

- цілочисловій змінній a присвоюється результат роботи допоміжної функції `chton` аргументом якої є n -й символ числа $s1$ ($a:=chton(s1[n])$);
- зменшується кількість кроків n на 1;
- у змінну r потрапляє добуток чисел a та b ;
- змінна c отримує результат цілочислового ділення r на систему числення sys ;
- від числа r віднімається добуток c на sys і присвоюється змінній d : $d:=r-(c*sys)$;
- у рядок e потрапляє конкатенація двох рядків, отриманих у результаті обробки командою `ntoch` чисел c та d : $e:=ntoch(c)+ntoch(d)$;
- збільшується кількість символів «0», які необхідно додати до рядка e (i);
- якщо i більша за 0, виконується цикл `for j:=1 to i-1 do e:=e+'0'`;
- у рядковій змінній p наращується сума чисел r та e шляхом виконанням команди `or_plus(p,e)`;
- завершення вкладеного циклу і продовження виконання основного.

Після перелічених дій у головному циклі здійснюються наступні операції: збільшується кількість символів «0», які необхідно додати в кінець суми p , отриманої у вкладеному циклі (змінна $i1$); якщо число $i1$ більше за 0, до суми p циклічно додається символ «0» у кількості $i1-1$; рядкова змінна $p2$ отримує суму чисел $p2$ та p ., тобто суму результатів обох циклів.

По завершенню роботи достатньо складного циклу множення виконується доробка результату $p2$: якщо його довжина менша або дорівнює сумі кількості знаків після коми обох операндів $n1$, отриманій перед основним циклом множення, то на початок результату додаються символи «0» у кількості $n1-length(p2)$; якщо $n1$ більша за 0, командою `insert(comma,p2,length(p2)-n1+1)` у результат вставляється кома; при

наявності дробової частини видаляються зайві символи «0» у кінці результату; видаляється кома, якщо вона є останнім символом у $p2$, і визначається знак результату за допомогою змінної ms типу `boolean`; функції `mult` присвоюється значення рядка $p2$.

Операція ділення реалізована за допомогою трьох функцій: `divide` (ділення з остачею), `edivide` (ділення у стовпчик) та `dividewr` (ділення без остачі). Головною з них є `divide`, яка циклічно викликає `edivide`, яка в свою чергу використовує `dividewr`. Почнемо опис реалізації операції ділення надвеликих цілих та дробових чисел у всіх СЧ з функції `divide`.

Як прийнято при реалізації подібних операцій на початку основної підпрограми ділення виконується перевірка коректності вхідних операндів функцією `check` та ініціалізуються необхідні змінні: глобальна `op_minusb` отримує значення `true` (при операції віднімання не використовуватиметься можливість обробки дробових операндів); якщо хоча б одне з чисел від'ємне, у змінну `ds` типу `boolean` записується результат роботи допоміжної функції обробки знаку `sign` (в іншому випадку `ds` дорівнює `false`).

Перед виконанням основного циклу ділення дробові числа оброблюються нами наступним чином:

- у ГЗ $z1$ та $z2$ записується кількість знаків після коми обох операндів відповідно $s1$ та $s2$: $z1 := \text{length}(s1) - \text{pos}(\text{comma}, s1)$;
- найбільша з двох кількостей зберігається в змінній pc ;
- виконуються дії, щоб позбутися коми в обох або в одному з чисел: формується рядок dp із початковим змістом «1», в кінець якого циклом `for` додаються символи «0» у кількості pc , отриманій на попередньому кроці; обидва числа перемножуються з рядком dp , у результаті чого вони стають цілими числами.

Після завершення функцією перелічених підготовчих операцій виконуються дії для формування результату ділення із заданою точністю: у рядок кінцевого результату x записується результат роботи функції ділення у стовпчик `edivide`, після чого в його кінець додається кома, та до глобальної

змінної *od*, яка використовувалась попередньою функцією для збереження остачі від ділення, додається символ «0»; виконується головний цикл ділення «while (*prec*>0) and (*od*<>'00')» («поки не буде досягнута задана користувачем точність та остача не нульова»), який включає в себе наступні дії:

- частка від ділення *od* на друге число *s2* заноситься в рядок *r*: *r:=edivide(od,s2)*;
- рядок кінцевого результату *x* поєднується з рядком *r* (*x:=x+r*);
- до остачі від ділення *od* додається символ «0»;
- задана точність *prec* зменшується на 1.

Далі виконується доробка результату: при необхідності видаляється зайва кома, якщо вона є останнім символом, додається знак в залежності від стану вже описаної змінної *ds*, видаляються зайві символи «0» в дробовій частині рядка результату.

Зупинимось більш детально на функції ділення в стовпчик *edivide*, яка використовується головною підпрограмою *divide*. На її початку здійснюється перевірка, чи не дорівнює один з операндів «0», оскільки на нуль ділити не можна. Наступний цикл формує рядок *o* із символів першого операнду *s1*, доки число, сформоване в ньому не буде більшим або рівним другому операнду *s2* та не будуть оброблені усі символи *s1*. Якщо довжина першого операнду *s1* дорівнює довжині другого операнду *s2*, рядковій змінній результату *r* присвоюється частка від цілочислового ділення *dividewr s1* на *s2*, в іншому випадку виконується цикл, перед яким встановлюється початкова кількість оброблюваних символів *i* на 1 та рядку *o* присвоюється перший символ операнду *s1*. Отже, цикл «while *i*<length(*s1*)+1», який імітує ділення у стовпчик, включає в себе наступні дії:

- число *o* ділиться на число *s2* за допомогою функції *dividewr i* частка зберігається в рядку *t*: *t:=dividewr(o,s2)*;
- до рядка результату *r* додається отриманий рядок *t*: *r:=r+t*;
- у *t* записується добуток останнього числа майбутньої частки *r* на операнд *s2*: *t:=mult(r[length(r)],s2)*;

- від числа o віднімається число t : $o := \text{minus}(o, t)$;
- перевірка рядка o : якщо на попередньому кроці був отриманий нульовий результат, рядок o стає пустим: $\text{if } o = '0' \text{ then } o := ''$;
- перехід на наступний символ шляхом збільшення лічильника i ;
- до рядка o додається i -й символ операнду $s1$.

Після завершення даного циклу або в іншому випадку після отримання частки за допомогою функції `dividewr` при необхідності виконується видалення зайвих нулів на початку рядка результату r за допомогою функції `zerodelb`. Підпрограма `edivide` повертає вміст змінної r .

Функція ділення чисел без остачі `dividewr` має дуже коротку реалізацію та базується на простому циклі «`while (compare(s1,s2)='>' ) or (compare(s1,s2)='=')`» («поки перший операнд більше другого або дорівнює йому»): на кожному кроці від першого числа $s1$ віднімається друге $s2$ і рядковий лічильник r збільшується на 1 за допомогою команди `or_plus(r,'1')`; після виконання циклу глобальній змінній `od`, в якій зберігається залишок від ділення, присвоюється значення рядка $s1$; якщо в ході циклічних перетворень рядок цілочислової частки r виявився пустим, його вміст змінюється на «0».

Описаний спосіб ділення довгих чисел у всіх СЧ має один серйозний недолік: частка дуже великих чисел обчислюється занадто довго. На відміну від інших арифметичних операцій, час її розрахунку може досягати кількох хвилин. Наприклад, при тестуванні помічено, що ділення між собою двох 34-символьних чисел тривало близько двох хвилин, а усі інші операції виконувались набагато швидше (наприклад, операція множення тих самих чисел виконувалась менше однієї секунди). Ми припускаємо, що така низька швидкодія спричинена не дуже вдалим способом отримання цілої частини за допомогою функції `dividewr`. Не дивлячись на свою дуже коротку реалізацію, вона циклічно повторює дуже велику кількість однакових трудомістких дій, особливо це помітно при діленні дуже довгого числа на дуже коротке. Тому операція ділення є найповільнішою з усіх інших і потребує повної переробки для досягнення необхідної швидкодії для роботи з надвеликими числами.

Для тестування усіх арифметичних операцій нами була створена консольна програма, в якій виконуються усі дії за допомогою вище описаних підпрограм, помічається загальна тривалість обробки однакових операндів чотирма алгоритмами. Результати виконання арифметичних операцій, реалізованих нами мовою програмування Pascal, повністю співпадають з результатами майже аналогічного онлайн-калькулятора numsys.ru, який здійснює ті самі операції, але працює з числами не довше 50 символів. Тому числа, довше 50 символів нами не випробувались, оскільки не було знайдено аналогічного ПЗ для порівняння. Результати виконання обчислень представлені на рисунку 4, при їх порівнянні слід обов'язково звертати увагу на вказану СЧ («Number system») та кількість знаків після коми операції ділення.

```

C:\Users\Maximka\Desktop\ConsoleTest\project1.exe
Number system: 9 Divide precision: 25
-1,2345167 + -5678 = -5680,2345167
-1,2345167 - -5678 = 5676,6543722
-1,2345167 / -5678 = 0,0001870453740187736711168
-1,2345167 * -5678 = 7257,7103442
TIME: 0 hours 0 minutes 1 seconds Operand 1 length: 10 Operand 2 length: 5
Number system: 11 Divide precision: 25
-0,0002 + 3,2 = 3,1AA9
-0,0002 - 3,2 = -3,2002
-0,0002 / 3,2 = -0,00006A06A06A06A06A06A
-0,0002 * 3,2 = -0,00064
TIME: 0 hours 0 minutes 0 seconds Operand 1 length: 7 Operand 2 length: 3
Number system: 17 Divide precision: 25
3,1234 + 100,456 = 103,5794
3,1234 - 100,456 = -GE,332D
3,1234 / 100,456 = 0,031170496662576EE4EE708EC
3,1234 * 100,456 = 312,67FDB57
TIME: 0 hours 0 minutes 0 seconds Operand 1 length: 6 Operand 2 length: 7
Number system: 20 Divide precision: 25
3,1234 + 10000000000 = 10000000003,1234
3,1234 - 10000000000 = -3JJJJJJJJG,1HGG
3,1234 / 10000000000 = 0,00000000031234
3,1234 * 10000000000 = 31234000000
TIME: 0 hours 0 minutes 2 seconds Operand 1 length: 6 Operand 2 length: 11
Number system: 22 Divide precision: 25
12345167 + 5678 = 1234A7DF
12345167 - 5678 = 1233LGKL
12345167 / 5678 = 4C9L,AL57J9IG4C5H064BHKEHOD8DK
12345167 * 5678 = 5HEK23KC8BC
TIME: 0 hours 0 minutes 0 seconds Operand 1 length: 8 Operand 2 length: 4
Number system: 26 Divide precision: 25
35,1 + 1,8 = 36,9
35,1 - 1,8 = 33,J
35,1 / 1,8 = 2B,D
35,1 * 1,8 = 44,F8
TIME: 0 hours 0 minutes 0 seconds Operand 1 length: 4 Operand 2 length: 3
Number system: 32 Divide precision: 25
3,6 + 0,027 = 3,627
3,6 - 0,027 = 3,5TP
3,6 / 0,027 = 1DV,34UKNDV34UKNDV34UKNDV34UK
3,6 * 0,027 = 0,072A
TIME: 0 hours 0 minutes 0 seconds Operand 1 length: 3 Operand 2 length: 5
Number system: 33 Divide precision: 25
Number1: ABCDEF9876543210,FEDCBA1 Number2: 10F1E2D3C4B5A60798,89706A5B4C3D2E1F
+ = 10PCQFRILCIBFA39A8,NNKCHK6B4C3D2E1F
- = -104N1LVLT3W51US87,PRQK50484C3D2E1F
* = AG30RHAKIFRC46L58KWMQTCWAIANB049,PEFKAAN9NL8C8AV8JU885JF
/ = 0,0AGNUOGRHQ5VGADGEWK4M9FBI
TIME: 0 hours 0 minutes 15 seconds Operand 1 length: 24 Operand 2 length: 35
Number system: 36 Divide precision: 30
N1: T01U2VW34QR5S67N089P,EKFLD1ICBHA N2: W0V1U2T3S4R5Q6P7O8N9,MALBKCJDIEHFG
+ = TW2P4PY6WVIBICWCGMZ,0V0WXM1PTVXPG
- = S4OZ11T2ZCM001ZIFZZMF,S9U9T6YYT2YUUK
* = PSQN9DLNBKCPV046CHPHXKTXVILEHRU2WNTL9EH602,V07YPMH9KAYU895EZKH8TY9EG
/ = W,LOFGVT93K8BHE11FXNAGTMTIYLOESNI
TIME: 0 hours 2 minutes 11 seconds Operand 1 length: 34 Operand 2 length: 34
Done!

```

Рисунок 3.2.3 –Виконання арифметичних операцій у різних СЧ.

3.2.4. Обробка арифметичних виразів

Калькулятор містить також дві функції для обробки арифметичних виразів, які складаються з кількох дій. Перша з них `strproc` виділяє операції в дужках і передає їх у другу функцію `strprocwб`, яка послідовно виконує арифметичні дії без дужок.

Обробка дужок здійснюється циклом `while`, який працює поки рядок вхідного виразу `s` не пустий. Результат виконання обчислень без дужок зберігається у змінній `t2`, яка на кожному кроці стає пустою. Якщо був знайдений символ відкриваючої дужки, у цілочислову змінну `p` поміщується її позиція, у рядок `t1` потрапляють усі символи перед дужкою шляхом виконання команди `copy(s,1,p-1)`, і скопійований вираз видаляється із вхідного рядка разом із відкриваючою дужкою. Подібним чином додаються символи перед закриваючою дужкою в рядок `t2`. Після цього циклом `for` здійснюється пошук дужок у `t2`, вони видаляються і отриманий арифметичний вираз обчислюється іншою функцією `strprocwб`. До рядка кінцевого виразу `r` додається зміст `t1`, який містить числа та дії за межами дужок, та `t2`, який містить результат обчислень у дужках. Після завершення таких циклічних перетворень і отримання результатів усіх операцій у дужках здійснюється обчислення виразу `r` функцією `strprocwб`.

Дана функція знаходиться на стадії тестування, при її роботі можливі помилки (особливо при виконанні операцій «великої вкладеності»), тому за замовчуванням можливість вводити символи дужок відключена у програмі (їх можна задіяти, активувавши перемикач «Тестовий режим» у вікні користувача). Підпрограма не включає в себе перевірку коректності усього арифметичного виразу (якщо користувач введе неправильний вираз для розрахунку або помилково не поставить дужку, калькулятор може невірно його розрахувати або видати повідомлення про помилку).

Функція обробки арифметичних виразів без дужок `strprocwб` використовується для обчислень коректно введених арифметичних виразів, які складаються з більше ніж двох операндів. З її допомогою можна

обчислювати вирази, які складаються з цілих та дробових знакових чисел і мають вигляд $a+b-c$ або $a+(-b)*c/-d$ (тобто припускається, що користувач увів від'ємний знак числа прямо після символу арифметичної операції без дужок). Дана функція має більш складний принцип дії, ніж `strprocwb`.

Перед основним циклом формування результату здійснюється ініціалізація змінної `b` типу `boolean`, призначеної для встановлення знаку числа; цілочислова змінна `pr` приймає значення вказаної користувачем точності ділення (ГЗ `pres`) для її майбутнього відновлення, оскільки у процесі розрахунків `pres` зменшується; здійснюється пошук і видалення всіх дужок; формується множина `set1`, до якої додається символ коми, а також цифри вказаної користувачем СЧ способом, висвітленим при описі допоміжної функції `check`; здійснюється перевірка першого символу вхідного рядка: якщо він дорівнює «-», `b` отримує значення `true` та знак видаляється; формується перший операнд `s1`, до якого додаються лише ті символи, які входять у множину `set1` (це означає, що якщо користувач увів число, яке є некоректним для вказаної СЧ, воно буде сформовано не повністю, що може спричинити невірний результат); якщо `b` має позитивне значення, до першого операнду додається знак, і сформоване число видаляється з вхідного рядка. Основний цикл обчислення арифметичного виразу виконується доки рядок вхідного виразу не пустий, включає в себе дії у такій послідовності:

- змінна `b`, яка «відповідає» за знак, отримує значення `false`;
- на початку кожної ітерації циклу глобальна змінна `pres` (кількість знаків після коми у результаті роботи операції ділення) відновлюється із вже описаної `pr`;
- глобальні `op_plusb` та `op_minusb` отримують значення `false`;
- очищується рядок другого числа `s2`;
- у змінну `c` типу `char` записується перший символ вхідного рядка всього виразу, який означає арифметичну операцію, необхідну для виконання, після чого він видаляється;
- формується число `s2` разом із його знаком шляхом виконання вже вище

описаних операцій для $s1$, яке видаляється із основного рядка;

- після отримання необхідного другого операнду виконується арифметична операція, вказана символом c та її результат циклічно оновлюється у $s1$.

Після завершення циклу функція `strprocwb` отримує значення рядка $s1$.

На даному етапі розвитку проекту описана підпрограма має один серйозний недолік: відсутня послідовність команд перевірки коректності введеного виразу. У випадку введення чисел з непридатними символами операнди отримують лише цифри перед першим неправильним розрядом, а решта символів просто ігнорується, це може спричинити некоректність результату, про що користувач не буде повідомлений. На щастя, екранна клавіатура дозволяє вводити лише ті символи, які є доступними у вибраній СЧ, що дозволяє частково уникнути помилок при формуванні арифметичного виразу користувачем. У майбутньому планується вдосконалення даної функції.

3.3 Конвертування чисел з однієї системи числення в іншу

Перетворення цілих та дробових додатних чисел з однієї СЧ в іншу здійснюється за допомогою функцій вище описаних арифметичних операцій. Воно реалізоване нами за допомогою двох підпрограм: основної `numberconvert` та допоміжної `convertfromdec`. Перша з них переводить числа у десяткову СЧ за такою формулою: $A_{10} = a_{n-1} \cdot q^{n-1} + \dots + a_0 \cdot q^0 + a_{-1} \cdot q^{-1} + \dots + a_{-m} \cdot q^{-m}$, де a – розряд числа, n – його порядковий номер, q – початкова СЧ; для розрядів цілої частини n додатне, для розрядів дробової частини n від’ємне. Друга функція `convertfromdec` перетворює числа з десятикової СЧ у будь-яку іншу (не меншу за 2 та не більшу за 36), для цілої частини послідовно повторюючи операцію її ділення на СЧ певну кількість разів, поміщуючи на початок рядка результату його залишки у вигляді відповідних символів; дробова частина перетворюється навпаки шляхом її множення на СЧ і результат формується з цілих частин добутків. Окрім усіх процедур, функцій та глобальних змінних калькулятора конвертер систем числення

додатково використовує глобальні цілочислові змінні `sys1` та `sys2` для запам'ятовування відповідно вхідних та цільових СЧ, а також додаткову допоміжну функцію `power` для піднесення розрядів чисел у степінь з цілим додатним або дробовим показником.

Допоміжна функція `power` має коротку просту реалізацію, у якості вхідних параметрів вона отримує два рядки: `s` (вхідне число) та `p` (показник степеня, який завжди знаходиться в десятковій СЧ). На її початку ініціалізується змінна `b` типу `boolean`, яка отримує значення `true`, якщо степінь від'ємний (у цьому випадку символ «-» видаляється з рядка `p`). Початковий рядок результату `t` приймає значення «1», оскільки добуток не може починатись з порожнього рядка або з 0. Після цього виконується цикл `for i:=1 to strtoint(p) do t:=mult(s,t)`, в якому вхідне число `s` множиться само на себе `p` разів. Якщо змінна `b` отримала позитивне значення, число 1 ділиться на отриманий результат для піднесення у від'ємний степінь. У якості результату функція повертає рядок `t`.

Зупинимось більш детально на підпрограмі конвертування чисел з десятикової СЧ у будь-яку іншу `convertfromdec`, яка працює з точністю, вказаною користувачем у відповідному полі з назвою «Точность деления». Перед початком двох основних циклів перетворень виконуються наступні дії:

- ГЗ `sys` отримує значення 10, це означає, що усі арифметичні операції будуть виконуватись лише у десятковій СЧ;
- змінна `pt` запам'ятовує точність ділення `rges` для її майбутнього відновлення;
- рядковий залишок від ділення `od` ініціалізується із значенням «0»;
- допоміжні рядки `r`, `t`, `t1`, `fs` отримують пусті значення;
- при відсутності дробової частини у змінну `fs` потрапляє весь вхідний рядок, в іншому випадку ціла частина виділяється командою `copy(s,1,pos(comma,s)-1)` та у вхідному рядку вона замінюється на «0».

Цикл конвертування цілої частини виконується поки не буде досягнута задана точність та змінна `fs` не отримає значення «0». Він включає в себе такі

команди: у рядку *fs* на кожному кроці зберігається частка від ділення *fs* на цільову СЧ *sys2*, перетворену з цілого числа на рядок; якщо після такої дії рядок залишку від ділення *od* пустий, йому присвоюється «0»; у цілочислову змінну *t1* потрапляє вміст залишку *od*, перетворений з рядка на число стандартною функцією *strtoint*; до рядка *r* додається символ, отриманий допоміжною функцією *ntoch* від *t1*; значення точності *prec* зменшується на 1.

Після отримання цілої частини у цільовій СЧ виконується ще кілька підготовчих дій до другого циклу: рядковій змінній майбутнього кінцевого результату *r1* присвоюється результат роботи попереднього циклу *r*; рядок *r* очищується; точність конвертування та ділення *prec* відновлюється із змінної *pt*. Якщо перед основним циклом була встановлена наявність дробової частини, то починає роботу другий цикл «while (*prec*>0) and (*fp*<>'0')», в якому виконуються наступні інструкції:

- у рядок *t* потрапляють всі цифри перед комою шляхом виконання команди *t:=copy(s,1,pos(comma,s)-1)*;
- вхідний рядок *s*, ціла частина в якому перед початком даного циклу була замінена на «0», множиться на цільову СЧ *sys2* і результат зберігається в *s*;
- формується рядок *ip* з цілої частини *s* вже описаним способом;
- ціле число *pt* отримує значення *ip*, перетворене на *integer*;
- у змінній *ip* відновлюється значення *pt*, отримане в результаті роботи команди *ntoch*;
- вміст *ip* додається в кінець результату *r*;
- якщо рядок *s* містить лише цілу частину (*pos(comma,s)=0*), *fp* отримує значення «0», що стає ознакою останньої ітерації даного циклу;
- якщо рядок *ip*, доданий до результату, не дорівнює цілій частині *t*, отриманій на початку ітерації циклу, то ціла частина рядка *s* замінюється на «0»;
- зменшується точність *prec* на 1, і при необхідності повторюються вказані операції.

Після завершення циклу отримана дробова частина r додається після коми рядка кінцевого результату $r1$, якщо другий цикл виконувався; здійснюється доробка результату шляхом видалення зайвих символів «0» на початку рядка та зайвої коми в кінці рядка (при необхідності). Отже, функція `convertfromdec` отримує значення $r1$.

Основна функція конвертера `numberconvert` служить для перетворення числа з будь-якої СЧ у десяткову. Усі арифметичні операції для перетворення здійснюються у вхідній СЧ $sys1$, тому змінна sys отримує значення $sys1$, її значення дублюється також у цілому числі nst (для відновлення). Якщо вхідна СЧ $sys1$ дорівнює 10, то дана функція завершується з результатом роботи вище описаної підпрограми `convertfromdec`. У іншому випадку виконуються наступні підготовчі операції: якщо перший символ вхідного рядка – «0» і в ньому є кома, то така ціла частина в ньому замінюється на «1»; рядкова змінна $r1$, яка буде використовуватись у циклі, отримує пусте значення; дробова частина вхідного операнду виділяється в рядок ds командою `copy(s,pos(comma,s)+1,length(s)-pos(comma,s))`, після чого вона видаляється із вхідного рядка s ; цілочислова змінна k отримує довжину s , меншу на 1; рядок результату ініціалізується з символом «0»; запускається цикл «while $k > -1$ » для перетворення цілої частини вхідного рядка у цільову СЧ, містить наступні дії:

- у рядок $r1$ потрапляє результат зведення початкової СЧ у степінь k ;
- символ s отримує цифру вхідного числа s з індексом $length(s)-k$, яка перетворюється функцією `chton` на ціле число n ;
- рядок $r1$ оновлює своє значення шляхом множення самого себе на перетворене у рядок n ;
- до рядка кінцевого результату додається число $r1$: $t1 := plus(r1, t1)$;

Якщо рядок ds не порожній, то після виконання команди $k := -length(ds)$ починає роботу цикл конвертування дробової частини «for $i := 1$ to $length(ds)$ », який виконує ті самі операції, що й попередній цикл, але початкова СЧ $sys1$ зводиться у від'ємний степінь i .

Після одного або двох таких циклів (у залежності від числа) виконується перевірка умови: якщо цільова СЧ не 10, то початкова sys1 отримує значення 10 і результат попередніх перетворень t1 додатково оброблюється функцією `convertfromdec`, після чого при необхідності виконуються дії для корекції результату, зокрема, видаляються зайві нулі в цілій частині; значення початкової СЧ sys1 відновлює значення, введене користувачем, із змінної `nst`.

Результати перетворень зображені на рисунку 3.3.1:

```

C:\Users\Maximka\Desktop\ConsoleTest\project1.exe
Number 0,1726354 in system 8 =
0,3D675FFFFFFFFFA18D in system 16 with precision 20
TIME: 0 hours 0 minutes 1 seconds Number length: 9

Number 1B2A3647589 in system 12 =
119903776377 in system 10 with precision 20
TIME: 0 hours 0 minutes 0 seconds Number length: 11

Number G1F2E3,D4C5B67A89 in system 17 =
201CEEC,BAB1036855D2C3755860 in system 15 with precision 20
TIME: 0 hours 0 minutes 3 seconds Number length: 17

Number 0,L2K3J4I5H6G7A8B9CFDE in system 22 =
0,YKWGYDJVQ059YSQ1VR68 in system 36 with precision 20
TIME: 0 hours 1 minutes 12 seconds Number length: 22

Number FEDCBAPQRB20,17BHNUV in system 32 =
3J3IG2A32GQOM,110FM7L81MKC560760C7 in system 27 with precision 20
TIME: 0 hours 0 minutes 2 seconds Number length: 20

Number 908,63 in system 10 =
1110001100,10100001010001111010111000010100011 in system 2 with precision 35
TIME: 0 hours 0 minutes 0 seconds Number length: 6

DONE!

