КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

«Система автоматизованого складання варіантів завдань лабораторних робіт, пов'язаних з обчисленнями»

Виконав студент 4-го курсу
Максим КУЗЬМИК _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Тетяна КАРНАУХ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 23 ілюстрації, 10 джерел посилань.

СКЛАДАННЯ ВАРІАНТІВ, LATEX, ДЕРЕВА, КЛАСИ

Об’єктом роботи є розроблення та застосування специфічного синтаксичного дерева, як шаблона для утворення виразів.

Метою роботи є створення системи автоматизованого складання завдань для лабораторних робіт, пов’язаних з обчисленнями.

Інструменти розроблення: мова програмування Python, інтегроване середовище розроблення PyCharm 2022.2 (Community Edition).

Результати роботи: розроблено та реалізовано алгоритм породження виразів за шаблоном; побудовано шаблон, за яким генеруються різні варіанти виразів; реалізовано систему, яка генерує задану кількість варіантів.

Створювана система спрямована на полегшення процесу складання варіантів та забезпечення різноманітності завдань для студентів, що дозволить покращити ефективність та якість навчального процесу. Хоч вона націлена на вивчення програмування, але використані в ній ідеї можуть бути адаптовані і для інших навчальних дисциплін.

Для подальших досліджень перспективним напрямком може бути автоматична генерація шаблонів заданої складності (наприклад, з певною кількістю операцій чи з обов’язковою присутністю певних елементів).

ЗМІСТ

С.

ВСТУП

Актуальність роботи. Дисципліни, що розпочинають вивчення програмування в навчальних закладах, наприклад "Програмування", "Основи програмування" тощо передбачають виконання низки лабораторних робіт. Доволі класичними є завдання з обчислення виразів, які розвивають навички роботи з стандартною бібліотекою математичних функцій та закріплюють знання правил обчислення операторів мови програмування. Використання з року в рік одних тих самих завдань провокує порушення академічної доброчесності з боку студентів, тому щороку виникає потреба складати значну кількість різних і у той самий час приблизно однакових за складністю завдань, що є доволі трудомісткою задачею. До того ж, людині доволі складно вручну контролювати рівень складності, коли складається порядку 100 різних варіантів. Розроблювана система призначена полегшити цей процес, бо система сама буде створювати варіанти завдань тільки потрібно буде задати їх кількість.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає у створенні системи автоматизованого складання завдань для лабораторних робіт, пов’язаних з обчисленнями.

Щоб мета роботи була досягнута, потрібно:

- Підібрати засоби для зображення виразів.
- Розробити алгоритм генерування виразів.
- Реалізувати програмне забезпечення, яке генерує задану кількість виразів.

Об’єктом розроблення є специфічне синтаксичне дерево, що являє собою шаблон для утворення виразів.

Засоби розроблення: мова програмування Python, інтегроване середовище розроблення PyCharm 2022.2 (Community Edition).

Можливі сфери застосування. Створювана система спрямована на полегшення процесу складання варіантів та забезпечення різноманітності завдань для студентів, а також для покращення ефективності та якості навчального процесу. Хоч вона націлена на вивчення програмування, але використані в ній ідеї можуть бути адаптовані і для інших навчальних дисциплін.

1 ВИКОРИСТОВУВАНІ ПОНЯТТЯ ТА ЗАСОБИ

1.1 Зображення виразів синтаксичними деревами

Дерева – це нелінійний абстрактний тип даних з ієрархічною структурою [1]. Складається з вузлів, які з'єднанні за допомогою посилань. Ця структура походить від одного вузла, що має назву корінь дерева, та має піддерева, які з'єднані з коренем і також листки- це вузли, що не мають дочірніх вузлів.

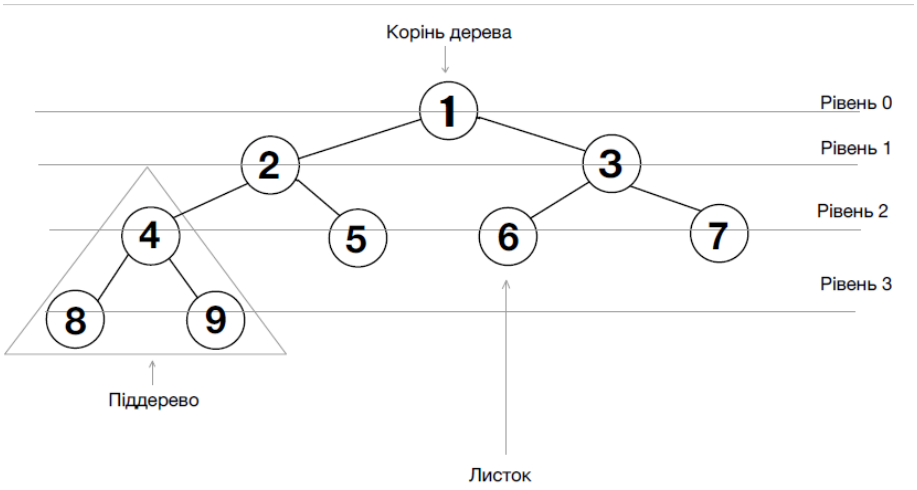

Рисунок 1.1 – Зразок дерева

Синтаксичне дерево виразу — це впорядковане дерево, мітками проміжних вершин якого є позначення операцій (оператори, позначки функцій), а мітками листів — операнди.

Описати будову синтаксичного дерева можна паралельно з будовою виразу.

Якщо arg є операндом (константа, змінна; перелік залежить від того, що саме дозволено використовувати у виразах, що означаються), то йому відповідає дерево з єдиного вузла, міткою якого є цей операнд.

Якщо є два вирази $expr_1$ та $expr_2$, яким відповідають дерева T_1 та T_2 з

коренями v_1 та v_2 відповідно, і є бінарний оператор op, то виразу $(expr_1)op(expr_2)$, відповідає дерево, коренем якого є новий вузол, його дітьми є вузли v_1 та v_2, а міткою op (див. рис. 1.2).

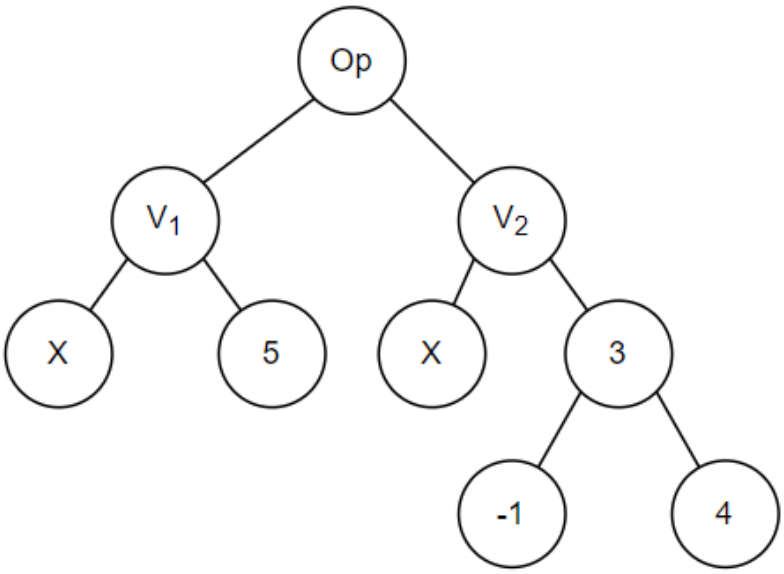

Рисунок 1.2 – Приклад опису побудови

Зазначимо, що синтаксичне дерево не потребує запису у своїй структурі дужок, адже їх функцію природним чином виконує структура дерева.

Наприклад, виразу $2 * (3 + 4) - 5$ відповідає таке синтаксичне дерево.

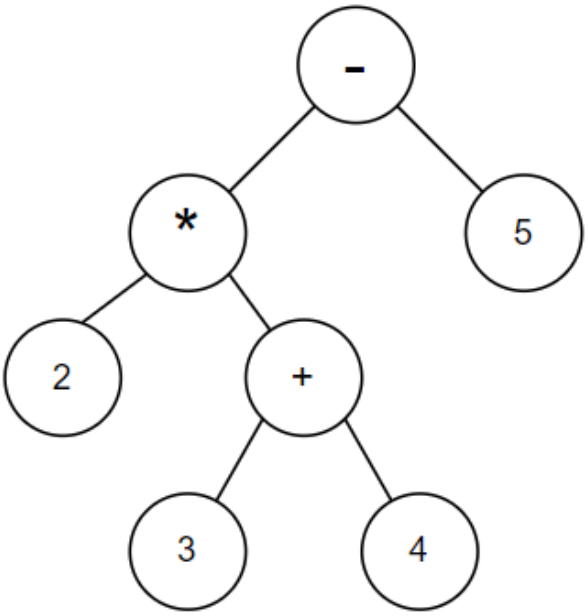

Рисунок 1.3 – Синтаксичне дерево виразу

1.2 Патерн Composite

Патерн Composite є структурним патерном проектування, який дозволяє об'єднати об'єкти в ієрархію таким чином, щоб їх можна було використовувати як один об'єкт або колекцію об'єктів [2]. Основна ідея цього патерну полягає в тому, що кожен об'єкт у структурі може бути розглянутий як компонент, незалежно від того, чи є він простим об'єктом чи складовим компонентом, який може містити інші об'єкти. Усі компоненти мають спільний інтерфейс, що дозволяє їх використовувати уніфіковано.

Основні компоненти патерна Компонувальник включають:

Компонент Component - визначає спільний інтерфейс для всіх об'єктів у структурі, незалежно від того, чи є вони простими об'єктами чи складовими компонентами.

Компонент Leaf - представляє простий об'єкт, який не може містити інші об'єкти.

Компонент Composite - представляє компонент, який може містити інші об'єкти, включаючи прості компоненти та інші складові компоненти.

Основна перевага патерна Компонувальник полягає в тому, що він дозволяє працювати зі складними структурами об'єктів так само, як зі стандартними об'єктами. Він спрощує кодування, підтримку коду та розширення функціональності.

