|  |  |  |
| --- | --- | --- |
| 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-6-2024 1 курса  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шестаков Е.Д.  «\_\_\_» \_\_\_\_\_\_\_\_2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рубцова М.Б.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#__RefHeading___1)

[Алгоритм решения 3](#__RefHeading___2)

[Тестирование 4](#__RefHeading___3)

[Код программы 4](#__RefHeading___4)

Код программы...................................................................................................................5

# Постановка задачи

Используя структуру стека, подсчитать значение арифметического выражения, записанного в обратной польской записи (постфиксной записи). В выражении используются знаки операций `+`, `-`, `\*`, `/`, а операнды являются вещественными положительными числами. Во вводимой пользователем строке числа и знаки операций разделяются одним пробелом.

# Алгоритм решения

Программа принимает от пользователя строку с выражением в постфиксной нотации, где операнды и операторы разделены пробелами. Сначала строка очищается от лишних пробелов по краям с помощью strip(), затем разбивается на отдельные элементы методом split(). Каждый элемент последовательно проверяется: если это число (проверяется вспомогательной функцией \_is\_number()), оно преобразуется в вещественный формат float и помещается в стек. Если элемент является оператором (+, -, \*, /), программа извлекает из стека два последних числа (при этом сначала извлекается второй операнд, затем первый), выполняет соответствующую операцию и помещает результат обратно в стек. Особое внимание уделяется операции деления - перед её выполнением проверяется, не равен ли второй операнд нулю. После обработки всех элементов в стеке должен остаться ровно один элемент - итоговый результат вычислений. Если это условие не выполняется, выражение считается некорректным.

**Ключевые проверки:**

* Деление на ноль: if b == 0: raise ValueError("Деление на ноль")
* Недостаток операндов: if len(self.stack) < 2: raise ValueError("Недостаточно операндов")
* Некорректные символы: else: raise ValueError(f"Неизвестный токен: {token}")

Автоматически проверяют базовые операции через assert в функции run\_tests()

# Тестирование

Тест 1: Простое сложение

Введите выражение: 3 5 +

Результат: 8.0

Тест 2: Комбинированное выражение

Введите выражение: 15 7 1 1 + - / 3 \* 2 1 1 + + -

Результат: 5.0

Тест 3: Ошибка - деление на ноль

Введите выражение: 4 0 /

Ошибка: Деление на ноль недопустимо

Тест 4: Ошибка - недостаток операндов

Введите выражение: 5 +

Ошибка: Недостаточно операндов для операции

Тест 5: Ошибка - некорректный символ

Введите выражение: 2 x +

Ошибка: Неизвестный символ: 'x'

Тест 6: Выход из программы

Введите выражение: exit

До свидания!

# Код программы

Сыллка на код https://github.com/akovve/-2

class Node:

"""Узел для реализации стека на связном списке"""

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

self.value = value

self.next = None

class Stack:

"""Реализация стека на связном списке"""

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

self.top = None

self.size = 0

def push(self, value):

"""Добавление элемента в стек"""

new\_node = Node(value)

new\_node.next = self.top

self.top = new\_node

self.size += 1

def pop(self):

"""Извлечение элемента из стека"""

value = self.top.value

self.top = self.top.next

self.size -= 1

return value

def peek(self):

"""Просмотр верхнего элемента без извлечения"""

if self.is\_empty():

raise IndexError("Попытка просмотра пустого стека")

return self.top.value

def is\_empty(self):

"""Проверка на пустоту"""

return self.top is None

def clear(self):

"""Очистка стека"""

self.top = None

self.size = 0

class PostfixCalculator:

"""Калькулятор постфиксных выражений"""

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

self.stack = Stack()

def evaluate(self, expression):

"""Вычисление постфиксного выражения"""

self.stack.clear()

tokens = expression.strip().split()

for token in tokens:

if self.\_is\_number(token):

self.stack.push(float(token))

elif token in '+-\*/':

self.\_apply\_operator(token)

else:

raise ValueError(f"Недопустимый токен: '{token}'")

if self.stack.size != 1:

raise ValueError("Некорректное выражение")

return self.stack.pop()

def \_apply\_operator(self, operator):

"""Применение оператора"""

if self.stack.size < 2:

raise ValueError("Недостаточно операндов")

b = self.stack.pop()

a = self.stack.pop()

if operator == '+':

self.stack.push(a + b)

elif operator == '-':

self.stack.push(a - b)

elif operator == '\*':

self.stack.push(a \* b)

elif operator == '/':

if b == 0:

raise ValueError("Деление на ноль")

self.stack.push(a / b)

def \_is\_number(self, token):

"""Проверка, является ли токен числом"""

try:

float(token)

return True

except ValueError:

return False

def main():

"""Интерактивный интерфейс калькулятора"""

calc = PostfixCalculator()

print("\n//====================================================//")

print("|| КАЛЬКУЛЯТОР ПОСТФИКСНЫХ ВЫРАЖЕНИЙ ||")

print("||====================================================||")

print("|| Пример: 3 5 + 2 \* → 16 ||")

print("|| Доступные операции: + - \* / ||")

print("|| Для выхода: 'exit', 'quit', 'выйти' ||")

print("//====================================================//")

while True:

try:

user\_input = input("\nВведите выражение: ").strip()

if user\_input.lower() in {'exit', 'quit', 'выйти'}:

print("\nРабота завершена. До свидания!")

break

if not user\_input:

print("Ошибка: Пустой ввод")

continue

result = calc.evaluate(user\_input)

print(f"Результат: {result:.2f}" if result % 1 else f"Результат: {int(result)}")

except ValueError as e:

print(f"Ошибка: {e}")

except Exception as e:

print(f"Неожиданная ошибка: {e}")

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

main()