Національний технічний університет України

"Київський Політехнічний Інститут"

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Курсова робота

з дисципліни "Системне програмування" на тему

«Парсер математичних виразів MathParser»

Виконав:

студент ІІІ-го курсу ФІОТ,

групи ІО-22, Краевий О.В

Керівник: Стіренко С.Г.

Київ, 2014 р.

# Зміст

# 1. Технічне завдання …...............................................................................................3

# 1.1 Загальні відомості …...................................................................................3

# 1.2 Призначення розробки …...........................................................................4

# 1.3 Вимоги до розроблюваного продукту……………………………...…...4

1.3.1 Вимоги до вхідних даних………………………………….……4

1.3.2 Вимоги до виводу даних……………………………….….…….5

1.3.3 Вимоги до часових характеристик……………………………..5

1.3.4 Вимоги до надійності програми…………………………….…..5

1.3.5 Вимоги до часу відновлення після відказу………………….…5

1.3.6 Відмови через некоректні дїї користувача……………….….…5

1.3.7 Вимогу до видів обслуговування………………………….…....5

1.3.8 Вимоги до чисельності та кваліфікації персоналу……….…....5

1.3.9 Вимоги до апаратного забезпечення …………….………….....5

1.4 Етапи розробки та тестування…………………………………….…..…6

# 2. Вступ …....................................................................................................................7

# 3. Обгрунтування роботи програми …......................................................................9

# 3.1 Обгрунтування граматики……………….………………………..9

# 4. Алгоритм роботи програми ……….....................................................................13

# 4.1 Алгоритм роботи парсера….………………………………………….13

# 5. Опис програмних модулів… …...........................................................................14

# 6. Тестування………………. …................................................................................15

# 7. Висновки …...........................................................................................................16

# Список використаної літератури …........................................................................17

# Додаток А. Лістинг програми ……………………………………………..……..18

# Додаток Б. Скріншот JUnit тестів………………………………………….……..19

# Додаток В. Скріншот виконання програми............................................................20

# 1. Технічне завдання

## 1.1 Загальні відомості

## Повне ім'я розроблювального програмного продукту - “MathParser”

## Умовне позначення - “MP”

## Розробник — студенти групи ІО-22 НТУУ “КПІ” ФІОТ, кафедри ОТ, Краєвий О.В., Мулявко Д.С.

## Замовник — викладач НТУУ “КПІ” ФІОТ, кафедри ОТ, Стіренко С.Г.

## Матеріали, на основі яких була проведена розробка -

## - Матеріл лекцій «Системне програмування», 5-й семестр, ФІОТ, кафедра ОТ.

## - Матеріал лабораторних робіт, розроблених Пустоваровим В.І.

## Планові строки розробки — з 01.11.2014 по 30.12.2014

## Фінансування розробки програмного продукту — стипендіальні виплати студентам Краєвому О.В. та Мулявко Д.С.

## Порядок оформлення — алгоритм та опис роботи програми

## 

## 1.2 Призначення розробки

## Функціональне призначення

Програмний продукт “МР” призначений для парсингу та обчислення математичних виразів.

**Експлуатаційне призначення**

“MP” повинен вільно експлуатуватися замовником

**Ціль розробки**

Ціллю розробки є створення програмного забезпечення, яке буде відрізнятися від раніше запропонованих на ринку аналогів багатьма рисами :

1) Швидкодія. Розроблюваний програмний продукт повинен мати менший час обробки вхідних даних, та має забезпечувати більшу ефективність, ніж вже існуючі аналоги.

2) Кросплатформеність. Програмний продукт має виконувати усі свої функції на усіх найбільш поширених операційних системах (Windows, Linux, Mac OS)

**1.3. Вимоги до розроблювального продукту**

Вимоги до виконуваних функцій

1. функція відкриття існуючого файлу
2. функція копіювання даних із певного файлу
3. функція виводу результатів в зручному для користувача форматі

**1.3.1 Вимоги до вхідних даних**

1) можливість відкриття файлів тільки формату \*.txt, які містять математичні вирази в певному форматі.