Цей патерн часто використовується в програмуванні інтерфейсів користувача, управлінні деревами, структурах документів та інших ситуаціях, де необхідно працювати зі структурованою колекцією об'єктів.

1.3 Запис математичних виразів у системі LaTeX

LaTeX – це система підготовки документів для високоякісного друку [3]. Використовується в багатьох наукових та технічних галузях, став стандартом для написання різних статей, дисертацій, звітів та інших документів. У LaTeX дуже зручно описувати математичні формули, графіки, таблиці. Тому було обрано саме цю високорівневу мову для створення кінцевого файлу.

У LaTeX є спеціальний синтаксис для запису арифметичних виразів та математичних функцій, що дозволяє зручно відображати різні математичні символи та операції [4]. Ось основні особливості запису арифметичних виразів у LaTeX:

Запис математичного виразу. Щоб позначити, що текст є математичним виразом, використовуються знаки долара ($). Наприклад, $x + y$ позначає суму двох змінних x і y і відображається як $x + y$.

Математичні символи. LaTeX має велику кількість математичних символів. Наприклад, для позначення грецької літери "альфа" використовується \alpha , а для грецької літери "бета" - \beta. Існують також спеціальні команди для зображення графічних операторів, таких як ≤, ≥ тощо.

Математичні функції: LaTeX має багато вбудованих математичних функцій, таких як \sin для синуса, \cos для косинуса, \log для логарифма та багато інших. Вони можуть бути використані для зручного відображення математичних функцій у виразах.

Щоб показати дріб, потрібно використати команду \frac{чисельник}{знаменник}. Наприклад, \frac{1}{2} відображає дріб $\frac{1}{2}$.

Для піднесення числа a до степеня b використовують команду a^b. Наприклад, 2^3 показує піднесення числа 2 до кубу, тобто 2^3. Команда ‘‘^’’ діє на наступний символ, якщо потрібно на декілька, використовуються дужки “{}”.

Для зображення нижніх індексів можна використовувати команду _

(позначається символом підкреслення), наприклад a_b задає a_b, \log_a b у графіці відображається як $\log_a b$.

Фігурні дужки: Для позначення групи виразів, які мають бути розглянуті разом, можна використовувати фігурні дужки { }. Наприклад, \sqrt{2x + 1} показує корінь квадратний з виразу 2x + 1, тобто $\sqrt{2x+1}$.

Цей код відображає математичні вирази, включаючи функцію косинуса, дроби, піднесення до степеню та використання фігурних дужок для групування виразів:

```
$y = \cos(x)$

$\frac{1}{2} + \frac{3}{4}$

$e^{x}$

$f(x) = \left(\frac{1}{2x} + \frac{3}{4}\right) \cdot \sqrt{x^2 + 1}$
```

На рисунку 1.4 показано, що виведе код

$$y = \cos(x)$$
$$\frac{1}{2} + \frac{3}{4}$$
$$e^{x}$$
$$f(x) = \left(\frac{1}{2x} + \frac{3}{4}\right) \cdot \sqrt{x^2 + 1}$$

Рисунок 1.4 – вивід прикладу коду LaTeX

В останньому прикладі фігурні дужки використовуються для групування двох дробових виразів ($\frac{1}{2x}$ і $\frac{3}{4}$) разом зі значенням квадратного кореня (x^2+1). Вони дозволяють показати, що ці вирази пов'язані між собою та мають бути розглянуті як одна група.

Використання фігурних дужок допомагає чітко вказати, які частини виразу належать до групи.

З допомогою LaTeX ви можете зручно відображати складні математичні вирази з точністю та стилізацією, необхідними для завдань, пов'язаних з обчисленнями.

2 ВИКОРИСТОВУВАНЕ ЗОБРАЖЕННЯ ВИРАЗУ

2.1 Загальний підхід до зображення виразів у програмі

Побудову виразу будемо здійснювати, будуючи його синтаксичне дерево. У такому випадку проміжні вузли дерева задають операції, що застосовуються до виразів, зображуваних деревами, що є безпосередніми нащадками вузла [5]. Мітками листових вершин будуть аргументи.

Для зображення дерева в програмі сконструюємо ієрархію класів, що будуть задавати вузли дерева.

Будову дерева можна вести двома способами: або використовувати один клас вузлів, параметризуючи вузли функціями та операторами, які цей вузол задає, або використати ієрархію класів вузлів, де вузли, що відповідають, наприклад, різним операторам, задаються об’єктами різних класів [6]. Другий підхід більше притаманний об’єктно-орієнтованому програмуванню і за збільшення функціоналу дозволить уникнути складної умовної логіки, коли за різних параметрів треба буде по-різному здійснювати обчислення чи побудову виразу. Отже, будуємо ієрархію класів вузлів.

2.2 Базовий клас вузлів

Базовий клас вузлів Base у поточній версії забезпечує такий набір операцій, що мають вигляд методів:

calculate — виконує обчислення виразу з коренем у вузлу,

latex — будує латех-формулу виразу з коренем у вузлу,

node — повертає мітку вершини.

Обчислення виразу відбувається рекурсивно за структурою дерева: за обчислених значень підвиразів до отриманих значень застосовується операція, що відповідає кореню.

Для побудови латех-формули слід враховувати тип операції (префіксна – оператор передує аргументам, або бінарна інфіксна – оператор записується між аргументами; постфіксні операції у цій роботі не розглядаємо, адже вони використовуватись не будуть).

Загально відомо, що обхід синтаксичного дерева в симетричному порядку (спочатку обходиться ліва дитина, потім корінь, потім права дитина) дозволяє побудувати формулу виразу [7]. Але ту саму побудову рядка з виразом насправді можна виконати і за оберненого обходу дерева (спочатку обійти всіх дітей і отримати їх вирази, а потім побудувати вираз для кореня) [8]. Зважаючи на те, що використовуються не тільки інфіксні, але й префіксні операції, то побудову будемо виконувати оберненим обходом дерева, запам’ятовуючи вирази піддерев у змінних.

Синтаксичне дерево однозначно задає повний дужковий запис (тобто запис, в якому явно розставлено всі дужки), і цей запис можна було б і будувати. Але в такому випадку загальна кількість дужок не сприятиме читабельності виразу. Отже, будемо явно задавати ті місця виразу, де треба поставити дужки. Отже, вводимо клас вузлів Parenthethis (див. рис. 2.1), похідний від Base, який призначений зображувати дужки. Крім цього класу безпосередніми нащадками Base є Argument, Base1, Base2, про які мова піде далі.

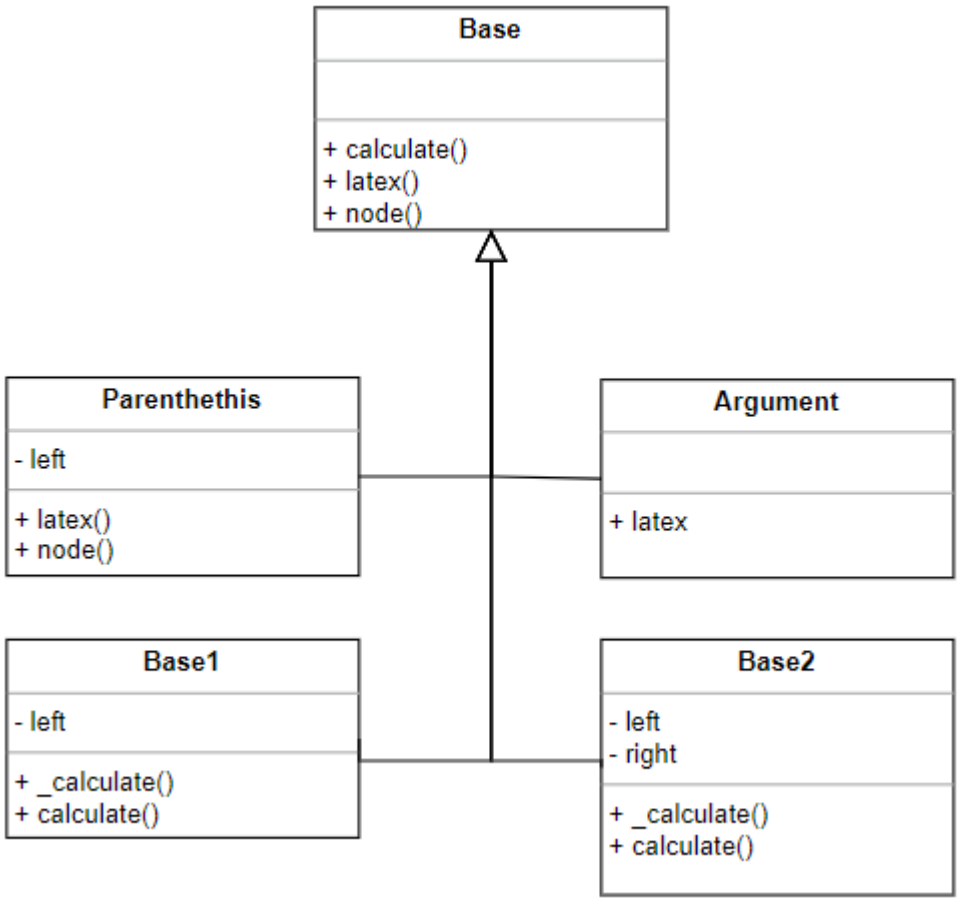

Рисунок 2.1 – Клас Base та його нащадки

2.3 Вузли, що зображують аргументи

Далі розробимо ієрархію класів для вузлів-аргументів (листових). Аргументом може бути іменована константа (як число π), числова константа (тоді в об’єкті вузла слід зберігати значення) або змінна (тоді в об’єкті вузла зберігаємо її ім’я). Усі аргументи мають спільну логіку побудови латех-формули: мітка вузла, взята у фігурні дужки. Підвирази будемо заключати у фігурні дужки через особливості запису латеху. Наприклад, x_13 у латесі позначає x13, а не x13. Для отримання останнього слід використовувати x_{13}.

