|  |  |  |
| --- | --- | --- |
| 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 1 | | |
|  | | Работу выполнил  студент группы ИТ-7,8-2024 1 курса  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пушкарев С. Н.  «\_\_\_» \_\_\_\_\_\_\_\_2024 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рубцова М. Б.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование 3](#_Toc153130029)

[Код программы 3](#_Toc153130030)

[Инструкция по применению стилей и оформлению работы 4](#_Toc153130031)

# Постановка задачи

Используя структуру стека, вычислить выражение, содержащее две операции: поиск минимума (обозначается m(<число1>,<число2>)) и поиск максимума (обозначается M(<число1>,<число2>)). Операции могут быть вложенными, например, M(15, m(16,8)) (в данном выражении ищем минимум из 16 и 8, а потом ищем максимум от результата m(16,8) и 15, ответ 15). В качестве аргументов могут использоваться только целые положительные числа. Строка, вводимая пользователем, корректна и не содержит пробелов.

# Алгоритм решения

1. Пользователь вводит строку вида M(15,m(16,8)) - это выражение с операциями M (максимум) и m (минимум), в котором могут быть вложенные скобки.
2. Программа посимвольно читает строку:
   1. Если находит число - собирает его целиком;
   2. Если находит букву M или m - вызывается команда максимум или минимум;
   3. Если находит скобки - задается приоритет операций.
3. Элементы по очереди добавляются в стек.
4. Когда программа находит закрывающую скобку:
   1. Берет два последних числа из стека;
   2. Берет последнюю операцию (min или max);
   3. Выполняется действие;
   4. Результат добавляется в стек.
5. Программа сначала обрабатывает внутреннее выражение, а потом внешнее.
6. Оставшееся в стеке число - ответ.

# Тестирование

Ввод: M(15,m(16,8))

Вывод: 15

Ввод: m(5,M(10,3))

Вывод: 5

Ввод: M(m(1,2),m(3,4))

Вывод: 3

Ввод: m(m(1,2),M(3,4))

Вывод: 1

Ввод: M(100,200)

Вывод: 200

Ввод: m(100,200)

Вывод: 100

Ввод: M(M(5,6),m(7,8))

Вывод: 7

Ввод: m(50,m(25,75))

Вывод: 25

Ввод: M(1,M(2,M(3,4)))

Вывод: 4

Ввод: m(9,M(4,m(2,10)))

Вывод: 4

# Код программы

class StackEmptyError(Exception): # класс исключения для создания собственного типа ошибки

pass

class Stack: # класс стека

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

self.\_items = [] # хранение элементов стека

def push(self, item):

self.\_items.append(item) # добавление элемента в стек

def pop(self):

if self.is\_empty():

raise StackEmptyError("Попытка извлечь элемент из пустого стека.")

return self.\_items.pop() # извлечение элемента из стека

def peek(self):

if self.is\_empty():

raise StackEmptyError("Попытка посмотреть верхушку пустого стека.")

return self.\_items[-1] # просмотр верхнего элемента

def is\_empty(self):

return len(self.\_items) == 0 # проверка стека на пустоту

def size(self):

return len(self.\_items) # размер стека

class MathOperations: # класс для нахождения минимума/максимума

@staticmethod # декоратор для того, чтобы можно было использовать функцию без создания экземпляра класса

def custom\_min(a, b):

if a < b:

return a

return b

@staticmethod

def custom\_max(a, b):

if a > b:

return a

return b

class ExpressionEvaluator: # класс для вычисления выражения из стека

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

self.expression = expression # сохранение строки

self.stack = Stack() # инициализация стека

def evaluate(self):

i = 0

while i < len(self.expression): # посимвольное чтение строки

char = self.expression[i]

if char.isdigit(): # если символ - число

num = ''

while i < len(self.expression) and self.expression[i].isdigit(): # собираем число целиком

num += self.expression[i]

i += 1

self.stack.push(int(num)) # добавляем число в стек

continue

elif char in ('m', 'M', '(', ',', ')'):

self.stack.push(char)

i += 1

return self.\_process\_stack() # обработка стека

def \_process\_stack(self):

temp\_stack = Stack()

while not self.stack.is\_empty(): # рассматриваем стек в обратном порядке

token = self.stack.pop()

temp\_stack.push(token)

return self.\_evaluate\_tokens(temp\_stack) # обработка стека

def \_evaluate\_tokens(self, tokens):

while tokens.size() > 1:

temp = Stack()

while not tokens.is\_empty():

token = tokens.pop()

if token == ')': # если нашли закрывающую скобку, начинаем собирать выражение

arg2 = temp.pop()

comma = temp.pop()

arg1 = temp.pop()

open\_bracket = temp.pop()

func = temp.pop()

if func == 'm':

result = MathOperations.custom\_min(arg1, arg2)

elif func == 'M':

result = MathOperations.custom\_max(arg1, arg2)

else:

raise ValueError(f"Неизвестная операция: {func}")

temp.push(result) # результат операции добавляем обратно в стек

else:

temp.push(token)

tokens = temp

return tokens.pop()

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

try:

expression = input("Введите выражение (например, M(15,m(16,8))): ")

evaluator = ExpressionEvaluator(expression)

result = evaluator.evaluate()

print(f"Результат: {result}")

except Exception as e:

print(f"Ошибка: {e}")