|  |  |  |
| --- | --- | --- |
| 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 ПМИ-ИТ-7-2024 1 курса  Чеповецкий А.В Фамилия И.О.  «12» июня 2024 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc200637940)

[Алгоритм решения 3](#_Toc200637941)

[1. Валидация выражения: 3](#_Toc200637942)

[2. Вычисление выражения: 3](#_Toc200637943)

[3. Структуры и подходы: 3](#_Toc200637944)

[Тестирование 4](#_Toc200637945)

[1. Минимум из 5 и 10 | m(5,10) 4](#_Toc200637946)

[2. Максимум из 5 и 10 | M(5,10) 4](#_Toc200637947)

[3. Внутри m(7,4)=4, затем M(3,4)=4 | M(3,m(7,4)) 4](#_Toc200637948)

[4. M(3,7)=7, M(1,2)=2, затем m(7,2)=2 | m(M(3,7),M(1,2)) 4](#_Toc200637949)

[5. m(1,2)=1, m(4,5)=4, M(3,4)=4, M(1,4)=4 | M(m(1,2),M(3,m(4,5))) 5](#_Toc200637950)

[6. Глубокая вложенность | m(m(m(9,8),7),6) 5](#_Toc200637951)

[7. Смешанные вложенности | M(5,m(4,M(1,2))) 5](#_Toc200637952)

[8. Ошибка: недостающий аргумент | m(1,) 5](#_Toc200637953)

[9. Ошибка: несбалансированные скобки | M(5,5 5](#_Toc200637954)

[10. Ошибка: недопустимый символ | X(1,2) 6](#_Toc200637955)

[11. Ошибка: более двух аргументов | M(3,4,5) 6](#_Toc200637956)

[Код программы 6](#_Toc200637957)

# Постановка задачи

Используя структуру стека, вычислить выражение, содержащее две операции: поиск минимума (обозначается m(<число1>,<число2>)) и поиск максимума (обозначается М(<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

## Валидация выражения:

* Разрешены только символы 0–9, m, M, (, ), ,.
* Проверка баланса скобок: каждое открытие ( должно иметь соответствующее закрытие ).
* Проверка, что каждая функция m(...) и M(...) имеет ровно два аргумента.

## Вычисление выражения:

* Используется стек (структура LIFO).
* Выражение обрабатывается справа налево:
* Если обнаружено число, оно добавляется в стек.
* Если обнаружена функция m(...) или M(...), из стека извлекаются два аргумента, вычисляется минимум или максимум, результат кладётся обратно в стек.
* В конце в стеке должен остаться единственный результат.

## Структуры и подходы:

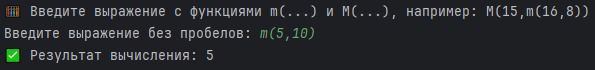
Используется классExpressionEvaluator с методами:

* is\_valid\_expression() — проверка корректности выражения;
* has\_valid\_arguments() — проверка количества аргументов;
* evaluate() — вычисление результата.

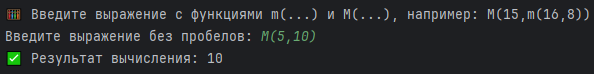
В качестве структуры данных выбран список Python, используемый как стек, поскольку он поддерживает методы .append() и .pop(), что удобно и эффективно при реализации LIFO.

# Тестирование

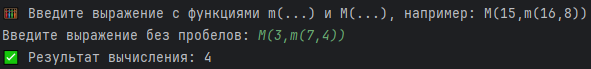
## Минимум из 5 и 10 | m(5,10)



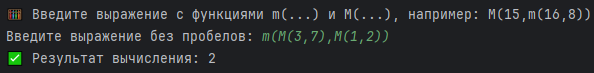
## Максимум из 5 и 10 | M(5,10)



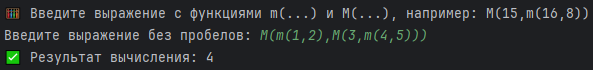
## Внутри m(7,4)=4, затем M(3,4)=4 | M(3,m(7,4))



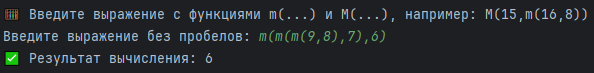
## M(3,7)=7, M(1,2)=2, затем m(7,2)=2 | m(M(3,7),M(1,2))



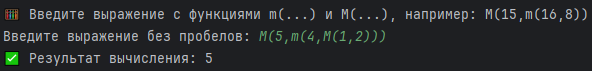
## m(1,2)=1, m(4,5)=4, M(3,4)=4, M(1,4)=4 | M(m(1,2),M(3,m(4,5)))



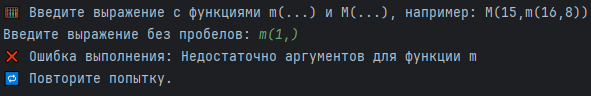
## Глубокая вложенность | m(m(m(9,8),7),6)