Загальний клас аргументів Argument успадковується від базового класу вузлів Base та задає реалізацію методу latex, який далі успадковується його нащадками. Класи, похідні від Argument, зображені на рис. 2.2.

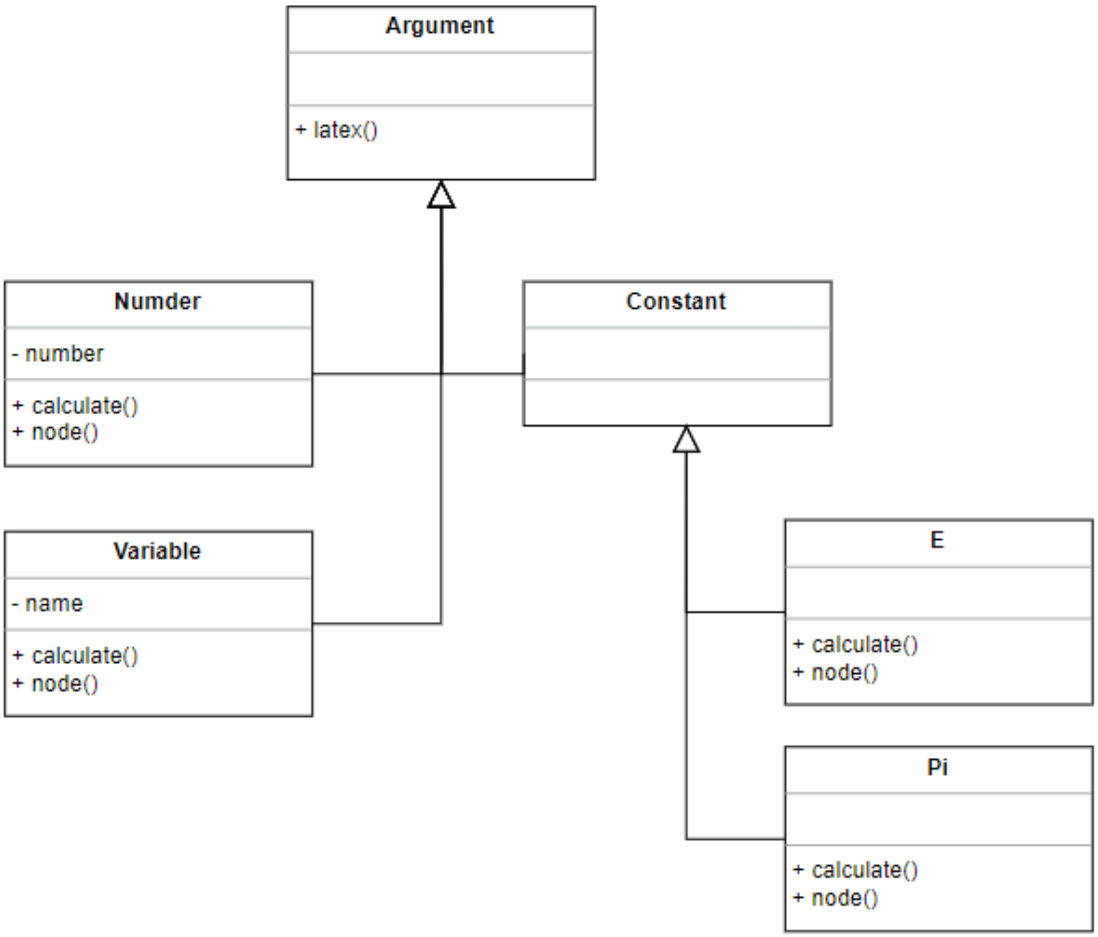

Рисунок 2.2 – Клас Argument та похідні від нього

Клас Variable зображує змінну, при конструюванні отримує ім’я змінної та запам’ятовує його в об’єкті класу, одночасно реєструючи в таблиці імен змінних – словнику, що задає відповідність між іменами та значеннями. На початку іменам зіставляється значення None. Міткою такого вузла є ім’я змінної, а значенням – значення змінної.

Клас Number зображує число, при конструюванні отримує числове значення та запам’ятовує його в об’єкті класу. Міткою такого вузла є зображення числа, а значенням – саме запам’ятоване число.

Клас Constant призначений зображувати іменовані константи. З одного боку, константу можна було б задати параметрично або як специфічну змінну. У даній реалізації використовуються математичні константи *e* та π, які реалізовані як класи-нащадки E та Pi класу Constant. У цих класах перевизначається метод calculate, який повертає математичне значення константи, та метод node, який повертає зображення константи для використання у латех. Так для константи π має бути рядок \pi.

2.4 Вузли, що зображують унарні операції

Клас Base1, який також похідний від Base, зображує унарні операції. Він задає спільну логіку створення вузла унарної операції: операція має єдиний аргумент, а також логіку обчислення: щоб обчислити вузол, слід операцію вузла застосувати до значення єдиної дитини.

Проте побудова латех-формули не є однаковою для всіх унарних операцій: у деяких випадках дужки ставимо (фактично операцією є застосування функції), у деяких випадках просто записуємо оператор попереду аргументу. Тому клас Base1 має двох нащадків Prefix1 та Func1, які розгалужують побудову латех-формули.

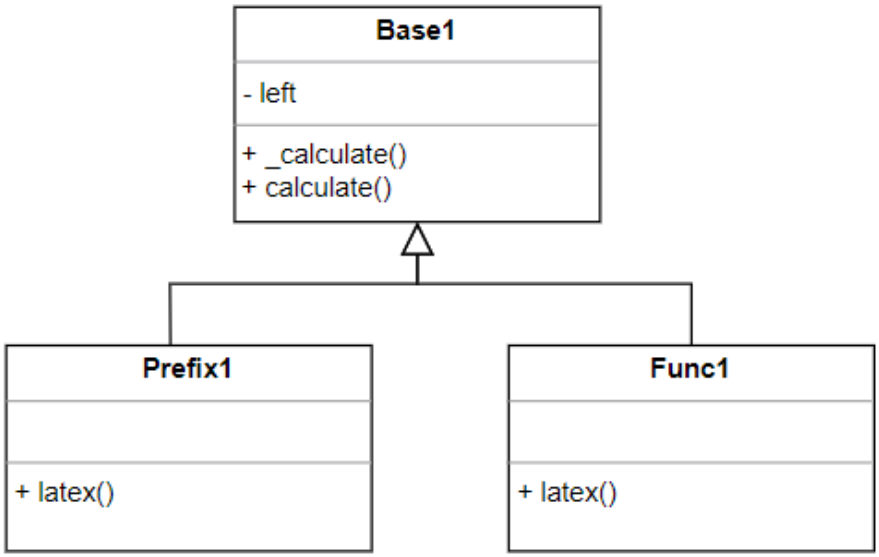

Рисунок 2.3 – Клас Base1 та похідні від нього

Клас Prefix1 унарних операторів зображує префіксні оператори (при побудові латех-формули дужки навколо аргументу не ставимо). Його нащадками є класи:

— Minus1 – зображує унарний мінус,

— Plus1 – зображує унарний плюс,

— Exp – зображує константу e у степені,

— Sqrt – зображує корінь квадратний.

Кожен з цих класів задає обчислення своєї операції та мітку вершини.

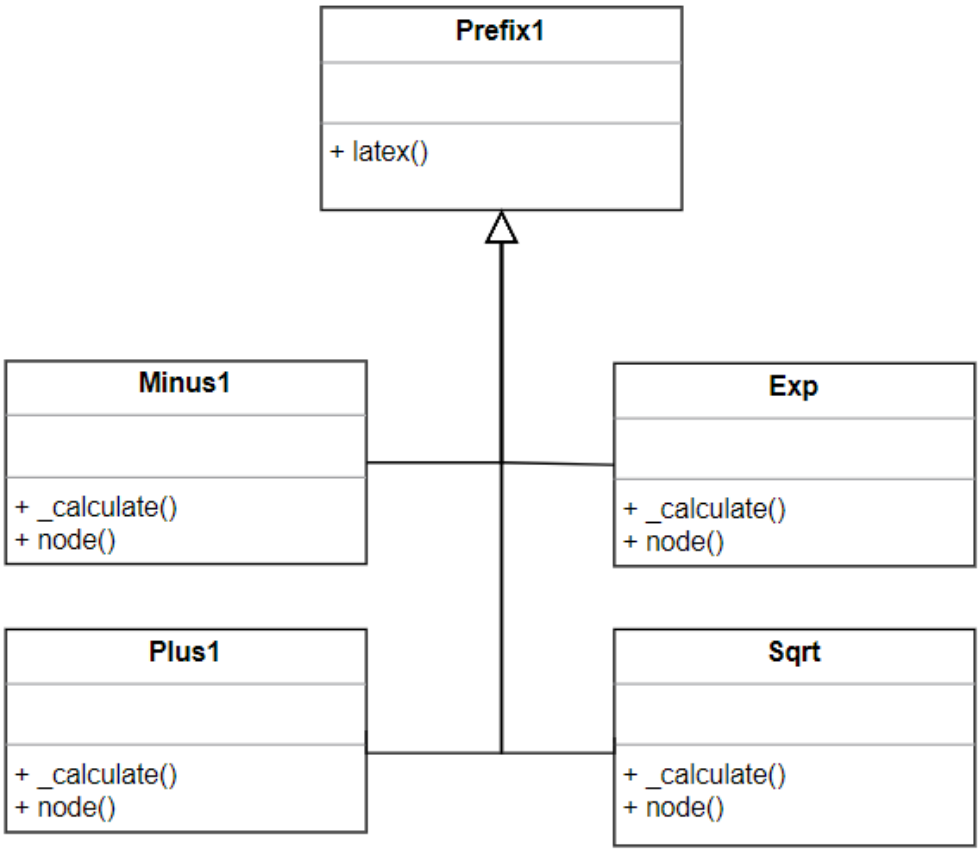

Рисунок 2.4 – Клас Prefix1 та похідні від нього

Клас Func1 унарних операторів зображує функції одного аргументу (при побудові латех-формули дужки навколо аргументу ставимо). Його нащадками є класи:

— Sin – зображує сінус,

— Cos – зображує косінус,

— Tg – зображує тангенс,

— Ctg – зображує котангенс,

— Arcsin – зображує арксінус,

— Arccos – зображує арккосінус.