Для додання змінних та операцій використовуються певні символи :

- **“/”** - операція ділення

- “\*” - операція множення

- **“+, -”** - операція додавання та віднімання

- “**a-z”** – змінні

- **“0-9”** - числові позначення

- **«sin, cos, tg …»** - стандартні математичні функції (інші математичні функції задаються так само, як і в математиці)

**1.3.2 Вимоги до виводу даних:**

1. можливість виводу результатів обчислень до консолі

**1.3.3 Вимоги до часових характеристик**

Вимог до часових характеристик не пред’являється.

**1.3.4 Вимоги до надійності програми**

Надійне функціонування програми повинно бути забезпечено виконанням замовником сукупності організаційно-технічних заходів:

1. організація надійного та безперебійного живлення технічних засобів
2. використання ліцензійного програмного забезпечення
3. регулярне випробування своїх технічних засобів на наявність вірусів.

**1.3.5 Вимоги до часу відновлення після відказу**

Вимоги до часу відновлення не пред’являються.

**1.3.6 Відмови через некоректні дїї користувача**

Відмова програми можлива через некоректні дії користувача при взаємодії з операційною системою. Для захисту від таких випадків необхідно забезпечити роботу користувача без надання адміністраторських повноважень.

**1.3.7 Вимогу до видів обслуговування**

Програма не вимагає будь-яких видів обслуговування

**1.3.8 Вимоги до чисельності та кваліфікації персоналу**

Мінімальна кількість персоналу, необхідного для роботи програми, має становити не менше 2 штатних одиниць - системний адміністратор і користувач програми.

## 1.3.9 Вимоги до апаратного забезпечення

Процесор — будь-який на базі AMD, Intel з тактовою частотою від 1 Ггц

Оперативна пам'ять — від 512 МБ

Місце на жорсткому диску — 1Гбайт

Відеопроцесор — інтегрований, або дискретний будь-якої вбудованої місткості

Операційна система — будь-яка сімейства Windows, Linux, Mac OS

Периферійні пристрої — клавіатура та миша

**1.4 Етапи розробки та тестування**

1) Розробка токенайзера. Відповідальний — Краєвий О.В. Строки розробки — 01.11.14 — 08.11.14

2) Розробка проміжної мови. Відповідальний — Краєвий О.В. Строки розробки — 09.11.14 — 16.11.14

3) Розробка будувальника дерева розбору. Відповідальні — Краєвий О.В., Мулявко Д.С. Строки розробки — 17.11.14 — 25.11.14

4) Розробка відвідувача гілок дерева розбору. Відповідальний — Мулявко Д.С. Строки розробки — 25.11.14 — 05.12.14

5) Розробка документації. Відповідальний – Мулявко Д.С. Строки розробки - 06.12.14 – 10.12.14

6) Тестування програмного продукту. Відповідальні — Краєвий О.В., Мулявко Д.С. Строки тестування — 11.12.14 — 17.12.14

# 2. Вступ

Іноді необхідно прочитати дані, введені користувачем, наприклад, формули та вирази, якусь просту розмітку, або навіть форматовані строки. Ці вхідні дані повинні бути розібрані та перетворені у форму, яку програма може обробляти. Напевно, потрібно побудувати ієрархію об'єктів, або потрібно вивести результат, як стрічку, яка відображає те, що ввів користувач. Крім того, парсер повинен повідомити користувачу, чи правильно він ввів вхідні дані, та попередити його про помилку.

Процес парсингу вхідних даних користувача містить наступні кроки.

1) Вхідний повинен бути розділений на лексеми. Лексемою може бути один символ, або набір символів. У математичному вираженні оексеми можуть включати в себе такі оператори, як знак плюс '+' або мінус '-', але й імена змінних, як імена 'X' або функції як «sin». Для парсеру XML лексемою можуть бути '<' або імена тегів. Завдання розпізнавання лексем бере на себе токенайзер.

