|  |  |  |
| --- | --- | --- |
| 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-5-2024 1 курса  Семешко А.А  «12» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc200646248)

[Алгоритм решения 3](#_Toc200646249)

[1. Постановка задачи 3](#_Toc200646250)

[2. Ключевая идея 3](#_Toc200646251)

[3. Подробный алгоритм 4](#_Toc200646252)

[4. Корректность 5](#_Toc200646253)

[5. Сложность 5](#_Toc200646254)

[6. Итог 5](#_Toc200646255)

[Тестирование 5](#_Toc200646256)

[Код программы 6](#_Toc200646257)

# 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

## 1. Постановка задачи

Нужно вычислить значение корректного выражения, построенного из двух бинарных операций:

* m(a,b) — минимум двух положительных целых a и b;
* M(a,b) — максимум тех же чисел.

Операции могут вкладываться любым образом. Пробелов нет, аргументы — только положительные целые.

## **2. Ключевая идея**

Разбор выражения выполняется слева направо в один проход, используя **два стека**:

* стек ops хранит ещё не обработанные операции m или M;
* стек vals хранит числовые аргументы и промежуточные результаты.

При встрече закрывающей скобки ) на вершинах стеков лежат ровно два аргумента соответствующей функции и сама операция. Выполняем её, кладём результат обратно в vals.

## 3. Подробный алгоритм

**Валидация входа**

Проверяем, что строка не пуста.

Допустимы только цифры, буквы m/M, скобки и запятые.

Баланс скобок отслеживается счётчиком balance; лишняя ) или незакрытая ( вызывает ошибку.

1. **Основной однопроходный разбор**

ops ← пустой стек

vals ← пустой стек

i ← 0

пока i < длина(expr):

ch ← expr[i]

если ch – цифра:

читаем целое число полностью

push(число) в vals

продолжить

если ch == 'm' или 'M':

push(ch) в ops

если ch == ')':

b ← pop(vals)

a ← pop(vals)

op ← pop(ops)

res ← min(a,b) если op == 'm', иначе max(a,b)

push(res) в vals

i ← i + 1 # '(', ',' просто пропускаем

конец цикла

если в ops что-то осталось или в vals не один элемент:

ошибка «некорректное выражение»

ответ ← pop(vals)

1. **Стек без встроенных списков**  
   Реализован на односвязном списке узлов Node(value, next).  
   Методы: push, pop с проверкой пустого стека и \_\_len\_\_ для len(stack).

## 4. Корректность

Инвариант: перед обработкой каждого символа стеки содержат ровно данные подвыражений, полностью просмотренных слева. Когда видим ), два верхних числа и верхняя операция относятся именно к этой скобке; выполняем, инвариант сохраняется. По завершении остаётся один результат — значение всего выражения.

## 5. Сложность

**Время** — O(n), где n — длина строки (один линейный проход).

**Память** — O(d), где d — максимальная глубина вложенности, поскольку храним два стека той же глубины.

## 6. Итог

Алгоритм использует только примитивные операции языка (str, int, len, чтение/вывод) и собственные структуры данных. Он простой, линейный по времени, требует памяти лишь пропорционально глубине вложенности и легко доказывается по инварианту стека.

# Тестирование

| **№** | **Входная строка** | **Ожидаемый результат** | **Что проверяем** |
| --- | --- | --- | --- |
| 1 | m(4,9) | 4 | Один простейший m |
| 2 | M(4,9) | 9 | Один простейший M |
| 3 | M(15,m(16,8)) | 15 | Вложенный m внутри M |
| 4 | m(M(10,2),M(3,5)) | 5 | Вложенные M-операции внутри m |
| 5 | M(m(12,7),m(M(14,2),9)) | 9 | Смешанные уровни вложенности слева и справа |
| 6 | m(m(m(10,20),30),M(5,m(8,4))) | 5 | «Глубокая» левая цепочка + смешанная правая ветка |
| 7 | M(123456,m(999999,1000000)) | 999999 | Работу с большими многозначными числами |
| 8 | m(M(7,9),m(8,8)) | 8 | Чередование строчных/прописных m/M и равные арг-ты |
| 9 | M(m(100,m(90,M(80,m(70,M(60,m(50,40)))))), m(M(200,180),m(170,M(160,150)))) | 160 | Длинное («стрессовое») выражение ≈ 80 симв. и 6 уровней вложенности |
| 10-a | m(1,2)) | **Ошибка:** Лишняя закрывающая скобка | Случай «лишняя )» |
| 10-b | M(3,4#) | **Ошибка:** Недопустимый символ: # | Запрещённый символ |
| 10-c | M(m(1,2),3 | **Ошибка:** Скобки не сбалансированы | Нет закрывающей ) |