```

Рисунок 3.3.1 – Результати роботи конвертера систем числення

Усі надані результати майже повністю співпадають з результатами, наданими онлайн-сервісом `numsys.ru`: розбіжності починаються лише після 10-12 знаків після коми, що можна усунути збільшенням точності в інтерфейсі користувача. Також при тестуванні помічена недостатня швидкодія обробки чисел з довгою дробовою частиною: вона може тривати в деяких випадках до кількох хвилин, що спричинено недостатньою раціональністю реалізації алгоритму ділення.

3.4 Модуль розрахунку ваги металопрокату

Сукупність певних елементів інтерфейсу користувача віконної версії калькулятора дають змогу автоматизувати розрахунок ваги металопрокату за

вказаними у відповідні поля даними. Даний модуль складається зі списків «Вид металла», «Тип і розмір металопроката», полів «Кількість м.», «Товщина», «Питома вага (кг/п.м)», «Вага з урахуванням кількості» та кнопки «Обчислити вагу». В залежності від вибраного елементу списку «Вид металу» активуються усі інші поля та списки: якщо користувач вибрав «Листовий», поле «Товщина» стає активним, а два «випадні» списки «Тип і розмір металопроката» та «Питома вага (кг/п.м)» вимикаються; в усіх інших випадках, при виборі «Сортовий» або «Труби», неактивним стає лише поле «Товщина».

Елементи списку «Тип и розмір металопрокату» завантажуються з певних файлів в залежності від вибраного виду металу:

- якщо користувачем вибрано «Сортовий», перелік завантажується з текстового файлу `Names&Dimensions_sorted.txt`. При виборі елементів списку «Тип и розмір металопрокату» у полі «Питома вага (кг/п.м)» відображаються довідкові дані, які зчитуються з файлу `Weights_sorted.txt` циклом `for i:=0 to GostMetallType.ItemIndex do readln(f,s)`;
- якщо вибрано «Труби», список найменувань завантажується з файлу `Names&Dimensions_pipe.txt`, а довідкові дані отримуються з файлу `Weights_pipe.txt` тим самим способом.

При натисканні на кнопку «Вычислить вес» виконуються такі дії:

- система числення (поле `ns.text`) отримує значення 10, оскільки даний модуль працює лише в десятковій СЧ;
- в залежності від активності певних полів здійснюється перевірка коректності введених даних функцією `checkfield` типу `boolean`, яка реалізована подібно до функції `check` у модулі калькулятора, але перевіряє числа лише у десятковій СЧ, при помилці видає повідомлення: «Введіть коректні дані!», переставляє фокус на поле введення кількості (оскільки після натискання на кнопку він змінився), та повертає результат `false` (при правильності рядка результат – `true`);
- при коректності введеної інформації формується арифметичний вираз в

залежності від вибраного елементу списку «Вид металу»: якщо активним є елемент «Листовий», то вираз формується з чисел у полях «Кількість м.» та «Товщина» за формулою відповідно $\text{quantity.text} * \text{thickness.text} * 7,85$; у іншому випадку формула для розрахунку $\text{quantity.text} * \text{unit_weight.text}$ («Кількість м.» * «Питома вага (кг/п.м)»);

- сформований рядковий вираз `s` потрапляє у поле для введення арифметичних операцій (`ae.text:=s`);
- викликається процедура натискання на кнопку «= $\Rightarrow$ » і здійснюються обчислення із вказаними даними;
- поле «Вага з урахуванням кількості» отримує результат обчислень калькулятором і фокус повертається на поле «Кількість м.» (команда `quantity.SetFocus`).

Довідкова інформація, яка використовується даним модулем, може доповнюватись та редагуватись користувачем без зміни коду програми, хоча на даному етапі розвитку проекту така можливість реалізована не дуже зручно: найменування видів металопрокату та удільна вага кожного з них зберігаються у різних файлах. Тому користувач має пам'ятати, в якому саме файлі яка інформація знаходиться та при зміні даних слідкувати за відповідністю номерів рядків, що дуже зручно робити за допомогою розширених текстових реакторів, таких як Notepad++. На наш погляд, єдиним недоліком даного модуля є недостатня зручність зміни та доповнення довідкових даних.

3.5 Модулі розрахунків ваги, довжини діаметра та розривної сили каната або троса

Сукупність таких елементів інтерфейсу користувача як «випадні» списки «Розміри троса (мм)» та «Обчислити», поля «Вага 1 м троса (кг)», «Кількість метрів» та «Вага троса (кг)», кнопка «Обчислити вагу троса», а також відповідних їм підпрограм становить модуль розрахунку ваги троса. Він створений для надання можливості швидкого обчислення ваги троса за

заданою у програмі формулою та з використанням довідкових даних лише у десятковій СЧ.

Список «Розміри троса (мм)» включає в себе лише два елементи: «За окружністю» і «За діаметром». Елементи з даними завантажуються з файлу `rope_size&weight` в список, який знаходиться прямо біля нього, в залежності від вибраного пункту. Елементи даного списку розміщуються у вказаному текстовому файлі в такому вигляді: «За окружністю» «За діаметром»/дані. Отже, формування відповідних переліків здійснюється такими двома циклами:

```
if type_of_size.Text='по окружности' then
  while not eof(f) do begin
    readln(f,s);
    delete(s,pos(' ',s),length(s)-pos(' ',s)+1);
    rope_s.Items.Add(s);
  end else
    while not eof(f) do begin s:="";
      readln(f,s);
      delete(s,1,pos(' ',s));
      delete(s,pos('/',s),length(s)-pos('/',s)+1);
      rope_s.Items.Add(s);
    end;
```

де `rope_s` – назва списку, `f` – змінна типу `text` (відкритий файл), `s` – рядок.

В залежності від вибраних елементів у поле «Вага 1 м троса (кг)» потрапляють відповідні довідкові дані, які в текстовому файлі знаходяться після символу «/».

Список «Розрахувати» містить у собі два пункти, вибір одного з яких змінює назви деяких елементів інтерфейсу: при виборі пункту «вага» поле, яке знаходиться поряд, отримує назву «Кількість метрів», кнопка перейменовується на «Обчислити вагу троса», а останнє поле результату – на

«Вага троса (кг)»; вибір елементу «довжину» відповідно формує назви «Кількість кілограм», «Обчислити довжину троса» та «Довжина троса (м)».

Натискання на кнопку під назвою «Обчислити вагу троса» або «Обчислити довжину троса» викликає попередню перевірку поля з назвами «Кількість метрів» або «Кількість кілограм» вище згаданою функцією `checkfield` (у коді програми дане поле має лише одну назву `kgm`), встановлює значення поля СЧ на 10, формує арифметичний вираз таким чином: якщо користувач обчислює вагу, то вказана кількість метрів ділиться на вагу 1 м троса, в іншому випадку ці два поля перемножуються калькулятором, результат потрапляє в поле під назвою «Вага троса (кг)» або «Довжина троса (м)». Якщо користувач обчислював вагу, до поля результату роботи даного модуля додається надпис «кг», якщо довжину – «м».

Проаналізувавши роботу цього невеликого модуля можна стверджувати, що він реалізований більш вдало, ніж попередній, оскільки має більш досконалу систему доповнення та редагування довідкових даних: достатньо відкрити відповідний текстовий файл і, пам'ятаючи про те що числами перед пробілом є окружність, після пробілу – довжина, а після символу «/» – вага (яка є спільною для «окружності» та «діаметра»), можна легко змінити та доповнити існуючі дані, при цьому не заплутуючись у різних файлах. Проте некоректно змінений текстовий файл може спричинити помилки при роботі даного модуля.

Останній «інженерний» модуль поєднує в собі можливості розрахунку розривного зусилля і діаметра канату. Він складається з двох списків «Режим роботи каната» та «Тип каната в наявності», поля з довідковою інформацією «Коефіцієнт», поля «Зусилля (кг)» для введення даних користувачем, двох полів для результату: «Розривне зусилля каната (кг)» і «Діаметр каната (мм)».

Список «Режим роботи каната» завантажується з файлу `rope-cond.txt`, в якому довідкова інформація для відповідного режиму роботи знаходиться після символу «/». При виборі елементу даного списку довідкові дані

потрапляють у поле «Коефіцієнт». Список «Тип каната в наявності» складається з трьох полів: «Трьохпрядний конопляний», «Сталевий (6х36 - 216 дротів)» та «Сталевий (6х37 - 222 дротів)», вибір одного з яких відповідно ініціалізує один з трьох файлів: rope-diameter_hemp.txt або rope-diameter_iron216.txt або rope-diameter_iron222.txt та частково повторює дії кнопки запуску обчислень для оновлення результату.

Кнопка «Обчислити розривне зусилля та діаметр каната» після всіх стандартних підготовчих дій формує простий арифметичний вираз за формулою «Зусилля (кг)» * «Коефіцієнт», присвоює результат його обчислення полю «Розривне зусилля каната (кг)», при необхідності округлює його командою `inttostr(round(strtfloat(e3.text)))` (оскільки користувачу не потрібна дуже точна вага), після чого в залежності від вибраного елементу списку «Тип каната в наявності» відкриває для читання один з трьох вище згаданих файлів та здійснює пошук діаметра каната наступним циклом:

```
while not (eof(f)) do begin
  readln(f,s);
  if strtoint(bursting_effort.text)<=strtoint(copy(s,1,pos('/',s)-1))
  then begin delete(s,1,pos('/',s));
    rope_d.text:=s; closefile(f); exit; end; end;
```

в якому `f` – файл типу `text`, `s` – зчитаний рядок, а `bursting_effort` – поле «Розривне зусилля каната (кг)». Якщо у циклі не відбувся вихід з підпрограми, процедура встановлює полю «Діаметр каната (мм)» значення «Велике зусилля!». Той самий цикл виконується при виборі іншого елементу списку «Тип каната в наявності» за умовою якщо поле «Розривне зусилля каната (кг)» не є пустим (тобто кнопка для відповідного розрахунку вже була натиснута користувачем після введення коректних даних).

Висновки до розділу

Розроблений програмний комплекс виконання інженерних розрахунків являє собою комплексне програмне рішення, призначене для автоматизації

обчислювальних процесів на металообробних підприємствах шляхом реалізації широкого спектру функціональних можливостей, що включають виконання арифметичних операцій з надвеликими числами у різних системах числення, конвертування числових даних між системами числення та здійснення специфічних інженерних розрахунків з використанням довідкової інформації. Програмний комплекс було розроблено за допомогою вільно розповсюджуваного середовища розробки Lazarus та мови програмування Free Pascal для використання у віконному режимі, що забезпечує його доступність та зручність експлуатації на підприємствах різного рівня технічного оснащення.

Принципи роботи програми та інтерфейс користувача побудовані на основі концепцій візуального, об'єктно-орієнтованого та процедурного програмування, що забезпечує високий рівень зручності взаємодії користувача з програмним забезпеченням. Структура програмного комплексу умовно поділяється на три основні функціональні частини, а саме калькулятор для виконання арифметичних операцій, конвертер систем числення та модулі інженерних розрахунків, які на поточному етапі розвитку проекту містять можливості обчислення ваги металопрокату і троса, довжини троса, розривного зусилля та необхідного діаметра каната. Усі екранні елементи інтерфейсу користувача розташовані на одній формі та можуть бути умовно поділені на три групи, включаючи клавіші екранного введення символів чисел та арифметичних операцій, кнопки та поля введення модулів інженерних розрахунків, а також допоміжні елементи управління. Особливістю реалізації інтерфейсу є використання єдиної процедури для всіх кнопок екранного введення, що зчитує відповідний символ з поля підказки натиснутої кнопки та передає його для введення у відповідне поле, така реалізація є найбільш раціональною та сприяє уникненню зайвого однотипного коду порівняно з попередніми версіями програми.

Модуль виконання арифметичних операцій представляє собою сукупність основних і допоміжних функцій, призначених для здійснення

обчислень над цілими та дробовими знаковими числами надвеликої довжини у всіх системах числення від двійкової до тридцяти шосткової на основі принципів довгої арифметики. Для збереження операндів у пам'яті та виконання різних операцій над ними використовується тип даних `string`, максимальна довжина якого завдяки використанню середовища розробки Lazarus разом з відповідною директивою становить до двох гігабайт, що фізично забезпечує можливість виконання обчислень над числами дуже великої довжини. Вибір рядкового типу даних для збереження операндів головним чином обумовлений наявністю різноманітних готових зручних функцій для обробки та перетворень введеної інформації, можливістю використовувати не тільки звичні традиційні символи цифр та достатньою максимальною довжиною операндів для практичних потреб підприємства.

Допоміжні функції калькулятора виконують дії, спільні для декількох арифметичних операцій, та використовуються головним чином для уникнення повторювальних дій, що частково може утруднити розуміння коду програми іншими спеціалістами, однак чіткі назви функцій і наявність коментарів не дають втратити сутність змісту при розбиранні програмного коду. До складу допоміжних функцій входять процедури перетворення цифр на символи та навпаки, функції нормалізації та вирівнювання вхідних операндів з різною довжиною цілої або дробової частин, процедури видалення зайвих нульових символів на початку та в кінці рядкових чисел, функція порівняння строкових чисел між собою, підпрограма перевірки відповідності рядкового числа заданій системі числення та функція визначення знаку результату для операцій множення та ділення в залежності від знаків вхідних операндів. Кожна арифметична операція за виключенням множення реалізована декількома функціями, які отримують в якості вхідних параметрів два рядки та повертають результат також рядкового типу після виконання відповідних обчислень.

Арифметичні операції додавання та віднімання реалізовані за допомогою основних функцій `plus` та `minus`, які включають в себе дії для

перевірки коректності введених даних і визначення знаку результату, та допоміжних функцій `op_plus` і `op_minus`, які виконують відповідні операції над числами без урахування знаку. Вибір допоміжних функцій здійснюється за певними принципами залежно від знаків операндів, зокрема якщо обидва числа від'ємні або обидва додатні викликається підпрограма додавання без знаку, а якщо одне з чисел від'ємне а інше додатне операнди віднімаються функцією віднімання без знаку. Механізм виконання операції додавання базується на алгоритмі додавання в стовпчик з урахуванням переносу розрядів, при цьому спочатку визначається позиція коми вихідного результату шляхом визначення найдовшої дробової частини одного з двох операндів, здійснюється нормалізація операндів, коми видаляються і подальші операції виконуються як над цілими числами з циклічною обробкою кожного розряду. Операція віднімання реалізована подібним чином з використанням алгоритму віднімання в стовпчик, при цьому додатково здійснюється порівняння чисел для визначення необхідності зміни їх місць та встановлення відповідного знаку результату.

Арифметичні операції множення та ділення мають більш складну реалізацію порівняно з операціями додавання та віднімання. Підпрограма множення здійснює обчислення шляхом використання двох вкладених циклів, де основний цикл обробляє всі розряди першого операнду використовуючи результат роботи другого вкладеного циклу який проходить по всім розрядам другого операнду, при цьому реалізований алгоритм нагадує спосіб виконання множення у стовпчик з накопиченням проміжних сум. Позиція коми в результаті множення визначається як сума кількостей знаків після коми обох операндів, після чого виконується доробка результату шляхом додавання необхідних нулів на початок результату та видалення зайвих символів в кінці. Операція ділення реалізована за допомогою трьох функцій, де головна функція `divide` циклічно викликає функцію ділення у стовпчик `edivide`, яка в свою чергу використовує функцію ділення без остачі `dividewr`, при цьому дробові числа спочатку перетворюються на цілі шляхом

множення на відповідний коефіцієнт, після чого виконується ділення з заданою користувачем точністю. Важливо відзначити що операція ділення має серйозний недолік у вигляді недостатньої швидкодії при обчисленні частки дуже великих чисел, оскільки функція отримання цілої частини циклічно повторює велику кількість однакових трудомістких дій, що робить операцію ділення найповільнішою з усіх інших і вимагає повної переробки для досягнення необхідної швидкодії.

Обробка арифметичних виразів здійснюється за допомогою двох функцій, де перша виділяє операції в дужках і передає їх у другу функцію яка послідовно виконує арифметичні дії без дужок. Функція обробки виразів з дужками працює циклічно виділяючи вирази всередині дужок та обчислюючи їх за допомогою функції обробки виразів без дужок, після чого результати підставляються в загальний вираз для подальших обчислень. Функція обробки арифметичних виразів без дужок використовується для обчислень коректно введених арифметичних виразів які складаються з більше ніж двох операндів, при цьому перший операнд формується з урахуванням його знаку та вказаної системи числення, після чого основний цикл послідовно виділяє арифметичні операції та другі операнди виконуючи відповідні обчислення та циклічно оновлюючи результат. Дані функції знаходяться на стадії тестування і за замовчуванням можливість використання дужок відключена в програмі.

Конвертування чисел з однієї системи числення в іншу здійснюється за допомогою функцій арифметичних операцій та реалізоване двома підпрограмами, де основна функція `numberconvert` переводить числа у десяткову систему числення за відповідною формулою з урахуванням позиційних розрядів, а допоміжна функція `convertfromdec` перетворює числа з десятикової системи числення у будь-яку іншу допустиму систему шляхом послідовного ділення цілої частини на цільову систему числення та множення дробової частини на неї. Конвертер систем числення додатково використовує глобальні цілочислові змінні для запам'ятовування вхідних та

цільових систем числення, а також допоміжну функцію піднесення розрядів чисел у степінь з цілим додатним або дробовим показником. Результати перетворень майже повністю співпадають з результатами наданими онлайн-сервісами при цьому розбіжності починаються лише після десяти-дванадцяти знаків після коми що можна усунути збільшенням точності в інтерфейсі користувача, однак при тестуванні помічена недостатня швидкодія обробки чисел з довгою дробовою частиною що спричинено недостатньою раціональністю реалізації алгоритму ділення.

Модуль розрахунку ваги металопрокату становить сукупність елементів інтерфейсу користувача які надають можливість автоматизувати розрахунок ваги металопрокату за вказаними у відповідні поля даними, при цьому елементи списків завантажуються з відповідних текстових файлів в залежності від вибраного виду металу. Активація елементів інтерфейсу здійснюється в залежності від вибраного виду металопрокату, зокрема при виборі листового металу стає активним поле введення товщини а списки деактивуються, в усіх інших випадках неактивним стає лише поле товщини. При натисканні на кнопку обчислення ваги здійснюється перевірка коректності введених даних, формується арифметичний вираз в залежності від вибраного елемента списку виду металу за відповідною формулою, після чого викликається процедура обчислення калькулятором та результат відображається у відповідному полі. Довідкова інформація яка використовується даним модулем може доповнюватись та редагуватись користувачем без зміни коду програми, хоча на даному етапі розвитку проекту така можливість реалізована не дуже зручно оскільки найменування видів металопрокату та удільна вага кожного з них зберігаються у різних файлах що вимагає уважного слідкування за відповідністю номерів рядків при зміні даних.

Модулі розрахунків ваги довжини діаметра та розривної сили каната або троса представляють собою спеціалізовані функціональні компоненти програмного комплексу призначені для швидкого обчислення відповідних

параметрів за заданими у програмі формулами та з використанням довідкових даних. Модуль розрахунку ваги троса включає елементи інтерфейсу для вибору розмірів троса по окружності або по діаметру, при цьому елементи з даними завантажуються з текстового файлу в залежності від вибраного пункту з використанням циклічних операцій для формування відповідних переліків. В залежності від вибраних елементів у відповідне поле потрапляють довідкові дані про вагу одного метра троса, при цьому користувач може обирати між обчисленням ваги за заданою довжиною або довжини за заданою вагою що змінює назви відповідних елементів інтерфейсу та формули розрахунку. Модуль розрахунку розривного зусилля і діаметра каната поєднує можливості обчислення обох параметрів з використанням списків режимів роботи каната та типів каната в наявності, при цьому довідкові дані завантажуються з відповідних текстових файлів а обчислення здійснюються за формулою множення зусилля на коефіцієнт з подальшим пошуком відповідного діаметра каната в довідковій інформації.

Таким чином розроблений програмний комплекс виконання інженерних розрахунків являє собою функціонально завершене програмне рішення яке успішно реалізує поставлені цілі щодо автоматизації обчислювальних процесів на металообробних підприємствах. Програма забезпечує виконання арифметичних операцій над надвеликими числами з високою точністю в усіх допустимих системах числення, надає можливості конвертування числових даних між різними системами числення та здійснює специфічні інженерні розрахунки з використанням довідкової інформації яка може бути відредагована та доповнена користувачами без необхідності зміни програмного коду. Реалізація програмного комплексу базується на раціональних алгоритмах довгої арифметики. Виявлені в ході тестування недоліки зокрема недостатня швидкодія операції ділення надвеликих чисел та необхідність вдосконалення функцій обробки арифметичних виразів з дужками визначають напрямки подальшого розвитку проекту для досягнення оптимальної продуктивності та функціональності програмного забезпечення.

РОЗДІЛ 4. БІБЛІОТЕКИ ДЛЯ РОБОТИ З ДОВГОЮ АРИФМЕТИКОЮ

Сучасні інженерно-наукові обчислення дедалі частіше виходять за межі можливостей стандартних числових типів мов програмування. Задачі високоточної обробки даних, моделювання фізичних процесів, криптографічні застосування та чисельні методи вимагають використання чисел великої розрядності та керованої точності обчислень. У зв'язку з цим особливої актуальності набувають бібліотеки довгої арифметики, які забезпечують підтримку обчислень із практично необмеженою кількістю цифр.

На сьогодні існує низка програмних бібліотек, серед яких найбільш відомими є GNU Multiple Precision Arithmetic Library (GMP), та їхні похідні MPIR, MPFR тощо. Дані бібліотеки надають ефективні реалізації базових арифметичних операцій, використовуючи оптимізовані алгоритми, що автоматично адаптуються до розрядності чисел. Водночас універсальність таких рішень супроводжується зростанням складності використання, накладними витратами на перетворення типів даних та обмеженою пристосованістю до прикладних інженерних задач.

У цьому розділі розглядаються сучасні бібліотеки довгої арифметики, наведено їх функціональні можливості, переваги й обмеження, а також обґрунтовано доцільність розробки та застосування спеціалізованих реалізацій, орієнтованих на конкретні класи інженерно-наукових розрахунків. Також описуються власні рішення як спроба створити більш ефективні аналоги функцій популярних бібліотек.

4.1 Загальна структура проекту для роботи з довгою арифметикою

Наш програмний продукт з довгої арифметики має чітко вибудовану модульну структуру, у якій кожен файл відповідає за окремий рівень функціональності – від базових допоміжних засобів до високорівневого тестування та порівняльного аналізу з бібліотечними реалізаціями.

Центральною точкою входу до програми є файл `main.cpp`, який

підключає заголовок `bignum.h` і виконує роль керуючого модуля. Саме в ньому ініціалізуються тестові сценарії, запускаються генератори випадкових чисел, викликаються операції довгої арифметики та здійснюється вимірювання їх коректності й продуктивності. Таким чином, `main.cpp` інтегрує всі підсистеми проекту в єдиний експериментально-обчислювальний комплекс.

Основний клас власної реалізації довгої арифметики описаний у файлі `bignum.h`. Він визначає інтерфейс класу `bignum`, який інкапсулює базові арифметичні операції – додавання, віднімання, множення і ділення над наддовгими числами, представленими у вигляді символьних масивів. Передавання довжини числа окремим параметром дозволяє уникнути повторного обчислення розміру рядка та підвищити продуктивність при масових обчисленнях. Оголошення глобальних констант `MaxBignumLength` і `MaxFractPartLength` задає єдині обмеження на розмір чисел у всьому проекті та забезпечує узгодженість між модулями.

Файл `OptimizedBignum.h` відповідає за альтернативну, бібліотечно орієнтовану реалізацію арифметичних операцій. Клас `OptimizedBignum` має такий самий інтерфейс, як і `bignum`, що дозволяє без змін у тестовій логіці порівнювати власні алгоритми з реалізаціями на основі бібліотек MPFR та MPFR. У цьому ж файлі оголошено допоміжні функції перетворення десяткових рядків у раціональне подання `mpq_t` та назад, а також функції піднесення до степеня й комплексної перевірки результатів. Такий підхід дозволяє використовувати бібліотечну арифметику як еталон при перевірці коректності власної реалізації.

Генерація та допоміжна обробка наддовгих чисел винесена у файл `RandomBignum.h`. Він містить набір функцій для порівняння чисел, пошуку позиції десяткової точки, виведення чисел, перестановки операндів, а також генерації випадкових цілих і дробових наддовгих чисел у заданих діапазонах або з певною довжиною. Наявність цих засобів забезпечує повноцінне стохастичне тестування арифметичних операцій на різних вхідних даних.

Найнижчий рівень проекту реалізовано у файлі `accessories.h`, який виконує роль універсального допоміжного модуля. У ньому зосереджені кросплатформні засоби введення з клавіатури, робота з часом, генерація випадкових чисел стандартних типів, операції зі строками та масивами, а також підключення стандартних бібліотек і бібліотек довгої арифметики GMP, MPIR та MPFR. Саме тут здійснюється ініціалізація генераторів випадкових чисел і задання загальної точності для обчислень з плаваючою комою.

Засоби компіляції проекту реалізовані у вигляді двох окремих сценаріїв для різних інструментальних ланцюжків. Файл `GCC MSYS2.bat` призначений для компіляції за допомогою компілятора `g++` у середовищі MSYS2. У ньому виконуються очищення попередніх збірок, компіляція всіх вихідних файлів у єдиний виконуваний файл та лінування з бібліотеками GMP, GMPXX та MPFR. Сценарій також містить заготовки для можливої збірки у вигляді динамічної бібліотеки, що свідчить про потенційну розширюваність проекту. Після успішної компіляції програма автоматично запускається.

Альтернативний сценарій `Visual Studio 2022.bat` забезпечує збирання проекту засобами Microsoft Visual C++. У ньому виконується ініціалізація середовища Visual Studio, компіляція вихідних файлів за стандартом C++17 та оптимізаціями, а також лінування з бібліотеками MPIR і MPFR у форматі, сумісному з Visual Studio. Окремо налаштовуються шляхи до бібліотек і виконується ручне очищення об'єктних файлів. Такий підхід дозволяє запускати проект як у відкритому середовищі GCC, так і в середовищі MSVC без зміни вихідного коду.

Проект розроблено з урахуванням вимог кросплатформності, що дозволяє використовувати його без змін вихідного коду в різних операційних системах – Windows та Linux. Це досягається завдяки застосуванню стандарту мови C++, використанню переносимих бібліотек довгої арифметики (GMP/MPIR/MPFR), а також умовній компіляції у допоміжних модулях для обробки системно-залежних функцій введення з клавіатури.