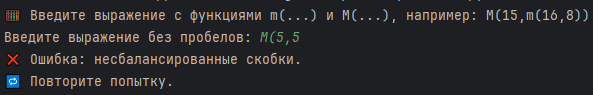
## Смешанные вложенности | M(5,m(4,M(1,2)))



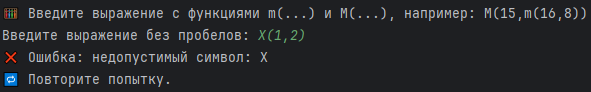
## Ошибка: недостающий аргумент | m(1,)



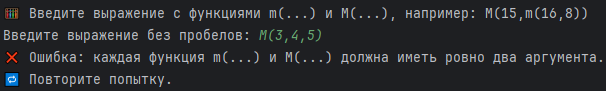
## Ошибка: несбалансированные скобки | M(5,5



## Ошибка: недопустимый символ | X(1,2)



## Ошибка: более двух аргументов | M(3,4,5)



# Код программы

class ExpressionEvaluator:

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

self.stack = [] # Стек для хранения чисел в процессе вычисления выражения

def is\_valid\_expression(self, expr):

"""

Проверка на корректность выражения:

допустимые символы, сбалансированные скобки, корректные аргументы у функций

"""

allowed = "0123456789mM()," # допустимые символы

bracket\_balance = 0 # счётчик открытых/закрытых скобок

for char in expr:

if char not in allowed:

print("❌ Ошибка: недопустимый символ:", char)

return False

if char == '(':

bracket\_balance += 1

elif char == ')':

bracket\_balance -= 1

# если закрыли скобку раньше открытия — ошибка

if bracket\_balance < 0:

print("❌ Ошибка: лишняя закрывающая скобка.")

return False

# проверка на несбалансированность скобок

if bracket\_balance != 0:

print("❌ Ошибка: несбалансированные скобки.")

return False

# проверка, что каждая функция имеет ровно два аргумента

if not self.has\_valid\_arguments(expr):

print("❌ Ошибка: каждая функция m(...) и M(...) должна иметь ровно два аргумента.")

return False

return True

# Проверка, что у всех функций m(...) и M(...) два аргумента

def has\_valid\_arguments(self, expr):

i = 0

while i < len(expr):

# ищем функцию вида m(...) или M(...)

if expr[i] in 'mM' and i + 1 < len(expr) and expr[i + 1] == '(':

i += 2 # пропускаем символ функции и открывающую скобку

depth = 1 # отслеживаем глубину вложенности

args = [] # список аргументов текущей функции

current = '' # текущий аргумент как строка

# собираем содержимое функции до закрывающей скобки

while i < len(expr) and depth > 0:

if expr[i] == '(':

depth += 1

current += expr[i]

elif expr[i] == ')':

depth -= 1

if depth > 0:

current += expr[i]

elif expr[i] == ',' and depth == 1:

# разделяем аргументы по запятой на верхнем уровне

args.append(current.strip())

current = ''

else:

current += expr[i]

i += 1

args.append(current.strip()) # добавляем последний аргумент

if len(args) != 2:

return False

else:

i += 1

return True

# Основной метод: вычисляет выражение, используя стек

def evaluate(self, expr):

self.stack.clear() # очищаем стек перед вычислением

i = len(expr) - 1 # начинаем обход выражения справа налево

while i >= 0:

if expr[i].isdigit():

# считываем многозначное число справа налево

number = ''

while i >= 0 and expr[i].isdigit():

number = expr[i] + number # цифры собираются в обратном порядке

i -= 1

self.stack.append(int(number)) # кладём число в стек

elif expr[i] in 'mM':

op = expr[i] # сохраняем тип операции: min или max

i -= 2 # пропускаем 'm(' или 'M('

# проверяем, что в стеке есть два числа для применения функции

if len(self.stack) < 2:

raise ValueError(f"Недостаточно аргументов для функции {op}")

# извлекаем два аргумента из стека

a = self.stack.pop()

b = self.stack.pop()

# вычисляем min или max и сохраняем результат в стек

result = min(a, b) if op == 'm' else max(a, b)

self.stack.append(result)

else:

# пропускаем символы '(', ')', ',' и т.п.

i -= 1

# в конце в стеке должен остаться один элемент — результат

if len(self.stack) != 1:

raise ValueError("Некорректное выражение: стек не сбалансирован.")

return self.stack[0]

# Пользовательский интерфейс

def main():

print("🧮 Введите выражение с функциями m(...) и M(...), например: M(15,m(16,8))")

evaluator = ExpressionEvaluator()

while True:

expression = input("Введите выражение без пробелов: ")

# проверка корректности выражения

if evaluator.is\_valid\_expression(expression):

try:

result = evaluator.evaluate(expression)

print("✅ Результат вычисления:", result)

break # выходим после успешного вычисления

except Exception as e:

# ошибка при выполнении вычислений

print("❌ Ошибка выполнения:", e)

print("🔁 Повторите попытку.\n")

else:

# если выражение некорректно, повторяем ввод

print("🔁 Повторите попытку.\n")

# Запуск основной функции

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

main()

https://github.com/Amaranthik/ikm-prjct