# Код программы

<https://github.com/EnotikonZ/IKM/blob/main/v1>  
  
# 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.  
*"""  
Калькулятор для выражений вида m(…), M(…).  
  
Формат:  
 • m(a,b) — минимум двух положительных целых a и b  
 • M(a,b) — максимум двух положительных целых a и b  
 • Операции можно вкладывать друг в друга, например:  
 M(15,m(16,8))  
 • Разрешены только символы: цифры 0-9, буквы m/M, скобки ( ), запятая ,  
 • Пробелов быть не должно.  
 • Пустая строка завершает работу калькулятора.  
"""*# ──────────────────── Узел и стек ───────────────────────────────────────────  
class Node: # узел односвязного списка  
 def \_\_init\_\_(self, value, nxt=None): # значение + ссылка  
 self.value = value # само значение  
 self.next = nxt # ссылка на следующий узел  
  
class Stack: # стек LIFO на односвязном списке  
 def \_\_init\_\_(self):  
 self.head = None # вершина стека  
 self.size = 0 # текущее количество элементов  
  
 def push(self, value): # кладём элемент наверх  
 self.head = Node(value, self.head) # новый узел → старая вершина  
 self.size += 1 # увеличиваем размер  
  
 def pop(self): # снимаем верхний элемент  
 if self.head is None: # пустой стек → ошибка  
 raise ValueError("Стек пуст")  
 value = self.head.value # сохраняем значение  
 self.head = self.head.next # сдвигаем вершину  
 self.size -= 1 # уменьшаем размер  
 return value # возвращаем извлечённое  
  
 def \_\_len\_\_(self): # позволяет вызвать len(stack)  
 return self.size  
  
# ──────────────────── Валидация выражения ──────────────────────────────────  
def validate(expr):  
 *"""Проверяет допустимые символы и баланс скобок."""* if len(expr) == 0: # пустая строка  
 raise ValueError("Пустая строка")  
 balance = 0 # счётчик скобок  
 for ch in expr:  
 if not (('0' <= ch <= '9') or ch in "mM(),"):  
 raise ValueError("Недопустимый символ: " + ch)  
 if ch == '(':  
 balance += 1 # открыли скобку  
 elif ch == ')':  
 balance -= 1 # закрыли скобку  
 if balance < 0:  
 raise ValueError("Лишняя закрывающая скобка")  
 if balance != 0:  
 raise ValueError("Скобки не сбалансированы")  
  
# ──────────────────── Основной вычислитель ─────────────────────────────────  
def eval\_min\_max(expr):  
 *"""Вычисляет корректное выражение из m/M, используя два стека."""* validate(expr) # защита от дурака  
  
 ops = Stack() # стек операций  
 vals = Stack() # стек чисел  
 i = 0 # текущая позиция курсора  
  
 while i < len(expr):  
 ch = expr[i]  
  
 if '0' <= ch <= '9': # начинаем читать число  
 num = 0  
 while i < len(expr) and '0' <= expr[i] <= '9':  
 num = num \* 10 + int(expr[i]) # «сдвиг» + цифра  
 i += 1  
 vals.push(num) # кладём число в стек  
 continue # уже на новом символе  
  
 if ch == 'm' or ch == 'M': # встретили имя функции  
 ops.push(ch) # сохраняем тип операции  
 elif ch == ')': # закрылась пара аргументов  
 b = vals.pop() # второй аргумент  
 a = vals.pop() # первый аргумент  
 op = ops.pop() # операция m/M  
 if op == 'm': # min  
 vals.push(a if a < b else b)  
 else: # max  
 vals.push(a if a > b else b)  
 # символы '(' и ',' пропускаем  
 i += 1 # шаг вперёд  
  
 if len(vals) != 1 or len(ops) != 0: # в конце должен остаться 1 результат  
 raise ValueError("Некорректное выражение")  
 return vals.pop()  
  
# ──────────────────── Дружественный REPL ───────────────────────────────────  
def repl():  
 *"""Интерактивный режим: принимает выражения, выводит результат."""* print(\_\_doc\_\_.strip()) # выводим инструкцию из docstring  
 print() # пустая строка-отступ  
 while True:  
 expr = input(">>> ").strip() # читаем ввод пользователя  
 if expr == "": # пустая строка → выход  
 print("До свидания!")  
 break  
 try:  
 result = eval\_min\_max(expr) # вычисляем выражение  
 print("Результат:", result) # печатаем ответ  
 except ValueError as err: # ловим ошибки парсинга  
 print("Ошибка:", err)  
  
# ──────────────────── Точка входа ──────────────────────────────────────────  
if \_\_name\_\_ == "\_\_main\_\_":  
 repl()