|  |  |  |
| --- | --- | --- |
| 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 9 | | |
|  | | Работу выполнил  студент группы ИТ-6-2024 1 курса  Плеханов Д. В.  «12» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

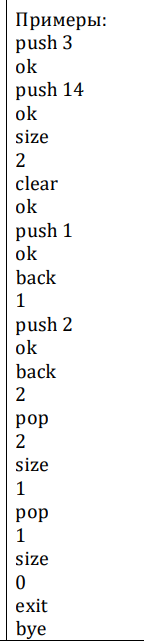
[Тестирование 4](#_Toc153130029)

[Код программы 13](#_Toc153130030)

[Инструкция по применению стилей и оформлению работы 17](#_Toc153130031)

# Постановка задачи

# Реализуйте структуру данных "стек". Напишите программу, содержащую описание стека и моделирующую работу стека, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы: push n - Добавить в стек число n (значение n задается после команды). Программа должна вывести ok. pop - Удалить из стека последний элемент. Программа должна вывести его значение. back - Программа должна вывести значение последнего элемента, не удаляя его из стека. size - Программа должна вывести количество элементов в стеке. clear - Программа должна очистить стек и вывести ok. exit - Программа должна вывести bye и завершить работу. Входные данные: команды управления стеком вводятся в описанном ранее формате по 1 на строке. Гарантируется, что набор входных команд удовлетворяет следующим требованиям: максимальное количество элементов в стеке в любой момент не превосходит100, все команды pop и back корректны, то есть при их исполнении в стеке содержится хотя бы один элемент. Выходные данные: протокол работы со стеком, по 1 сообщению в строке.



# Алгоритм решения

**Постановка задачи**

**Цель:**  
Реализовать структуру данных **стек** (LIFO - Last In, First Out) с поддержкой основных операций:

* push n – добавление элемента n в стек (вывод ok),
* pop – удаление и возврат верхнего элемента (вывод значения),
* back – просмотр верхнего элемента без удаления (вывод значения),
* size – получение текущего количества элементов в стеке,
* clear – очистка стека (вывод ok),
* exit – завершение программы (вывод bye).

**Требования:**

* Максимальный размер стека – **100 элементов**.
* Все команды pop и back корректны (стек не пуст при их вызове).
* Ввод команд осуществляется построчно.
* После каждой команды выводится соответствующее сообщение.

## Подробный алгоритм решения

Используется **связный список** (на основе узлов StackNode), где каждый элемент хранит:

* value – значение,
* next – ссылку на следующий узел.
* **Основные методы класса Stack**

1. **push(value)**
   * Проверяет, не превышен ли лимит (max\_size = 100).
   * Создает новый узел и добавляет его в вершину стека.
   * Увеличивает счетчик size.
   * Возвращает "ok".
2. **pop()**
   * Проверяет, не пуст ли стек.
   * Сохраняет значение верхнего узла.
   * Перемещает указатель top на следующий узел.
   * Уменьшает size.
   * Возвращает сохраненное значение.
3. **back()**
   * Проверяет, не пуст ли стек.
   * Возвращает значение верхнего узла без изменений стека.
4. **size()**
   * Возвращает текущее количество элементов (self.\_\_size).
5. **clear()**
   * Сбрасывает top в None и size в 0.
   * Возвращает "ok".

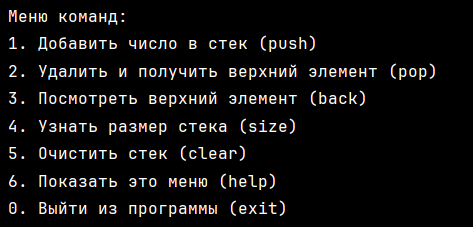
* **Обработка команд**

Программа работает в бесконечном цикле, пока не получит команду exit:

1. Считывает команду.
2. Вызывает соответствующий метод:
   * push n → stack.push(n),
   * pop → stack.pop(),
   * и т. д.
3. Выводит результат операции.

