|  |  |  |
| --- | --- | --- |
| 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](#_Toc201199473)

[Алгоритм решения 3](#_Toc201199474)

[1. Валидация выражения: 3](#_Toc201199475)

[2. Вычисление выражения: 3](#_Toc201199476)

[3. Структуры и подходы: 4](#_Toc201199477)

[Тестирование 4](#_Toc201199478)

[1. Минимум из 5 и 10 | m(5,10) 4](#_Toc201199479)

[2. Максимум из 5 и 10 | M(5,10) 5](#_Toc201199480)

[3. Внутри m(7,4)=4, затем M(3,4)=4 | M(3,m(7,4)) 5](#_Toc201199481)

[4. M(3,7)=7, M(1,2)=2, затем m(7,2)=2 | m(M(3,7),M(1,2)) 5](#_Toc201199482)

[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](#_Toc201199483)

[6. Глубокая вложенность | m(m(m(9,8),7),6) 5](#_Toc201199484)

[7. Смешанные вложенности | M(5,m(4,M(1,2))) 5](#_Toc201199485)

[8. Ошибка: недостающий аргумент | m(1,) 6](#_Toc201199486)

[9. Ошибка: несбалансированные скобки | M(5,5 6](#_Toc201199487)

[10. Ошибка: недопустимый символ | X(1,2) 6](#_Toc201199488)

[11. Ошибка: более двух аргументов | M(3,4,5) 6](#_Toc201199489)

[Код программы 7](#_Toc201199490)

# 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 собственная реализация стека на основе односвязного списка (структура LIFO).
* Выражение обрабатывается справа налево:
* Если обнаружено число, оно полностью считывается и помещается в стек.
* Если обнаружена функция m(...) или M(...), из стека извлекаются два аргумента, вычисляется минимум или максимум, результат помещается обратно в стек.
* В конце в стеке должен остаться единственный элемент — итоговый результат.

## Структуры и подходы:

 Используется два класса:

* Stack — собственная реализация стека на основе односвязного списка.
* ExpressionEvaluator — основной класс обработки выражения:
  + is\_valid\_expression() — проверка корректности выражения;
  + has\_valid\_arguments() — проверка количества аргументов;
  + evaluate() — выполнение вычисления.

 Реализация собственного стека вместо встроенного списка позволяет:

* полностью контролировать работу со стеком;
* избавиться от использования встроенных структур Python;
* показать работу со структурами данных «с нуля».

 Применяется обработка исключений:

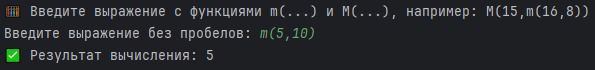
* При отсутствии достаточного количества аргументов для функции (например, при ошибках в выражении) возбуждается ValueError. Исключение перехватывается в основном цикле программы и позволяет выводить пользователю понятные сообщения об ошибках.

 Применяются принципы ООП:

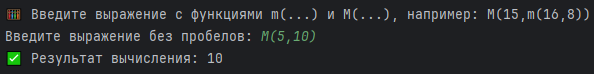
* **Инкапсуляция** — каждый класс инкапсулирует свою логику: стек — работу со структурой данных, парсер — обработку выражения.
* **Отсутствие наследования** — поскольку задача ограничена, использование наследования не требуется.
* **Отсутствие полиморфизма** — все функции строго определены.

# Тестирование

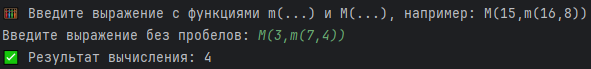
## Минимум из 5 и 10 | m(5,10)



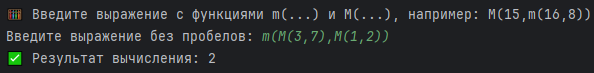
## Максимум из 5 и 10 | M(5,10)



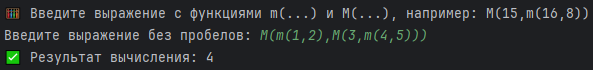
## Внутри m(7,4)=4, затем M(3,4)=4 | M(3,m(7,4))



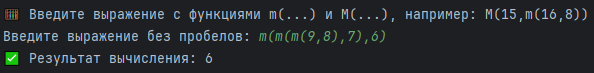
## M(3,7)=7, M(1,2)=2, затем m(7,2)=2 | m(M(3,7),M(1,2))



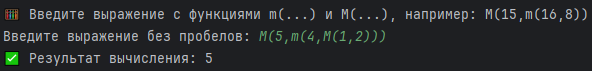
## m(1,2)=1, m(4,5)=4, M(3,4)=4, M(1,4)=4 | M(m(1,2),M(3,m(4,5)))



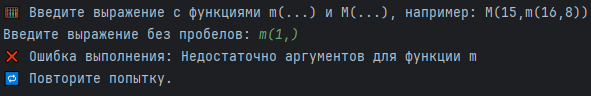
## Глубокая вложенность | m(m(m(9,8),7),6)