Файл `accessories.h` містить відповідні директиви препроцесора, які автоматично підлаштовують реалізацію низькорівневих функцій під конкретну платформу, що гарантує однакову логіку роботи програми незалежно від середовища виконання.

Сценарій компіляції для Linux реалізовано у вигляді Bash-скрипта, який автоматизує процес збирання та запуску програми. На початковому етапі скрипт змінює обмеження на розмір стеку, збільшуючи його до 16 МБ. Це є важливим з огляду на специфіку роботи з наддовгими числами і великою кількістю рекурсивних або глибоко вкладених викликів, що можуть використовувати значні обсяги стекової пам'яті. Налаштування стека виконується засобами оболонки та не потребує прав адміністратора.

Далі здійснюється компіляція всіх вихідних файлів проекту за допомогою компілятора `g++` з увімкненим режимом попереджень. Усі реалізації – керуючий модуль, допоміжні засоби, власна довга арифметика, оптимізований модуль та генератор випадкових чисел – компілюються в один виконуваний файл і лінуються з бібліотеками `gmp` та `gmpxx`, які забезпечують бібліотечну підтримку високоточної арифметики. Після завершення компіляції виконується перевірка коду повернення для визначення успішності збирання, що підвищує надійність автоматизованого сценарію.

У разі успішної компіляції програма автоматично запускається, після чого сценарій аналізує код завершення. Окремо передбачено виявлення аварійного завершення через помилку сегментації, що є особливо важливим при тестуванні операцій над великими обсягами даних та роботі з динамічною пам'яттю. Такий контроль дозволяє оперативно діагностувати проблеми коректності реалізації алгоритмів довгої арифметики.

У цілому структура проекту демонструє чітке розподілення функцій між різними модулями, можливість порівняння власних і бібліотечних реалізацій, а також високий рівень відтворюваності результатів експериментів, що є важливим для інженерно-наукових досліджень з довгої

арифметики, а наявність окремих сценаріїв компіляції для Windows і Linux підтверджує повну кросплатформеність проекту та забезпечує відтворюваність експериментальних результатів у різних середовищах розробки і виконання, що є суттєвою перевагою для дослідницьких і навчальних застосувань.

4.2 Генератори випадкових наддовгих чисел

Під час тестування алгоритмів довгої арифметики важливу роль відіграють генератори випадкових чисел великої розрядності. Вони дозволяють створювати широкий спектр тестових даних, зокрема знакові цілі та дробові числа довільної довжини, що є необхідним для оцінки коректності та продуктивності арифметичних операцій. Якість таких генераторів безпосередньо впливає на повноту тестового покриття та достовірність експериментальних результатів, особливо при стохастичному тестуванні.

Для формування тестових даних у роботі використано функцію *char* GetRandomBignum(int length, bool SignMinus, bool FloatingPoint)*, та функцію з тим самим ім'ям, але з іншими аргументами: *char* GetRandomBignum(char* min, char* max)*.

Перша з них генерує випадкове наддовге число у символьному представленні із заданою довжиною (змінна *CountOfDigits*). Функція підтримує як цілі, так і дробові числа (задається логічним параметром *FloatingPoint*), а також надає можливість визначення знаку числа (логічна змінна *SignMinus*).

На першому етапі обчислюється фактична довжина майбутнього представлення з урахуванням можливого знака «-» та десяткової точки. Під число динамічно виділяється пам'ять, що забезпечує коректну роботу для чисел максимальної допустимої довжини та виключає переповнення буфера.

У випадку генерації від'ємного числа у нульову позицію записується символ мінуса, після чого починається формування цифрової частини. Для чисел із дуже малою кількістю цифр (менше двох) дробова форма

автоматично вимикається, що запобігає створенню некоректних або вироджених переважно дробових значень.

У режимі генерації дробових чисел випадковим чином визначається позиція десяткової точки. При цьому реалізовано додаткові обмеження:

- перша цифра не може бути нульовою, щоб уникнути неінформативних чисел виду 0.xxx;
- кінцева цифра дробової частини гарантовано ненульова (від 1 до 9), що виключає проблему тривіальних нульових залишків та спрощує перевірку операцій ділення.

У разі генерації цілого числа перша цифра також вибирається ненульовою, а всі наступні є рівномірно випадковими в діапазоні від 0 до 9. Такий підхід гарантує збереження заданої розрядності числа та унеможлиблює появу ведучих нулів. Після завершення генерації рядок коректно завершується нуль-термінатором і повертається як динамічно виділений символьний масив.

Функція `char* GetRandomBignum(char* min, char* max)` реалізує генерацію випадкового наддовгого числа у текстовому поданні, яке гарантовано лежить у заданому діапазоні `min – max`, причому діапазон може включати як додатні, так і від’ємні, цілі та дробові значення. Уся логіка побудована так, щоб жоден з крайових або проміжних випадків не приводив до виходу за межі діапазону чи до синтаксично некоректного числа.

На верхньому рівні `GetRandomBignum` відповідає за знакову логіку, тоді як усі допоміжні функції працюють лише з невід’ємними числами без знака. Спочатку виділяється пам’ять під результуюче число та допоміжний буфер `ZeroOrOne`, який використовується як технічний мінімум або максимум ("0" або "1") під час генерації чисел у змішаних знакових діапазонах. Змінна `SignMinus` фіксує факт наявності мінуса у результаті.

Далі розглядаються всі можливі комбінації знаків меж. Якщо і `min`, і `max` від’ємні, то знак «-» відкидається, а генерація зводиться до задачі отримання числа в діапазоні $|\max| - |\min|$. Після цього знак мінуса примусово

додається до результату, оскільки будь-яке число в такому діапазоні також буде від'ємним. Якщо ж $\min$ від'ємне, а $\max$ додатне, то випадковим чином з рівною ймовірністю обирається генерація або від'ємного числа в діапазоні $-\min - 1$, або додатного в діапазоні $0 - \max$. Якщо ж обидві межі невід'ємні, то число генерується напряму без знакових трансформацій.

Безпосередня побудова числового значення виконується функцією `SetRandomUnsignedBignum`, яка формує невід'ємну частину числа відповідно до символічних меж $\min$ і $\max$. Вона спершу аналізує позицію десяткової точки в обох межах, визначаючи, чи є число цілочисловим або дробовим. Далі завжди формується ціла частина викликом `SetRandomIntBignum`, після чого, за потреби, додається дробова частина.

Функція `SetRandomIntBignum` відповідає за генерацію цілої частини числа, дотримуючись обмежень в рамках $\min$ та $\max$. Вона випадково вибирає довжину числа в допустимому інтервалі, після чого порозрядно будує число. Якщо початкові цифри $\min$ і $\max$ збігаються, вони копіюються в результат примусово. У момент появи першої відмінності виконується контрольований випадковий вибір цифри з відповідного допустимого діапазону. Подальші цифри заповнюються вільно, але так, щоб не виникало ведучих нулів і щоб число не перевищило межі. Така поетапна стратегія дозволяє уникнути повного порівняння чисел і значно спрощує контроль кордонів.

Функція `SetRandomFloatBignum` додає дробову частину, гарантуючи, що дробове представлення також не вийде за межі. Для цього дробові частини $\min$ та $\max$ виділяються окремо, вирівнюються по довжині шляхом доповнення нулями, після чого аналогічно цілій частині виконується поцифрова генерація з контролем меж. Довжина дробової частини обирається випадково, але в допустимих межах, і наприкінці усуваються зайві нулі з кінця, щоб уникнути некоректних представлень, наприклад, 12.3400 або 12..

Підпрограма `SetRandomUnsignedBignum` виконує роль інтегратора між

цілочисловою і дробовою генерацією. Вона приймає рішення, чи потрібно додавати дробову частину, залежно від того, чи мають межі десяткову точку і чи може випадкове ціле значення вийти за межі. У складних прикордонних випадках вона динамічно конструює нові штучні межі з максимально дозволеною або мінімально дозволеною дробовою частиною, забезпечуючи коректність навіть тоді, коли ціла частина дорівнює межовому значенню.

Після завершення генерації у `GetRandomBignum` відбувається детальна перевірка: контролюється коректність символів, відсутність зайвих нулів, правильність розташування десяткової точки, а також те, що число не перевищує `max` і не є меншим за `min`. У разі виявлення помилки програма аварійно завершується з діагностичним повідомленням, що підкреслює дослідницьку спрямованість і орієнтацію на виявлення рідкісних дефектів.

У підсумку дана система функцій реалізує надійний, детерміновано коректний генератор випадкових наддовгих чисел, який підтримує знак, довільну довжину, дробову частину та строгі числові межі. Це робить її придатною для масового тестування алгоритмів довгої арифметики, де статистична випадковість має поєднуватися з абсолютною коректністю результатів.

4.3 Арифметичні операції над наддовгими числами

У рамках даної роботи були створені власні реалізації базових арифметичних операцій над числами довільної довжини, поданими у вигляді символьних рядків. На відміну від бібліотечних підходів, де внутрішня структура чисел прихована, запропоновані функції безпосередньо оперують цифрами, десятковою крапкою та знаком, що дозволяє детально контролювати обчислювальний процес, аналізувати проміжні стани та забезпечувати роботу як з цілими, так і з дробовими наддовгими числами. Реалізації побудовані за класичними алгоритмами “стовпчикової” арифметики, адаптованими для довільної розрядності та символьного представлення.

4.3.1 Операція додавання

Функція `plus` класу `bignum` реалізує операцію додавання двох великих чисел, представлених у вигляді масивів символів з підтримкою як цілих так і дробових чисел з урахуванням їх знаків, при цьому алгоритм базується на принципі додавання в стовпчик з обробкою переносу розрядів та вирівнюванням операндів за позицією десяткової коми.

Робота алгоритму починається з обробки граничних випадків: якщо один операнд дорівнює нулю, функція повертає копію другого числа, уникаючи зайвих обчислень. Це спрощує логіку, зменшує накладні витрати.

Далі виконується аналіз знаків операндів. Якщо обидва числа від'ємні, запам'ятовується факт негативного результату, знак мінус вилучається з обох рядків, а подальше додавання відбувається для їх модулів. У випадку різних знаків додавання коректно зводиться до віднімання: сума додатного та від'ємного числа еквівалентна різниці їх абсолютних значень, тому викликається функція `minus` з відповідним порядком аргументів. Таким чином досягається повторне використання коду та усувається дублювання.

Після визначення знакової ситуації виділяється буфер для результату достатньої довжини з запасом. Далі обчислюються позиції десяткової точки в обох операндах. Саме на цьому етапі алгоритм розгалужується на два принципово різні сценарії: додавання чисел із та без дробової частини.

Якщо обидва операнди є цілими, додавання виконується з правого краю рядків у напрямку до старших розрядів. Алгоритм враховує можливу різну довжину чисел: якщо одне число довше за інше, спочатку складаються співпадаючі розряди, а залишкові розряди довшого числа просто переносяться в результат з урахуванням переносу. На кожному кроці обчислюється сума двох цифр і переносу, а новий перенос формується при перевищенні значення дев'яти. У найпростішому випадку, коли довжини чисел однакові, обробка відбувається симетрично без додаткових перевірок.

Якщо хоча б один з операндів містить дробову частину, процедура стає складнішою. Спершу обчислюється кількість цифр у дробовій та цілій

частинах кожного числа. На основі цих даних визначається загальна довжина результату та позиція десяткової точки. Алгоритм починає додавання з дробової частини, вирівнюючи довжини шляхом концептуального доповнення коротшої дробової частини нулями. Додавання дробових цифр виконується справа наліво з урахуванням переносу, після чого в результат примусово вставляється десяткова крапка.

Після завершення роботи з дробовою частиною алгоритм переходить до додавання цілих частин. Тут знову враховується різна кількість цифр операндів. Як і в цілочисловому випадку, спочатку складаються спільні розряди, а потім, за необхідності, обробляються залишкові старші цифри довшого числа з урахуванням можливого переносу. На всіх етапах перенос поширюється строго з молодших розрядів у старші.

Після завершення додавання виконується нормалізація результату. З кінця рядка видаляються зайві нулі у дробовій частині, а якщо після цього десяткова крапка стає крайнім символом, вона також вилучається. Далі перевіряється фінальний перенос: якщо після обробки всіх розрядів він залишився рівним одиниці, у старшому розряді результату формується цифра '1'. Знак мінус додається якщо було визначено, що обидва операнди від'ємні.

На завершальному етапі виконується косметичне упорядкування рядка результату. Якщо на початку буфера залишився службовий символ-заповнювач, він зсувається, щоб забезпечити коректний початок числового представлення. Також відновлюються початкові вказівники на операнди, що гарантує відсутність побічних ефектів для коду, який викликав дану функцію. Отже, функція повертає динамічно виділений рядок, який містить точний результат додавання двох довільних за довжиною чисел у символьному форматі.

4.3.2 Операція віднімання

Функція `minus` класу `bignum` реалізує операцію віднімання двох наддовгих чисел, поданих у символьному вигляді, із повною підтримкою

знака та дробової частини. Логіка побудована таким чином, щоб максимально коректно відтворити письмове віднімання, доповнивши його попереднім аналізом чисел і зведенням задачі до найбільш простого для обчислень вигляду.

На початковому етапі функція перевіряє, чи є обидва операнди рівними. У такому випадку результатом віднімання завжди є нуль, тому одразу формується та повертається рядок "0", що дозволяє уникнути подальших обчислень. Далі виконується порівняння чисел. Якщо перше число менше за друге, фіксується, що результат буде від'ємним, і, залежно від комбінації знаків, операнди міняються місцями, щоб фактичне віднімання завжди виконувалось між більшим та меншим числом. Такий прийом суттєво спрощує алгоритм, оскільки дозволяє уникнути обробки від'ємних результатів на рівні цифрової арифметики.

Після цього аналізуються знаки операндів. Якщо обидва числа від'ємні, їх знаки тимчасово відкидаються, а самі числа за потреби переставляються місцями, оскільки віднімання двох від'ємних чисел еквівалентне відніманню їх модулів у зворотному порядку. Якщо ж перший операнд додатний, а другий від'ємний, операція віднімання зводиться до додавання модулів чисел, що забезпечує повторне використання логіки функції plus. У цьому випадку знак результату формується окремо, якщо він був визначений раніше як від'ємний.

Перед початком обчислень алгоритм виділяє буфер достатнього розміру для результату та визначає позиції десяткових крапок в обох числах. Якщо обидва операнди є цілими, застосовується класичний алгоритм віднімання справа наліво. Кожна пара цифр віднімається з урахуванням запозичення з наступного старшого розряду. У випадку, коли поточна цифра зменшувана з урахуванням запозичення менша за цифру від'ємника, виконується «позики» одиниці з наступного розряду, що еквівалентно додаванню десятка до поточної цифри. Після завершення віднімання усуваються провідні нулі, щоб результат набув канонічного вигляду.

Якщо хоча б один з операндів має дробову частину, функція переходить до складнішого сценарію. Спершу визначається кількість цифр у дробових та цілих частинах кожного числа. На основі цих даних обчислюється довжина результату та позиція десяткової крапки. Віднімання починається з дробової частини, причому коротша дробова частина концептуально доповнюється нулями. В залежності від співвідношення довжин дробових частин можливе попереднє формування «дев'яток» у результаті, що відповідає механізму запозичення при відніманні чисел з різною дробовою довжиною.

Після завершення обробки дробових розрядів у результаті вставляється десяткова крапка, а дробова частина нормалізується шляхом видалення зайвих нулів з кінця. Далі алгоритм переходить до віднімання цілих частин. Знову застосовується поцифрове віднімання з контролем запозичення та коректною обробкою різної кількості розрядів. Особлива увага приділяється ситуації, коли всі провідні цифри цілої частини стають нульовими – у такому разі вони видаляються, але гарантовано зберігається щонайменше одна значуща цифра.

На фінальному етапі до результату додається знак мінус, якщо раніше було встановлено, що різниця має бути від'ємною. Вихідний рядок приводиться до компактного та коректного представлення без надлишкових нулів, зайвих символів або некоректного розташування десяткової точки. Отже, функція повертає новий динамічно виділений рядок, який містить точний результат операції віднімання для двох довільних за довжиною чисел у символьному представленні.

4.3.3 Операція множення

Функція `multiply` реалізує множення двох наддовгих чисел, поданих у вигляді символьних рядків, і за своєю логікою дуже близька до класичного «шкільного» алгоритму множення в стовпчик, розширеного для підтримки знака та дробової частини. Уся обробка виконується на рівні символів, без

використання готових числових типів.

На самому початку функція виконує швидку перевірку на тривіальний випадок: якщо хоча б один із операндів дорівнює нулю і складається з одного символу, то результат множення також є нулем. У цьому випадку одразу виділяється мінімальний буфер, формується рядок "0" і функція завершується, уникаючи зайвих обчислень.

Далі виконується аналіз знаків операндів. Якщо лише один із двох чисел має від'ємний знак, результат множення буде від'ємним, що фіксується у змінній `SignMinus`. Після цього обидва числа, якщо вони були від'ємними, переводяться до додатного вигляду: знак мінус відкидається, а довжина рядка коригується. Таким чином, уся подальша арифметика працює лише з модулями чисел, а знак результату додається в кінці.

Після підготовки знаків виділяється пам'ять для результату та допоміжного масиву `multiplication`, який використовується для зберігання проміжних добутків. Довжина результату обчислюється із запасом і дорівнює сумі довжин операндів з урахуванням можливого знака. Масив результату попередньо ініціалізується символами '0', що значно спрощує подальше додавання часткових результатів.

Наступним кроком визначаються позиції десяткових крапок в обох числах. Якщо жоден з операндів не містить дробової частини, алгоритм переходить до спрощеного сценарію множення цілих чисел. У цьому випадку спочатку множиться перше число на останню цифру другого числа, формуючи початковий результат. Далі для кожної наступної цифри другого множника обчислюється частковий добуток у масиві `multiplication`, який відповідає одному «ряду» табличного множення. Цей частковий добуток зсувається на відповідну кількість позицій і додається до загального результату з урахуванням перенесення. Після завершення всіх ітерацій із початку результату послідовно видаляються провідні нулі для канонічного вигляду результату.

Якщо хоча б один з операндів є дробовим, активується складніша гілка

алгоритму. На цьому етапі підраховується загальна кількість значущих цифр без урахування десяткової крапки для кожного числа. За потреби операнди міняються місцями так, щоб перший мав не менше цифр, ніж другий, що зменшує кількість ітерацій і дещо оптимізує обчислення. Далі визначаються розміри дробових частин обох чисел і загальна довжина дробової частини результату, яка дорівнює їх сумі.

Позиція десяткової крапки в результаті визначається наперед і фіксується у масиві результату. Після цього множення виконується фактично так само, як і для цілих чисел, але з урахуванням того, що цифри з обох сторін десяткових крапок беруть участь у множенні. Алгоритм проходить по цифрах другого операнда справа наліво, формує часткові добутки для кожної цифри першого операнда, накопичує перенос і додає ці часткові результати до основного масиву результату, коректно обходячи позицію десяткової крапки.

Після завершення всіх операцій виконується нормалізація результату. Спочатку з кінця дробової частини видаляються всі зайві нулі; якщо після цього дробова частина виявляється порожньою, десяткова крапка також вилучається. Далі з початку цілої частини видаляються провідні нулі, але так, щоб не порушити коректність числа і не знищити єдину значущу цифру. Після цього довжина результату коригується відповідно до фактичного вмісту рядка.

На фінальному етапі, якщо під час аналізу знаків було встановлено, що результат має бути від'ємним, у нульову позицію рядка записується символ мінуса. Додатково виконується ще одна перевірка дробової частини: якщо вона присутня, з її кінця прибираються всі нульові символи, аби результат не містив надлишкових даних.

Отже, дана функція повертає новий динамічно виділений рядок, який містить точний результат множення двох наддовгих чисел у символьному представленні, з коректно обробленими знаком, десятковою частиною, переносами та мінімальною, канонічною формою запису.

4.3.4 Операція ділення

Функція `divide` реалізує алгоритм ділення двох наддовгих чисел, поданих у символьному вигляді, за логікою, близькою до класичного ділення «в стовпчик», але з повною підтримкою від’ємних значень, дробових частин, обчислення десяткових розрядів та виявлення періодичних дробів. Уся обробка виконується на рівні символів, без використання вбудованої арифметики довільної точності.

На самому початку обробляється тривіальний випадок: якщо ділене або дільник дорівнює нулю і представлений одним символом, функція одразу повертає рядок "0". Це дозволяє уникнути некоректних ситуацій та зайвих обчислень. Далі створюються локальні копії обох операндів, оскільки в процесі ділення вони можуть тимчасово змінюватися. Паралельно визначається знак результату за стандартним правилом множення знаків: якщо знаки операндів різні, результат буде від’ємним.

Після цього з’ясовуються позиції десяткових крапок у дільнику та діленому. Якщо хоча б один з операндів є дробовим, функція нормалізує обидва числа, перетворюючи їх на цілі. Для цього десяткові крапки фізично видаляються з рядків, а кількість цифр після крапки вирівнюється шляхом доповнення менш точного операнда необхідною кількістю нулів. Таким чином зберігається числова еквівалентність, але далі всі обчислення виконуються над цілими числами. Паралельно виконується видалення провідних нулів, щоб уникнути помилок при порівнянні та оцінці довжин.

Після нормалізації формується початковий дільник (`dividend`), який складається з перших цифр діленого, достатніх для порівняння з дільником. Якщо ділене за модулем є меншим за дільник, у результат одразу записується 0, а обчислення переходить до формування дробової частини. В іншому випадку починається формування цілої частини результату.

Щоб прискорити процес, попередньо обчислюються всі кратні дільника від 1 до 10. Кожне з цих значень зберігається окремо разом з інформацією про фактичну довжину та початковий індекс значущих цифр. Надалі це

дозволяє швидко визначати, на яке максимальне однозначне число можна помножити дільник, щоб результат не перевищував поточний залишок.

Основний цикл ділення працює так само, як і ділення у стовпчик. На кожному кроці з поточного залишку вилучаються провідні нулі, оцінюється його реальна довжина і шляхом послідовного порівняння з попередньо підготовленими кратними дільника визначається максимально можливий множник від 0 до 9. Отримана цифра додається до результату, а з поточного залишку віднімається відповідний добуток дільника. Після цього до залишку «зносяться» наступні цифри діленого, і процес повторюється, доки не буде оброблено всю цілу частину.

Після завершення обчислення цілої частини перевіряється, чи залишок дорівнює нулю. Якщо це не так, і якщо результат ще не містить десяткової крапки, вона додається, і алгоритм переходить до обчислення дробової частини. Для цього до залишку послідовно приписуються нулі, і кожен наступний розряд дробової частини обчислюється тим самим механізмом порівняння та віднімання, що й для цілої частини.

Особливістю реалізації є виявлення періодичних дробів. Усі проміжні залишки зберігаються в масиві. Якщо під час обчислення дробової частини з'являється залишок, який уже траплявся раніше і має ту саму довжину, це означає початок періоду. У такому разі результат доповнюється дужками: перед першою цифрою періодичної частини вставляється '(', а після останньої – ')', після чого обчислення зупиняється.

Після завершення ділення результат нормалізується: з кінця дробової частини видаляються зайві нулі, а якщо дробова частина повністю зникає, прибирається і десяткова крапка. Далі, якщо результат має бути від'ємним, на початку рядка встановлюється знак мінуса.

На фінальному етапі всі тимчасові зміни операндів відновлюються, а вся динамічно виділена пам'ять, використана для проміжних результатів, таблиць кратних, залишків і допоміжних масивів, коректно звільняється. Функція повертає новий рядок, який містить точний результат ділення двох

наддовгих чисел у символьному представленні з підтримкою дробової частини, періодичних дробів і коректного знака.

4.4 Основні арифметичні операції бібліотеки MPIR

У функціях `plus`, `minus`, `multiply` та `divide` класу `OptimizedBignum` реалізовано принципово інший підхід до роботи з числами у символьному представленні порівняно з «ручними» алгоритмами додавання, віднімання, множення та ділення. Тут уся арифметична логіка повністю делегується бібліотеці MPIR, а власний код виконує роль тонкого шару перетворення між рядками та внутрішнім раціональним форматом бібліотеки.

Кожна з чотирьох функцій починається однаково. Вхідні параметри `bignum1` і `bignum2` передаються як вказівники на символи разом з явною довжиною. Оскільки MPIR працює з нуль-термінованими рядками, спочатку для кожного операнда виділяється новий буфер розміром `length + 1`. У нього копіюється рівно відповідна кількість символів, після чого вручну додається символ завершення рядка `'\0'`. Таким чином код гарантує коректність рядка незалежно від того, як саме ці дані зберігалися зовні.

Далі для кожної операції створюються три змінні типу `mpq_t`: два операнди та результат. Тип `mpq_t` у MPIR представляє раціональне число у вигляді дробу, який складається з чисельника та знаменника, що дозволяє точно відображати як цілі, так і десяткові числа без втрати точності. Кожна з цих змінних обов'язково ініціалізується через `mpq_init`, оскільки MPIR використовує власні механізми керування пам'яттю.

Після ініціалізації відбувається ключовий етап – перетворення десяткового рядка у внутрішній раціональний формат MPIR. Це робить допоміжна функція `decimal_str_to_mpq`. Вона аналізує символи рядка, обробляє знак, цілу і дробову частини та формує відповідний чисельник і знаменник. Якщо рядок некоректний або перетворення не вдається, функція повертає ненульове значення. У такому випадку всі вже ініціалізовані дані формату `mpq_t` очищуються через `mpq_clear`, проміжні буфери з рядками

звільняються, а сама функція повертає NULL, сигналізуючи про помилку.

Якщо обидва операнди перетворені успішно, подальша логіка для кожної операції дуже проста і надійна, оскільки використовуються готові високорівневі функції MPIR. Для додавання викликається `mpq_add`, яка обчислює суму двох раціональних чисел з автоматичним приведенням до спільного знаменника і скороченням дробу. Для віднімання використовується `mpq_sub` з аналогічною поведінкою. Для множення викликається `mpq_mul`, яка перемножує чисельники і знаменники, після чого скорочує результат. Для ділення використовується `mpq_div`, але перед цим окремо перевіряється, чи не дорівнює другий операнд нулю, за допомогою `mpq_sgn`. Якщо дільник дорівнює нулю, виконання переривається і повертається NULL, що запобігає некоректному діленню.