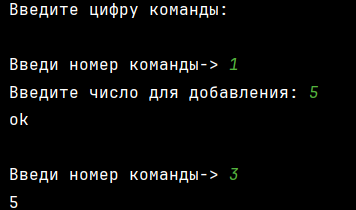
* **Ошибки**
* При переполнении стека (push при size >= 100) – игнорируется (по условию гарантируется, что такого не будет).
* Пустой стек в pop/back – также не возникает (по условию).

# Тестирование

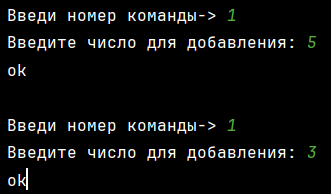
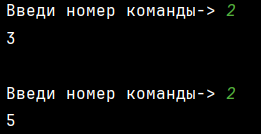


# Тесты для класса Stack:

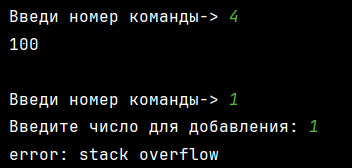
1. **Тест метода push:**
   * Добавление одного элемента и проверка, что он действительно добавлен.



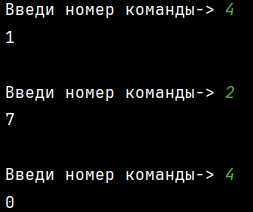
* + Добавление нескольких элементов и проверка их порядка.

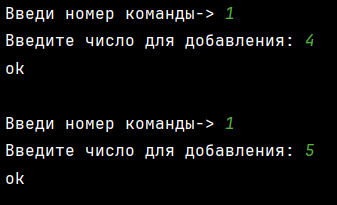
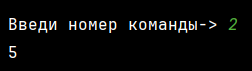
* + Попытка добавления элемента в заполненный стек (должна возвращать ошибку переполнения).



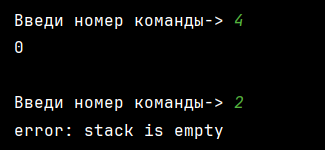
1. **Тест метода pop:**
   * Удаление элемента из стека с одним элементом и проверка, что стек стал пустым.



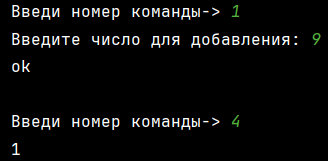
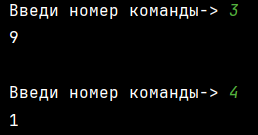
* + Удаление элемента из стека с несколькими элементами и проверка, что удален последний добавленный элемент.

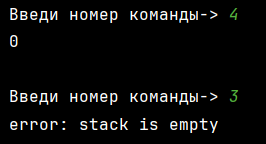
* + Попытка удаления из пустого стека (должна возвращать ошибку).



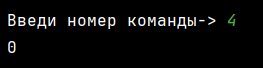
1. **Тест метода back:**
   * Проверка значения верхнего элемента без его удаления.

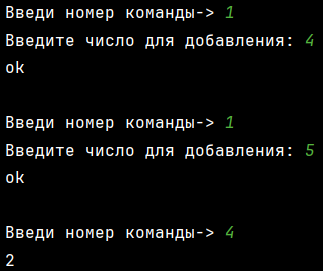
* + Проверка на пустом стеке (должна возвращать ошибку).



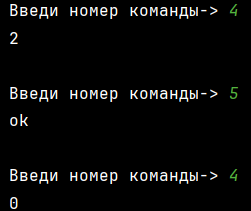
1. **Тест метода size:**
   * Проверка размера пустого стека.



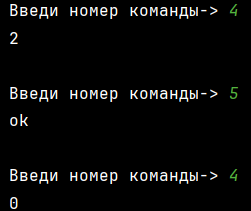
* + Проверка размера после добавления нескольких элементов.



