|  |  |  |
| --- | --- | --- |
| 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-7 1 курса  Шмидт К. К.  «10» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc200464932)

[Алгоритм решения 3](#_Toc200464933)

[Тестирование 3](#_Toc200464934)

[Код программы 4](#_Toc200464935)

# 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.

Замечание: Постфиксной формой записи выражения a b называется запись, в которой знак операции размещен за операндами a b .

Примеры:

a - b → a b –

a \* b + c → a b \* c +

a \* (b + c) → a b c + \*

a + b / c / d \* e → a b c / d / e \* +

# Алгоритм решения

1. Разобьём введённое выражение на токены по пробелам.
2. Инициализируем стек пустым. Считываем токены слева направо.
3. Если очередной токен – вещественное число, добавляем его в стек.
4. Если очередной токен – знак операции, извлекаем из стека два числа, проводим над ними операцию и складываем результат в стек. Если двух чисел в стеке нет, выбрасываем исключение.
5. Если токен не является ни числом, ни знаком операции, выбрасываем исключение.
6. Когда обработали все токены, проверяем, что в стеке осталось ровно одно число. Если это верно, то выводим его в качестве результата, иначе выбрасываем исключение.

# Тестирование

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Вывод программы** |
| 2 1 – | 1 | 1.0 |
| 2.5 30 \* 2 + | 77 | 77.0 |
| 2.5 30 2 + \* | 80 | 80.0 |
| 2 6 3 / 2 / 5 \* + | 7 | 7.0 |
| 2 + | *Неверный ввод* | ОШИБКА! Недостаточно чисел для операции +: нужно хотя бы 2 |
| 1 2 $ | *Неверный ввод* | ОШИБКА! Некорректная операция: $ |
| 1 2 3 + | *Неверный ввод* | ОШИБКА! Недостаточно операций для завершения вычислений. |

# Код программы

def is\_float(s):

    """

    Проверяет, является ли строка представлением вещественного числа

    """

    try:

        float(s)

        return True

    except ValueError:

        return False

class Node:

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

        self.data = data

        self.next = next

class Stack:

    def \_\_init\_\_(self):

        # верхушкой стека считаем голову (первый элемент) односвязного списка

        self.head = None

        # из-за особенностей односвязного списка для быстрого вычисления длину нужно хранить отдельно

        self.len = 0

    def isempty(self):

        return self.head is None

    def push(self, data):

        self.len += 1

        if self.head == None:

            self.head = Node(data)

        else:

            newnode = Node(data, self.head)

            self.head = newnode

    def pop(self):

        if self.isempty():

            return None

        self.len -= 1

        poppednode = self.head

        self.head = self.head.next

        return poppednode.data

    def top(self):

        if self.isempty():

            return None

        else:

            return self.head.data

    def clear(self):

        # для очищения односвязного списка достаточно обнулить ссылку на голову

        self.head = None

        self.len = 0

    def length(self):

        return self.len

class Calculator:

    def \_\_init\_\_(self):

        self.stack = Stack()

    def \_perform(self, op: str):

        if is\_float(op):

            self.stack.push(float(op))

        else:

            if self.stack.length() < 2:

                raise Exception(f"Недостаточно чисел для операции {op}: нужно хотя бы 2")

            x = self.stack.pop()

            y = self.stack.pop()

            match op:

                case '+':

                    self.stack.push(x + y)

                case '-':

                    self.stack.push(y - x)

                case '\*':

                    self.stack.push(x \* y)

                case '/':

                    self.stack.push(y / x)

                case \_:

                    raise Exception(f"Некорректная операция: {op}")

    def calculate(self, formula: str):

        try:

            ops = formula.split()

            for op in ops:

                self.\_perform(op)

            if self.stack.length() > 1:

                raise Exception(f"Недостаточно операций для завершения вычислений.")

            return self.stack.pop()

        finally: # действия здесь будут выполнены независимо от того, произошла ли ошибка

            self.stack.clear()

if \_\_name\_\_ == '\_\_main\_\_':

    completed = False

    calc = Calculator()

    while True:

        formula = input("Введите выражение в обратной польской записи (для выхода ничего не пишите и нажмите Enter): ")

        if formula == '':

            break

        try:

            result = calc.calculate(formula)

            print("Результат вычислений:", result)

        except ZeroDivisionError:

            print("ОШИБКА! Деление на 0.")

            print("Попробуйте ещё раз.")

        except Exception as e:

            print("ОШИБКА!", e.args[0])

            print("Попробуйте ещё раз.")