Після виконання відповідної арифметичної операції результат зберігається у змінній типу `mpq_t`. Щоб повернути його у вигляді звичного символьного представлення, викликається функція `mpq_to_decimal_str`. Вона перетворює раціональне число у десятковий рядок, зазвичай з точною дробовою частиною, без накопичення похибок, властивих числам з плаваючою комою. Після цього динамічно виділений рядок стає готовим до повернення.

На фінальному етапі у кожній функції виконується повне очищення всіх тимчасових ресурсів. Усі `mpq_t` змінні звільняються через `mpq_clear`, а допоміжні рядки `str1` і `str2` звільняються через `free`.

Отже, логіка описаних функцій полягає не в реалізації арифметичних алгоритмів на рівні цифр, а в безпечному й ефективному переході від символьного представленням чисел до потужного внутрішнього механізму MPIR. Усі складні аспекти роботи з великими числами, дробами, знаками, скороченням і точністю перекладені на бібліотеку, тоді як власний код зосереджений на перевірках, перетвореннях форматів і керуванні пам'яттю.

4.5 Обчислення степеня за допомогою бібліотек MPFR та MPFR

Функція `power` реалізує універсальний та багаторівневий алгоритм піднесення довільно точних раціональних чисел до раціонального степеня, поєднуючи точні обчислення на основі MPFR з чисельною апроксимацією через MPFR у випадках, коли точне представлення неможливе або недоцільне. Загальною ідеєю реалізації є максимальна спроба зберегти точність результату в раціональному вигляді `mpq_t`, і лише за необхідності переходити до високоточної плаваючої арифметики з адаптивною точністю.

На початку роботи функція виконує перевірку області визначення. Якщо основа є від'ємною, а показник не є цілим числом, операція над дійсними числами втрачає сенс, тому функція одразу повертає код помилки. Далі обробляються вироджені випадки: будь-яке число в нульовому степені дорівнює одиниці, а нуль у додатному степені дорівнює нулю; натомість нуль у від'ємному степені вважається помилкою.

Після цього показник степеня розкладається на чисельник і знаменник у вигляді двох цілих чисел p і q , оскільки тип `mpq_t` зберігає число саме як дріб. Додатково фіксується знак показника, що дозволяє надалі розрізняти додатний і від'ємний степінь без втрати загальності. Цей крок є ключовим, бо вся подальша логіка спирається саме на представлення степеня через дріб.

Якщо виявляється, що знаменник показника дорівнює одиниці, тобто степінь є цілим, використовується спеціалізована логіка для цілочисельного піднесення. Для відносно малих значень показника, обмежених за кількістю бітів, застосовується точний алгоритм бінарного піднесення до степеня. У допоміжній функції `mpq_pow_integer_exp` реалізовано класичний метод «швидкого піднесення до степені», при якому результат накопичується шляхом множення на поточну основу лише тоді, коли відповідний біт показника дорівнює одиниці, а сама основа на кожному кроці підноситься до квадрату. Цей підхід забезпечує логарифмічну складність за показником і дозволяє зберегти повністю точне раціональне представлення результату.

Якщо показник цілий, але від'ємний, функція переходить до

еквівалентного перетворення задачі: обчислюється обернене значення основи і виконується піднесення вже до додатного показника. Для цього використовується допоміжна підпрограма інверсії раціонального числа, яка міняє місцями чисельник і знаменник та виконує канонізацію дробу.

У випадку, коли цілий показник є занадто великим для ефективного точного обчислення, або коли він не вміщається у безпечні обмеження, алгоритм переходить до наближеного обчислення. Тут починає працювати бібліотека MPFR, яка дозволяє виконувати операції з плаваючою комою з довільною бітовою точністю. Значення основи і показника переводяться в тип `mpfr_t`, після чого запускається функція `mpfr_pow`. Виконується спроба повернути отриманий результат у раціональний вигляд шляхом перетворення десяткового рядка з повною точністю у дріб. Якщо це не вдається через втрату значущих цифр або переповнення, точність збільшується, і спроба повторюється, доки не буде досягнуто прийнятного результату або не буде перевищено граничне обмеження точності.

Якщо показник є раціональним числом із знаменником, більшим за одиницю, алгоритм спочатку намагається виконати обчислення точно. Для цього використовується ідея представлення виразу $a^{p/q}$ у вигляді кореня q -го степеня від a^p . У допоміжній функції `try_exact_rational_power` спочатку окремо підноситься до степеня p чисельник і знаменник основи, а потім перевіряється, чи є результат точним q -тим степеневим коренем для обох цілих значень. Для цього застосовується функція `mpz_root`, після якої результат перевіряється зворотним піднесенням до степеня. Якщо обидва корені виявляються точними, формується раціональний результат, який за потреби інвертується у випадку від'ємного показника. Особливо враховується знак основи: для непарного знаменника можливе коректне винесення знаку, тоді як для парного знаменника від'ємна основа відразу визнається некоректною.

Якщо ж точне раціональне обчислення неможливе, тобто корені не є цілими або параметри виходять за допустимі межі, знову використовується

чисельний підхід на основі MPFR. Функція наближеного обчислення застосовує адаптивну схему підвищення точності: починаючи з фіксованої кількості бітів, вона поступово збільшує точність у залежності від величини показника степеня і характеру результату, намагаючись уникнути переповнення, втрати точності або появи NaN чи нескінченності.

Отже, функція `power` повертає не лише саме значення, а й інформацію про характер обчислення. Код завершення 0 означає, що результат отримано точно у вигляді раціонального числа MPFR, код 1 сигналізує про успішне наближене обчислення з використанням MPFR, а значення -1 вказує на помилку або математично некоректну операцію. Отже, реалізований алгоритм поєднує строгість раціональної арифметики з гнучкістю високоточної чисельної обробки, забезпечуючи коректну і стабільну роботу в широкому класі задач піднесення до степеня.

4.5.1 Порівняння розробленого рішення з іншими реалізаціями

Слід зазначити, що бібліотека GMP/MPFR має подібну функцію, проте вона оперує лише значеннями типу `long long`. Наша реалізація є більш досконалою з точки зору функціональності, точності та надійності, хоча й вона має деякі обмеження і в певних випадках краще використовувати GMP/MPFR.

Перш за все, базова відмінність – це клас вхідних даних. Стандартні функції на основі `long long` приймають лише цілі показники в обмеженому діапазоні (типово $\pm 2^{63-1}$), тоді як наша реалізація працює з показником і основою як з раціональними числами `mpq_t` (співвідношення цілих чисел). Це дозволяє робити операції виду $\text{base}^{p/q}$ (раціональні степені), обробляти від'ємні показники (за інверсією) і взагалі оперувати експонентами значно більшого діапазону (`mpz/mpq` замість `long long`). У практиці це означає, що можна коректно обчислювати, наприклад, кубічний корінь раціональних чисел, дробові степені з великими чисельниками/знаменниками або експоненти, що не вміщаються в 64 біти – сценарії, для яких «довгих `long`

long» недостатньо або які взагалі недоступні у GMP-версії, орієнтованій на long long.

Наша реалізація спочатку намагаючись знайти точний раціональний результат без переходу в плаваючу арифметику: якщо експонента має форму p/q з невеликими p і q , виконується спроба обчислити $\text{num}(\text{base})^{p^{(1/q)}}$ і $\text{den}(\text{base})^{p^{1/q}}$ за допомогою функцій `mpz_pow_ui` і `mpz_root`. Якщо обидва q -ті корені виявляються точними цілими, то результат збирається як точний `mpq` (раціонал), зберігається в канонічному вигляді і повертається. Такий шлях дає гарантію математичної точності (без наближень) коли це можливо – такого підходу у версії з long long немає, бо вона просто не працює з раціональними степенями і не робить аналізу простих коренів для точного представлення.

Якщо точного раціонального представлення отримати не вдається, реалізація робить адаптивне повернення на MPFR: вона переводить `mpq` -> `mpfr` з деякою початковою точністю (`MPFR_PREC_BITS`), виконує обчислення степеня у `mpfr` і намагається потім перетворити результат назад у формат `mpq` шляхом отримання десяткового рядка з повною точністю. Якщо перетворення невдале або результат виявляється нерегулярним (NaN/Inf/нуль через переповнення або затухання), функція підвищує точність і повторює спробу. Цей adaptive-loop дає баланс між точністю й продуктивністю: при складних випадках ми прагнемо підвищити точність, доки отримане наближення не стане коректно конвертованим у раціонал або поки не вичерпано допустиму точність. Такий підхід дозволяє отримати результат для широкого класу задач, включно з дуже складними раціональними або ірраціональними експонентами, тоді як реалізація на long long або взагалі не підтримує ці сценарії, або змушує переводити все в подвійний double з певною втратою точності.

Один із важливих моментів – коректна робота зі знаком. У нас реалізовано строгі перевірки: від’ємна основа з нецілим (тобто з парним знаменником) експонентом вважається нереальною у дійсній області і

обробляється як помилка (функція повертає -1). При цілих показниках знак обробляється коректно (наприклад, $-a^{\text{odd}}$ дасть від'ємний результат). Код також коректно обробляє крайні випадки $\text{base}==0$, $\text{exp}==0$ тощо та повертає математично визначені чи помилкові стани відповідно. У рішеннях на базі `long long` часто не враховані всі граничні випадки або їхня обробка тривіальна і нестача інформації про характер експонента призводить до невизначеності.

Проаналізувавши швидкодію, можна стверджувати, що рішення, вбудовані в бібліотеки GMP/MPFR, у випадках простих цілих степенів і особливо коли `exponent` вміщується в `long long`, майже завжди швидші. GMP має спеціалізовані, оптимізовані функції (`mpz_powm`, `mpz_pow_ui` та інші) з низькорівневою оптимізацією, що використовують машиноорієнтовані примітиви і добре масштабуються для великих цілих показників. Наша реалізація, будучи універсальною, робить додаткові кроки: аналіз розкладу показника, можливу інверсію для від'ємних експонентів, спроби точних коренів, і врешті кілька ітерацій MPFR з підвищенням точності. У випадках, коли потрібне швидке піднесення цілої основи до невеликого цілого показника, GMP-варіант на `long long` буде простіший і швидший. Тому наше рішення можна вважати функціонально більш загальним, точнішим та більш надійним для раціональних випадків з більшою кількістю параметрів.

Технічні обмеження і правила застосування реалізації також чітко прописані в коді: для ефективного і точного визначення точних коренів `try_exact_rational_power` накладає практичні обмеження на розміри `p` і `q` (перевірка `mpz_fits_ulong_p`), щоб не намагатися підняти надто великі числа до надвеликих степенів у вигляді `mpz_pow_ui`, що могло б бути недосяжним з точки зору пам'яті та часу. Це розумний компроміс: ми отримаємо точні результати там, де це реалістично, і не впадемо у непомірну витрату ресурсів, переходячи до MPFR, якщо задача стає надто «важкою» для дуже точного підходу. У GMP вибір відповідної функції, навпаки, покладається на користувача і вона не має універсального рішення з довільними раціоналами.

Можемо описати кілька конкретних сценаріїв, які показують переваги нашого підходу: приклад 1: обчислення $8^{1/3}$. GMP з long long не підтримує дробові степені – доведеться переводити в mpf_double або mpfr вручну й ризикувати неточною відповіддю. Наша функція виявляє, що $\text{num}(\text{base})^1=8$ та $\text{den}(\text{base})^1=1$ і що корінь третього ступеня від 8 дає ціле число 2, тобто повертає точний mpq 2/1. Приклад 2: обчислення $9/4^{3/2}$. Тут base і exp є раціональними числами, а точне представлення має вигляд $(9^3)^{(1/2)} / (4^3)^{(1/2)}$ – наша спроба обчислити корінь може дати точний результат у форматі mpq (якщо корені цілі). Вбудоване рішення GMP на базі long long знову не підходить. Приклад 3: дуже великий показник, що не вміщується в тип long long – наша функція оперує mpz експонентом та, за необхідності, використовує бінарне піднесення з mpq_row_integer_exp, тоді як бібліотечне рішення має обмеження.

Отже, наша реалізація є архітектурно більш досконалою в сенсі універсальності, числової точності та надійності. Вона дозволяє: працювати з раціональними показниками (p/q), повертати точні результати у форматі mpq, коли це можливо, коректно обробляти від’ємні показники через інверсію, адаптивно використовувати MPFR при складних випадках. Все це робить її придатною для широкого класу задач зі строгими вимогами до точності та загальної коректності. Проте в контексті виконання простих цілих степенів з малими показниками GMP/MPFR (низькорівневе mpz_row_ui або аналогічні) залишаються швидшими і потребують менше ресурсів. Тому наше рішення можна вважати функціонально багатшим, точнішим і більш універсальним.

4.6 Робота з арифметичними виразами на базі довгої арифметики

Для аналізу та обчислення значення арифметичних виразів, які можуть складатись з чотирьох базових дій та дужок, була розроблена функція CalculateExpression. Вона є програмним інтерпретатором інфіксних виразів з довільною точністю чисел, який поєднує синтаксичний аналіз, керування пріоритетами операцій та виконання самих арифметичних дій над числами з

використанням довгої арифметики. В основі рішення лежить класичний алгоритм обчислення виразів із використанням двох стеків – для операндів і для операторів – адаптований до роботи з числовими рядками великої довжини та власними й бібліотечними реалізаціями арифметичних операцій.

Допоміжна функція `strdup_s` виконує безпечне дублювання рядка з явним керуванням пам'яттю, що є критично важливим, оскільки всі проміжні значення у виразі зберігаються як динамічно виділені рядки. Функції `is_op_char`, `precedence` та `is_right_assoc` формалізують граматику виразу, визначаючи допустимі оператори, їхній пріоритет і асоціативність. Таким чином, логіка обчислення не є безпосередньою частиною головної підпрограми, а винесена в окремі абстракції, що спрощує розширення, наприклад, додавання операції піднесення до степеня.

Для зберігання проміжних даних використовується власна реалізація стеку `StrStack`, яка підтримує динамічне розширення буфера. Це дозволяє без обмежень обробляти довгі вирази з великою кількістю операцій і дужок. Стек операторів зберігає символи операцій та дужки, тоді як стек операндів містить рядкові подання чисел або результати проміжних обчислень.

Ключову роль у виконанні арифметики відіграють функції `ApplyOperation_OutputOn` та `ApplyOperation_OutputOff`. Обидві з них мають одну й ту саму логіку обчислення, але відрізняються режимом роботи: перша використовується для налагодження та детального виводу кожного кроку обчислення, друга – для «тихого» режиму, орієнтованого на продуктивність при тестуванні великої кількості операцій або при роботі з дуже довгими числами. Ці функції реалізовані у вигляді шаблонів, усередині яких відбувається виклик відповідних методів класів довгої арифметики: `bignum` (власні рішення) або `OptimizedBignum` (рішення, які використовують бібліотеки `MPIR` та `MPFR`). Після кожної операції результат додатково перевіряється зворотною дією (наприклад, для додавання виконується віднімання), що дозволяє автоматично виявляти логічні помилки реалізації. Для ділення та множення використовується спеціалізована перевірка

узгодженості результату з урахуванням максимальної довжини дробової частини.

Головна функція `CalculateExpression` виконує безпосередній розбір і обчислення арифметичного виразу. Вона проходить рядок символ за символом, ігноруючи пропуски, розпізнаючи числа, оператори та дужки. Числа, включно з від'ємними та дробовими, зчитуються як послідовності символів і одразу поміщаються в стек операндів. Для унарного мінуса реалізовано спеціальну логіку, яка або зчитує від'ємне число цілком, або неявно додає нуль перед мінусом, забезпечуючи коректну інтерпретацію виразу.

Після знаходження оператора алгоритм порівнює його пріоритет із іншими в стеку, і при необхідності виконує відкладені дії. При цьому суворо дотримується порядок операндів: лівий операнд завжди знімається зі стеку другим, що критично важливо для коректності операцій віднімання та ділення. Обробка дужок забезпечує локальність обчислень: усі операції всередині пари дужок виконуються до виходу за її межі.

Після завершення проходу по рядку всі залишкові оператори зі стеку послідовно застосовуються до операндів. Якщо в кінці обчислення в стеку чисел залишається більше одного елемента або в стеку операторів залишаються дужки, це вважається синтаксичною помилкою виразу, і виконання аварійно завершується. У разі успішного обчислення повертається єдиний рядок – результат обчислення з довільною точністю.

Отже, реалізований підхід поєднує класичний алгоритм розбору арифметичних виразів з високоточними власними арифметичними операціями та вбудованими механізмами самоперевірки. Це робить систему не лише універсальною для обчислення складних виразів над наддовгими числами, але й надійною з точки зору виявлення помилок та валідації коректності результатів, що є особливо важливим у наукових і тестових застосуваннях.

4.7 Тестування рішень для роботи з довгою арифметикою

Тестування рішень на базі довгої арифметики реалізовано через комплексну систему автоматизованої генерації тестових даних, виконанню арифметичних операцій та перевірці правильності результату шляхом зворотних обчислень. Використання всієї системи демонструє головна функція `main`, реалізована у файлі `main.cpp`, яка викликає декілька різних варіантів проведення перевірок: з детальним виведенням інформації про кожну виконану дію та її результат та без нього, з перевіркою вірності обчислень та без неї, а також «тихий» режим без виводу повідомлень для отримання інформації про швидкодію реалізованих рішень.

4.7.1 Створення тестових пар

З метою перевірки коректності та швидкості роботи реалізованих операцій була створена функція `GetBignumPairs`, яка призначена для генерування випадкових пар великих чисел у символьному представленні з контрольованим розподілом довжин і діапазонів значень. На логічному рівні вона реалізує двоетапний механізм: спочатку формується набір числових діапазонів, а вже потім з кожного такого діапазону генерується задана кількість випадкових пар наддовгих чисел у символьному представленні.

На початку роботи функція виводить повідомлення про старт генерації та фіксує час початку виконання для подальшого вимірювання тривалості. Далі обчислюється кількість можливих довжин чисел як різниця між максимальною та мінімальною кількістю цифр. Для цього діапазону створюється масив `lengths`, який заповнюється всіма допустимими довжинами чисел у зростаючому порядку, починаючи з `MinCountOfDigits`. Після заповнення цей масив випадково перемішується, що забезпечує рівномірний розподіл довжин між подальшими етапами генерації.

Потім формуються два масиви, `MinLengths` і `MaxLengths`, які відповідають мінімальним та максимальним довжинам чисел для кожного діапазону. Для кожного з діапазонів (змінна `CountOfRanges`) обирається

довжина з уже перемішаного масиву `lengths`. Якщо кількість діапазонів перевищує кількість можливих довжин, відбувається циклічне повторне використання довжин. Після цього масиви мінімальних і максимальних довжин додатково перемішуються незалежно один від одного, що ще більше знижує кореляцію між діапазонами.

На наступному етапі створюється масив `BignumRanges`, кожен елемент якого є парою граничних значень чисел у конкретному діапазоні. Для кожного з них генеруються два випадкові великі числа заданої довжини з випадковою наявністю знака та десяткової частини. Ці два числа інтерпретуються як межі діапазону. Щоб гарантувати коректність інтервалу, виконується порівняння отриманих чисел: якщо перше з них виявилось більшим за друге, вони міняються місцями, забезпечуючи інваріант «нижня межа $\leq$ верхня межа».

Після побудови всіх числових діапазонів розпочинається основна фаза генерації пар чисел. Створюється масив `BignumPairs`, розмір якого відповідає кількості необхідних пар. Генерація відбувається ітеративно, із циклічним перебором діапазонів. Для кожної пари обирається поточний діапазон, і з нього незалежно генеруються два випадкові числа, кожне з яких гарантовано лежить між відповідною мінімальною та максимальною межами. Таким чином кожна пара чисел є внутрішньо узгодженою і належить одному конкретному числовому інтервалу.

Щоб уникнути регулярності та повторюваності, після повного проходу по всіх діапазонах масив `BignumRanges` випадково перемішується, і процес повторюється з новим порядком діапазонів. Це дозволяє рівномірно змішувати пари з різних числових інтервалів у загальному вихідному масиві.

Після завершення генерації всіх пар функція звільняє всю тимчасово виділену пам'ять, зокрема масиви довжин та самі числові діапазони, що не входять до фінального результату. Далі фіксується час завершення роботи, обчислюється тривалість виконання і виводиться діагностична інформація.

Поверненим значенням функції є тривимірний масив `char***`, де кожен

елемент представляє окрему пару великих чисел у символьному форматі. Отриманий результат придатний для масового тестування арифметичних операцій, бенчмаркінгу або статистичного аналізу, оскільки забезпечує різноманіття довжин, знаків, форматів і числових діапазонів при контрольованій структурі даних.

4.7.2 Тестування арифметичних операцій

Подана система функцій реалізує повний цикл функціонального та продуктивного тестування арифметичних операцій над наддовгими числами у символьному представленні з використанням як власної реалізації довгої арифметики (`bignum`), так і бібліотечних реалізацій на базі MPIR (`OptimizedBignum`) та безпосередньо типу `mpq_t`. Загальна ідея тестування полягає у поєднанні двох підходів: математичної перевірки коректності результатів через обернені операції та точного вимірювання часу виконання для порівняння швидкодії різних реалізацій.

Функція `TestAllBignumOperations` є демонстраційною і призначена для покрокової перевірки всіх чотирьох базових арифметичних операцій для кожної пари чисел. Вона створює екземпляр класу, який задається шаблоном `BigNumClass`, завдяки чому одна й та сама логіка може застосовуватися як до власної реалізації, так і до бібліотечної. Для кожної пари чисел послідовно виконується додавання, після чого результат перевіряється відніманням другого операнда; аналогічно віднімання перевіряється додаванням, множення – діленням, а ділення – за допомогою спеціальної перевірки узгодженості частки. Кожен проміжний і фінальний результат виводиться у форматованому вигляді, що полегшує візуальний аналіз. У разі виявлення невідповідностей збільшується лічильник помилок, який наприкінці виводиться користувачу.

Функція `TestOneBignumOperationWithCheck` реалізує більш компактний і масштабний варіант тестування, орієнтований на статистичну перевірку та вимірювання часу. Вона приймає символ операції та набір пар чисел і,

залежно від обраної операції, виконує відповідну арифметичну дію для всіх пар. Коректність перевіряється не шляхом порівняння з еталонним значенням, а через застосування оберненої операції. Такий підхід дозволяє виявляти логічні помилки без залучення еталону. Час виконання вимірюється за допомогою `std::chrono`, що надає можливість аналізу часових витрат.

Функція `TestOneBignumOperation` призначена виключно для вимірювання швидкодії без перевірки коректності. Вона багаторазово виконує задану операцію для всіх пар чисел, звільняючи лише пам'ять для результатів. Відсутність додаткових перевірок дозволяє оцінити «чисту» вартість обчислень у кожній реалізації та порівняти ефективність алгоритмів.

Окремий набір функцій, `TestOneMpq_tBignumOperation` та `TestOneMpq_tBignumOperationWithCheck`, реалізує аналогічну логіку тестування, але безпосередньо на типі `mpq_t` бібліотеки MPFR. У першому випадку виконується лише обчислення і вимірювання часу, у другому – алгебраїчна перевірка коректності результатів з використанням рівностей виду $(a + b) - b = a$, $(a * b) / b = a$ або $(a / b) * b = a$, за умови що відповідні значення не дорівнюють нулю. Для цього створюється допоміжне нульове значення типу `mpq_t`, яке використовується для коректного аналізу граничних випадків.

Послідовність викликів тестових функцій побудована таким чином, щоб забезпечити коректне порівняння різних реалізацій. Спочатку виконуються тести з перевіркою та без перевірки для власного класу `bignum`, далі аналогічні тести виконуються для класу `OptimizedBignum`, який використовує функціональність MPFR. Після цього всі символічні пари чисел конвертуються у формат `mpq_t`, що супроводжується окремим вимірюванням часу на конверсію. Завершальним етапом є тестування арифметичних операцій безпосередньо над типом `mpq_t` у варіантах з перевіркою вірності результату та без неї.

Завершальний код звільняє всю динамічно виділену пам'ять як для символічних представлень чисел, так і для об'єктів `mpq_t`, що гарантує

відсутність витоків пам'яті та коректне завершення програми. У сукупності описаний підхід формує повноцінний тестовий стенд, який дозволяє одночасно оцінювати точність, стабільність і продуктивність власної реалізації довгої арифметики у прямому порівнянні з промисловими бібліотечними рішеннями.

4.7.2.1 Перевірка вірності результату операції ділення

Також слід детально зупинитись на рішеннях для перевірки вірності результату операції ділення, оскільки обернена дія, як у випадку інших арифметичних операцій, часто не зможе встановити, чи є частка коректною. Тому ми створили рішення, яке реалізує детальну і надійну перевірку коректності результату операції ділення для великих чисел, представлених у десятковому вигляді, з підтримкою як скінченних, так і періодичних дробів з урахуванням обмеженої точності обчислень.

Функція `IsDivisionConsistentOutputOn` реалізує повну перевірку коректності ділення з виводом усіх проміжних кроків. Якщо результат Q є періодичним десятковим дробом, він перевіряється точно: Q перетворюється у дріб $Q_{\text{num}}/Q_{\text{den}}$, після чого перевіряється рівність $Q_{\text{num}} * B = Q_{\text{den}} * A$ за допомогою точної арифметики великих чисел. Для неперіодичного результату використовується перевірка з допустимою похибкою. Обчислюється добуток $QB = Q * B$ та різниця $\text{Diff} = QB - A$. Далі береться модуль різниці і порівнюється з ε , де ε обирається як максимум між $|A| * 10^{-p}$ або $|QB| * 10^{-p}$ та мінімальним порогом 10^{-p} . Якщо $|\text{Diff}|$ не перевищує ε , результат ділення вважається коректним.

Функція `IsDivisionConsistent` реалізує ту саму логіку, але без детального виводу в консоль. Її призначення – швидка автоматизована перевірка результату. Вона використовує ті самі два сценарії: точну перевірку для періодичних дробів і наближену перевірку з контрольованою похибкою для звичайних десяткових результатів.

Ці обидві функції також використовують допоміжні підпрограми, які

працюють із десятковими рядками та готують їх до коректних порівнянь. Функція `normalize_positive_decimal` зводить додатний десятковий запис до канонічної форми: вона прибирає зайві початкові нулі в цілій частині, видаляє кінцеві нулі у дробовій частині, коректно обробляє відсутність дробової частини та гарантує, що результат завжди має принаймні символ «0». Це потрібно для того, щоб різні текстові представлення одного й того ж числа не заважали правильному порівнянню.