Кожен з цих класів також задає обчислення своєї операції та мітку вершини.

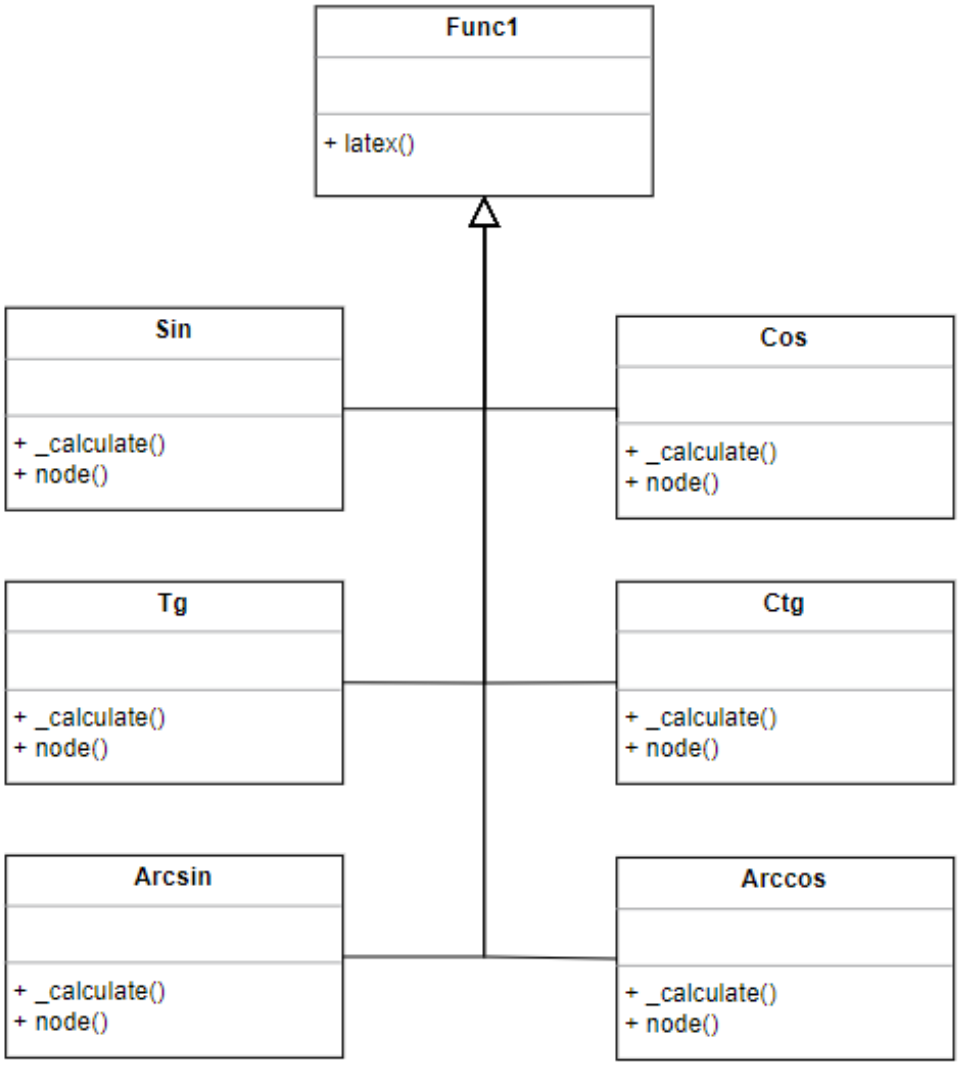

Рисунок 2.5 – Клас Func1 та похідні від нього

2.5 Вузли, що зображують бінарні операції

Клас Base2, який також похідний від Base, зображує бінарні операції. Він задає спільну логіку створення вузла бінарної операції: операція має два аргументи, а також логіку обчислення: щоб обчислити вузол, слід операцію вузла застосувати до значень дітей.

Знов-таки, побудова латех-формули не є однаковою для всіх бінарних операцій: у деяких випадках відповідний оператор є інфіксним (оператор записується між своїх аргументів), у деяких випадках оператор є префіксним (спочатку записується оператор, а потім два його аргументи). Тому клас Base2 має двох нащадків Infix2 та Prefix2, які розгалужують побудову латех-формули.

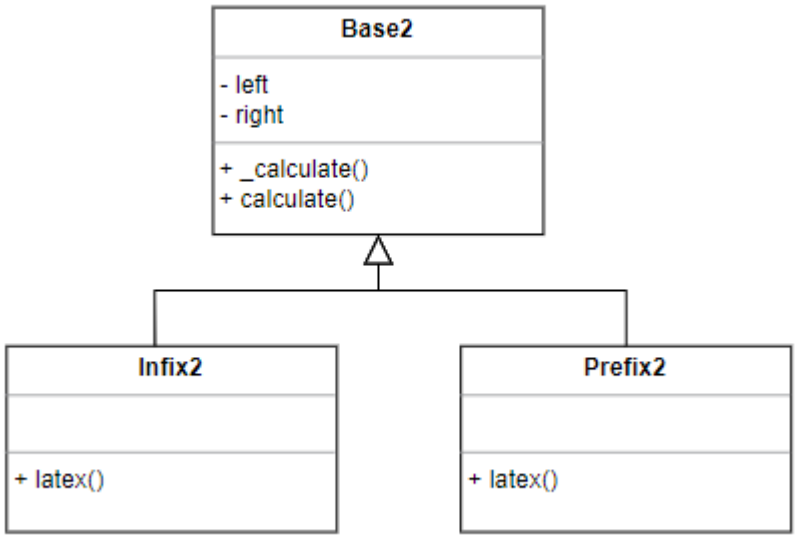

Рисунок 2.6 – Клас Base2 та похідні від нього

Клас Infix2 бінарних операторів зображує інфіксні оператори. Його нащадками є класи:

— Minus2 – зображує бінарний мінус,

— Plus2 – зображує бінарний плюс,

— Mult2 – зображує множення,

— TrueDiv – зображує дійсне ділення,

— Div2 – зображує ділення націло,

— Mod2 – зображує остачу від ділення націло,

— Power2 – зображує піднесення до степеню.

Кожен з цих класів задає обчислення своєї операції та мітку вершини.

Оскільки знак множення в математичних виразах може опускатись, то для забезпечення цієї можливості, до класу Mult2 було додано нащадки Mult2Mute та Mult2None, які замість знаку множення використовують або пробіл, або порожній рядок.

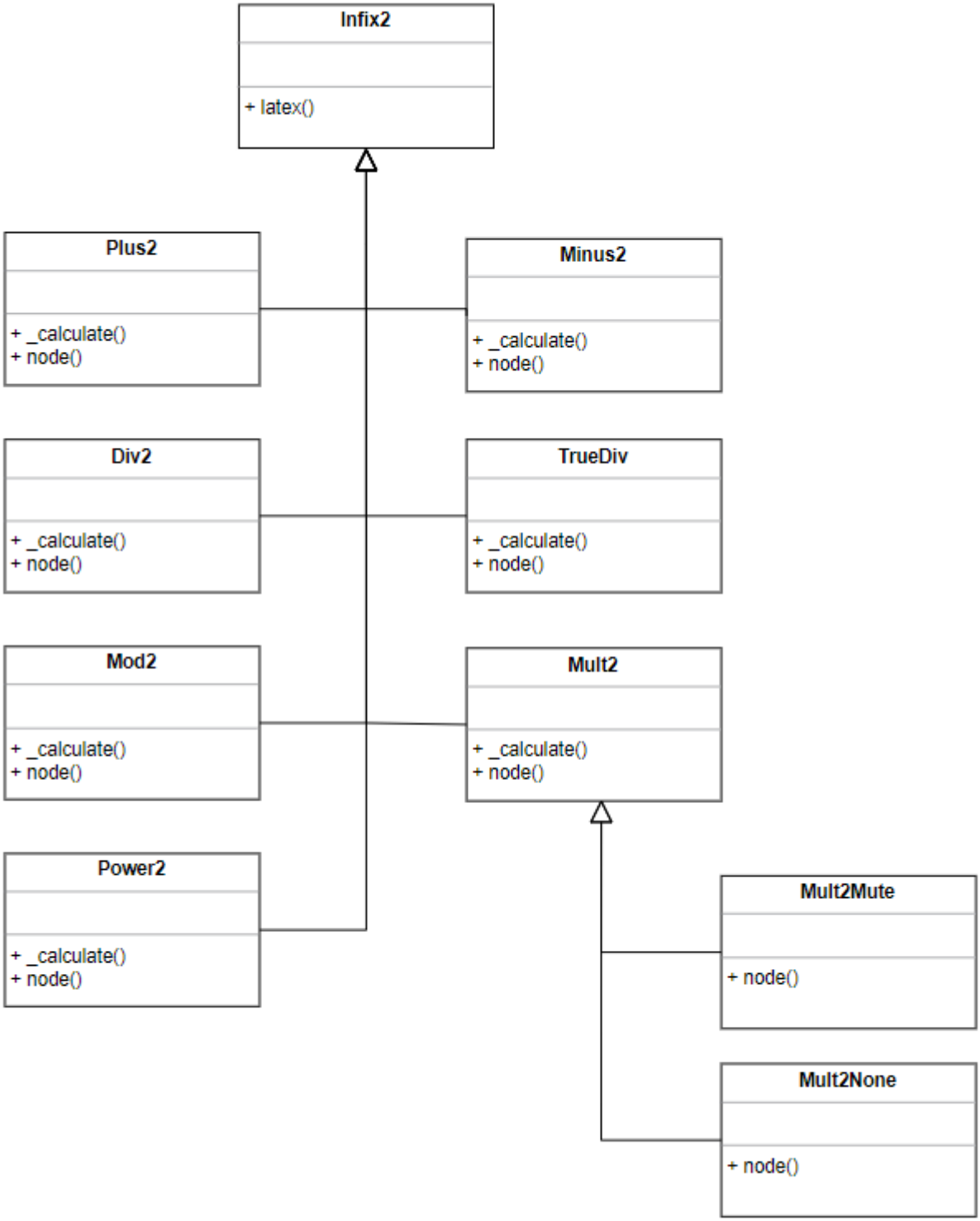

Рисунок 2.7 – Клас Infix2 та похідні від нього

Клас Prefix2 бінарних операторів зображує префіксні оператори. Його нащадками є класи:

— Fraction – зображує дріб,

— Log – зображує логарифм.