2) Повинна бути побудована граматика . Граматика описує набір правил, які описують, на які вхідні дані користувача розраховує програма. Граматика працює на рівні лексем. Просте правило для розбору суми двох змінних може бути таким

**sum -> var + var**

На цьому рівні, граматиці не робить різниці між різними змінними і крім двох "var" лексем правило працює на різних змінних. Створення граматики — це більш абстрактний процес, і, як правило, робиться не на комп'ютері

3) Граматика перенаправляється у аналізатор. Існують різні алгоритми для генерації коду парсеру. Залежно від складності граматики ця задача часто здійснюється за допомогою генератора парсерів. Генераторів парсерів генерує код, якому ви можете дати, як вхідні дані, лексеми. Згенерований код буде оцінювати лексеми і вирішувати питання за правилами граматики. У цій роботі використовується, відносно, проста граматиа. Для простих граматик парсер може бути закодований з використанням рекурсивного алгоритму спуску.

4) Застосовуються конкретні правила граматики та робляться відповідні дії. Ці дії необхідні, щоб перетворити вхідний рядок у структуру даних, яку можна використовувати в рамках програми. Дії можуть передбачати генерування деревоподібної структури об'єктів або запису шматку відформатированного рядку у потік.

# 3. Обґрунтування роботи програми

# 3.1 Обгрунтування граматики

# Перед проектуванням граматики, ми повинні визначити всі термінальні символи, які ми очікуємо.

Символ Значення

PLUSMINUS + або -

MULTDIV \* або /

RAISED ^

FUNCTION будь-яка мат. функція:

sin, cos, exp, sqrt, ...

OPEN\_BRACKET (

CLOSE\_BRACKET )

NUMBER будь-яке число

VARIABLE ім'я змінної

## Всі ці символи, що може вводити користувач. Ми припускаємо, що токенізатор вже розділити внесок в цих термінальних символів і аналізатор тепер отримує потік цих символів.

## Ми хочемо, щоб наш парсер працював лише з дозволеними виразами. Ми також хочемо, щоб генерувати внутрішнє подання цього виразу. Це означає, що ми повинні переконатися, що математичні операції здійснюються в правильному порядку. У виразі типу 3 + 2 \* 5 множення повинно бути проведені спочатку, тому що множення має вищий пріоритет, ніж складання.

## На верхньому рівні вираз можна інтерпретувати як суму одного або декількох умов. Перший доданок можна почати з плюсом чи мінусом.

## expression -> signed\_term sum\_op

## sum\_op -> PLUSMINUS term sum\_op

## sum\_op -> EPSILON

## Коли ці правила розроблені необхідно переконатися, що ми завжди знаємо, як застосувати правило, просто читаючи наступний символ з вхідного сигналу. signed\_term може бути терміналом , який передує PLUSMINUS або просто терміналом

## signed\_term -> PLUSMINUS термін

## signed\_term -> термін

## Тепер ми як і раніше таким же чином з результатом. Кожен член є результатом одного або декількох факторів.

## term -> factor term\_op

## term\_op -> MULTDIV signed\_factor term\_op

## term\_op -> EPSILON

## Signed\_factor є чинником, який може почати з PLUSMINUS або просто фактором.

## signed\_factor -> factor PLUSMINUS

## signed\_factor -> factor

## Коефіцієнт може бути як функцією, так і аргументом, який може бути збільшений до певної степені.

## factor -> argument factor\_op

## factor\_op -> RAISED signed\_factor

## factor\_op -> EPSILON

## Друге правило тут дозволяє простий аргумент, тому що factor\_op може бути усуненим за допомогою правила EPSILON. Вона також дозволяє експоненційність а допомогою третього правила.