Функція `abs_normalized` видаляє можливий знак «+» або «-» і повертає нормалізовану додатну копію числа. Вона застосовується перед порівняннями модулів чисел, щоб знак не впливав на арифметичну логіку.

Функція `cmp_pos_dec` порівнює два додатних нормалізованих десяткових рядки за абсолютним значенням. Спочатку порівнюються довжини цілих частин, далі – посимвольне значення цифр цілої частини, і лише якщо вони рівні, здійснюється порівняння дробових частин із доповненням відсутніх цифр нулями. Це забезпечує числове порівняння без використання плаваючої крапки.

Функція `shift_decimal_down` реалізує ділення числа на 10 у степені `digits` шляхом логічного зсуву десяткової крапки вправо. Вона коректно працює як із цілими, так і з дробовими числами, дописує необхідну кількість нулів у дробову частину та наприкінці нормалізує результат, прибираючи зайві нулі.

Функція `make_relative_epsilon` (хоча у фінальних перевірках явно не використовується, її логіка частково реалізована вручну) концептуально формує допустиму похибку як максимум із двох значень: відносної похибки $|X| * 10^{-p}$ та мінімальної абсолютної похибки 10^{-p} . Це дозволяє уникнути надто жорстких перевірок для малих чисел.

Функція `PowerOf10` повертає десятковий рядок, що відповідає числу 10^n . Вона використовується під час побудови знаменників у перетворенні десяткових дробів у звичайні раціональні дроби.

У сукупності ці функції забезпечують універсальний механізм

перевірки ділення: вони коректно працюють із довільно великими числами, враховують знак, підтримують періодичні дробі, уникають помилок порівняння через різні текстові представлення чисел і дозволяють контролювати допустиму похибку залежно від заданої кількості знаків після плаваючої крапки.

4.7.2.2 Результати тестування основних арифметичних дій

Тестування рішень для роботи з довгою арифметикою відбувалось як у середовищі Windows з використанням компіляторів GCC MSYS2 та Visual Studio 2022, так і на віртуальній машині VMware з гостьовою ОС Linux Ubuntu, що доводить повну кросплатформеність випробуваних нами рішень. Нами було проведено два види перевірок: з малими числами (1-20 символів) при великій кількості пар, та з великими числами (від ста цифр до мільйона), але на 10 парах. Час на виконання 1 млн. операцій показано в секундах у таблицях:

Таблиця 4.7.2.2.1 – Порівняння швидкодії операції PLUS:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)	1.499	36.876	3.294
MSVS 2022	1.575	46.947	3.443
GCC (Linux)	0.177	1.558	0.159

Таблиця 4.7.2.2.2 – Порівняння швидкодії операції MINUS:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)	1.332	36.353	3.293
MSVS 2022	1.250	47.037	3.454
GCC (Linux)	0.232	1.597	0.171

Таблиця 4.7.2.2.3 – Порівняння швидкодії операції MULTIPLY:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)	3.802	37.211	3.362
MSVS 2022	2.917	51.818	7.591
GCC (Linux)	1.321	1.606	0.192

Таблиця 4.7.2.2.4 – Порівняння швидкодії операції DIVIDE:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)	46.269	31.214	3.352
MSVS 2022	44.834	43.970	8.512
GCC (Linux)	6.780	1.544	0.232

Отримані результати демонструють чітку різницю у швидкодії між власними операціями над малими числами (1-20 символів) та їх бібліотечними аналогами, а також значну залежність продуктивності від середовища компіляції та операційної системи. У середовищі GCC MSYS2 під Windows власна реалізація арифметики показує перевагу над MPIR при роботі зі строковим поданням чисел майже у всіх операціях, особливо при додаванні та відніманні, які виконуються у десятки разів швидше. Навіть множення та ділення, що потребують складніших алгоритмічних кроків, у власній реалізації залишаються помітно швидшими за MPIR при роботі з числами малої довжини, хоча збільшення кількості тестових пар до мільйона приводить до стрімкого зростання часу, особливо для операції ділення. Після перетворення всіх пар чисел від рядкового формату до раціонального (тип даних mpq_t) бібліотека MPIR демонструє суттєву перевагу в швидкості роботи за нашу реалізацію через значне зниження накладних витрат.

Visual Studio 2022 демонструє схожу тенденцію, проте абсолютні часи виконання у Windows загалом гірші, ніж у Linux, як для власних алгоритмів, так і для MPIR. Власні реалізації знову виявляються значно швидшими за MPIR у рядковому форматі, тоді як MPQ-обчислення скорочують різницю. У

випадку великої кількості операцій Windows-компілятори призводять до відчутно більшого часу виконання, особливо в MPIR, де операції множення та ділення можуть бути повільнішими у два-три рази, ніж аналогічні під Linux. Це свідчить про можливий вплив як особливостей компілятора Visual Studio, так і відмінностей у реалізації низькорівневих оптимізацій MPIR під Windows.

Найбільш показовими є результати, отримані в середовищі Linux з використанням компілятора GCC, де спостерігається найвища загальна продуктивність. Власна реалізація арифметики демонструє мінімальні часи обробки навіть при мільйоні пар чисел, а MPIR у MPQ-форматі працює надзвичайно швидко та стабільно. У Linux бібліотека MPIR має не лише менші накладні витрати при перетвореннях чисел, але й краще використовує оптимізовані алгоритми множення та ділення завдяки ефективнішій реалізації низькорівневих операцій (особливо у 64-бітному оточенні GCC). У результаті цього MPIR під Linux працює швидше, ніж відповідні реалізації у Windows, а подекуди навіть досягає продуктивності, близької до власних алгоритмів, особливо при використанні типу `mpq_t`.

Загальний аналіз показує, що для малих чисел і великої кількості операцій власні алгоритми, орієнтовані на роботу з короткими рядками, мають суттєву перевагу, адже MPIR створений для універсальної високоточної арифметики та не оптимізований на випадок дуже коротких операндів у великій кількості. Водночас MPQ-формат демонструє, що основні затримки в MPIR пов'язані саме з обробкою рядків, тоді як самі базові операції додавання, віднімання та множення у форматі раціональних чисел є ефективнішими.

Порівняння середовищ компіляції чітко підтверджує платформозалежність продуктивності: GCC у Linux стабільно забезпечує найкращий час, GCC MSYS2 – середній результат, а Visual Studio 2022 – найповільніший, особливо при використанні MPIR. Це пояснюється тим, що стороння бібліотека в першу чергу була створена під Linux. Компіляція її

останньої версії, яка включає низькорівневі оптимізації, спеціально реалізовані навіть для деяких поколінь процесорів Intel, за допомогою портованої версії GCC MSYS2 показує найкращу швидкодію в середовищі Windows. Для Visual Studio 2022 нам не вдалось отримати останню версію MPIR, саме тому ми бачимо її нижчу продуктивність. Це свідчить про те, що за однакової логіки високорівневих операцій вирішальну роль відіграють компіляційні оптимізації, реалізація інструкцій на рівні ОС та ефективність внутрішніх модулів бібліотек.

Ефективність реалізацій істотно залежить від формату подання чисел, особливостей використаних бібліотек та середовища компіляції. Власна реалізація демонструє значну перевагу у випадках великої кількості малих операндів, MPIR у MPQ-форматі досягає найкращої продуктивності під Linux, а Windows-середовища є гіршими за швидкодією у всіх протестованих конфігураціях, насамперед через меншу оптимізованість низькорівневих операцій та іншу модель керування пам'яттю. Отже, для високопродуктивної обробки чисел невеликої довжини оптимальною платформою є GCC Linux, а власні алгоритми демонструють найкраще співвідношення швидкості та стабільності у всіх середовищах.

У ході наступного експериментального дослідження було проаналізовано вплив довжини чисел на продуктивність арифметичних операцій додавання, віднімання, множення та ділення при фіксованій кількості тестових пар (10) для двох реалізацій – власного алгоритму обчислень та бібліотечної MPIR, включно з форматом `mpq_t`. Результати демонструють суттєве зростання вартості операцій із збільшенням довжини вхідних чисел, однак характер цього зростання істотно залежить від реалізації та платформи. Слід зазначити, що при кількості цифр від 10000 дослідження швидкодії власних реалізацій операцій множення та ділення не проводилось через надмірно високі витрати оперативної пам'яті та часу. Швидкість роботи рішень при довжині цифр від ста тисяч до мільйона (10 пар):

Таблиця 4.7.2.2.5 – Порівняння швидкодії операції PLUS:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)	0.016	1.300	0.319
MSVS 2022	0.014	4.342	0.957
GCC (Linux)	0.106	6.484	1.135

Таблиця 4.7.2.2.6 – Порівняння швидкодії операції MINUS:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)	0.018	1.297	0.332
MSVS 2022	0.014	4.352	0.970
GCC (Linux)	0.118	6.471	1.137

Таблиця 4.7.2.2.7 – Порівняння швидкодії операції MULTIPLY:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)		1.925	0.652
MSVS 2022		6.784	1.867
GCC (Linux)		9.348	2.749

Таблиця 4.7.2.2.8 – Порівняння швидкодії операції DIVIDE:

Платформа	Власна реалізація	MPIR	MPIR (формат mpq_t)
GCC (Windows)		1.287	0.492
MSVS 2022		5.141	2.815
GCC (Linux)		7.376	3.027

У всіх середовищах операції додавання та віднімання виявилися найменш ресурсомісткими. Для чисел довжиною до тисячі символів час обчислення практично не відрізняється як між власною реалізацією, так і між MPIR, причому найвищу швидкість демонструє середовище Linux GCC. При

збільшенні розрядності до десятків і сотень тисяч власна реалізація продовжує працювати значно швидше за MPIR, оскільки в MPIR операції над великими дробами потребують багаторазових перетворень та нормалізуючих процедур. MPIR у форматі `mpq_t` демонструє стабільно кращі результати за MPIR у символьному представленні, але воно все одно суттєво повільніший за власний алгоритм при дуже великих числах.

Найбільша різниця спостерігається в операціях множення та ділення. Для діапазонів довжини 1000–10000 символів власна реалізація у Windows (MSYS2 або Visual Studio) потребує від 5 до 15 секунд на множення та від 50 до 70 секунд на ділення, тоді як MPIR виконує ці самі операції за десятки мілісекунд. Це демонструє високу оптимізованість MPIR для цілочисельної арифметики, проте лише коли операції виконуються у внутрішньому форматі `mpq_t`. Натомість MPIR, що працює з десятковими рядками, показує багаторазове падіння швидкодії, оскільки у цьому режимі бібліотека змушена виконувати регулярні перетворення між форматами.

При ще більших розрядностях, до сотень тисяч і мільйона, продуктивність власної реалізації знижується плавно, натомість MPIR (особливо в символьному режимі) демонструє експоненційне збільшення часу – окремі операції множення та ділення займають десятки секунд під Windows і до десятків секунд у Linux. MPIR у форматі `mpq_t` зберігає відносно кращий рівень продуктивності, але зростання все одно суттєве, особливо в Windows. Порівняння середовищ показує, що Linux стабільно перевершує Windows у продуктивності MPIR, причому виграш становить у середньому від 20% до 300% залежно від операції й розрядності чисел. Код, скомпільований за допомогою GCC і Visual Studio в середовищі Windows демонструє близькі результати, але Visual Studio суттєво повільніше працює з MPIR при надвеликих числах. Тестова програма стабільно працює при використанні трьох різних компіляторів, що видно на наступних скріншотах:

```

C:\Windows\system32\cmd.exe
MIN COUNT OF DIGITS:      1; MAX COUNT OF DIGITS :      10; MAX FRACT PART LENGTH:      10
MAX BIGNUM LENGTH :      20; COUNT OF BIGNUM PAIRS: 1000000;
-----
Creating 1000000 random bignum pairs... Time: 4.520 seconds
Testing operation PLUS with check... Count of errors: 0 Time: 2.803 seconds
Testing operation MINUS with check... Count of errors: 0 Time: 3.219 seconds
Testing operation MULTIPLY with check... Count of errors: 0 Time: 34.962 seconds
Testing operation DIVIDE with check... Count of errors: 0 Time: 61.627 seconds
Testing operation PLUS... Time: 1.499 seconds
Testing operation MINUS... Time: 1.332 seconds
Testing operation MULTIPLY... Time: 3.802 seconds
Testing operation DIVIDE... Time: 46.269 seconds
-----GNU Multiple Precision Arithmetic Library (MPIR)-----
Testing operation PLUS with check... Count of errors: 0 Time: 73.142 seconds
Testing operation MINUS with check... Count of errors: 0 Time: 72.471 seconds
Testing operation MULTIPLY with check... Count of errors: 0 Time: 74.307 seconds
Testing operation DIVIDE with check... Count of errors: 0 Time: 92.352 seconds
Testing operation PLUS... Time: 36.876 seconds
Testing operation MINUS... Time: 36.353 seconds
Testing operation MULTIPLY... Time: 37.211 seconds
Testing operation DIVIDE... Time: 31.214 seconds
Converting bignum pairs into mpq_t format... Time: 12.496 seconds
-----GNU Multiple Precision Arithmetic Library (MPIR) MPQ_T format-----
Testing operation PLUS with check... Count of errors: 0 Time: 6.250 seconds
Testing operation MINUS with check... Count of errors: 0 Time: 6.341 seconds
Testing operation MULTIPLY with check... Count of errors: 0 Time: 6.432 seconds
Testing operation DIVIDE with check... Count of errors: 0 Time: 6.465 seconds
Testing operation PLUS... Time: 3.294 seconds
Testing operation MINUS... Time: 3.293 seconds
Testing operation MULTIPLY... Time: 3.362 seconds
Testing operation DIVIDE... Time: 3.352 seconds
DONE.

```

Рисунок 4.7.2.2.1 – Результати роботи програми при використанні компілятора GCC у середовищі Windows (1 млн. числових пар)

```

Terminal
MIN COUNT OF DIGITS:      1; MAX COUNT OF DIGITS :      10; MAX FRACT PART LENGTH:      10
MAX BIGNUM LENGTH :      20; COUNT OF BIGNUM PAIRS: 1000000;
-----
Creating 1000000 random bignum pairs... Time: 14.981 seconds
Testing operation PLUS with check... Count of errors: 0 Time: 0.423 seconds
Testing operation MINUS with check... Count of errors: 0 Time: 0.476 seconds
Testing operation MULTIPLY with check... Count of errors: 0 Time: 4.501 seconds
Testing operation DIVIDE with check... Count of errors: 0 Time: 10.614 seconds
Testing operation PLUS... Time: 0.177 seconds
Testing operation MINUS... Time: 0.232 seconds
Testing operation MULTIPLY... Time: 1.321 seconds
Testing operation DIVIDE... Time: 6.780 seconds
-----GNU Multiple Precision Arithmetic Library (MPIR)-----
Testing operation PLUS with check... Count of errors: 0 Time: 3.293 seconds
Testing operation MINUS with check... Count of errors: 0 Time: 3.304 seconds
Testing operation MULTIPLY with check... Count of errors: 0 Time: 3.358 seconds
Testing operation DIVIDE with check... Count of errors: 0 Time: 4.241 seconds
Testing operation PLUS... Time: 1.558 seconds
Testing operation MINUS... Time: 1.557 seconds
Testing operation MULTIPLY... Time: 1.606 seconds
Testing operation DIVIDE... Time: 1.544 seconds
Converting bignum pairs into mpq_t format... Time: 0.950 seconds
-----GNU Multiple Precision Arithmetic Library (MPIR) MPQ_T format-----
Testing operation PLUS with check... Count of errors: 0 Time: 0.343 seconds
Testing operation MINUS with check... Count of errors: 0 Time: 0.352 seconds
Testing operation MULTIPLY with check... Count of errors: 0 Time: 0.395 seconds
Testing operation DIVIDE with check... Count of errors: 0 Time: 0.425 seconds
Testing operation PLUS... Time: 0.159 seconds
Testing operation MINUS... Time: 0.171 seconds
Testing operation MULTIPLY... Time: 0.192 seconds
Testing operation DIVIDE... Time: 0.232 seconds
DONE.

```

Рисунок 4.7.2.2.1 – Результати роботи програми при використанні компілятора GCC у середовищі Linux (1 млн. числових пар)

Отже, було експериментально підтверджено, що власна реалізація краще масштабується при екстремально великих числах, натомість MPIR незрівнянно ефективніший для малих та середніх розрядностей, особливо при використанні внутрішнього представлення `mpq_t` без попередніх витрат на перетворення від символьного формату. Важливою закономірністю є також залежність продуктивності від операційної системи: Linux чітко домінує в обробці великих чисел незалежно від реалізації. Це свідчить про те, що вибір ОС суттєво впливає на ефективність обчислень і має враховуватися при проектуванні програмних систем високоточних розрахунків.

4.7.3 Тестування операції піднесення до степеня

Також у рамках нашої роботи було створено рішення для перевірки вірності результату піднесення до степеня. Функція `TestMpq_tPower` реалізує повний цикл тестування даної операції для числових даних, представлених у вигляді рядків та перетворених у формат раціональних чисел `mpq_t`. Логіка побудована так, щоб забезпечити коректне завантаження даних, виконання обчислень, перевірку точності результату, його перетворення у зручний для аналізу формат та повне звільнення пам'яті для уникнення витоків.

На початку роботи створюється допоміжне значення нульового раціонального числа: формується рядок "0", після чого виділяється пам'ять під структуру `mpq_t`, ініціалізується змінна та викликається функція `decimal_str_to_mpq`, яка перетворює текстове представлення у формат `mpq_t`. Це значення надалі використовується лише як службовий об'єкт і не впливає на основний цикл тестування.

Результати усіх операцій записуються у файл `power_results.txt`. Основна робота виконується в циклі за кількістю пар чисел `CountOfBignumPairs`. Для кожної пари виділяється динамічна пам'ять під три змінні `mpq_t`: основу степеня, показник степеня та результат. Кожна з них ініціалізується викликом `mpq_init`. Обидва рядкові операнди – основа та показник – перетворюються функцією `decimal_str_to_mpq`, після чого у стандартний

потік виводиться текст із зазначенням номера тесту та числових значень операндів.

Операція піднесення до степеня виконується власною функцією `power`, яка приймає три структури `mpq_t`: основу, показник та результат. Після обчислення результат проходить через функцію `validate_power_comprehensive`, яка виконує багаторівневу перевірку для довільно точних раціональних чисел і призначена для незалежної верифікації значень, отриманих як точними, так і наближеними методами. Основна ідея цієї реалізації полягає у використанні кількох числових критеріїв, які зменшують ризик прийняття хибного результату через особливості подання чисел або похибки округлення.

Перевірка починається з аналізу тривіальних граничних випадків. Якщо показник степеня дорівнює нулю, функція очікує, що результат дорівнює одиниці, і проводить точне порівняння з раціональним числом 1. Такий підхід дозволяє уникнути використання чисельних методів у ситуаціях, де правильний результат відомий наперед. Аналогічно обробляється випадок нульової основи при додатному показнику, для якого результат має дорівнювати нулю. У цих випадках верифікація виконується в рамках раціональної арифметики MPFR.

Для всіх інших випадків основним методом перевірки стає чисельне порівняння з еталонним значенням, обчисленим засобами MPFR із підвищеною бітовою точністю. Значення основи, показника і результату переводяться з типу `mpq_t` у `mpfr_t`. Після цього за допомогою функції `mpfr_pow` незалежно обчислюється значення степеня з високою точністю, що слугує чисельним еталоном. Якщо при цьому виникають особливі значення, зокрема нескінченність або NaN, результат одразу визнається некоректним, оскільки це свідчить про вихід за область чисельної стабільності.

Далі виконується безпосереднє оцінювання відносної похибки між результатом, який підлягає перевірці, і еталонним значенням. Для цього обчислюється абсолютна різниця між ними, яка потім ділиться на модуль

еталонного значення. Такий підхід забезпечує масштабно-інваріантну перевірку, тобто коректно працює як для дуже великих, так і для дуже малих чисел. Якщо отримане значення відносної похибки не перевищує заданого порогового значення *tolerance*, результат визнається коректним.

У ситуаціях, коли пряме чисельне порівняння є ненадійним або не дає позитивного результату, використовується альтернативний логарифмічний метод перевірки, реалізований у допоміжній функції *validate_power_result_log*. Цей підхід ґрунтується на тотожності $\log(ab) = b \log(a)$ і є більш стабільним у випадках екстремальних значень, коли безпосереднє обчислення степеня може призводити до переповнення або втрати точності. Метод застосовується лише для додатних значень основи та результату, оскільки логарифм для від'ємних і невід'ємних чисел у дійсній області не визначений. У логарифмічному просторі обчислюється різниця між $\log(\text{result})$ і $\text{exp} \cdot \log(\text{base})$; якщо ця різниця за абсолютним значенням не перевищує допустиму межу, результат вважається коректним. Інтерпретація цієї різниці спирається на апроксимацію $\log_{10}(1+\epsilon) \approx \epsilon$ для малих похибок, що зв'язує логарифмічну похибку з відносною.

Якщо ж і прямий, і логарифмічний методи не підтверджують коректність, функція переходить до додаткової евристичної перевірки, яка застосовується лише для цілих показників степеня. У цьому випадку використовується обернене піднесення: обчислюється степінь результату з показником, оберненим до початкового, тобто фактично виконується спроба відновити початкову основу з результату, який перевіряється. Отримане значення порівнюється з початковою основою з використанням відносної похибки, але з більш м'яким порогом точності, оскільки ця перевірка накопичує похибки двох степеневих операцій. Успішне відновлення основи з достатньою точністю розглядається як непрямий доказ коректності вихідного результату.

Таким чином реалізується комплексний підхід до верифікації операції піднесення до степеня, комбінуючи точні порівняння, високоточні чисельні

обчислення, логарифмічні тотожності та евристичні обернені перевірки. Використання кількох незалежних стратегій дозволяє підвищити надійність перевірки, особливо в умовах довільної точності, де стандартні методи тестування виявляються недостатніми або нестабільними.

Після завершення перевірки `mpq_t`-значення переводиться у формат `mpf_t` з фіксованою точністю 256 біт, після чого цей об'єкт форматовано переводиться у звичайний десятковий рядок і. Спочатку викликається `gmp_snprintf` із `NULL`-буфером, що дозволяє визначити точну довжину рядка, після чого виділяється відповідна кількість пам'яті, і результат разом з обома операндами записується в текстовий файл повторним викликом `gmp_snprintf`.

Після завершення кожної ітерації всі `mpq_t`-змінні очищаються викликами `mpq_clear`, а динамічно виділена пам'ять звільняється через `free`. Після завершення всього циклу також очищається службовий об'єкт "zero". На завершальному етапі файл із результатами закривається.

Слід зазначити, що наше рішення, яке має такі широкі засоби тестування, підвищує довіру до повернутого результату й дозволяють виявляти випадки, коли повернення до наближення не надало коректного значення. GMP/MPIR у своїй мінімальній формі не мають такого механізму.

Результати роботи даної функції видно на відповідному зображенні:

```

C:\Windows\system32\cmd.exe
MIN COUNT OF DIGITS: 1; MAX COUNT OF DIGITS : 10; FRACT PART LENGTH: 10
MAX BIGNUM LENGTH : 20; COUNT OF BIGNUM PAIRS: 10;
-----
Creating 10 random bignum pairs... Time: 0.001 seconds
Converting bignum pairs into mpq_t format... Time: 0.000 seconds
-----GNU Multiple Precision Arithmetic Library (MPIR) MPQ_T format-----
1. base: -18.08527476767; exp: -14398.608827341... Result is NOT valid.
-----
2. base: 24; exp: -98358035... Result is valid.
-----
3. base: 0.4; exp: 0.12553... Result is valid.
-----
4. base: 3.744140022; exp: 3.69037... Result is valid.
-----
5. base: -1.94639612; exp: -1.89977814... Result is NOT valid.
-----
6. base: -74; exp: -925... Result is valid.
-----
7. base: -5; exp: -5... Result is valid.
-----
8. base: 62.110738647; exp: 60.69486470479... Result is valid.
-----
9. base: 51.17816597; exp: -5518... Result is valid.
-----
10. base: -57618.8007877; exp: -6899.6421985252... Result is NOT valid.
-----
DONE.

```

Рисунок 4.7.3.1 – Тестування операції піднесення до степеня

Слід зазначити, що наше рішення не працює з від'ємним показником степеня, тому при тестуванні вони помічаються як невірні.

Отже, функція реалізує послідовний та коректно керований процес тестування степеневі операції: вона виконує імпорт даних, обчислення, незалежну перевірку точності, перетворення результатів у текстовий формат, протоколювання у файл та повне звільнення ресурсів. Це забезпечує надійність тестування, відтворюваність результатів та відсутність витоків пам'яті навіть при обробці великої кількості числової інформації.

4.7.4 Перевірка роботи аналізатора арифметичних виразів

У межах системи тестування рішень на основі довгої арифметики була створена функція `GetRandomArithmeticExpression`, яка призначена для автоматичного формування коректних випадкових арифметичних виразів із заданою кількістю операцій, що використовуються як вхідні дані для перевірки механізмів аналізу та обчислення. Такий підхід дозволяє моделювати широкий спектр сценаріїв обчислень, включно з різними комбінаціями операторів, знакових цілих та дробових чисел, а також вкладених підвиразів.

Логіка функції починається з перевірки параметра `OperationCount`. Якщо кількість операцій менша або дорівнює нулю, функція одразу повертає одне випадкове число великої розрядності, згенероване викликом `GetRandomBigNum`, тобто в цьому випадку результатом є тривіальний вираз без операторів. Якщо ж кількість операцій додатна, формується повноцінний арифметичний вираз, який складається з числових операндів та операторів.

Далі визначається загальна кількість токенів виразу як $2 * \text{OperationCount} + 1$, що відповідає класичній структурі інфіксного запису, де кількість чисел на одиницю більша за кількість операторів. Виділяється масив вказівників `parts`, у який послідовно записуються рядки чисел і рядки операторів. У циклі генерується кожен числовий операнд як випадкове довге число з довільною кількістю цифр у заданому діапазоні, випадковим знаком

та, за необхідності, дробовою частиною. Між цими числами, окрім останнього, вставляються оператори з множини $+$, $-$, $*$ та $/$, причому кожен оператор оформлюється як окремий рядок із пробілами зліва та справа, що спрощує подальший аналіз.