Кожен з цих класів задає обчислення своєї операції та мітку вершини. Клас Log додатково перевизначає метод latex, бо у виразі основу логарифму слід записувати як нижній індекс.

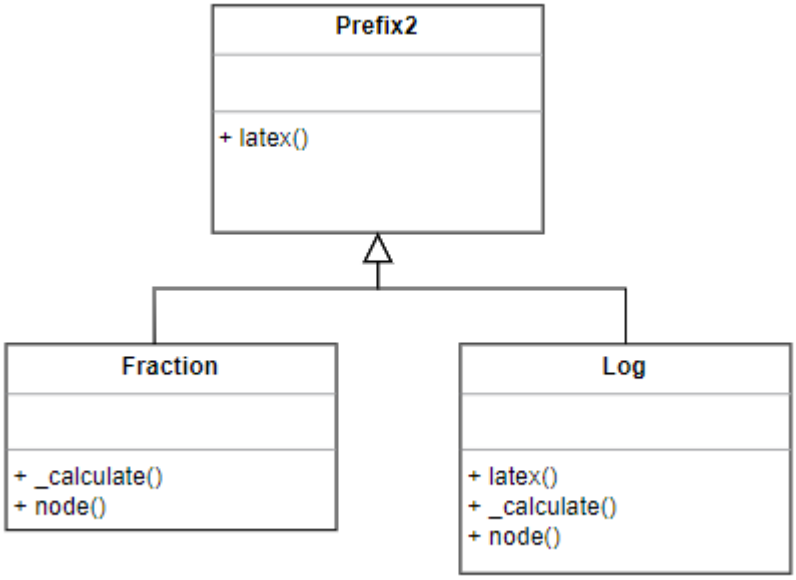

Рисунок 2.8 – Клас Prefix2 та похідні від нього

3 ШАБЛОНИ ДЛЯ ПОРОДЖЕННЯ ВИРАЗІВ

Для отримання різних виразів уявимо таке. Нехай синтаксичне дерево задає не сам вираз, а тільки його структуру. Наприклад, значення операнду у вузлу вибирається з певної множини чисел, або оператор у вузлу може вибиратися з певного переліку операторів, або взагалі вузол задає перелік піддерев, які можуть бути використані для побудови конкретного дерева.

Тоді таке дерево буде шаблоном, за яким можна отримувати вже звичайні синтаксичні дерева виразів. Прив’яжемо утворення звичайного дерева за деревом-шаблоном до операції next. Отже, за операцією next шаблон має утворювати примірник конкретного синтаксичного дерева [9]. При цьому спочатку утворюються конкретні піддерева, а потім до них додається конкретний корінь.

Більш детально розглянемо випадки, що саме може варіюватись при побудові виразу, та реалізацію породження виразу.

Випадок 1. Змінюється число, зображуване листом.

Цьому випадку відповідатиме клас LeafFabric, конструктор якого приймає перелік альтернатив параметрів для утворення листа, а також клас, об’єкт якого має генеруватись.

Перелік альтернатив та клас запам’ятовується в об’єкті. Перелік альтернатив перемішується. Також об’єкт пам’ятає останню використану альтернативу. За запиту на утворення чергового примірника створюється об’єкт заданого класу та ініціалізується черговою альтернативою з переліку. Якщо всі альтернативи в переліку вичерпано, то перелік перемішується і обхід альтернатив починається з початку.

Наприклад, об’єкт LeafFabric(range(2, 31), Number) буде породжувати цілі числа від 2 до 30 включно.

У проєкті використано такі фабрики листів:

```
N = LeafFabric(range(2, 31), Number)
One = LeafFabric(range(1, 2), Number)
M = LeafFabric(range(45, 71), Number)
A = LeafFabric(range(3, 16), Number)
X = LeafFabric('x', Variable)
```

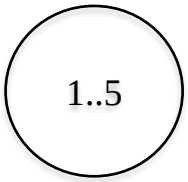

Рисунок 3.1 – Вузол дерева-шаблону, що дозволяє породжувати різні числові значення з діапазону 1..5

Випадок 2. Змінюється операція у корені, яка застосовується до виразів, породжених шаблонами-дітьми.

Цьому випадку відповідатиме клас NodeFabric, конструктор якого приймає перелік альтернатив для класу кореня дерева, а також перелік дерев-шаблонів для отримання підвиразів.

Перелік альтернатив та перелік дерев-шаблонів запам’ятовується в об’єкті. Перелік альтернатив перемішується. Також об’єкт пам’ятає останню використану альтернативу. За запиту на утворення чергового примірника з переліку альтернатив обирається черговий клас (якщо всі альтернативи в переліку вичерпано, то перелік перемішується і обхід альтернатив починається з початку), породжуються піддерева та утворюється вузол обраного класу, що ініціалізується породженими піддеревами.

Наступний шаблон задаватиме вирази вигляду змінна ± число

```
linear = NodeFabric((Minus2, Plus2), (X, A))
```

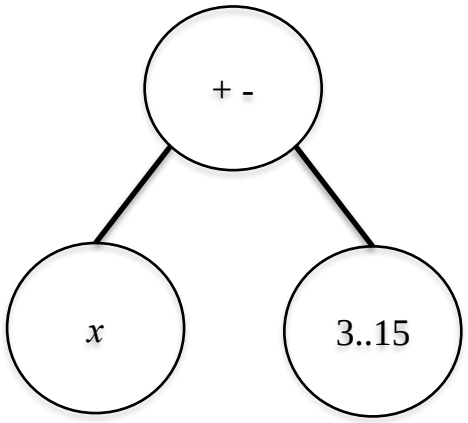

Рисунок 3.2 – Дерево-шаблон, що відповідає шаблону linear

За необхідності використати в шаблоні конкретний оператор чи константу, можна створити шаблон з однією альтернативою, наприклад:

```
pi = NodeFabric((Pi,), tuple())
e = NodeFabric((E,), tuple())
```

Наведені шаблони породжують вузли з константами π та e відповідно.

Шаблон для виразів вигляду (змінна ± число) може бути таким

```
linear_p = NodeFabric((Parenthethis,), (linear,))
```

Випадок 3. Використовуються дерева різної структури.

Цьому випадку відповідатиме клас FabricFabric, конструктор якого приймає перелік альтернатив, що є шаблонами для побудови дерев.

Перелік альтернатив запам’ятовується в об’єкті. Перелік альтернатив перемішується. Також об’єкт пам’ятає останню використану альтернативу. За запиту на утворення чергового примірника з переліку альтернатив обирається чергова (якщо всі альтернативи в переліку вичерпано, то перелік перемішується і обхід альтернатив починається з початку) і для неї відбувається генерація дерева.

Якщо expr1, expr2, expr3 позначають шаблони виразів, то об’єкт

```
FabricFabric(expr1, expr2, expr3, expr4)
```

задає вибір одного з цих шаблонів для подальшого використання у виразі.

4 ГЕНЕРАЦІЯ ВИРАЗІВ ДЛЯ ЛАБОРАТОРНИХ РОБІТ

4.1 Шаблон для генерації виразів

Генерація виразу відбувається за шаблоном main, який будується кодом, наведеним нижче.

Задаються шаблони листових вузлів, що містять числа.

```
N = LeafFabric(range(2, 31), Number)
One = LeafFabric(range(1, 2), Number)
M = LeafFabric(range(45, 71), Number)
A = LeafFabric(range(3, 16), Number)
```

Задається шаблон листових вузлів, що задають змінні. Хоч змінна усього одна, але для коректної роботи шаблону в цілому, усі його вузли мають задавати шаблони.

```
X = LeafFabric('x', Variable)
```

Шаблон для виразів вигляду змінна ± число.

```
linear = NodeFabric((Minus2, Plus2), (X, A))
```

Шаблон для виразів вигляду (змінна ± число).

```
linear_p = NodeFabric((Parenthethis,), (linear,))
```

Для констант π та *e* також маємо побудувати вузли-шаблони.

```
pi = NodeFabric((Pi,), tuple())
e = NodeFabric((E,), tuple())
```

Наступний шаблон задає вирази вигляду $f\left(\frac{N}{M}\right)$, де f одна із функцій sin, cos.

```
part1 = NodeFabric((Sin, Cos),
                   (NodeFabric((Fraction,), (N, M)),))
```

Наступний шаблон задає вирази вигляду $\frac{A}{(x \pm N)(x \pm M)}$, де A, N, M – числа.

```
zz = NodeFabric((Fraction,),
        (A, NodeFabric((Mult2None,), (linear_p, linear_p)),))
```