## Аргумент - це значення або повне вираження в квадратних дужках.

## argument -> value

## argument -> FUNCTION argument

## argument -> OPEN\_BRACKET expression CLOSE\_BRACKET

## Третє правило дозволяє нам будувати вирази довільної складності, тому що ми можемо робити вирази у виразах, як довго, як вони укладені в квадратні дужки. Друге правило дозволяє функціональні вирази, які будуть використовуватися в якості аргументів. Такі вирази, як sin х можна використовувати без будь-яких дужок х, тому що аргумент функції може бути просто значенням.

## Але що відбувається, коли вираз читає щось на зразок sin 2 \* х. Це призводить до того, що sin 2 \* х розглядається як sin (2 \* х) або (sin 2) \* х. У моєму випадку, за допомогою простого LL (1) граматики та ігноруючи пусті місця, ми не будемо в змозі реалізувати таку поведінку. Це означає, що граматика завжди буде інтерпретувати вираз як sin (2) \* х.

## Нарешті, значення може бути числом або змінною

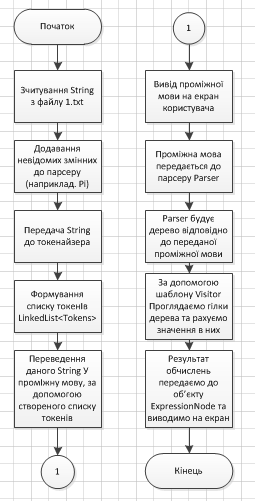
## value -> NUMBER

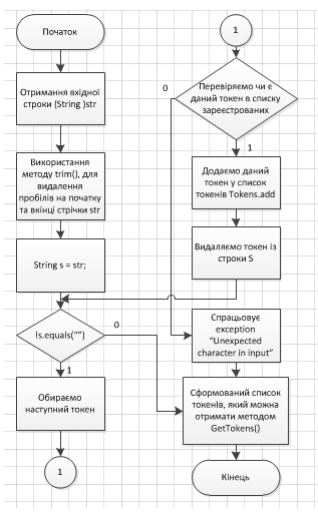
## value -> VARIABLE

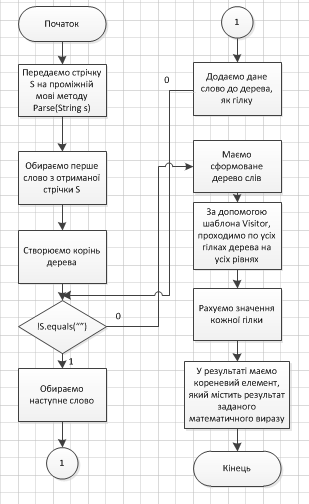
## Це повна граматика для математичних виразів.

## 

# 4. Алгоритм роботи програми 4.1 Алгоритм роботи токенайзера







# 5. Опис програмних модулів

До складу проекту консольної прикладної програми, орієнтованої на можливість використання в числі інших бібліотек стандартних функцій входять модулі з наступними вхідними файлами:

1. AdditionExpressionNode.java – ланка дерева розбору, що відповідає сумі
2. ConstantExpressionNode.java – ланка дерева розбору, що відповідає константі
3. EvaluationException.java – виключна ситуація, що трапляється при встановленні рівності між виразами
4. ExponentiationExpressionNode.java — ланка дерева розбору, що відповідає за експоненту
5. ExpressionNode.java – ланка дерева розбору, що відповідає за вираз
6. ExpressionNodeVisitor.java – інтерфейс, що реалізує шаблон Visitor. У ньому прописані методи, за допомогою яких можна перейти на ланку дерева розбору
7. FunctionExpressionNode.java – ланка дерева розбору, що відповідає за функцію
8. MultiplicationExpressionNode.java – ланка дерева розбору, що відповідає за множення
9. Parser.java – парсер математичних виразів
10. ParserException.java – виключна ситуація при парсингу
11. SequenceExpressionNode.java – ланка дерева розбору, що відповідає за послідовність виразів
12. SetVariable.java – візітор, що встановлює змінну та дає їй ім'я
13. Token.java - лексема
14. Tokenizer.java – токенайзер, що розбиває вираз на лексеми
15. VariableExpressionNode.java – ланка дерева розбору, що відповідає за змінну

**6. Тестування**Для тестування “MP” було обрано JUnit тесты.  
Створено 3 тести:

1. Перевірка коректності ввода даних
2. Перевірка коректної роботи токенайзера
3. Перевірка фінального результату.

