|  |  |  |
| --- | --- | --- |
| 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 11 | | |
|  | | Работу выполнил  студент группы ПМИ-ИТ-7-2024 1 курса  Клешнин Д.В. Фамилия И.О.  «12» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2025 | | |

Оглавление

[Постановка задачи 3](#_Toc201409372)

[Алгоритм решения 5](#_Toc201409373)

[**1.** **Структура баржи:** 5](#_Toc201409374)

[**2.** **Инициализация:** 5](#_Toc201409375)

[**3.** **Обработка операций:** 5](#_Toc201409376)

[**4.** **Погрузка (+ A B):** 5](#_Toc201409377)

[**5.** **Выгрузка (- A B):** 6](#_Toc201409378)

[**6.** **Контроль ошибок:** 6](#_Toc201409379)

[**7.** **Финальная проверка:** 6](#_Toc201409380)

[**8.** **Интерфейс пользователя:** 6](#_Toc201409381)

[Обработка исключений и ООП в программе 7](#_Toc201409382)

[Тестирование 8](#_Toc201409383)

[**1.** **Базовый корректный сценарий:** 8](#_Toc201409384)

[**2.** **Выгрузка из пустого отсека:** 9](#_Toc201409385)

[**3.** **Неверный тип топлива при выгрузке:** 9](#_Toc201409386)

[**4.** **Превышение лимита бочек P:** 9](#_Toc201409387)

[**5.** **Баржа не пуста после всех операций:** 10](#_Toc201409388)

[**6.** **Неверный формат строки:** 10](#_Toc201409389)

[**7.** **Недопустимый номер отсека:** 10](#_Toc201409390)

[**8.** **Несколько отсеков:** 11](#_Toc201409391)

[**9.** **Неверный символ действия:** 11](#_Toc201409392)

[**10.** **Погрузка и выгрузка в разных отсеках:** 11](#_Toc201409393)

[Код программы 12](#_Toc201409394)

# Постановка задачи

Баржа. На барже располагается K грузовых отсеков. В каждый отсек можно поместить некоторое количество бочек с одним из 10 000 видов топлива. Причём извлечь бочку из отсека можно лишь в случае, если все бочки, помещённые в этот отсек после неё, уже были извлечены. Таким образом, в каждый момент времени в каждом непустом отсеке имеется ровно одна бочка, которую можно извлечь, не трогая остальных. Будем называть такие бочки крайними.

Изначально баржа пуста. Затем она последовательно проплывает через N доков, причём в каждом доке на баржу либо погружается бочка с некоторым видом топлива в некоторый отсек, либо выгружается крайняя бочка из некоторого отсека. Однако, если указанный отсек пуст, либо если выгруженная бочка содержит не тот вид топлива, который ожидалось, следует зафиксировать ошибку. Если на баржу оказывается погружено более P бочек или если после прохождения всех доков она не стала пуста, следует также зафиксировать ошибку. От вас требуется либо указать максимальное количество бочек, которые одновременно пребывали на барже либо зафиксировать ошибку.

Входные данные: в первой строке три целых числа N, K и P (1 ≤ N, K, P ≤ 100 000). Далее следует N строк с описанием действия, выполняемого в очередном доке. Если в нём происходит погрузка, то строка имеет вид «+ A B», где A — номер отсека, в который помещается бочка, а B — номер вида топлива в ней. Если же док занимается разгрузкой, то строка имеет вид «- A B», где A — номер отсека, из которого извлекается бочка, а B — номер ожидаемого вида топлива.

Выходные данные: вывести либо одно число, равное искомому максимуму в случае безошибочного прохождения баржой маршрута, либо вывести слово «Error» в противном случае.

Примеры  
Пример:

(входные данные)

6 1 2

+ 1 1

+ 1 2

- 1 2

- 1 1

+ 1 3

- 1 3

(выходные данные)

2

# Алгоритм решения

1. **Структура баржи:**

* Баржа реализована в виде класса Barge.
* Каждому отсеку соответствует стек (list), чтобы реализовать принцип LIFO (последний пришёл — первый ушёл).
* Все отсеки хранятся в списке self.compartments, индексация начинается с 1 (0-й элемент не используется).

1. **Инициализация:**

* При создании экземпляра Barge задаются:
* num\_compartments — число отсеков.
* max\_capacity — максимально допустимое число бочек на борту.
* Ведётся подсчёт:
* total\_barrels — текущее количество бочек.
* max\_barrels — максимальное количество бочек, которое одновременно находилось на барже.

1. **Обработка операций:**

* Каждая строка операции разбирается методом process().
* Проверяется корректность формата (+ A B или - A B) и преобразуются аргументы.
* Производится проверка:
* На существование указанного отсека.
* На корректность действия (+ или -).

1. **Погрузка (+ A B):**

* Бочка типа B помещается в отсек A (в стек).
* Увеличивается счётчик всех бочек.
* Проверяется превышение лимита P.
* Обновляется максимум max\_barrels, если нужно.

1. **Выгрузка (- A B):**

* Проверяется, не пуст ли отсек.
* Сверяется, соответствует ли верхняя бочка ожидаемому типу топлива B.
* Если всё корректно, бочка удаляется и счётчик бочек уменьшается.

1. **Контроль ошибок:**

* При любой ошибке (неправильный формат, пустой отсек, неверный тип топлива, превышение лимита) устанавливается флаг self.error и запоминается сообщение self.error\_message.