Після формування базової лінійної послідовності tokenів реалізується механізм випадкового додавання дужок. Для цього обчислюється максимально можлива кількість пар дужок, виходячи з довжини виразу, після чого випадковим чином визначається фактична кількість таких пар. Для кожної пари обирається коректний лівий індекс, який обов'язково вказує на числовий token, та відповідний правий індекс, що також відповідає числу і розташований правіше мінімум на одну операцію. Обрані підвирази модифікуються шляхом додавання відкритої дужки на початок лівого операнда та закритої дужки в кінець правого операнда. Такий підхід гарантує синтаксичну коректність дужок і водночас забезпечує випадкову зміну пріоритетів обчислень у виразі.

На завершальному етапі обчислюється сумарна довжина всіх tokenів, виділяється один суцільний буфер під кінцевий рядок виразу, після чого всі частини послідовно з'єднуються в єдиний рядок. Паралельно відбувається звільнення проміжної пам'яті, що використовувалася для окремих tokenів. У результаті функція повертає коректний випадковий арифметичний вираз довільної складності, який може містити довгі числа, різні арифметичні оператори та вкладені дужки, і придатний для подальшого тестування алгоритмів обчислення виразів у системі довгої арифметики.

Також було створено фрагмент коду, який реалізує повний цикл тестування виконання операцій над випадковими виразами та з детальним вимірюванням часу виконання. Логіка починається з оголошення функціонального вказівника `ApplyOperation`, який визначає, яким саме способом виконуватимуться арифметичні операції над числами у процесі обчислення виразів: з детальним виводом усіх виконаних дій чи без нього.

Після ініціалізації обраного режиму роботи виводиться повідомлення

про початок обчислення виразів. Якщо активовано режим з виведенням (`ApplyOperation_OutputOn`), додатково виводиться перехід на новий рядок, щоб результати кожного обчислення відображались у зручному для аналізу форматі. Далі фіксується початковий момент часу за допомогою `chrono::system_clock::now()`, що дозволяє точно виміряти сумарні часові витрати на виконання всього тестового циклу.

Основна частина коду зосереджена у циклі, який виконується задану кількість разів (`CountOfExpressions`). На кожній ітерації спочатку генерується випадковий арифметичний вираз шляхом виклику функції `GetRandomArithmeticExpression`. Кількість операцій у виразі також визначається випадково в межах, заданих змінними `MinCountOfOperations` та `MaxCountOfOperations`. Таким чином забезпечується різноманітність тестових даних як за структурою виразів, так і за їх складністю, включно з різними комбінаціями операторів та вкладених дужок.

Створений рядок виразу передається до функції `CalculateExpression` разом із вказівником `ApplyOperation`. Усередині цієї функції виконується синтаксичний аналіз виразу, коректне врахування пріоритетів операцій та дужок, а всі арифметичні дії над операндами реалізуються через обрану функцію `ApplyOperation`. У результаті повертається рядок, що містить числовий результат обчислення виразу у форматі довгого числа. Після завершення обчислення виділена під вираз та його результат пам'ять звільняється викликами `free`, що запобігає витокам пам'яті навіть при великій кількості тестових ітерацій.

Після завершення циклу фіксується кінцевий момент часу, що дозволяє обчислити загальну тривалість виконання тестування. Якщо був обраний режим без детального виведення, перед фінальним повідомленням додається пробіл для коректного форматування консолі. Далі виводиться кількість попереджень, пов'язаних з операцією ділення, яка накопичується у глобальній змінній `CountOfDivisionWarnings`. Цей показник відображає випадки, коли результат ділення є чисельно допустимим, але може містити

похибки округлення або періодичні дробу, що не повністю відтворюються у заданій точності.

Завершальним етапом є виведення сумарного часу виконання тесту в секундах, що дає змогу оцінити продуктивність алгоритмів обчислення випадкових арифметичних виразів у цілому. Повідомлення DONE. сигналізує про успішне завершення всіх обчислень, а виклик getch() зупиняє виконання програми до натискання клавіші, що зручно під час інтерактивного аналізу результатів. Отже, наше рішення забезпечує комплексне функціональне тестування механізму обчислення складних випадкових виразів на основі реалізацій довгої арифметики. Приклади результатів роботи даної функції:

```
Administrator: C:\Windows\system32\cmd.exe
66084.782844 + 6 = 66090.782844
66090.782844 - 6 = 66084.782844
-----
result: 66090.782844
=====
696691 + 619521968 - -62 / 5.21072975 / (755700 / -45758 - (-15 / -8.5623 / 1.8)) + -313970
36
696691 + 619521968 = 620218659
620218659 - 619521968 = 696691
-----
-62 / 5.21072975 = -11.8985253457
-11.8985253457 * 5.21072975 = -61.9999999999
-61.9999999999 - -62 = 0.0000000001
-----
755700 / -45758 = -16.5151448926
-16.5151448926 * -45758 = 755699.999995908
755699.999995908 - 755700 = -0.0000044092
-----
-15 / -8.5623 = 1.751865737
1.751865737 * -8.5623 = -14.9999999999
-14.9999999999 - -15 = 0.0000000001
-----
1.751865737 / 1.8 = 0.9732587427
0.9732587427 * 1.8 = 1.7518657368
1.7518657368 - 1.751865737 = -0.0000000002
-----
-16.5151448926 - 0.9732587427 = -17.4884036353
-17.4884036353 + 0.9732587427 = -16.5151448926
-----
-11.8985253457 / -17.4884036353 = 0.6803665785
0.6803665785 * -17.4884036353 = -11.8985253447
-11.8985253447 - -11.8985253457 = 0.0000000001
-----
620218659 - 0.6803665785 = 620218658.3196334215
620218658.3196334215 + 0.6803665785 = 620218659
-----
620218658.3196334215 + -31397036 = 588821622.3196334215
588821622.3196334215 - -31397036 = 620218658.3196334215
-----
result: 588821622.3196334215
=====
Count of division warnings: 1; Time: 15.983 seconds
DONE.
```

Рисунок 4.7.4.1 – Приклад детального виведення на екран результатів роботи функції для аналізу випадкових виразів.

На даному зображенні видно детальний розбір випадкового арифметичного виразу та послідовні дії, які виконувались разом з

оберненими операціями для перевірки вірності обчислень.

```

C:\Windows\system32\cmd.exe
MIN COUNT OF DIGITS: 10; MAX COUNT OF DIGITS : 100; FRACT PART LENGTH : 100
MAX BIGNUM LENGTH : 200; COUNT OF BIGNUM PAIRS: 10;
COUNT OF EXPR. : 10000; MIN COUNT OF OPS. : 10; MAX COUNT OF OPS. : 100
-----
Calculating expressions... Count of division warnings: 953; Time: 26.237 seconds
DONE.
  
```

Рисунок 4.7.4.2 – Приклад виведення на екран результатів роботи функції для аналізу випадкових виразів у режимі масового тестування.

Отримані результати відображають поведінку та обчислювальні властивості реалізованого механізму генерації та обчислення випадкових арифметичних виразів над довгими числами за умов підвищеної складності вхідних даних: у даному експерименті кожне число має довжину від 10 до 100 цифр із дробовою частиною до 100 знаків, а кожен вираз містить від 10 до 100 арифметичних операцій, що формує надзвичайно різноманітний і глибоко вкладений простір обчислень. Загальна кількість згенерованих і обчислених виразів становить 10000, що забезпечує статистично значущу вибірку для оцінки стабільності та продуктивності алгоритму. Функція генерації створює не лише лінійні послідовності операндів і операторів, а й випадковим чином додає дужки, що призводить до складних дерев обчислень із різною глибиною вкладеності. Це активно навантажує механізм розбору виразів, стекову модель обчислень та реалізації довгої арифметики.

Час виконання на рівні приблизно 26 секунд свідчить про суттєві обчислювальні витрати, що є очікуваним результатом з огляду на поєднання трьох факторів: великої кількості виразів, значної довжини чисел та наявності дорогих за складністю операцій множення і особливо ділення. На відміну від додавання та віднімання, які мають лінійну складність відносно довжини числа, ділення довгих дробових чисел із контролем точності потребує багаторазових ітерацій та перевірок коректності результату.

Показник у 953 попередження щодо ділення є важливою характеристикою поведінки системи. Він означає, що при виконанні деякої кількості операцій ділення було виявлено ситуації, коли результат не є строго узгодженим із зворотною перевіркою з урахуванням обмеження на

максимальну довжину дробової частини. Це не свідчить про критичні помилки реалізації, а радше демонструє неминучі наслідки роботи з наближеними дробовими результатами при фіксованій точності, особливо в умовах багатокрокових обчислень, де похибки можуть накопичуватися.

Водночас важливо підкреслити, що механізм не припиняє виконання при таких ситуаціях, а коректно фіксує їх як попередження, що відповідає науково коректному підходу до чисельних обчислень. Це дозволяє відокремити фатальні помилки алгоритму від допустимих чисельних відхилень, зумовлених природою операцій над довгими дробовими числами.

У цілому результати демонструють, що реалізована система здатна стабільно обробляти великі масиви складних арифметичних виразів із довгими числами, забезпечуючи контроль коректності на кожному етапі та надаючи кількісні метрики як продуктивності, так і чисельної стабільності. Це підтверджує придатність розробленого підходу для використання в дослідницьких і тестових задачах, пов'язаних із перевіркою та порівнянням реалізацій довгої арифметики.

4.8 Інтеграція рішень для роботи з довгою арифметикою в інший програмний комплекс

З метою розширення функціональності існуючого комплексу для виконання інженерно-наукових розрахунків у галузі машинобудування було вирішено інтегрувати рішення з довгої арифметики, написані мовами C та C++, у проект на базі середовища Lazarus та мови Free Pascal у вигляді динамічної бібліотеки DLL. Для цього необхідно виконати низку концептуальних і технічних змін, які забезпечать коректну міжмовну взаємодію, стабільний бінарний інтерфейс та передбачувану роботу з пам'яттю.

4.8.1 Загальні правила міжмовної взаємодії

Для успішного вбудовування проект на C та C++ має бути чітко

розділений на внутрішню логіку та публічний програмний інтерфейс. Усі функції довгої арифметики, які планується використовувати з Free Pascal, повинні бути винесені в окремий заголовковий файл інтерфейсу, що описує лише ті функції та змінні, які експортуються з DLL. При цьому для C++ критично важливо вимкнути манглінг («спотворення») імен, оскільки Free Pascal очікує C-сумісний бінарний інтерфейс програм (ABI). Для цього всі експортовані функції необхідно оголосити з використанням конструкції `extern "C"`. Це гарантує, що імена функцій у таблиці експорту DLL будуть простими та стабільними і не залежатимуть від компілятора C++.

Далі необхідно забезпечити явний експорт функцій та, за потреби, глобальних змінних. У середовищі Windows це досягається або за допомогою атрибутів `__declspec(dllexport)` у визначеннях функцій і змінних, або через використання `.def`-файлу з переліком експортованих символів. Бажано обрати один підхід і використовувати його послідовно. Для кросплатформенності доцільно застосувати макрос, який в Windows розкривається у `__declspec(dllexport)`, а в інших системах – у відповідний атрибут видимості або залишається порожнім. Таким чином, сигнатури експортованих функцій залишаються однаковими для всіх платформ.

Особливу увагу необхідно приділити типам даних. Усі типи, які перетинають межу DLL, мають бути максимально простими та сумісними з Free Pascal. Це означає, що слід уникати передачі C++ класів, шаблонів, `std::string`, `std::vector` та інших складних структур стандартної бібліотеки. Оптимальним рішенням є використання примітивних типів C (`int`, `long long`, `double`), вказівників на `char` для рядків, а також явних структур C, якщо це необхідно. Для рядків довгої арифметики потрібно чітко зафіксувати угоду: або функція повертає вказівник на динамічно виділений `char*`, який згодом звільняється спеціальною функцією з тієї ж DLL, або ж викликаюча сторона передає буфер і його розмір для запису результату. Другий варіант часто є безпечнішим при міжмовній інтеграції.

Окремим критичним аспектом є управління пам'яттю. Пам'ять,

виділена в DLL за допомогою `malloc` або `new`, не повинна звільнятися з боку Free Pascal без спеціально експортованої функції, оскільки менеджери пам'яті в різних мовах і середовищах можуть бути несумісними. Тому в проекті на C/C++ слід передбачити експортовані функції для звільнення результатів обчислень, наприклад `FreeBignumString` або `FreeResult`. У Free Pascal ці функції викликаються замість стандартного `Dispose` або `FreeMem`.

Глобальні змінні, такі як параметри точності, максимальна довжина дробової частини чи лічильники попереджень, також можуть бути зроблені видимими для Lazarus, однак це потребує особливої обережності. Для цього вони повинні бути оголошені як `extern` у заголовковому файлі та експортовані з DLL. У Free Pascal їм мають відповідати змінні з директивою `external`. Водночас з точки зору архітектури значно безпечніше сховати доступ до таких змінних через функції, які повертають або встановлюють значення змінних, наприклад `SetMaxFractPartLength` і `GetMaxFractPartLength`, що дозволяє контролювати коректність значень і уникати проблем.

Після підготовки DLL на стороні C/C++ у проекті Lazarus необхідно описати інтерфейс доступу до неї. Для цього у Free Pascal створюється окремий модуль, в якому за допомогою ключового слова `external` та директиви `cdecl` оголошуються всі імпортовані функції. Вказівка `cdecl` є принципово важливою, оскільки саме цю угоду виклику зазвичай використовують компілятори C і C++, і вона забезпечує коректну передачу параметрів та очищення стеку. Також у цьому модулі визначається ім'я бібліотеки DLL, яке має збігатися з фактичним іменем зібраного файлу.

Окрім цього, у проекті на C++ бажано мінімізувати використання глобального стану та статичних об'єктів, оскільки ініціалізація та видалення таких об'єктів з пам'яті в DLL може бути неочевидною з точки зору життєвого циклу бібліотеки в Lazarus. Якщо використання глобального стану неминуче, доцільно реалізувати явні функції ініціалізації бібліотеки, які відповідно викликаються з Free Pascal перед початком і завершенням роботи.

Отже, для інтеграції рішення з довгої арифметики у Lazarus необхідно

забезпечити C-сумісний ABI, явно експортувати функції та необхідні глобальні змінні, уніфікувати типи даних, продумати безпечну модель управління пам'яттю та описати відповідний Pascal-інтерфейс. Виконання цих кроків дозволить використовувати реалізовані алгоритми довгої арифметики як повноцінну DLL-бібліотеку в проектах Free Pascal без втрати продуктивності та збереження коректності обчислень.

4.8.2 Використання створених рішень у вигляді динамічної бібліотеки

У рамках даної роботи було реалізовано повний ланцюг інтеграції обчислювального ядра на мовах C та C++ у вигляді динамічної бібліотеки DLL з подальшим використанням її в середовищі Lazarus (мова Free Pascal). Ключова логіка побудована навколо чіткого розмежування внутрішньої реалізації довгої арифметики та зовнішнього інтерфейсу, який робиться сумісним із викликами з Pascal-коду.

Сценарій компіляції GCC MSYS2_BuildDLL.bat виконує роль точки входу для створення DLL. Він запускає компілятор g++ у режимі створення спільної бібліотеки через ключ -shared. Визначення макросу BUILD_DLL є принципово важливим, оскільки саме він активує експорт символів замість їх імпорту. До складання включаються всі cpp-файли, які містять реалізацію довгої арифметики, генераторів випадкових чисел, глобальних параметрів і тестової логіки. Зв'язування з бібліотеками gmp, gmpxx та mpfr гарантує коректну роботу зовнішніх числових типів, які використовуються в частині перевірок і тестів. Результатом виконання сценарію є файл bignums.dll, який фізично копіюється або підключається до проекту Lazarus.

Файл accessories.h був доповнений кодом, який визначає універсальний механізм експорту функцій для C та C++. Важливим є використання команди extern "C", яке відключає C++-манглінг імен, забезпечуючи стабільні символічні імена функцій у DLL. Це критично важливо для Free Pascal, який очікує C-сумісний ABI. Макрос DLL_EXPORT_FUNC інкапсулює

платформозалежні атрибути експорту, дозволяючи без змін компілювати код як у вигляді DLL, так і у вигляді звичайного виконуваного файлу.

Файл `exports.h` деталізує цю логіку для різних платформ. Для Windows використовується `__declspec(dllexport)` або `__declspec(dllimport)` залежно від того, чи визначено `BUILD_DLL`. Таким чином один і той самий заголовок коректно працює як під час побудови бібліотеки, так і під час її підключення. Для не-Windows платформ визначається спрощений варіант експорту, що робить код кросплатформним на рівні інтерфейсу.

Файл `globals.h` є центральним вузлом інтеграції, оскільки саме тут оголошуються глобальні змінні, які мають бути безпосередньо доступні з Lazarus. Кожна змінна позначена макросом `DLL_EXPORT`, що гарантує її фізичну присутність у таблиці експорту DLL. Ці змінні визначають параметри генерації чисел та виразів і виступають спільним станом між графічним інтерфейсом Lazarus і обчислювальним ядром. Окремо оголошується функція `UpdateLimits`, яка синхронізує похідні параметри довжини чисел і дробової частини після зміни вхідних значень.

У `globals.cpp` ці змінні отримують конкретні початкові значення та фактичну пам'ять. Функція `UpdateLimits` реалізує захисну логіку, яка запобігає некоректним параметрам, таким як нульова або від'ємна довжина числа, та автоматично перераховує максимальну довжину дробової частини і повного числа. Оскільки ця функція експортується, вона може викликатися безпосередньо з Lazarus для приведення внутрішнього стану бібліотеки у цілісний вигляд.

Файл `bignums.pas` у проєкті Lazarus виконує роль Pascal-інтерфейсу до DLL. Тут явно задається угода виклику `cdecl`, яка повністю відповідає угоді, використаній у C та C++. Функції `TestExpressions` і `UpdateLimits` оголошуються як `external`, що означає відсутність їх реалізації в Pascal-коді та перенаправлення викликів у DLL. Особливо важливим є використання директиви `var` для глобальних змінних, оскільки вона дозволяє Free Pascal працювати з ними як із зовнішніми змінними, а не через гетери або сетери.

Таким чином змінні `MinCountOfDigits`, `MaxCountOfDigits`, `CountOfExpressions`, `MinCountOfOperations` та `MaxCountOfOperations` стають повністю спільними між C/C++ та Lazarus.

Для тестування механізму інтеграції в існуючий обчислювальний комплекс було додано кнопку «Перевірити обчислення», при натисканні на яку відбувається генерування та обчислення заданої кількості випадкових арифметичних виразів, результати виводяться у відповідне текстове поле журналу. Значення про мінімальну та максимальну довжину числа, кількість арифметичних виразів, мінімальну та максимальну кількість операцій в них вводяться користувачем у відповідні поля типу `TSpinEdit`, вони безпосередньо присвоюються глобальним змінним DLL. Після цього викликається підпрограма `UpdateLimits`, яка синхронізує внутрішні обмеження бібліотеки. Подальший виклик функції `TestExpressions` запускає повний цикл операцій всередині DLL. Повернений результат у вигляді `PChar` відображається в Метод-компоненті, що забезпечує простий і наочний обмін даними між C/C++-кодом і графічним інтерфейсом Lazarus.

Отже, ключовими елементами інтеграції є суворе дотримання C-сумісного ABI, явний експорт функцій і глобальних змінних, використання `extern "C"` для уникнення манглінгу (зміни) імен, коректне узгодження виклику та чітке розділення між внутрішньою реалізацією довгої арифметики і зовнішнім API. Така архітектура дозволяє легко використовувати рішення на мовах C/C++ у проекті Free Pascal/Lazarus як звичайну DLL-бібліотеку.

Після інтеграції даної функції графічний інтерфейс існуючого програмного продукту було доповнено:

EngineeringCalc 2.0

Вид металу: Листовий, Кількість м.: 0, Товщина: 0,068, Тип і розмір металопроката: 30, Питома вага (кг/п.м): 0,068, Вага з урахуванням кількості: 0,068

Розміри троса (мм): За окружністю: 30, Вага 1 м троса (кг): 0,068, Розрахувати: Вагу, Кількість метрів: 0, Вага троса (кг): 0,068

Режим роботи каната: Ручний, Коефіцієнт: 4,5, Тип каната в наявності: Трьохрядний конопляний, Зусилля (кг): 0, Обчислити розривне зусилля та діаметр каната: 0

Система числення: 2, Точність ділення: 10, Тестовий режим: ☐, Час в журналі: ☒, Розривне зусилля каната (кг): 0, Діаметр каната (мм): 0

Арифметичний вираз: , Очистити:

Початкова система числення: 10, Цільова система числення: 16, Конвертувати:

Мін. довжина числа: 1, Макс. довжина числа: 10, Кількість виразів: 1000, Мін. кількість операцій: 1, Макс. кількість операцій: 10, ПЕРЕВІРИТИ ОБЧИСЛЕННЯ:

Очистити журнал:

Очистити результат:

0 1 2 3 4 5 + ()

6 7 8 9 , +/- -)

A B C D E F *

G H I J K L /

M N O P Q R S

T U V W X Y Z =

=====

1000. 7126.12 * -4 + 326259

7126.12 * -4 = -28504.48

-28504.48 / -4 = 7126.12

-28504.48 + 326259 = 297754.52

297754.52 - 326259 = -28504.48

RESULT: 297754.52

=====

Count of errors: 0;

Count of division warnings: 34

Time: 0.053000 seconds

Рисунок 4.8.1 – Інтерфейс користувача програмного комплексу для виконання інженерно-наукових розрахунків у галузі машинобудування

З наступного виведення результатів роботи функції «Перевірити обчислення» видно, що дана функція повністю повторює логіку вище описаного метода для тестування роботи з випадковими виразами у консольному додатку. Дане рішення працює стабільно навіть при більш «жорстких» параметрах: було задано, що довжина чисел змінюється в діапазоні від 100 до 1000 цифр, дробова частина може досягати 1000 цифр, а кожен вираз містить від 10 до 100 арифметичних операцій. При роботі віконного застосунку спостерігалось «підвисання» програми на деякий час, хоча після завершення роботи програма успішно повернулась у нормальний стан та показала результати у вигляді наступного виведення:

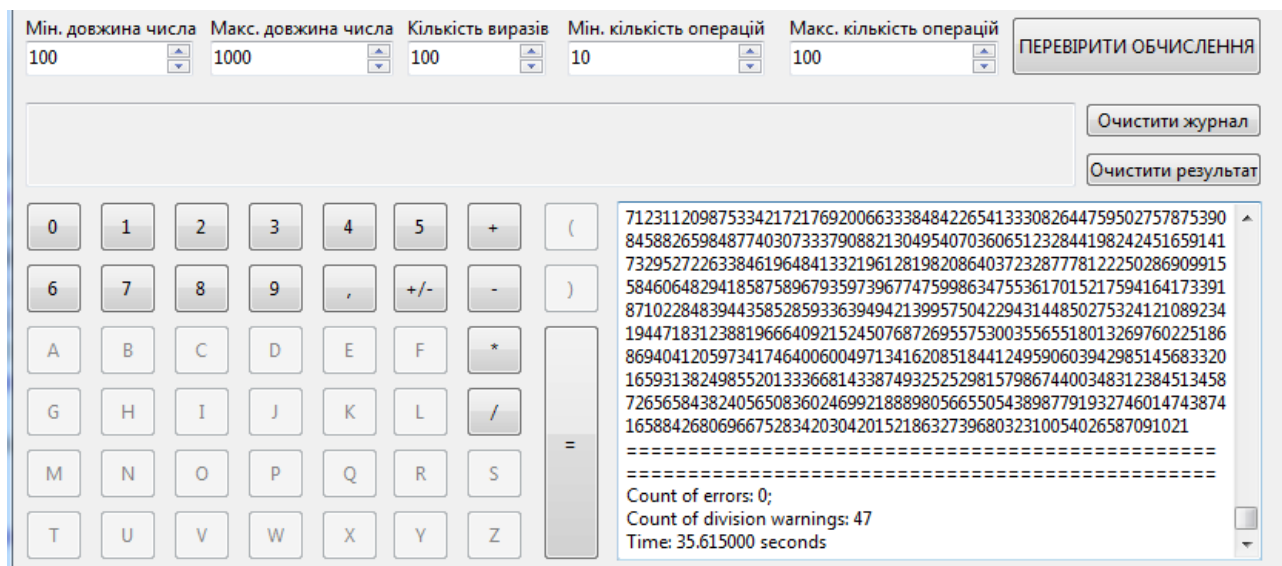


Рисунок 4.8.2 – Результати інтенсивного тестування функції «Перевірити обчислення»

Видно, що система успішно обробила 100 випадкових виразів без жодної критичної помилки, що підтверджує правильність реалізації базових арифметичних дій, алгоритмів роботи з довгою арифметикою та механізмів перевірки результатів. Зафіксована кількість попереджень при виконанні операцій ділення (47) не є ознакою некоректної роботи, а відображає специфіку обчислень над дійсними наддовгими числами з обмеженою максимальною довжиною дробової частини. Такі попередження виникають у випадках, коли результат ділення не може бути представлений точно в заданих межах точності або має періодичну дробову частину. Важливо, що такі випадки обробляються контрольовано та не призводять до аварійного завершення програми чи спотворення результатів обчислень.

Час виконання тесту, який становить близько 35,6 секунди, є очікуваним з огляду на значну довжину чисел, складність виразів та велику кількість операцій у кожному з них. Це підтверджує, що основним фактором часових витрат у даному режимі є не генерація даних, а саме обчислювальна складність операцій множення та ділення над наддовгими числами. Результати демонструють, що реалізоване рішення є функціонально коректним, достатньо стійким до складних обчислювальних сценаріїв і придатним для подальшої інтеграції в інші програми через DLL.

Висновки до розділу

У даному розділі магістерської дисертації було проаналізовано рішення для роботи з довгою арифметикою, особливості його реалізації, інтеграції та практичних результатів експлуатації.

Розроблений програмний продукт являє собою повноцінну систему для роботи з довгими знаковими цілими та дробовими числами, побудовану на мовах C та C++ із використанням власних алгоритмів арифметичних операцій та вибірковою інтеграцією бібліотек GMP/MPFR для перевірки вірності результатів і реалізації окремих складних операцій. Архітектура проекту орієнтована на масштабне автоматизоване тестування, генерацію випадкових чисел і складних арифметичних виразів, а також порівняльний аналіз швидкодії різних реалізацій у різних середовищах виконання.