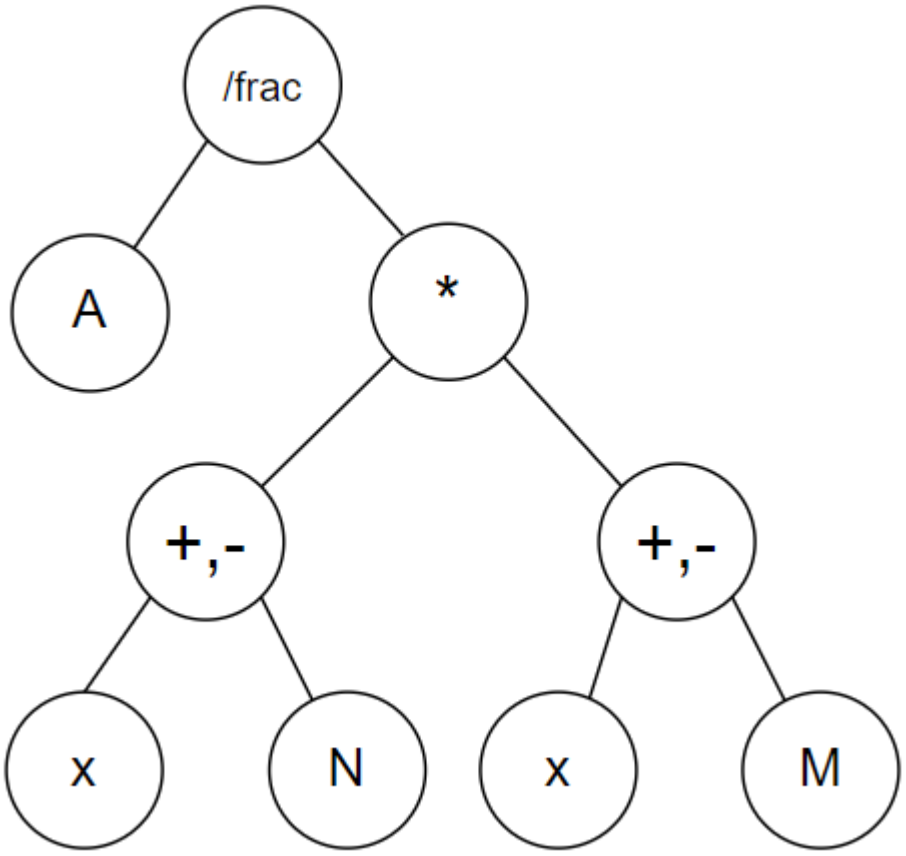

Рисунок 4.1 - Дерево шаблон, що відповідає шаблону zz

Наступний шаблон задає вирази вигляду $\frac{N\pi}{Me}$

```
z1 = NodeFabric((Fraction,),
          (NodeFabric((Mult2None,), (N, pi)),
           NodeFabric((Mult2None,), (M, e))))
```

Наступний шаблон задає вирази вигляду $\frac{Ne}{M\pi}$

```
z2 = NodeFabric((Fraction,),
          (NodeFabric((Mult2None,), (N, e)),
           NodeFabric((Mult2None,), (M, pi))))
```

Прикладами виразів, побудованих за шаблоном

```
z3_1 = NodeFabric((Mult2,), (FabricFabric(z1, z2), zz))
```

є :

$$\frac{19\pi}{45e} \cdot \frac{14}{(x-10)(x+8)}$$

$$\frac{16e}{51\pi} \cdot \frac{5}{(x+4)(x-9)}$$

$$\frac{27\pi}{66e} \cdot \frac{13}{(x+7)(x-3)}$$

Наступний шаблон задає вирази вигляду $f\left(\frac{N}{M}\right) \pm \frac{N\pi}{Me} \cdot \frac{A}{(x \pm N)(x \pm M)}$, де f одна із функцій sin, cos.

```
y1 = NodeFabric((Minus2, Plus2), (part1, z3_1))
```

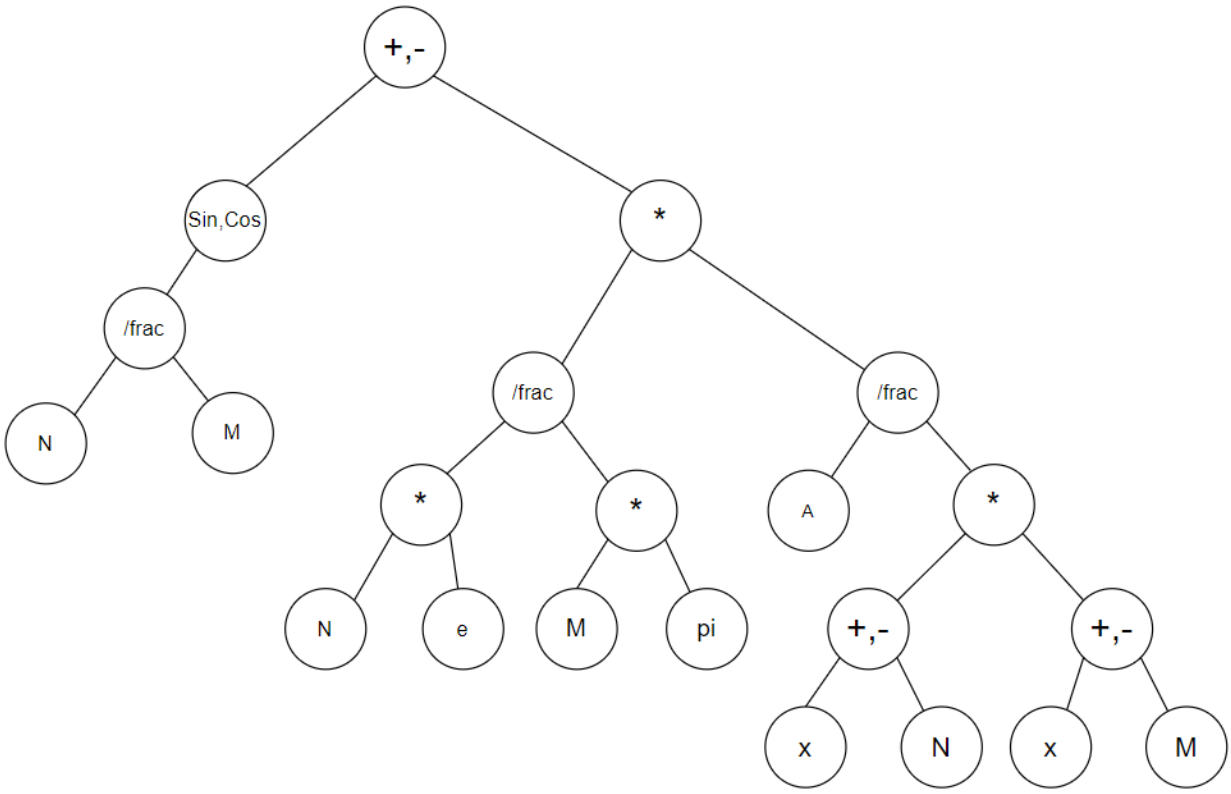

Рисунок 4.2 - Дерево шаблон, що відповідає шаблону y1

Наступний шаблон задає вирази вигляду $f\left(\frac{N}{M}\right) \pm N\pi$.

```
y2 = NodeFabric((Minus2, Plus2),
                (part1, NodeFabric((Mult2None,), (N, pi))))
```

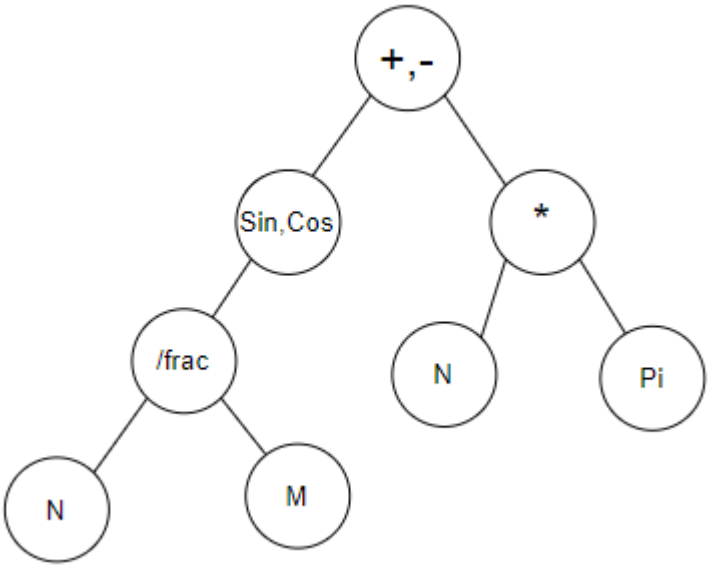

Рисунок 4.3 - Дерево шаблон, що відповідає шаблону y2

Наступний шаблон задає вирази вигляду $f\left(\frac{N}{M}\right) \pm N\pi \pm Me \cdot \frac{A}{(x \pm N)(x \pm M)}$.

```
y3 = NodeFabric((Minus2, Plus2),
     (y2, NodeFabric((Mult2,),
              (NodeFabric((Mult2None,), (M, e)), zz))))
```

Наступний шаблон задає вирази вигляду $f\left(\frac{N}{M}\right) \pm N\pi \pm Me \cdot \frac{A}{(x \pm N)(x \pm M)}$

або $f\left(\frac{N}{M}\right) \pm \frac{N\pi}{Me} \cdot \frac{A}{(x \pm N)(x \pm M)}$

```
y4 = FabricFabric(y1, y3)
```

Шаблон y5 задає вирази вигляду $A\, f(x \pm A)$

```
part3 = NodeFabric((Mult2Mute,),
      (A, NodeFabric((Sin, Cos, Arcsin, Arccos), (linear,))))
y5 = NodeFabric((Minus2, Plus2), (y4, part3))
```

Наступний шаблон задає вирази вигляду $\sqrt{x \pm N}$.

```
sqrt = NodeFabric((Sqrt,), (linear,))
```

