|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт Информационных Технологий | | |
| Кафедра Вычислительной техники | | |

| **ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №1**  «Преобразование алгебраического выражения в обратную польскую запись на Python» | |
| --- | --- |
|  | |
| **по дисциплине** | |
| **«**Теория формальных языков**»** | |
|  | |
| Выполнил студент группы ИКБО-43-23 | *Жаворонкова А.А.* |
| Принял ассистент | *Цынгалёв П.С.* |

| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2024 г. |  |
| --- | --- | --- |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2024 г. |  |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_heading=h.30j0zll)

[1 ЦЕЛЬ 4](#_heading=h.1fob9te)

[2 ЗАДАНИЕ 5](#_heading=h.3znysh7)

[2.1 Формулировка задания 5](#_heading=h.2et92p0)

[2.2 Математическая модель 5](#_heading=h.tyjcwt)

[2.3 Тестирование программы 6](#_heading=h.1t3h5sf)

[ЗАКЛЮЧЕНИЕ 7](#_heading=h.2s8eyo1)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 8](#_heading=h.3rdcrjn)

[ПРИЛОЖЕНИЯ 9](#_heading=h.1rplyrhsk38o)

# ВВЕДЕНИЕ

Изучение теории формальных языков занимает важное место в образовательной программе по программной инженерии. Эта область, являющаяся неотъемлемой частью компьютерных наук и теоретической информатики, посвящена анализу структуры, свойств формальных языков и моделированию их обработки. В условиях современного мира, где постоянно растут объемы данных и необходимость в эффективных методах их обработки, хранения и передачи, понимание основ формальных языков становится особенно актуальным.

Формальные языки находят применение в различных областях: от описания синтаксиса языков программирования до разработки сетевых протоколов и создания компиляторов. Они позволяют формализовать взаимодействие как между человеком и компьютером, так и между различными компонентами программного обеспечения. Основу теории формальных языков составляют такие понятия, как грамматики, автоматы и алгебраические структуры, которые используются для описания синтаксиса и семантики языков.

В данном отчете будет рассмотрен процесс преобразования выражений в обратную польскую запись.

# 1 ЦЕЛЬ

Преобразование алгебраического выражения в обратную польскую запись на Python.

# 2 ЗАДАНИЕ

## 2.1 Формулировка задания

На выбранном ЯП реализовать преобразование простого алгебраического выражения в обратную польскую запись с использованием стека и простейший калькулятор выражений в обратной польской записи с целыми числами.

Выражение для обработки: 10 + 3 \* (2 - 1/2) = 14,5

## 2.2 Математическая модель

Разработаем функцию, которая принимает арифметическое выражение в виде строки и преобразует его в постфиксную запись (обратную польскую нотацию). Для этого определим строковую переменную expression, содержащую исходное выражение. Зададим словарь для приоритетов операторов (+, -, \*, /), где большее число соответствует более высокому приоритету. Создадим список output для итогового выражения в постфиксной нотации, список tokens для хранения элементов (чисел, операторов и скобок), полученных путем разбиения строки expression с помощью регулярных выражений, и стек operators для временного хранения операторов и скобок.

При обработке каждого элемента из tokens алгоритм выполняет следующие шаги:

- Если текущий элемент (token) является числом (включая отрицательные значения), он добавляется в output.

- Если token — оператор, его приоритет сравнивается с приоритетами операторов в стеке. Если оператор на вершине стека имеет равный или более высокий приоритет, он переносится в output, после чего текущий оператор добавляется в стек.

- Если token — открывающая скобка, она помещается в стек.

- Если token — закрывающая скобка, все операторы из стека перемещаются в output до ближайшей открывающей скобки, которая затем удаляется из стека.

После обработки всех элементов из tokens оставшиеся в стеке операторы перемещаются в output. Функция возвращает постфиксную нотацию в виде строки.

Дополнительно создадим функцию для вычисления значения выражения в постфиксной нотации. В этой функции используется стек stack для промежуточных результатов и список tokens для разделения элементов выражения. В процессе обработки:

- Если token — число (включая отрицательные значения), оно добавляется в стек.

- Если token — оператор, из стека извлекаются два последних числа, над которыми выполняется соответствующая операция (сложение, вычитание, умножение или деление). Результат возвращается в стек.

После завершения вычислений функция возвращает единственный элемент из стека как итоговый результат.

Также реализуем функцию для проверки строки на наличие недопустимых символов (любых символов, кроме цифр, операторов +, -, \*, /, скобок и пробелов).

## 2.3 Тестирование программы

Проведём тестирование для данной программы на разных значениях. Результаты продемонстрируем на рисунке 1, 2.



**Рисунок 1 - Первое тестирование программы**



**Рисунок 2 - Второе тестирование программы**

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были достигнуты следующие результаты:

Реализовано преобразование алгебраического выражения в обратную польскую нотацию с использованием языка Python.

Создана программа, осуществляющая данное преобразование.

Проведено тестирование программы, результаты которого приведены в отчете.

Таким образом, поставленная цель работы — разработка программы для преобразования алгебраического выражения в обратную польскую запись на Python — была успешно достигнута.

# 

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Малявко А.А. Формальные языки и компиляторы: учебное пособие для вузов. - М., Изд-во Юрайт, 2019.
2. Свердлов С.З. Языки программирования и методы трансляции : учебное пособие для вузов - СПб., 2007, Id=65534.
3. Карпов Ю.Г. Теория и технология программирования. Основы построения трансляторов: учеб. пособие. - СПБ.: БХВ-Петербург, 2005, Id=64347.

# ПРИЛОЖЕНИЯ

Приложение А – код на языке Python

**ПРИЛОЖЕНИЕ А**

Код на языке Python

*Листинг А.1 – main.py*

| import re  def infix\_to\_postfix(expression):  precedence = {'+': 1, '-': 1, '\*': 2, '/': 2}  output = []  operators = []  tokens = re.findall(r'\d+\.?\d\*|[()+\*/-]', expression) # Нужно для деления на типо список из '10', '+', ... (я устал это прописывать)  for token in tokens:  if token.isdigit() or '.' in token: # Если это число  output.append(token)  elif token in precedence: # Если это оператор  while (operators and operators[-1] in precedence and  precedence[token] <= precedence[operators[-1]]):  output.append(operators.pop())  operators.append(token)  elif token == '(':  operators.append(token)  elif token == ')':  while operators and operators[-1] != '(':  output.append(operators.pop())  operators.pop() # Удаление '(' ато почему-то багалось  while operators:  output.append(operators.pop())  return ' '.join(output)  def evaluate\_postfix(expression):  stack = []  tokens = expression.split()  for token in tokens:  if token.isdigit() or '.' in token: # Если это число  stack.append(float(token))  else: # Если это оператор  b = stack.pop()  a = stack.pop()  if token == '+':  stack.append(a + b)  elif token == '-':  stack.append(a - b)  elif token == '\*':  stack.append(a \* b)  elif token == '/':  stack.append(a / b)  return stack[0] # По итогу возвращаем первый элемент в стеке, он же и ответ  expression = "10 + 3 \* (2 - 1/2)"  print(f"Дано: {expression}")  postfix\_expression = infix\_to\_postfix(expression)  print(f"Полиз: {postfix\_expression}")  result = evaluate\_postfix(postfix\_expression)  print(f"Результат: {result}") |
| --- |