Однією з ключових переваг рішення є гнучкий механізм генерації тестових даних. Реалізовані генератори випадкових наддовгих чисел і арифметичних виразів дозволяють контролювати діапазон довжин чисел, наявність дробової частини, знак, кількість операцій у виразах та глибину вкладеності дужок. Це забезпечує перевірку алгоритмів у широкому спектрі сценаріїв – від простих обчислень до обробки виразів із сотнями операцій над числами довжиною в тисячі та сотні тисяч цифр.

Система тестування побудована таким чином, що кожна арифметична операція не лише обчислюється, а й перевіряється на коректність зворотними діями або шляхом порівняння з бібліотечними реалізаціями. Це дозволило досягти нульової кількості логічних помилок у результатах навіть при роботі з дуже довгими операндами, що підтверджується результатами тестів, де кількість помилок («Count of errors») стабільно дорівнює нулю. Наявність попереджень при діленні свідчить не про помилки алгоритму, а про неминучі числові похибки, пов'язані з обмеженням довжини дробової частини.

Порівняльний аналіз продуктивності показав, що власні реалізації арифметичних операцій є значно ефективнішими за універсальні бібліотечні рішення GMP/MPFR у випадку роботи з великою кількістю коротких чисел.

Це пояснюється суттєвими накладними витратами бібліотек на ініціалізацію структур даних і універсальність алгоритмів. Водночас при роботі з наддовгими числами існуючі популярні рішення демонструють кращу масштабованість, особливо для операцій множення та ділення, що підтверджує доцільність комбінованого підходу, обраного в проекті.

Важливим результатом роботи є доведена кросплатформеність рішення. Проект для тестування довгої арифметики успішно компілюється та працює у середовищах Windows (GCC MSYS2, Visual Studio 2022) і Linux (GCC), демонструючи стабільні результати та передбачувану різницю у швидкодії, зумовлену особливостями компіляторів, стандартних бібліотек і управління пам'яттю в різних операційних системах. Linux-середовище продемонструвало найкращі часові показники, що узгоджується з відомими особливостями компілятора GCC та ефективністю системних викликів.

Окремої уваги заслуговує успішна інтеграція рішення у проект на мові програмування Free Pascal в середовищі Lazarus у вигляді DLL-бібліотеки. Чітке розмежування інтерфейсу експорту, використання необхідних директив, коректне оголошення глобальних змінних і функцій дозволили безпечно та прозоро керувати параметрами генерації й запускати складні обчислення безпосередньо з графічного інтерфейсу Lazarus. Це підтверджує універсальність розробленої архітектури та можливість її використання як обчислювального ядра в проектах на інших мовах програмування.

Результати тестів показують високу стабільність роботи системи. Час виконання у десятки секунд для настільки складних обчислень є прийнятним і свідчить про практичну придатність рішення для інженерних і наукових задач, де критично важливими є точність, контрольованість похибок і можливість детального тестування.

Отже, створене рішення для роботи з довгою арифметикою є комплексним, надійним і масштабованим. Воно поєднує власні оптимізовані алгоритми, бібліотечні реалізації, розвинену систему тестування та кросплатформену інтеграцію.

ВИСНОВКИ

Метою магістерської дисертації була розробка програмно-апаратного комплексу підтримки інженерних розрахунків для машинобудівного підприємства з можливістю виконання обчислень над цілими та дробовими знаковими числами великої довжини, перетворення чисел у різних системах числення з керованою точністю, автоматизації інженерних розрахунків із використанням довідкових даних, а також відображення результатів обчислень і часових витрат на їх виконання.

У ході написання роботи було проаналізовано велику кількість рішень, яке використовуються на сучасних машинобудівних підприємствах, докладно описані «прямі» аналоги розробленого програмного комплексу, висвітлені їх недоліки. Нами докладно прокоментовані алгоритми розробленої програми, висвітлені вимоги для її функціонування та підтримки іншими спеціалістами.

Програмний комплекс планується до введення в експлуатацію на підприємстві, що підтверджується довідкою про впровадження.

У ході виконання роботи, зокрема в межах розділу 4, було розроблено та детально проаналізовано програмну бібліотеку для роботи з довгою арифметикою, яка має чітко структуровану архітектуру та охоплює всі базові етапи роботи з наддовгими числами. Запропонована структура проекту дозволяє ізолювати реалізовувати, тестувати та оптимізувати окремі компоненти, що істотно підвищує масштабованість і розширюваність програмного комплексу.

Значну увагу приділено розробці генераторів випадкових наддовгих цілих і дробових чисел, а також створенню тестових пар і складних арифметичних виразів. Це дало змогу побудувати універсальне середовище для детальних випробувань, у якому результати обчислень перевіряються на великій кількості різнорідних вхідних даних, включаючи граничні випадки. Реалізовані генератори забезпечують контроль над довжиною чисел, знаком, наявністю дробової частини та кількістю операцій у виразах, що є важливою

перевагою порівняно з типовими тестовими наборами.

У роботі реалізовано власні алгоритми виконання основних арифметичних операцій над наддовгими числами: додавання, віднімання, множення та ділення. Для кожної з операцій було розроблено механізми внутрішньої перевірки коректності результатів, зокрема шляхом виконання обернених операцій. Окремо реалізовано перевірку узгодженості результатів ділення з урахуванням обмеженої точності дробової частини, що дозволяє виявляти потенційні числові неточності без зупинки обчислювального процесу.

Також було проведено детальне порівняння власних реалізацій арифметичних операцій з бібліотечними рішеннями MPIR та MPFR. Експериментальні результати показали, що для чисел малої довжини та великої кількості операцій власні реалізації мають істотно менші часові накладні витрати, ніж універсальні бібліотечні алгоритми. Водночас при роботі з наддовгими числами великої розрядності бібліотека MPIR демонструє кращу масштабованість, що підтверджує доцільність комбінованого підходу, реалізованого в даному проекті.

Окремою важливою частиною роботи є створення власного алгоритму піднесення до степеня, який не обмежується використанням цілочисельного показника типу `long long`, як це реалізовано в стандартних засобах GMP/MPIR. Запропоноване рішення дозволяє виконувати операцію піднесення до степеня з довільною раціональною експонентою, використовуючи можливості бібліотек MPIR та MPFR, що істотно розширює їх функціональність та робить придатними для складних інженерних і наукових розрахунків.

У межах дисертації також реалізовано повноцінний аналізатор арифметичних виразів на базі довгої арифметики, який підтримує пріоритети операцій, дужки та унарні оператори. Тестування цього модуля на великій кількості випадково згенерованих виразів підтвердило його коректність і стійкість. Отримані результати показали відсутність помилок обчислення та

наявність лише контрольованих попереджень при діленні, пов'язаних із обмеженою точністю представлення результатів.

Важливим практичним результатом роботи стала успішна інтеграція розробленої бібліотеки для роботи з довгою арифметикою в існуючий програмний комплекс, реалізований мовою Free Pascal у середовищі Lazarus. Було розроблено механізм експорту функцій і глобальних змінних у вигляді динамічної бібліотеки DLL, забезпечено коректну міжмовну взаємодію та продемонстровано практичне використання розроблених алгоритмів у графічному інтерфейсі користувача. Це підтверджує прикладну цінність запропонованого рішення.

Розроблений програмний комплекс був успішно випробуваний при вирішенні реальних виробничих задач. При порівнянні нашого продукту з іншими відомими, але менш гнучкими і функціонально обмеженими системами не було виявлено суттєвих розбіжностей в якості та швидкодії виконання покладених дій. Комплекс може вважатися програмно-апаратним, оскільки частково імітує принципи роботи апаратних обчислювальних пристроїв, зокрема калькуляторів, і водночас потребує відповідної апаратної та програмної підтримки для успішного функціонування.

Отримані результати роботи підтверджують доцільність подальшого розвитку розробленого рішення, зокрема в напрямі оптимізації операцій над наддовгими числами, розширення підтримуваних математичних функцій та інтеграції з іншими інженерними програмними системами. Програмний продукт планується до введення в експлуатацію на підприємстві, що підтверджується відповідною довідкою про впровадження, а результати роботи мають як практичну, так і наукову цінність. Це дозволяє розглядати його як повноцінну основу для подальшого розвитку, оптимізації та використання у багатомовних програмних системах разом з інженерними калькуляторами та науковими обчислювальними модулями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bailey D.H., Borwein J.M. High-precision arithmetic in mathematical physics // Mathematics of Computation. - 2021. - Vol. 90. - P. 2391-2421.
2. Brooks R.E. Comparative analysis of programming languages for HPC // Parallel Computing. - 2022. - Vol. 110. - P. 102-118.
3. Brown M.K., Johnson R.L. Microservices architecture for computational engineering applications // IEEE Transactions on Software Engineering. - 2022. - Vol. 48, No. 7. - P. 2156-2169.
4. Chen L., Wang X. Performance comparison of programming languages for scientific computing // Journal of Computational Science. - 2023. - Vol. 67. - P. 101-115.
5. Davis P.J. Software engineering principles for numerical applications // ACM Computing Surveys. - 2020. - Vol. 53, No. 2. - Article 35.
6. Evans E., Fowler M. Domain-driven design for engineering applications // IEEE Software. - 2020. - Vol. 37, No. 5. - P. 71-78.
7. Ferguson D.E., Young R.M. High-precision arithmetic in fluid dynamics simulations // Journal of Computational Physics. - 2021. - Vol. 445. - P. 110-125.
8. Granlund T., Montgomery P.L. Division by invariant integers using multiplication // ACM SIGPLAN Notices. - 2021. - Vol. 56, No. 6. - P. 78-91.
9. Harrison P.L., Miller J.S. Python for scientific computing: best practices // Computing in Science & Engineering. - 2021. - Vol. 23, No. 4. - P. 67-78.
10. Jackson D., Wing J.M. Lightweight formal methods for software architecture // IEEE Software. - 2019. - Vol. 36, No. 4. - P. 23-31.
11. Johnson K.R., White S.T. Layered architecture patterns for computational software // Software Architecture. - 2022. - Vol. 15. - P. 145-162.
12. Kumar S., Patel A. Python frameworks for scientific computing: A comparative study // Computing in Science & Engineering. - 2022. - Vol. 24, No. 3. - P. 45-56.

13. Lewis M.A., Brown D.K. GNU Multiple Precision Arithmetic Library in engineering // International Journal of Numerical Methods in Engineering. - 2020. - Vol. 121. - P. 4567-4582.
14. Möller N. Efficient implementation of the Karatsuba multiplication algorithm // Journal of Symbolic Computation. - 2021. - Vol. 104. - P. 834-851.
15. Nelson R.J., Green A.L. Java platforms for scientific applications // Journal of Systems and Software. - 2022. - Vol. 186. - P. 111-125.
16. Nielsen F., Anderson K.L. Software design patterns for numerical computation // Software: Practice and Experience. - 2022. - Vol. 52, No. 8. - P. 1723-1741.
17. Percival C. Cache-timing attacks on AES // Technical Report. - 2020. - 45 p.
18. Peterson L.R., Adams C.M. MPFR library applications in thermal analysis // Numerical Heat Transfer. - 2021. - Part B, Vol. 80. - P. 234-249.
19. Qin L., Zhou M. Component-based software architecture for engineering applications // IEEE Transactions on Software Engineering. - 2021. - Vol. 47, No. 9. - P. 1845-1858.
20. Roberts S.A., Thompson J.K. Architectural patterns for high-performance computing // Computer. - 2020. - Vol. 53, No. 7. - P. 42-51.
21. Smith T.J., Davis R.L. Modern Java frameworks for scientific computing // Software: Practice and Experience. - 2021. - Vol. 51, No. 10.
22. Taylor R.N., Medvidovic N. Software architecture: foundations, theory, and practice // ACM Transactions on Software Engineering and Methodology. - 2021. - Vol. 30, No. 1. - Article 8.
23. Turner M.K., Phillips S.R. Extensible computational frameworks // ACM Transactions on Mathematical Software. - 2020. - Vol. 46, No. 2. - Article 15.
24. Walker J.P., Moore K.S. Thermal analysis with high-precision arithmetic // International Journal of Heat and Mass Transfer. - 2021. - Vol. 178.
25. Williams R.S., Taylor A.M. Dynamic analysis of mechanical systems

with high precision // Journal of Sound and Vibration. - 2020. - Vol. 487.

26. Wilson A.B., Clark M.D. Newton-Raphson methods for arbitrary precision arithmetic // Numerical Algorithms. - 2022. - Vol. 89. - P. 1123-1142.

27. Xu Y., Li H. Design principles for scientific software systems // ACM Computing Surveys. - 2021. - Vol. 54, No. 3. - Article 62.

28. Zhang L., Wang P. Cross-platform GUI development for scientific applications // Software: Practice and Experience. - 2022. - Vol. 52, No. 6.

29. Zhang W., Liu C. FFT-based multiplication algorithms for large integers // Mathematics of Computation. - 2021. - Vol. 90. - P. 2789-2814.

30. Абрамов А.І., Петренко С.М. Методи проектування програмного забезпечення для інженерних розрахунків // Вісник НТУ "ХПІ". - 2020. - №15. - С. 12-18.

31. Андрущенко О.Т., Семенов В.В. Методології розробки складного ПЗ // Кібернетика та системний аналіз. - 2019. - №5. - С. 167-174.

32. Валуєв О.М., Горобець В.К. Оптимізація технологічних процесів // Технологія машинобудування. - 2020. - №15. - С. 34-41.

33. Васильєв О.П., Коваленко Т.А. Об'єктно-орієнтовані підходи в розробці інженерного ПЗ // Прикладна математика і механіка. - 2019. - Т. 83, №4. - С. 45-52.

34. Головка В.А., Мельник С.І. Теоретичні основи побудови обчислювальних систем // Математичні машини і системи. - 2019. - №4.

35. Гриценко В.І., Жук К.Д. Концептуальне моделювання інженерних систем // Проблеми програмування. - 2020. - №2-3. - С. 45-52.

36. Дмитрієв А.В., Козлов П.С. Оптимізація алгоритмів довгої арифметики // Математичне моделювання. - 2021. - Т. 33, №7. - С. 78-89.

37. Зубарєв В.П., Мельник О.І. Програмні засоби технологічного проектування // САПР і графіка. - 2021. - №7. - С. 45-52.

38. Іваненко М.С., Сидоренко В.К. Розподілені обчислення в задачах механіки суцільних середовищ // Механіка та машинобудування. - 2021. - №2. - С. 67-74.

39. Іщенко В.М., Руденко К.А. Швидкі алгоритми множення великих чисел // Кібернетика та комп'ютерні технології. - 2020. - №1. - С. 34-42.
40. Козаченко Ю.В., Савенко П.О. Python-бібліотеки для наукових обчислень // Наукові записки НаУКМА. - 2021. - Т. 4. - С. 78-85.
41. Левченко В.Д., Мороз О.В. Концептуальні основи побудови інженерних розрахункових систем // Системні дослідження та інформаційні технології. - 2020. - №3. - С. 78-85.
42. Матвієнко С.Г., Богуш В.М. Plugin-архітектура для розрахункових систем // Штучний інтелект. - 2021. - №3. - С. 89-96.
43. Олійник А.П., Романенко В.С. Мікросервісна архітектура для інженерних обчислень // Інформаційні технології в машинобудуванні. - 2021. - №8. - С. 34-41.
44. Остапенко Т.С., Кравченко О.П. Розрахунок зношування в парах тертя // Проблеми тертя та зношування. - 2020. - №4. - С. 45-52.
45. Романов В.І., Сергєєв К.О. Розширювана архітектура інженерних додатків // Системи проектування. - 2020. - №12. - С. 67-73.
46. Степанов К.І., Богданов А.Л. Алгоритми високоточної арифметики в інженерних розрахунках // Наукові вісті КПП. - 2019. - №4.
47. Тимченко А.В., Шевченко Д.М. Моделювання процесів обробки матеріалів // Вісник машинобудування. - 2021. - №8. - С. 23-29.
48. Усенко В.А., Петров С.Г. Проблеми точності в чисельних методах // Обчислювальна математика. - 2020. - Т. 45, №2. - С. 123-130.
49. Уткін В.Б., Федоров А.С. Java-технології в інженерних розрахунках // Комп'ютерні науки та інженерія. - 2021. - №2. - С. 56-63.
50. Фоменко Т.М., Кириченко О.В. Тришарова архітектура інженерних додатків // Комп'ютерно-інтегровані технології. - 2021. - №1.
51. Ярошенко М.П., Волков Д.С. Інструментальні засоби розробки інженерного ПЗ // Програмні системи і обчислювальні методи. - 2020. - №3.
52. Яценко І.О., Кулик Я.А. Розподілені обчислення в САЕ-системах // Автоматизація виробничих процесів. - 2020. - №4. - С. 78-85.

ДОДАТОК А: Довідка про впровадження**ДЕРЖАВНИЙ КОНЦЕРН «УКРОБОРОНПРОМ»
ДЕРЖАВНЕ ПІДПРИЄМСТВО****«СУДНОБУДІВНИЙ ЗАВОД ІМЕНІ 61 КОМУНАРА»**

Вул. Адміральська, 38 м Миколаїв 54001;

Телефон (+380512) 670093 факс (+380512) 670093

Ідентифікаційний код за ЄДРПОУ 14313240

URL: www.shipyard61.com.ua

E- mail: info@shipyard61.com.ua

Д О В І Д К А П Р О В П Р О В А Д Ж Е Н Н Я

Видана **Кононову Максиму Анатолійовичу** у тому, що матеріали його бакалаврської роботи **«Програмний комплекс підтримки інженерних розрахунків у машинобудуванні»** (керівник к. т. н., доц. Романкевич В. О. плануються до впровадження як самостійний програмний продукт «EngineeringCalc 2.0».

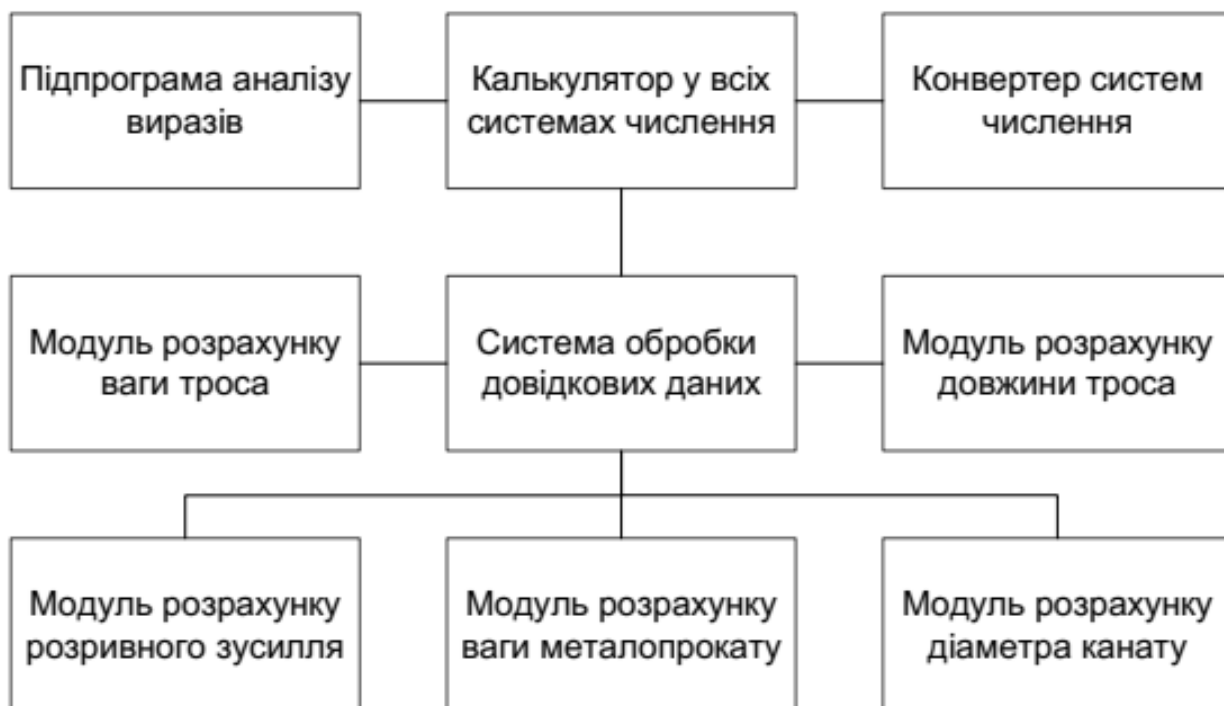
Розроблений комплекс успішно використовується та тестується на підприємстві. Завдяки даній програмі були суттєво зменшені часові витрати на розрахунок необхідної кількості різних видів металопродукату і тросу, розривного зусилля та діаметра канатів, збільшена точність обчислень, отримана можливість представлення результатів високоточних інженерних розрахунків у різних системах числення з метою їх шифрування. Крім того використання програми сприяє більш швидкому документуванню потреб у різноманітних матеріалах. Калькулятор може бути вдосконалений та доповнений новими функціональними можливостями в робочому порядку з урахуванням виробничих потреб.

Головний інженер

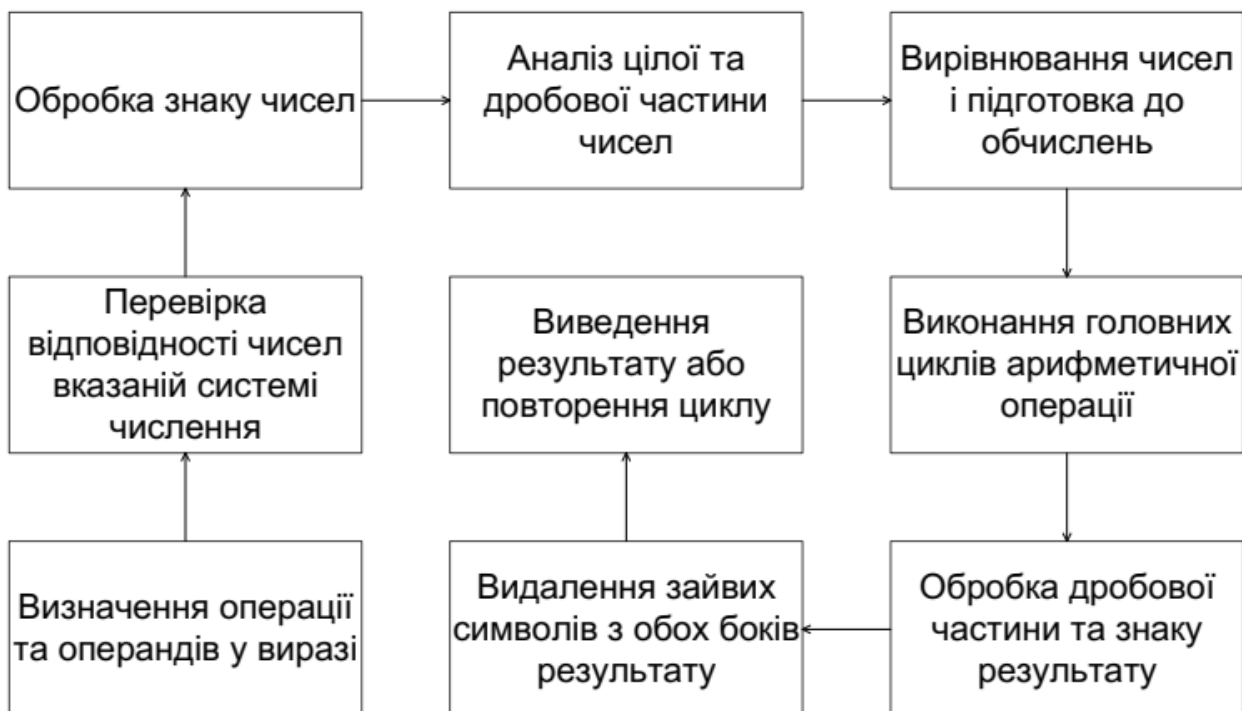
А. Р. Ярошевич

ДОДАТОК Б: Копії графічних матеріалів

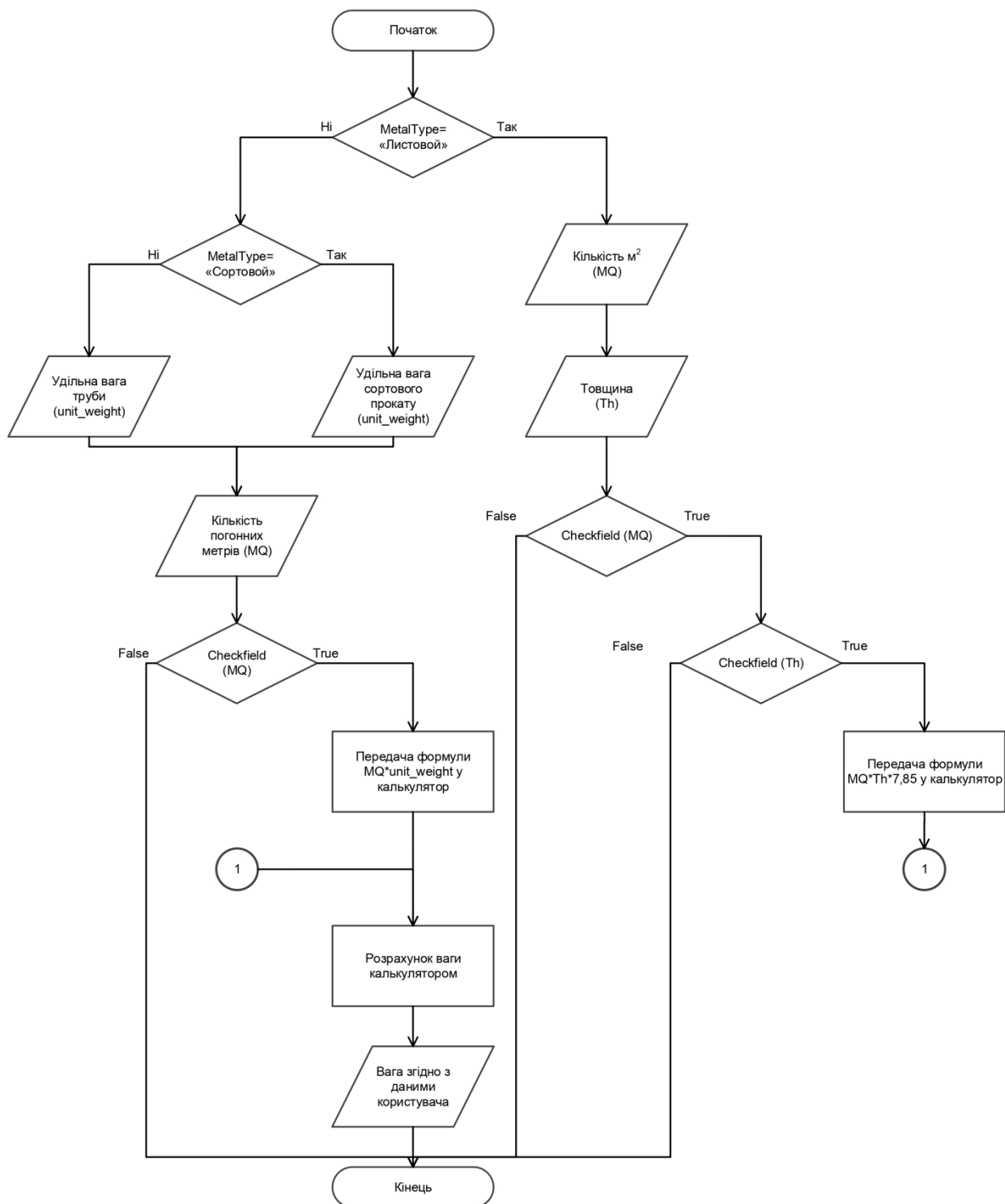
Будова програмного комплексу (схема структурна)