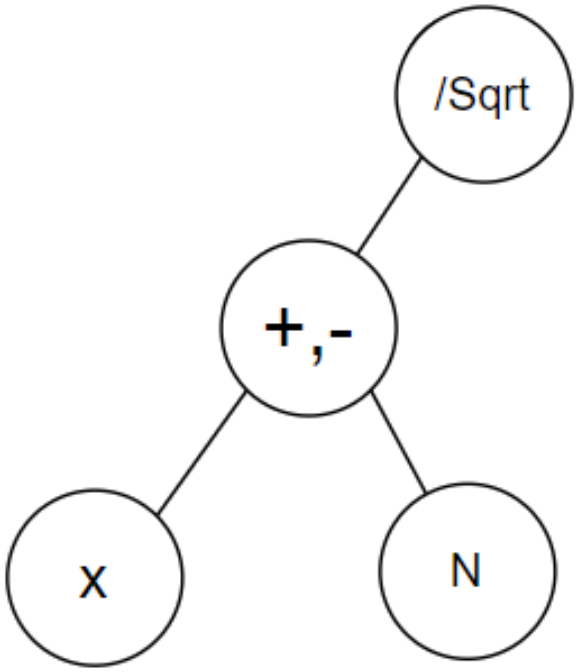

Рисунок 4.4 - Дерево шаблон, що відповідає шаблону sqrt

Наступний шаблон задає вирази вигляду $\frac{1}{\sqrt{x \pm N}}$.

```
z2 = NodeFabric((Fraction,), (One, sqrt,))
```

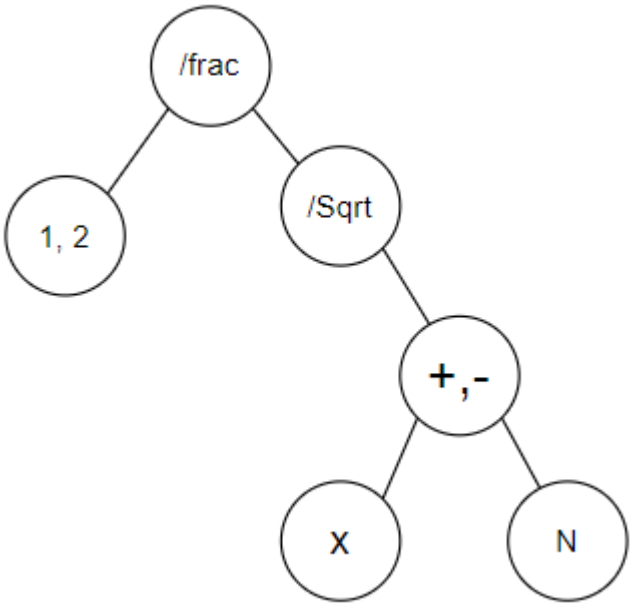

Рисунок 4.5 - Дерево шаблон, що відповідає шаблону z2

Прикладами виразів, побудованих за шаблоном

```
z3 = NodeFabric((Fraction, Log), (A, linear))
```

є

а) $\frac{8}{x-11}$,

б) $\log_6(x+7)$,

в) $\log_{10}(x-3)$.

Наступний шаблон задає вирази вигляду $\frac{A \pm \sqrt{x \pm N}}{(x \pm A)}$.

```
z4 = NodeFabric((Fraction, ),
            (NodeFabric((Plus2, Minus2),(A,sqrt)), linear))
```

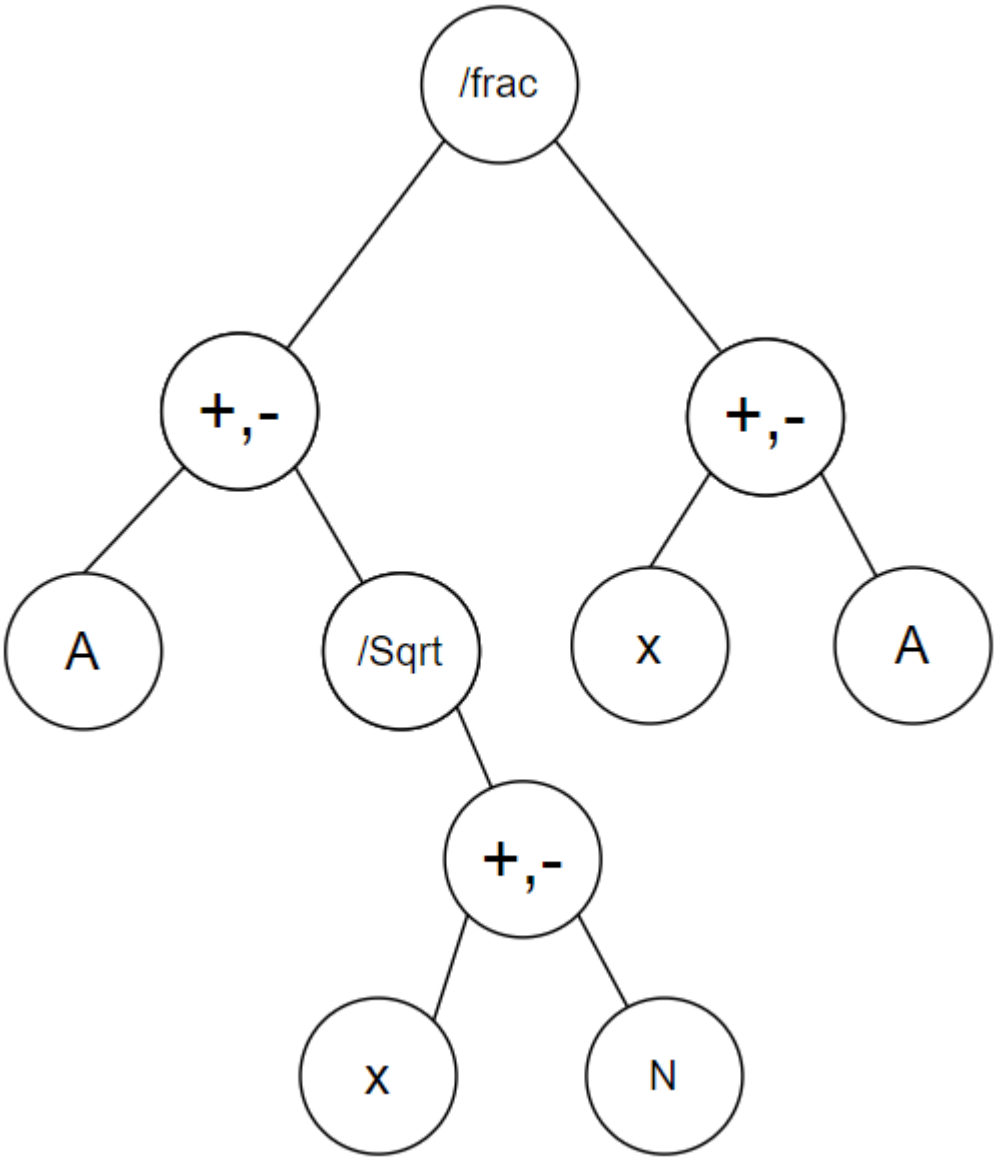

Рисунок 4.6 - Дерево шаблон, що відповідає шаблону z4

```
part4 = FabricFabric(sqrt, z2, z3, z4)
```

Шаблон main, що збирається з раніше зазначених частин,

```
main = NodeFabric((Minus2, Plus2), (y5, part4))
```

є тим шаблоном, за яким у програмі утворюватимуться вирази.

4.2 Застосування шаблону для генерації виразів

Приклади виразів, згенерованих за шаблоном main:

1)

```
{{{{{\cos({\frac{19}{61}})} - {{5} {\pi}}} - {{{66}
{e}}{\cdot}{\frac{4}{({{x} + {7}}) ({{x} - {9}})}}}} + {{15}\
{\cos({{x} + {10}})}}} + {\sqrt{{x} - {5}}}}
```

2)

```
{{{{\cos({\frac{9}{57}})} + {{\frac{{20} {e}}{{62}
{\pi}}}{\cdot}{\frac{11}{({{x} + {12}}) ({{x} - {13}})}}}} +
{{3}\ {\sin({{x} + {14}})}}} + {\frac{1}{\sqrt{{x} - {6}}}}}
```

3)

```
{{{{\sin({\frac{5}{60}})} + {{\frac{{30} {\pi}}{{64}
{e}}}{\cdot}{\frac{10}{({{x} - {8}}) ({{x} + {7}})}}}} - {{5}\
{\cos({{x} + {14}})}}} - {\frac{1}{\sqrt{{x} - {15}}}}}
```

Що маємо в результаті:

1)

$$\cos(\tfrac{19}{61}) - 5\pi - 66e \cdot \tfrac{4}{(x+7)(x-9)} + 15\,\cos(x+10) + \sqrt{x-5}$$

2)

$$\cos(\tfrac{9}{57}) + \tfrac{20e}{62\pi} \cdot \tfrac{11}{(x+12)(x-13)} + 3\,\sin(x+14) + \tfrac{1}{\sqrt{x-6}}$$

3)

$$\sin(\tfrac{5}{60}) + \tfrac{30\pi}{64e} \cdot \tfrac{10}{(x-8)(x+7)} - 5\,\cos(x+14) - \tfrac{1}{\sqrt{x-15}}$$

5 ПРОГРАМА ГЕНЕРАЦІЇ ВИРАЗІВ

5.1 Загальна структура програми

Програма генерації виразів використовує раніше розроблені класи LeafFabric, NodeFabric та FabricFabric для утворення шаблону-дерева. Зокрема, використовується шаблон main, побудований у підрозділі 4.1. Результуючий вираз будується з використанням класів, описаних у розділі 3.

Для роботи з програмою достатньо інтерпретатора Python3 [10].

Рядок запуску програми. Програма генерації виразів через командний рядок отримує ім’я файлу, в який слід записати згенеровані вирази у форматі латех, та кількість виразів, які потрібні. Використовуючи розроблений шаблон main програма генерує запитувану кількість виразів та записує їх у файл. До файлу додаються початок та кінцівка, щоб його можна було відразу відкомпілювати LaTeX.