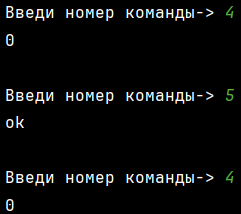
* + Проверка размера после очистки стека.



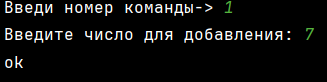
1. **Тест метода clear:**
   * Очистка непустого стека и проверка, что размер стал равен 0.



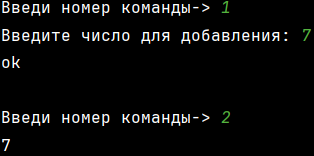
* + Очистка пустого стека.



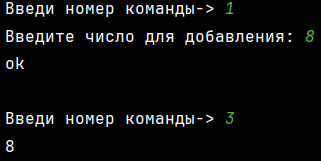
1. **Тесты для основной программы (main):**
2. **Тест обработки команд:**
   * Проверка реакции на команду push 7 (должна выводить "ok").



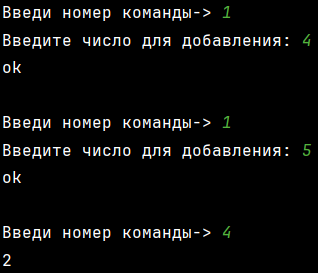
* + Проверка реакции на команду pop (должна выводить удаленный элемент).



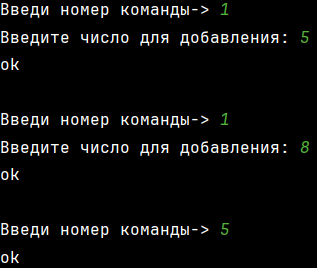
* + Проверка реакции на команду back (должна выводить последний элемент).



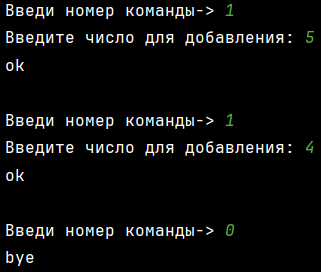
* + Проверка реакции на команду size (должна выводить текущий размер стека).



* + Проверка реакции на команду clear (должна выводить "ok").



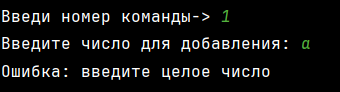
* + Проверка реакции на команду exit (должна выводить "bye" и завершать работу).



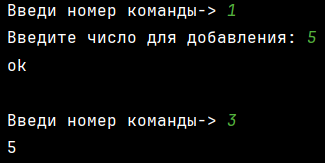
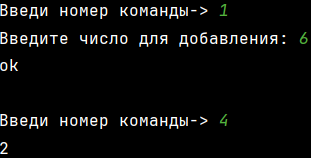
1. **Тест обработки некорректного ввода:**
   * Проверка реакции на ввод несуществующей команды.



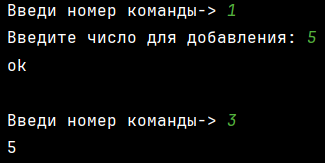
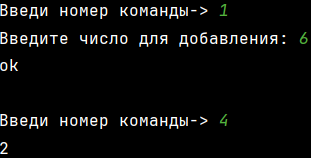
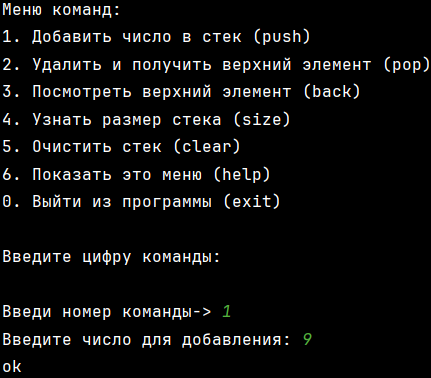
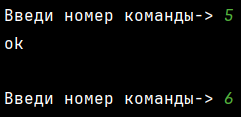
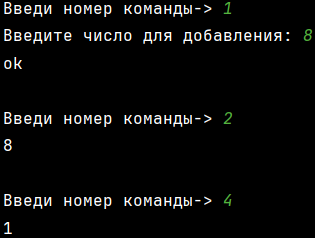
* + Проверка реакции на ввод некорректного значения для push (не число).