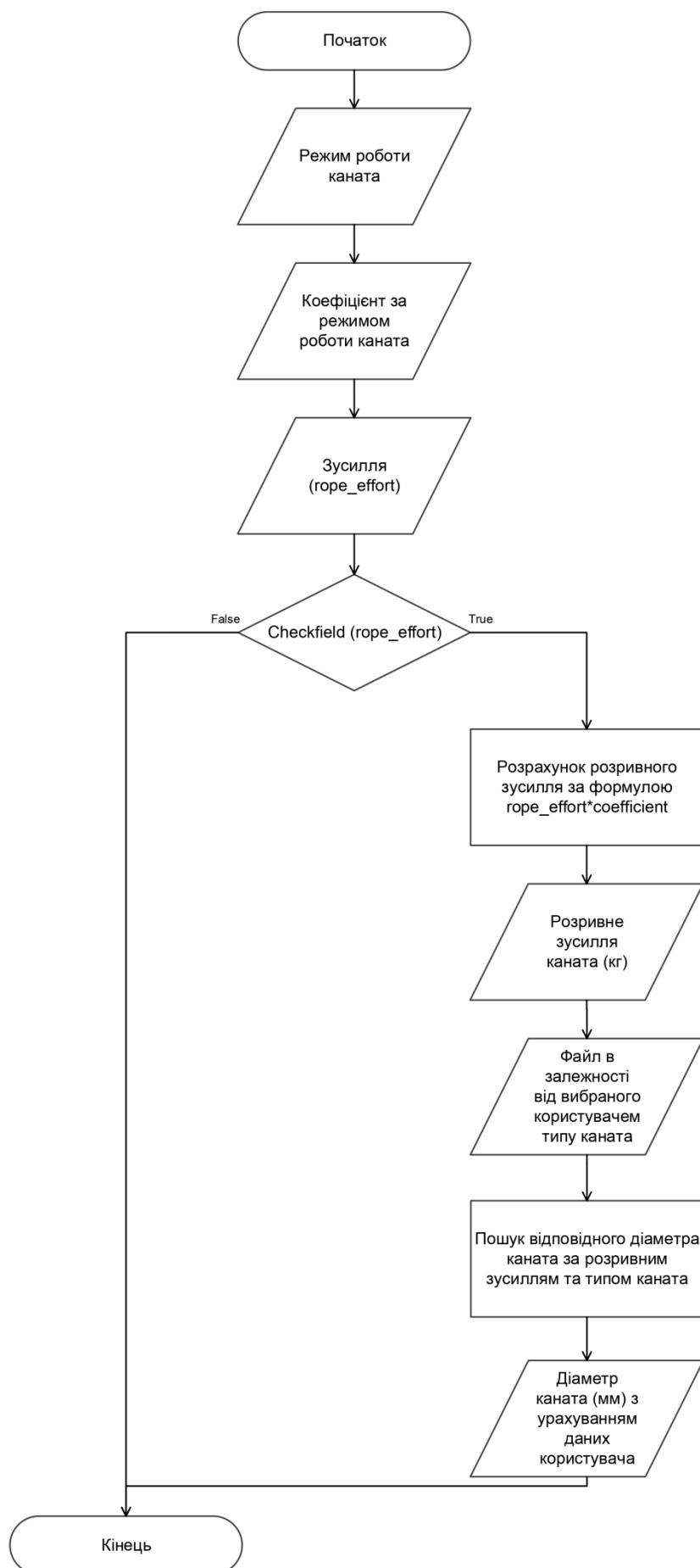
Робота калькулятора у різних системах числення (схема структурна)



Розрахунок ваги металопрокату (схема алгоритму)



Розрахунок розривного зусилля та діаметра каната (схема алгоритму)



ДОДАТОК В: Код програми

accessories.h

```

1  #pragma once
2  #include <iostream>
3  #if defined( WIN32) || defined( _WIN64)
4      #include <conio.h>
5      char my_getch(void);
6      char my_kbhit(void);
7  #else
8      #include <termios.h>
9      #include <unistd.h>
10     #include <fcntl.h>
11     char my_getch(void);
12     char my_kbhit(void);
13 #endif
14 #include <fstream>
15 #include <iomanip>
16 #include <limits.h>
17 #include <sstream>
18 #include <cfloat>
19 #include <cstring>
20 #include <cstdlib>
21 #include <algorithm>
22 #include <ctime>
23 #include <chrono>
24 #include <gmp.h>
25 #include <mpfr.h>
26 #include <stdlib.h>
27 #include <string.h>
28 #include <stdio.h>
29 #include <math.h>
30 #ifdef __cplusplus
31 #define DLL_EXPORT_FUNC extern "C" __declspec(dllexport)
32 #else
33 #define DLL_EXPORT_FUNC __declspec(dllexport)
34 #endif
35 #include "exports.h"
36
37 using namespace std;
38 const int FloatingPointNumberPrecision=2;
39 void InitFloatingPointAndRandomNumbers();
40 int GetCountOfDigits(int n);
41 char GetRandomDigit(char min,char max);
42 int GetRandomInt(int min,int max);
43 float GetRandomFloat(float min,float max);
44 double GetRandomDouble(double min, double max);
45 char GetRandomChar(char min,char max);
46 void ShuffleArrayOfCharacters(char* array);
47 void ShuffleArrayOfIntegers(int* array,int ArraySize);
48 void ShuffleString(string& str);

```

accessories.cpp

```

1  #include "accessories.h"
2  #include "globals.h"
3  #ifndef WIN32
4      char my_getch(void)
5      {
6          char my_kbhit(void)
7          {
8              #else
9              char my_getch(void){return getch();}
10             char my_kbhit(void){return kbhit();}
11             #endif
12             void InitFloatingPointAndRandomNumbers()
13             {
14                 int GetCountOfDigits(int n)
15                 {
16                     char GetRandomDigit(char min,char max)
17                     {
18                         int GetRandomInt(int min,int max)
19                         {
20                             char GetRandomChar(char min,char max)
21                             {

```

```

86 float GetRandomFloat(float min,float max)
87 {
93 double GetRandomDouble(double min,double max)
94 {
100 void ShuffleArrayOfCharacters(char* array)
101 {
112 void ShuffleArrayOfIntegers(int* array,int ArraySize)
113 {
123 void ShuffleString(string& str)
124 {
135

```

bignum.h:

```

1  #include "OptimizedBignum.h"
2  extern const int MaxBignumLength;
3  extern int MaxFractPartLength;
4  class bignum
5  {
6      public:
7      bignum();
8      char* plus(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
9      char* minus(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
10     char* multiply(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
11     char* divide(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
12 };

```

bignum.cpp

```

1  #include "bignum.h"
2  bignum::bignum(){}
3  char* bignum::plus(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
4  {
337 char* bignum::minus(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
338 {
529 char* bignum::multiply(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
530 {
822 char* bignum::divide(char* bignum1_orig,int Bignum1Length,char* bignum2_orig,int
Bignum2Length)
823 {
1342

```

OptimizedBignum.h

```

1  #include "RandomBignum.h"
2  extern const int MaxBignumLength;
3  extern int MaxFractPartLength;
4  int decimal_str_to_mpq(char *str, mpq_t rop);
5  char* mpq_to_decimal_str(mpq_t q);
6  int power(mpq_t rop, const mpq_t base, const mpq_t exp);
7  int validate_power_comprehensive(const mpq_t result, const mpq_t base, const mpq_t exp,
double tolerance);
8  class OptimizedBignum
9  {
10     public:
11     OptimizedBignum();
12     char* plus(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
13     char* minus(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
14     char* multiply(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
15     char* divide(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
16 };

```

OptimizedBignum.cpp

```

1  #include "OptimizedBignum.h"
2  #ifndef MPFR_PREC_BITS
3  #define MPFR_PREC_BITS 256
4  #endif
5  #ifndef MPFR_MAX_PREC
6  #define MPFR_MAX_PREC 65536
7  #endif
8  #ifndef MAX_EXACT_INT_EXP_BITS
9  #define MAX_EXACT_INT_EXP_BITS 30
10 #endif
11 #ifndef MAX_RATIONAL_ROOT_BITS
12 #define MAX_RATIONAL_ROOT_BITS 20
13 #endif

```

```

14   OptimizedBignum::OptimizedBignum(){}
15   int decimal_str_to_mpq(char *in_s, mpq_t out)
16   {
107  char* mpq_to_decimal_str(mpq_t q)
108  {
154  char* OptimizedBignum::plus(char* bignum1, int Bignum1Length, char* bignum2, int
      Bignum2Length)
155  {
190  char* OptimizedBignum::minus(char* bignum1, int Bignum1Length, char* bignum2, int
      Bignum2Length)
191  {
226  char* OptimizedBignum::multiply(char* bignum1, int Bignum1Length, char* bignum2, int
      Bignum2Length)
227  {
269  char* OptimizedBignum::divide(char* bignum1, int Bignum1Length, char* bignum2, int
      Bignum2Length)
270  {
312  static int mpq_invert_safe(mpq_t inv, const mpq_t src)
313  {
325  static void mpq_abs_safe(mpq_t dst, const mpq_t src)
326  {
330  static int mpfr_to_mpq_via_string_safe(mpq_t rop, mpfr_t f)
331  {
382  static int try_exact_root(mpz_t rop, const mpz_t n, unsigned long q)
383  {
393  static int mpq_pow_integer_exp(mpq_t rop, const mpq_t base, const mpz_t exp_int)
394  {
424  static int try_exact_rational_power(mpq_t rop, const mpq_t base, const mpz_t p, const mpz_t
      q, int negate_exp)
425  {
476  static int approximate_pow_with_retry(mpq_t rop, const mpq_t base, const mpq_t exp)
477  {
584  int power(mpq_t rop, const mpq_t base, const mpq_t exp)
585  {

703  int validate_power_result(const mpq_t result, const mpq_t base, const mpq_t exp, double
      tolerance)
704  {
771  int validate_power_result_log(const mpq_t result, const mpq_t base, const mpq_t exp, double
      tolerance)
772  {
825  int validate_power_comprehensive(const mpq_t result, const mpq_t base, const mpq_t exp,
      double tolerance)
826  {
891

```

RandomBignum.h

```

1   #include "accessories.h"
2   extern const int MaxBignumLength;
3   extern int MaxFractPartLength;
4   int GetPointPosition(char* number,int length);
5   bool IsLess(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
6   bool IsGreater(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
7   bool IsEqual(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
8   bool IsNotEqual(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length);
9   void DisplayBignum(char* number,int length);
10  void SetRandomIntBignum(char* number,int& length,char* min,char* max,int MinPointPos,int
      MaxPointPos);
11  void SetRandomFloatBignum(char* number,int& length,char* min,char* max,int MinPointPos,int
      MaxPointPos);
12  inline void SetRandomUnsignedBignum(char* number,int& length,char* min,char* max,bool
      SignMinus);
13  char* GetRandomBignum(int length,bool SignMinus,bool FloatingPoint);
14  char* GetRandomBignum(char* min,char* max);
15  void SwapBignums(char*& bignum1,int& Bignum1Length,char*& bignum2,int& Bignum2Length);

```

RandomBignum.cpp

```

1   #include "RandomBignum.h"
2   int GetPointPosition(char* number,int length)
3   {
10  bool IsLess(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
11  {
122  bool IsGreater(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
123  {

```

```

234 bool IsEqual(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
235 {
243 bool IsNotEqual(char* bignum1,int Bignum1Length,char* bignum2,int Bignum2Length)
244 {
252 void DisplayBignum(char* number,int length)
253 {
319 char* GetRandomBignum(int CountOfDigits,bool SignMinus,bool FloatingPoint)
320 {
355 void SetRandomIntBignum(char* number,int& length,char* min,char* max,int MinPointPos,int
MaxPointPos)
356 {
428 void SetRandomFloatBignum(char* number,int& length,char* min,char* max,int MinPointPos,int
MaxPointPos)
429 {
501 inline void SetRandomUnsignedBignum(char* number,int& length,char* min,char* max,bool
SignMinus)
502 {
591 char* GetRandomBignum(char* min,char* max)
592 {
692 void SwapBignums(char*& bignum1,int& Bignum1Length,char*& bignum2,int& Bignum2Length)
693 {
701

```

main.cpp

```

1  #include "bignum.h"
2  int MinCountOfDigits=1,MaxCountOfDigits=10;
3  int MaxFractPartLength=MaxCountOfDigits;
4  const int MaxBignumLength=MaxCountOfDigits+MaxFractPartLength;
5  int CountOfRanges=10,CountOfBignumPairs=10;
6  int CountOfDivisionWarnings=0;
7  int CountOfExpressions=1000,MinCountOfOperations=1,MaxCountOfOperations=10;
8  static char* normalize_positive_decimal(const char* s)
9  {
67  static char* abs_normalized(const char* x)
68  {
76  static int cmp_pos_dec(const char* a, const char* b)
77  {
105  char* shift_decimal_down(const char* A, int digits)
106  {
155  static char* make_relative_epsilon(const char* X, int p)
156  {
187  char* PowerOf10(int n)
188  {
196  template <typename BigNumClass> void ParseDecimalToFractionOutputOn(const char* str, char*&
numerator, char*& denominator)
197  {
276  template <typename BigNumClass> bool IsDivisionConsistentOutputOn(char* A, char* B, char* Q,
int precisionDigits)
277  {
386  template <typename BigNumClass> void ParseDecimalToFraction(const char* str, char*&
numerator, char*& denominator)
387  {
457  template <typename BigNumClass> bool IsDivisionConsistent(char* A, char* B, char* Q, int
precisionDigits)
458  {
554  template <typename BigNumClass> void TestAllBignumOperations(char*** BignumPairs)
555  {
638  template <typename BigNumClass> void TestOneBignumOperationWithCheck(char operation,char***
BignumPairs)
639  {
704  template <typename BigNumClass> void TestOneBignumOperation(char operation,char***
BignumPairs)
705  {
750  char*** GetBignumPairs(int MinCountOfDigits,int MaxCountOfDigits,int CountOfRanges,int
CountOfBignumPairs)
751  {
823  void TestOneMpq_tBignumOperation(char operation,mpq_t*** Mpq_tBignumPairs)
824  {
882  void TestOneMpq_tBignumOperationWithCheck(char operation,mpq_t*** Mpq_tBignumPairs)
883  {
983  void TestMpq_tPower(char*** BignumPairs)
984  {
1032  char* GetRandomArithmeticExpression(int OperationCount)
1033  {

```

```

1090 static char *strdup_s(const char *s)
1091 {
1100 static int is_op_char(char c) {return c == '+' || c == '-' || c == '*' || c == '/';}
1101 static int precedence(const char *op)
1102 {
1111 static int is_right_assoc(const char *op)
1112 {
1117 typedef struct
1118 {
1119     char **v;
1120     int top;
1121     int cap;
1122 } StrStack;
1123 static StrStack *stack_new(int cap)
1124 {
1132 static void stack_free_all(StrStack *s)
1133 {
1144 static int stack_push(StrStack *s, char *item)
1145 {
1157 static char *stack_pop(StrStack *s)
1158 {
1164 static char *stack_peek(StrStack *s)
1165 {
1169 static int stack_empty(StrStack *s){return s->top < 0;}
1170 static char* ApplyOperation_OutputOn(char *bignum1, char *bignum2, char *operation)
1171 {
1262 static char* ApplyOperation_OutputOff(char *bignum1, char *bignum2, char *operation)
1263 {
1338 char* CalculateExpression(const char *expr, char* ApplyOperation(char *bignum1, char *
bignum2, char *operation))
1339 {
1470 int main()
1471 {
1570

```

GCC MSYS2.bat

```

1 chcp 65001
2 echo off
3 cls
4 if exist program.exe (del program.exe)
5 C:\compilers\gcc_msys\ucrt64\bin\g++.exe main.cpp accessories.cpp bignum.cpp
OptimizedBignum.cpp RandomBignum.cpp -lgmp -lgmpxx -lmpfr -o program.exe
6 del program.obj
7 if exist program.exe (
8     cls
9     program.exe
10    echo.
11    pause
12 ) else (
13     echo.
14     pause
15 )

```

Visual Studio 2022.bat

```

1 @ECHO OFF
2 cls
3 if exist program.exe (del program.exe)
4 call "C:\compilers\Microsoft_Visual_Studio_2022\VC\Auxiliary\Build\vcvars64.bat"
5 REM === Compile sources ===
6 C:\compilers\Microsoft_Visual_Studio_2022\VC\Tools\MSVC\14.44.35207\bin\Hostx64\x64\cl.exe
/EHsc /O2 /std:c++17 /MT ^
7 /I"mpir\include" ^
8 /I"mpfr\include" ^
9 main.cpp accessories.cpp bignum.cpp OptimizedBignum.cpp RandomBignum.cpp
10 REM === Link objects with MPIR + MPFR ===
11 C:\compilers\Microsoft_Visual_Studio_2022\VC\Tools\MSVC\14.44.35207\bin\Hostx64\x64\link.exe
^
12 /OUT:program.exe main.obj accessories.obj bignum.obj OptimizedBignum.obj RandomBignum.obj ^
13 /LIBPATH:"mpir" mpir_Visual_Studio_2022.lib ^
14 /LIBPATH:"mpfr" mpfr_Visual_Studio_2022.lib
15 REM === Cleanup ===
16 del main.obj
17 del accessories.obj
18 del bignum.obj
19 del OptimizedBignum.obj
20 del RandomBignum.obj

```

```

21 set PATH=mpir;%PATH%
22 set PATH=mpfr;%PATH%
23 REM === Run if build succeeded ===
24 if exist program.exe (
25     cls
26     program.exe
27     echo.
28     pause
29 ) else (
30     echo.
31     pause
32 )

```

linux_compile.sh

```

1  #!/bin/bash
2  # Increase stack size limit
3  ulimit -s 65536 # 16MB stack size
4  echo "Stack limit set to: $(ulimit -s)KB"
5  # Compile the C++ program
6  g++ -Wall -o program main.cpp accessories.cpp bignum.cpp OptimizedBignum.cpp RandomBignum.cpp
7  -lmpxx -lomp -lmpfr
8  # Check if compilation succeeded
9  if [ $? -ne 0 ]; then
10     echo "Compilation failed."
11     read -n 1 -s
12     exit 1
13 fi
14 echo "Compilation successful."
15 # Run the program
16 ./program
17 # Check if program crashed
18 if [ $? -eq 139 ]; then
19     echo "Program crashed with segmentation fault."
20 else
21     echo "Program executed successfully."
22 fi
23 echo "Press any key to exit..."
24 read -n 1 -s
25 echo ""

```

Функція для запуску тестування випадкових виразів з бібліотеки dll

```

678 DLL_EXPORT_FUNC char* TestExpressions()
679 {
680     UsedMemory=0;
681     buffer[0] = '\0';
682     char* (*ApplyOperation)(char* a, char* b, char *op);
683     ApplyOperation=ApplyOperation_OutputOn;
684     WrittenMemory=snprintf(buffer+UsedMemory,BufferSize-UsedMemory,
685     "MIN COUNT OF DIGITS: %d; MAX COUNT OF DIGITS : %d; FRACT PART LENGTH : %d\nMAX BIGNUM
LENGTH : %d; COUNT OF BIGNUM PAIRS: %d;\nCOUNT OF EXPR. : %d; MIN COUNT OF OPS.
: %d; MAX COUNT OF OPS. : %d\n",MinCountOfDigits,MaxCountOfDigits,MaxFractPartLength,
MaxBignumLength,CountOfBignumPairs,CountOfExpressions,MinCountOfOperations,
MaxCountOfOperations);
686     UsedMemory += (size_t)WrittenMemory;
687     WrittenMemory=snprintf(buffer+UsedMemory,BufferSize-UsedMemory,
"=====\\n");
688     UsedMemory += (size_t)WrittenMemory;
689     /*char* RandomExpression=(char*)malloc(sizeof(char)*1000);
690     strcpy(RandomExpression,"2.590439 + 4.9922014 - 0.53322419 / 60644425 + -65618 *
-330.061 / 0.7422 + -75");
691     WrittenMemory=snprintf(buffer+UsedMemory,BufferSize-UsedMemory, "%s\\n",
RandomExpression);
692     UsedMemory += (size_t)WrittenMemory;
693     char*
RandomExpressionResult=CalculateExpression(RandomExpression,ApplyOperation_OutputOn);
694     WrittenMemory=snprintf(buffer+UsedMemory,BufferSize-UsedMemory,"RESULT:
%s\\n",RandomExpressionResult);
695     UsedMemory += (size_t)WrittenMemory;*/
696     InitFloatingPointAndRandomNumbers();

```

```

697     auto StartTime=chrono::system_clock::now();
698     for(int i=0; i<CountOfExpressions; i++)
699     {
700         char* RandomExpression=GetRandomArithmeticExpression(GetRandomInt(
701             MinCountOfOperations,MaxCountOfOperations));
702         WrittenMemory=sprintf(buffer+UsedMemory,BufferSize-UsedMemory, "%d. %s\n", i+1,
703             RandomExpression);
704         UsedMemory += (size_t)WrittenMemory;
705         char* RandomExpressionResult=CalculateExpression(RandomExpression,ApplyOperation);
706         WrittenMemory=sprintf(buffer+UsedMemory,BufferSize-UsedMemory,"RESULT: %s\n",
707             RandomExpressionResult);
708         UsedMemory += (size_t)WrittenMemory;
709         free(RandomExpression);
710         free(RandomExpressionResult);
711         WrittenMemory=sprintf(buffer+UsedMemory,BufferSize-UsedMemory,
712             "=====\n");
713         UsedMemory += (size_t)WrittenMemory;
714     }
715     auto EndTime=chrono::system_clock::now();
716     chrono::duration<double> duration=EndTime-StartTime;
717     WrittenMemory=sprintf(buffer+UsedMemory,BufferSize-UsedMemory,"Count of errors:
718         %d;\nCount of division warnings: %d\nTime: %.6f seconds",CountOfErrors,
719         CountOfDivisionWarnings,duration.count());
720     UsedMemory += (size_t)WrittenMemory;
721     return buffer;
722 }

```

exports.h

```

1  #pragma once
2  #ifdef WIN32
3  #ifdef BUILD_DLL
4      #define DLL_EXPORT extern "C" __declspec(dllexport)
5  #else
6      #define DLL_EXPORT extern "C" __declspec(dllimport)
7  #endif
8  #else
9      #define DLL_EXPORT extern "C"
10 #endif

```

globals.h

```

1  #pragma once
2  #include "exports.h"
3  DLL_EXPORT int MinCountOfDigits;
4  DLL_EXPORT int MaxCountOfDigits;
5  DLL_EXPORT int CountOfExpressions;
6  DLL_EXPORT int MinCountOfOperations;
7  DLL_EXPORT int MaxCountOfOperations;
8  extern int MaxFractPartLength;
9  extern int MaxBignumLength;
10 DLL_EXPORT void UpdateLimits(void);

```

globals.cpp

```

1  #include "globals.h"
2  int MinCountOfDigits=1;
3  int MaxCountOfDigits=10;
4  int CountOfExpressions=1000;
5  int MinCountOfOperations=1;
6  int MaxCountOfOperations=10;
7  int MaxFractPartLength=MaxCountOfDigits;
8  int MaxBignumLength=MaxCountOfDigits+MaxFractPartLength;
9  DLL_EXPORT void UpdateLimits(void)
10 {
11     if (MaxCountOfDigits < 1)
12         MaxCountOfDigits = 1;
13     MaxFractPartLength = MaxCountOfDigits;
14     MaxBignumLength = MaxCountOfDigits + MaxFractPartLength;
15 }

```

GCC MSYS2_BuildDLL.bat

```

1 chcp 65001
2 echo off
3 cls
4 if exist program.exe (del program.exe)
5 C:\compilers\gcc_msys\ucrt64\bin\g++ -shared -DBUILD_DLL -o ../\EngineeringCalc2.0\bignums.dll
   globals.cpp main.cpp accessories.cpp bignum.cpp RandomBignum.cpp -lgmp -lgmpxx -lmpfr
6 echo.
7 pause

```

bignums.pas

```

1 unit bignums;
2 {$mode objfpc}{$H+}
3 interface
4 function TestExpressions():PChar; cdecl; external 'bignums.dll';
5 procedure UpdateLimits; cdecl; external 'bignums.dll';
6 var
7     MinCountOfDigits: integer; cvar; external 'bignums.dll';
8     MaxCountOfDigits: integer; cvar; external 'bignums.dll';
9     CountOfExpressions: integer; cvar; external 'bignums.dll';
10    MinCountOfOperations: integer; cvar; external 'bignums.dll';
11    MaxCountOfOperations: integer; cvar; external 'bignums.dll';
12 implementation
13 end.

```

Функція запуску тестування випадкових виразів у середовищі Lazarus:

```

699 procedure TForm1.GenerateAndCalculate_ButtonClick(Sender: TObject);
700 var
701     output: PChar;
702 begin
703     MinCountOfDigits:=MinCountOfDigits_SpinEdit.Value;
704     MaxCountOfDigits:=MaxCountOfDigits_SpinEdit.Value;
705     CountOfExpressions:=CountOfExpressions_SpinEdit.Value;
706     MinCountOfOperations:=MinCountOfOperations_SpinEdit.Value;
707     MaxCountOfOperations:=MaxCountOfOperations_SpinEdit.Value;
708     UpdateLimits;
709     output:=TestExpressions();
710     memo.text:='';
711     memo.text:=output;
712 end;

```