За невідповідної кількості аргументів командного рядка користувачу виводиться довідка з використання програми.

```
"D:\Універ (D)\pythonProject8\venv\Scripts\python.exe" "D:\Універ (D)\pythonProject8\generate.py"
Використання програми:
перший аргумент: ім'я файлу для запису результату
другий аргумент: кількість виразів

наприклад,
generate.py tasks.tex 15


Process finished with exit code 0
```

Рисунок 5.1– вивід програми за некоректного командного рядка

5.2 Результат роботи програми

```
\documentclass{article}
\setlength\textwidth{190mm} \setlength\textheight{237mm}
\setlength\voffset{-18mm} \setlength\hoffset{-23mm}
\usepackage{amsmath}
\begin{document}


1) ${{{{{\cos({\frac{28}{49}})} - {{21} {\pi}}} + {{{47} {e}}{\cdot}{\frac{15}{({{{x} + {12}}) ({{x} -
{7}})}}}}} + {{14}\ {\arccos({{x} + {8}})}}} - {\sqrt{{x} - {5}}}}$

2) ${{{{\sin({\frac{12}{69}})} - {{\frac{{5} {e}}{{53} {\pi}}}{\cdot}{\frac{10}{({{{x} + {9}}) ({{x} -
{6}})}}}}} - {{3}\ {\sin({{x} - {4}})}}} + {\frac{13}{{x} + {11}}}}$

3) ${{{{\sin({\frac{24}{51}})} + {{\frac{{26} {\pi}}{{68} {e}}}{\cdot}{\frac{14}{({{{x} - {11}}) ({{x} +
{12}})}}}}} - {{15}\ {\cos({{x} + {5}})}}} - {\frac{1}{\sqrt{{x} - {8}}}}}$

4) ${{{{{\cos({\frac{4}{65}})} + {{13} {\pi}}} - {{{58} {e}}{\cdot}{\frac{7}{({{{x} + {4}}) ({{x} -
{13}})}}}}} + {{3}\ {\arcsin({{x} - {9}})}}} + {\frac{{6} - {\sqrt{{x} + {10}}}}{{x} - {14}}}}$

5) ${{{{{\cos({\frac{14}{45}})} + {{9} {\pi}}} + {{{52} {e}}{\cdot}{\frac{13}{({{{x} + {15}}) ({{x} -
{11}})}}}}} + {{9}\ {\arcsin({{x} + {5}})}}} + {\frac{1}{\sqrt{{x} + {7}}}}}$

6) ${{{{\sin({\frac{3}{60}})} + {{\frac{{2} {\pi}}{{55} {e}}}{\cdot}{\frac{4}{({{{x} - {6}}) ({{x} +
{8}})}}}}} - {{10}\ {\cos({{x} - {12}})}}} - {\log_{3} ({{x} + {4}})}}$

7) ${{{{\cos({\frac{29}{64}})} - {{\frac{{17} {e}}{{63} {\pi}}}{\cdot}{\frac{15}{({{{x} - {7}}) ({{x} +
{8}})}}}}} + {{11}\ {\arccos({{x} - {3}})}}} + {\sqrt{{x} + {12}}}}$

8) ${{{{{\sin({\frac{6}{46}})} - {{22} {\pi}}} - {{{59} {e}}{\cdot}{\frac{14}{({{{x} - {6}}) ({{x} +
{9}})}}}}} - {{10}\ {\sin({{x} - {13}})}}} - {\frac{{5} + {\sqrt{{x} + {10}}}}{{x} - {3}}}}$

9) ${{{{{\sin({\frac{10}{62}})} - {{18} {\pi}}} + {{{50} {e}}{\cdot}{\frac{13}{({{{x} - {6}}) ({{x} +
{12}})}}}}} + {{5}\ {\sin({{x} + {7}})}}} + {\frac{1}{\sqrt{{x} - {15}}}}}$
```

Рисунок 5.2 – Фрагмент побудованого програмою латех-файлу

1) $\cos(\frac{28}{49}) - 21\pi + 47e \cdot \frac{15}{(x+12)(x-7)} + 14\arccos(x+8) - \sqrt{x-5}$
2) $\sin(\frac{12}{69}) - \frac{5e}{53\pi} \cdot \frac{10}{(x+9)(x-6)} - 3\sin(x-4) + \frac{13}{x+11}$
3) $\sin(\frac{24}{51}) + \frac{26\pi}{68e} \cdot \frac{14}{(x-11)(x+12)} - 15\cos(x+5) - \frac{1}{\sqrt{x-8}}$
4) $\cos(\frac{4}{65}) + 13\pi - 58e \cdot \frac{7}{(x+4)(x-13)} + 3\arcsin(x-9) + \frac{6-\sqrt{x+10}}{x-14}$
5) $\cos(\frac{14}{45}) + 9\pi + 52e \cdot \frac{13}{(x+15)(x-11)} + 9\arcsin(x+5) + \frac{1}{\sqrt{x+7}}$
6) $\sin(\frac{3}{60}) + \frac{2\pi}{55e} \cdot \frac{4}{(x-6)(x+8)} - 10\cos(x-12) - \log_3(x+4)$
7) $\cos(\frac{29}{64}) - \frac{17e}{63\pi} \cdot \frac{15}{(x-7)(x+8)} + 11\arccos(x-3) + \sqrt{x+12}$
8) $\sin(\frac{6}{46}) - 22\pi - 59e \cdot \frac{14}{(x-6)(x+9)} - 10\sin(x-13) - \frac{5+\sqrt{x+10}}{x-3}$
9) $\sin(\frac{10}{62}) - 18\pi + 50e \cdot \frac{13}{(x-6)(x+12)} + 5\sin(x+7) + \frac{1}{\sqrt{x-15}}$
10) $\cos(\frac{23}{70}) - \frac{25e}{56\pi} \cdot \frac{9}{(x+4)(x-14)} - 11\arcsin(x-8) - \frac{3}{x+11}$
11) $\sin(\frac{7}{61}) + 16\pi - 57e \cdot \frac{10}{(x+14)(x-9)} - 12\cos(x-13) - \sqrt{x+6}$
12) $\cos(\frac{19}{48}) + \frac{20\pi}{54e} \cdot \frac{4}{(x+5)(x-8)} + 7\arccos(x-15) + \frac{15+\sqrt{x+10}}{x+5}$
13) $\sin(\frac{27}{67}) - 15\pi - 66e \cdot \frac{11}{(x-9)(x+7)} + 3\arcsin(x-12) + \frac{1}{\sqrt{x-6}}$
14) $\cos(\frac{8}{61}) - \frac{30e}{58\pi} \cdot \frac{4}{(x+14)(x+8)} - 13\sin(x-4) - \log_8(x+7)$
15) $\sin(\frac{11}{49}) + 21\pi + 55e \cdot \frac{12}{(x-6)(x-13)} + 3\cos(x+15) + \sqrt{x-11}$
16) $\cos(\frac{23}{65}) + \frac{4\pi}{64e} \cdot \frac{9}{(x+5)(x-14)} - 10\arccos(x+15) - \frac{9-\sqrt{x+5}}{x-7}$
17) $\cos(\frac{25}{50}) - 9\pi - 46e \cdot \frac{11}{(x-4)(x+14)} - 6\arccos(x+13) + \sqrt{x-3}$
18) $\sin(\frac{18}{51}) + \frac{29e}{69\pi} \cdot \frac{12}{(x-10)(x+8)} + 12\sin(x+7) - \frac{9+\sqrt{x-4}}{x-10}$

Рисунок 5.3 – Результат компіляції отриманого латех-файлу

5.3 Інструкція користувача

Для запуску програми необхідно:

1. Потрібно в терміналі написати generate.py, ім’я файлу та кількість виразів

Наприклад:

generate.py tasks.tex 15

2. У директорії створиться файл .tex , у якому буде записано результат генерації у форматі LaTeX.

3. Далі компілювати файл LaTeX і на виході матимемо .pdf файл з генерованими варіантами завдань

ВИСНОВКИ

У роботі розроблено ієрархію класів вузлів дерева, що враховує не тільки особливості побудови виразу, але й має елементи керування виведенням.

До розробленої ієрархії класів спроєктовано класи-шаблони вузлів, що здатні породжувати вирази.

Розроблено шаблон виразу, що містить математичні функції, константи π та e, ділення у вигляді дробу і породжує не всюди визначені функції. Такі вирази можуть бути використані в якості учбових завдань для курсу Програмування для засвоєння пріоритетів операцій та використання функцій математичної бібліотеки. Розроблена система класів фактично задає фреймворк. Вона може бути поповнена класами, що зображують й інші математичні функції. Також можна впровадити й інші шаблони.

Для подальших досліджень перспективним напрямком може бути автоматична генерація шаблонів заданої складності (наприклад, з певною кількістю операцій чи з обов’язковою присутністю певних елементів).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Data Structures and Algorithm – Tree. [Електронний ресурс] – Режим доступу до ресурсу : https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm
2. Design Patterns: Elements of Reusable Object-Oriented Software/[Gamma E., Helm R., Johnson R., Vlissides.J]. – AddisonWesley Professional, 1994. –416 p.
3. Introduction to Latex. [Електронний ресурс] – Режим доступу до ресурсу : https://www.latex-project.org/about/
4. Lamport L. LaTex2e. – 1994. [Електронний ресурс] – Режим доступу до ресурсу : https://www.ntg.nl/literatuur/lamport/latex2e.pdf
5. Wirth N. Algorithms and data structures. – Prentice Hall, 1985. –288 p.
6. Проценко В. С. Техніка програмування мовою Сі / Проценко В. С., Чаленко П. Й., Ставровський А. Б. – К.: Либідь, 1993. – 224 с.
7. Вступ до алгоритмів/ [Кормен Т., Лейзерсон Ч., Рівест Р., Стайн К.]. – К.І.С., 2019. – 1288 с.
8. Aho A. Data Structures and Algorithms / Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. – Pearson, 1983. – 448 p.
9. Built-in Functions — Python 3.11.3 documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/functions.html#next
10. Welcome to Python.org. [Електронний ресурс] – Режим доступу до ресурсу : https:/www.python.org/