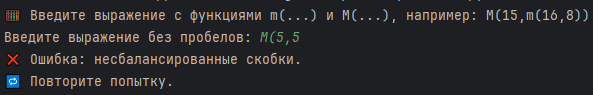
## Смешанные вложенности | M(5,m(4,M(1,2)))



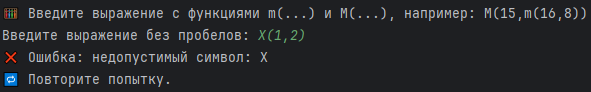
## Ошибка: недостающий аргумент | m(1,)



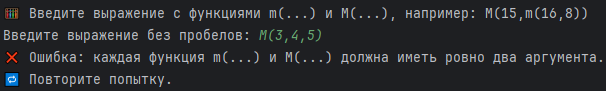
## Ошибка: несбалансированные скобки | M(5,5



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



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



# Код программы

# Узел связного списка для стека

**class** **Node:**

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

self**.***value* **=** value # значение узла

self**.next** **=** **None** # ссылка на следующий узел

# Реализация стека на основе связного списка

**class** **Stack:**

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

self**.***top* **=** **None** # вершина стека (ссылка на верхний элемент)

# Поместить значение в стек

**def** push**(**self**,** value**):**

node **=** Node**(**value**)** # создаём новый узел

node**.next** **=** self**.***top* # связываем новый узел с текущей вершиной

self**.***top* **=** node # новый узел становится вершиной

# Вытолкнуть значение из стека

**def** pop**(**self**):**

**if** self**.***top* **is** **None:**

**raise** **ValueError(**"Стек пуст"**)** # защита от пустого стека

value **=** self**.***top***.***value* # сохраняем значение текущей вершины

self**.***top* **=** self**.***top***.next** # переходим к следующему элементу

**return** value # возвращаем значение

# Очистить стек

**def** clear**(**self**):**

self**.***top* **=** **None**

# Подсчитать количество элементов в стеке

**def** \_\_len\_\_**(**self**):**

count **=** 0

current **=** self**.***top*

**while** current**:**

count **+=** 1

current **=** current**.next**

**return** count

# Основной класс — вычислитель выражений

**class** **ExpressionEvaluator:**

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

self**.***stack* **=** Stack**()**

# Проверка корректности введённого выражения

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

allowed **=** "0123456789mM(),"

bracket\_balance **=** 0 # баланс открытых и 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

**if** bracket\_balance **<** 0**:**

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

**return** **False**

**if** bracket\_balance **!=** 0**:**

**print(**"❌ Ошибка: несбалансированные скобки."**)**

**return** **False**

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

**print(**"❌ Ошибка: у функций m/M должно быть два аргумента."**)**

**return** **False**

**return** **True**

# Проверка, что у функций m и M два аргумента

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

i **=** 0

**while** i **<** **len(**expr**):**

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

i **+=** 2 # пропускаем имя функции и открывающую скобку

depth **=** 1 # отслеживаем вложенность скобок

args **=** **[]** # аргументы функции

current **=** '' # собираем текущий аргумент

**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**:**

args**.***append***(**current**.***strip***())**

current **=** ''

**else:**

current **+=** expr**[**i**]**

i **+=** 1

args**.***append***(**current**.***strip***())**

**if** **len(**args**)** **!=** 2**:** # проверяем, что аргументов ровно два

**return** **False**

**else:**

i **+=** 1

**return** **True**

# Основной метод вычисления выражения

**def** evaluate**(**self**,** expr**):**

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

i **=** **len(**expr**)** **-** 1 # обходим строку справа налево (обратный обход)

**while** i **>=** 0**:**

**if** expr**[**i**].***isdigit***():**

number **=** ''

# считываем число (учитываем многозначные числа)

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

number **=** expr**[**i**]** **+** number

i **-=** 1

self**.***stack***.***push***(int(**number**))**

**elif** expr**[**i**]** **in** 'mM'**:**

op **=** expr**[**i**]**

i **-=** 2 # пропускаем символ функции и '('

**if** **len(**self**.***stack***)** **<** 2**:**

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

a **=** self**.***stack***.***pop***()**

b **=** self**.***stack***.***pop***()**

# применяем min или max

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

self**.***stack***.***push***(**result**)**

**else:**

# игнорируем скобки и запятые

i **-=** 1

**if** **len(**self**.***stack***)** **!=** 1**:**

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

**return** self**.***stack***.***pop***()**

# Интерфейс пользователя

**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: "**)**

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

**try:**

result **=** evaluator**.***evaluate***(**expression**)**

**print(**"✅ Результат вычисления:"**,** result**)**

**break**

**except** **Exception** **as** e**:**

**print(**"❌ Ошибка выполнения:"**,** e**)**

**print(**"🔁 Повторите попытку.\n"**)**

**else:**

**print(**"🔁 Повторите попытку.\n"**)**

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

main**()**

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