“MP” було успішно пройдено усі тести.

# 7. Висновки

Відповідно до ТЗ було розроблено та реалізовано програмний продукт «MathParser». Вимоги, які були задані до програмного продукту у технічному завданні, були витримані. Розроблений продукт має більшу від існуючих аналогів швидкодію(на жаль лише для виразів, які складаються с менше ніж 100 лексем). Завдяки обраній мові реалізації – продукт є повністю кросплатформеною та самостійною програмою.

Функціональне та експлуатаційне призначення, задане у технічному завданні, було реалізовано на 100%.

Програма повністю виконує свої функції. Для перевірки правильності та точності обчислення – було проведено тестування за допомогою JUnit. JUnit тесты були пройдені успішно, що підтверджує стабільність та коректність роботи програмного забезпечення “MP”

**Список використаної літератури**

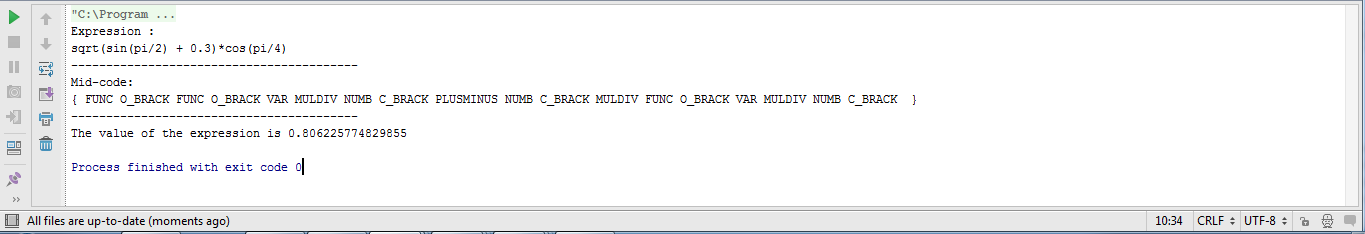
1. Матеріл лекцій «Системне програмування», 5-й семестр, ФІОТ, кафедра ОТ.

2. Матеріал лабораторних робіт, розроблених Пустоваровим В.І.

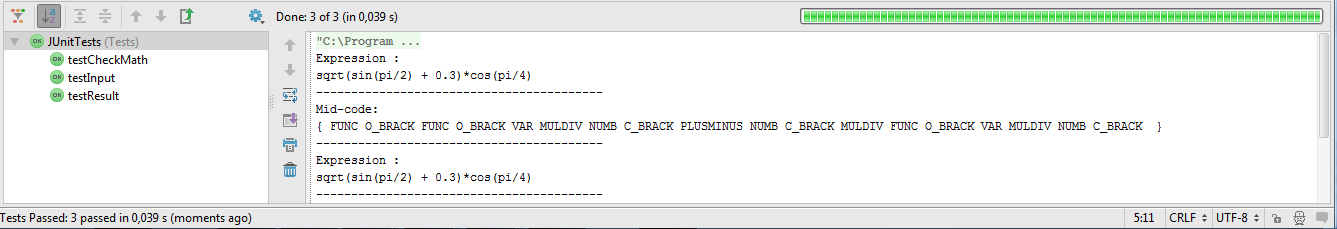
3. А. Ахо, Р. Сети, Дж. Ульман "Компиляторы: принципы, технологии и инструменты", М.: "Вильямс", 2001. 768 с.

# Додаток А. Лістинг програми

**Додаток Б. Скріншот виконання програми**



**Додаток В. Скріншот JUnit тестів**



# 