1. **Тест последовательности команд:**
   * Последовательное выполнение нескольких команд и проверка корректности вывода после каждой.

* + Проверка работы стека после очистки и повторного добавления элементов.

# Код программы

class StackNode:  
 *"""Узел стека."""* def \_\_init\_\_(self**,** value):  
 self.value = value  
 self.next = None  
  
  
class Stack:  
 *"""Класс, реализующий структуру данных стек."""* def \_\_init\_\_(self):  
 *"""Инициализирует стек с максимальным размером 100 элементов."""* self.\_\_top = None  
 self.\_\_size = **0** self.\_\_max\_size = **100** def push(self**,** value):  
 *"""  
 Добавляет элемент в стек.  
 """* if self.\_\_size >= self.\_\_max\_size:  
 return "error: stack overflow"  
 new\_node = StackNode(value)  
 new\_node.next = self.\_\_top  
 self.\_\_top = new\_node  
 self.\_\_size += **1** return "ok"  
  
 def pop(self):  
 *"""  
 Удаляет и возвращает верхний элемент стека.  
 """* if self.\_\_top is None:  
 return "error: stack is empty"  
 value = self.\_\_top.value  
 self.\_\_top = self.\_\_top.next  
 self.\_\_size -= **1** return str(value)  
  
 def back(self):  
 *"""  
 Возвращает верхний элемент без удаления.  
 """* if self.\_\_top is None:  
 return "error: stack is empty"  
 return str(self.\_\_top.value)  
  
 def size(self):  
 *"""  
 Возвращает количество элементов в стеке.  
 """* return str(self.\_\_size)  
  
 def clear(self):  
 *"""Очищает стек и возвращает подтверждение."""* self.\_\_top = None  
 self.\_\_size = **0** return "ok"  
  
  
def display\_menu():  
 print("\nМеню команд:")  
 print("1. Добавить число в стек (push)")  
 print("2. Удалить и получить верхний элемент (pop)")  
 print("3. Посмотреть верхний элемент (back)")  
 print("4. Узнать размер стека (size)")  
 print("5. Очистить стек (clear)")  
 print("6. Показать это меню (help)")  
 print("0. Выйти из программы (exit)")  
 print("\nВведите цифру команды:")  
  
  
def main():  
 stack = Stack()  
 print("Добро пожаловать в программу 'Стек'!")  
 display\_menu()  
  
 while True:  
 try:  
 user\_input = input("\nВведи номер команды-> ").strip()  
  
 if not user\_input:  
 continue  
  
 if user\_input == "1": # push  
 try:  
 value = int(input("Введите число для добавления: "))  
 print(stack.push(value))  
 except ValueError:  
 print("Ошибка: введите целое число")  
  
 elif user\_input == "2": # pop  
 print(stack.pop())  
  
 elif user\_input == "3": # back  
 print(stack.back())  
  
 elif user\_input == "4": # size  
 print(stack.size())  
  
 elif user\_input == "5": # clear  
 print(stack.clear())  
  
 elif user\_input == "6": # help  
 display\_menu()  
  
 elif user\_input == "0": # exit  
 print("bye")  
 break  
  
 else:  
 print("Неизвестная команда. Введите цифру от 0 до 6")  
  
 except KeyboardInterrupt:  
 print("\nПрограмма завершена")  
 break  
 except Exception:  
 print("Произошла ошибка. Попробуйте снова")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()