|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования Python»  Вариант 15 | | |
|  | | Работу выполнил  студент группы ИТ-8 1 курса  Надымова Александра Николаевна  «16» июня 2025 г. |
| Работу проверил  Рубцова Марина Борисовна  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc200704019)

[Цель 3](#_Toc200704020)

[Требования к программе: 3](#_Toc200704021)

[Алгоритм решения 4](#_Toc200704022)

[Структуры данных 4](#_Toc200704023)

[Шаги выполнения алгоритма 5](#_Toc200704024)

[Тестирование 6](#_Toc200704025)

[Код программы 6](#_Toc200704026)

[Инструкция по применению стилей и оформлению работы 10](#_Toc200704027)

# Постановка задачи

Используя структуру стека подсчитать значение арифметического

выражения, записанного в обратной польской записи (постфиксной записи)

при условии, что используются знаки операций +, –, \*, / и операнды являются

вещественными положительными числами. Во вводимой пользователем

строке числа и знаки операций разделяются одним пробелом.

## Цель

Создать корректную, устойчивую к ошибкам и удобную в использовании программу, вычисляющую арифметическое выражение в постфиксной записи.

## Требования к программе:

1. Обработка операндов: положительные вещественные числа (например, 5, 3.14, 0.5).
2. Поддержка операций: +, -, \*, /.
3. Ввод данных: строка, где числа и операторы разделены пробелами (например, 3 4 + 2 \*).
4. Вывод: результат вычисления или понятное сообщение об ошибке.
5. Обработка некорректного ввода:

* Неверный формат чисел.
* Неправильное количество операторов и операндов.
* Недопустимые символы.

# Алгоритм решения

Для решения задачи применяется структура данных "стек", так как принцип его работы идеально соответствует порядку выполнения операций в постфиксной записи: сначала операнды добавляются в стек, затем применяются операторы, которые извлекают верхние значения и возвращают результат обратно в стек.

## Структуры данных

1. Class Stack

Основные методы:

* push(data) — добавление элемента на вершину стека.
* pop() — извлечение элемента с вершины.
* is\_empty() — проверка на пустоту.

1. Class Validator

Проверка корректности введённого выражения:

* is\_operand(s) — проверка, что строка является положительным числом.
* is\_operator(s) — проверка, что символ — допустимая операция.
* verification\_of\_correctness(expr) — проверка баланса операндов и операторов.

1. Класс PostfixCalculator

Выполняет вычисление:

* Последовательно обходит выражение.
* Числа добавляются в стек.
* При встрече оператора — извлекаются два верхних значения, применяется операция, результат кладётся обратно.
* В конце должно остаться одно число — результат.
* В случае ошибок (деление на ноль, недостаточно операндов и пр.) выбрасываются исключения.

## ****Шаги выполнения алгоритма****

1. Пользователь вводит выражение в виде строки, где числа и операции разделены пробелами.
   * Если ввод “exit” – завершение работы программы.
2. Строка разбивается на отдельные элементы по пробелам.
3. Программа проверяет, что:
   * Все элементы являются либо положительными числами, либо допустимыми операциями (+, -, \*, /).
   * Количество чисел на один больше, чем количество операций.
4. Если проверка на корректность не пройдена, происходит возврат к пункту 1.
5. Начинается вычисление:
   * Поочерёдно перебираются элементы выражения.
   * Если элемент — число, оно помещается в стек.
   * Если элемент — операция:
     1. Из стека извлекаются два верхних числа.
     2. К ним применяется операция.
     3. Результат возвращается в стек.
   * Если на каком-либо шаге невозможно извлечь два числа — возникает ошибка: "Недостаточно операндов".
6. После завершения обработки всех элементов:
   * Если в стеке остался один элемент — это результат.
   * Если в стеке остались лишние элементы — возникает ошибка: "Лишние операнды".
7. Выводится либо результат, либо сообщение об ошибке.

# Тестирование

|  |  |  |
| --- | --- | --- |
| **Ввод** | **Вывод** | **Комментарий** |
| 1 2 + | 3.0 | Одна операция |
| 5 6 \* 2 6 + + 2 / | 19.0 | Несколько операций |
| 5 6 \* 2 6 + + 2 / | 19.0 | Ввод через множество пробелов |
| 3 | 3.0 | Просто число |
| 2.5 6 / 3.25 \* | 1.3541666666666667 | Вещественный числа |
| 10000000000000 100000000000 \* | 1e+24 | Большие числа |
| 0.0000000004 0.000045452 / | 8.800492827598347e-06 | Маленькие числа |
| + | Количество чисел должно быть ровно на 1 больше, чем операций! | Операции без операндов |
| 5 6 + 7 6 | Количество чисел должно быть ровно на 1 больше, чем операций! | Количество операций ≠ количество операнд - 1 |
| 3 а + | Некорректный элемент а | Ввод недопустимого символа |
| 5 0 / | Деление на 0! | Деление на 0 |
| 6 2 / 6 6 - / | Деление на 0! | Неочевидное деление на 0 |
| 1 + 2 | Недостаточно операндов для 1 операции '+' ! | Нарушение порядка операндов и операций |
|  | Пустой ввод. Повторите попытку | Пустой ввод |

# 

# Код программы

class NodeList:

"""Узел Стека."""

def \_\_init\_\_(self, data=None, next\_el=None):

self.data = data

self.next\_el = next\_el

class Stack:

"""Стек"""

def \_\_init\_\_(self):

self.top = None

def is\_empty(self):

"""Проверка, пуст ли стек"""

return self.top is None

def push(self, data):

"""Добавить элемент на вершину стека"""

if self.is\_empty():

self.top = NodeList(data)

else:

newnode = NodeList(data) # создаём новый узел

newnode.next\_el = self.top # связываем его с текущей вершиной

self.top = newnode # переназначаем вершину

def pop(self):

"""Удалить с вершины"""

if self.is\_empty():

return None

else:

poppednode = self.top # сохраняем текущую вершину

self.top = self.top.next\_el # переносим указатель к следующему элементу

poppednode.next\_el = None # разрываем связь вершины со стеком

return poppednode.data # возвращаем значение удаленной вершины

class Validator:

"""Класс первичной проверки на корректность введённых данных"""

@staticmethod

def is\_operand(val: str) -> bool:

"""Проверка, что в строке число и оно больше 0"""

try:

return float(val) >= 0

except ValueError:

return False

@staticmethod

def is\_operator(s: str) -> bool:

"""Проеряет, что символ является какой либо оперцией"""

return s in {"+", "-", "/", "\*"}

@classmethod

def verification\_of\_correctness(cls, expression: list[str]) -> bool:

"""Проверка на корректность вводимых данных"""

# счётчики для подсчёта количества операндов и операций

operand\_count = 0

operator\_count = 0

for elem in expression:

if cls.is\_operand(elem): # если число

operand\_count += 1

elif cls.is\_operator(elem): # если знак операции

operator\_count += 1

else: # если не то и не другое - это некорректный элемент

raise ValueError(f"Некорректный элемент {elem}")

# В постфиксной записи операторов должно быть на 1 меньше, чем операндов

if operand\_count == operator\_count + 1:

return True

else:

raise ValueError(

"Некорректный ввод. Количество чисел должно быть на 1 больше, чем операций!"

)

class PostfixCalculator:

"""Класс для вычисления постфиксных выражений."""

@staticmethod

def evaluate(expression: list[str]) -> float:

"""Подсчёт выражения"""

stack = Stack()

count = 0 # счётчик знаков операций

for element in expression:

# если число, то добавляем в стек

if Validator.is\_operand(element):

stack.push(float(element))

# если знак операции, вычисляем значение

elif Validator.is\_operator(element):

count += 1

second = stack.pop()

first = stack.pop()

# если закончились элементы в стеке

if first is None or second is None:

raise ValueError(

f"Недостаточно операндов для {count} операции '{element}' !"

)

if element == "+":

stack.push(first + second)

elif element == "-":

stack.push(first - second)

elif element == "\*":

stack.push(first \* second)

else:

if second == 0:

raise ValueError("Деление на 0!")

stack.push(first / second)

else:

raise ValueError(f"Некорректный элемент {element}!")

result = stack.pop() # достам элемент после всего прохода

# проверка что в стеке ещё остались элементы

if not stack.is\_empty():

raise ValueError("Лишние операнды в стеке!")

return result

if \_\_name\_\_ == "\_\_main\_\_":

# программа выполняется до тех пор пока не будет введено exit

while True:

user\_input = input(

"\nВведите выражение в одну строчку через пробел, "

"содержащее только положительные числа и знаки операций: +, –, \*, /, \n"

"либо 'exit' для выхода: \n"

)

# удвляем пробелы и преобразуем к строчным

if user\_input.strip().lower() == "exit":

break

# если пустой ввод

if not user\_input:

print("Пустой ввод. Повторите попытку.")

continue

# разделяем элементы по пробелам

expression = user\_input.split()

# сначала пытаемся проверить на корректность

try:

if Validator.verification\_of\_correctness(expression):

# пытаемся вычислить значение

try:

result = PostfixCalculator().evaluate(expression)

print(f"Результат: {result}")

# если вышла ошибка при попытке вычислить

except ValueError as e:

print(f"Ошибка: {e}") # вывод названия ошибки

# есливышла ошибка при попытке проверить корректность

except ValueError as e:

print(f"Ошибка: {e}")