1. **Финальная проверка:**

* После выполнения всех операций вызывается is\_empty(), чтобы убедиться, что баржа полностью пуста.
* Если ошибок не возникло, выводится максимальное число бочек, одновременно находившихся на борту.

1. **Интерфейс пользователя:**

* Программа снабжена текстовыми подсказками.
* Пользователь вводит сначала параметры N K P, затем N строк с операциями.
* Выводятся как сообщения об ошибках, так и итоговый успех с результатом.

# Обработка исключений и ООП в программе

1. **Где и почему используется обработка ошибок:**

* В программе не используются стандартные классы исключений Python (try-except), а реализована внутренняя обработка ошибок вручную через флаги self.error и self.error\_message.
* Почему так сделано: Производительность: не создаются объекты исключений при каждом нарушении. Контроль: вся логика ошибок сосредоточена в классе Barge, а не разбросана по разным try-except блокам. Требование задачи: по условию нужно зафиксировать ошибку в логике операции, а не техническую ошибку выполнения кода.
* Исключения (try-except) всё же используются: При чтении и преобразовании входных данных (например, преобразование int(), количество аргументов и т.д.) в функции main().

1. **Классы и объектно-ориентированные принципы:**

* Node — простой узел связного списка:

Хранит значение и ссылку на следующий элемент.

Нужен для реализации стека без использования list.

* Stack — собственная структура данных:

Реализует поведение стека (LIFO) через Node.

Используется вместо списка в каждом отсеке баржи.

Упрощает управление и повышает читаемость, код легче тестировать и расширять.

* Barge — модель всей баржи:

Содержит список отсеков (каждый — Stack).

Хранит общее количество бочек, максимум, ошибки.

Отвечает за бизнес-логику: погрузка, выгрузка, проверка операций.

1. **Инкапсуляция:**

* Переменные состояния (compartments, total\_barrels, error, max\_barrels, error\_message) скрыты внутри класса Barge. Доступ к этим данным осуществляется только через методы load(), unload(), process(), is\_empty().
* Все внутренние данные (top\_node, size, compartments, total\_barrels, error\_message) скрыты в соответствующих классах.

Пользователь работает только через методы (push, pop, load, unload, process), не обращаясь напрямую к структуре данных.

1. **Наследование:**

* В текущей версии наследование не используется, так как достаточно одного класса Barge.

1. **Полиморфизм:**

* Прямого полиморфизма через наследование в коде нет. Но косвенно применяется поведенческий полиморфизм через метод process(): Метод обрабатывает строки вида + A B или - A B и вызывает соответствующее поведение. Один метод — разные действия (погрузка или выгрузка) в зависимости от входа.

# Тестирование

1. **Базовый корректный сценарий:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Выгрузка из пустого отсека:**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Неверный тип топлива при выгрузке:**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Превышение лимита бочек P:**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Баржа не пуста после всех операций:**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Неверный формат строки:**

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Недопустимый номер отсека:**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Несколько отсеков:**

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Неверный символ действия:**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. **Погрузка и выгрузка в разных отсеках:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

# Код программы

|  |
| --- |
| **class** **Node:**  **def** \_\_init\_\_**(**self**,** value**,** next\_node**=None):**  self**.***value* **=** value # значение (тип топлива)  self**.next** **=** next\_node # ссылка на следующий узел (ниже по стеку)  **class** **Stack:**  **def** \_\_init\_\_**(**self**):**  self**.***top\_node* **=** **None** # вершина стека (верхняя бочка)  self**.***size* **=** 0 # количество элементов в стеке  **def** push**(**self**,** value**):**  # Добавление элемента на вершину стека  self**.***top\_node* **=** Node**(**value**,** self**.***top\_node***)**  self**.***size* **+=** 1  **def** pop**(**self**):**  # Удаление верхнего элемента стека  **if** self**.***is\_empty***():**  **raise** **IndexError(**"Стек пуст"**)**  value **=** self**.***top\_node***.***value*  self**.***top\_node* **=** self**.***top\_node***.next**  self**.***size* **-=** 1  **return** value  **def** top**(**self**):**  # Получение значения верхнего элемента  **if** self**.***is\_empty***():**  **raise** **IndexError(**"Стек пуст"**)**  **return** self**.***top\_node***.***value*  **def** is\_empty**(**self**):**  # Проверка, пуст ли стек  **return** self**.***top\_node* **is** **None**  **class** **Barge:**  **def** \_\_init\_\_**(**self**,** num\_compartments**,** max\_capacity**):**  # Инициализация баржи с пустыми отсеками, каждый представлен стеком  self**.***compartments* **=** **[**Stack**()** **for** \_ **in** **range(**num\_compartments **+** 1**)]** # индексация с 1  self**.***capacity* **=** max\_capacity # ёмкость баржи  self**.***total\_barrels* **=** 0 # текущее количество бочек  self**.***max\_barrels* **=** 0 # максимальное количество одновременно находившихся на борту  self**.***error* **=** **False**  self**.***error\_message* **=** ""  **def** load**(**self**,** compartment**,** fuel\_type**):**  # Погрузка бочки в указанный отсек  self**.***compartments***[**compartment**].***push***(**fuel\_type**)**  self**.***total\_barrels* **+=** 1  # Проверка на превышение допустимого количества бочек  **if** self**.***total\_barrels* **>** self**.***capacity***:**  self**.***error* **=** **True**  self**.***error\_message* **=** f"Превышено максимальное количество бочек на барже: {self**.***capacity*}"  self**.***max\_barrels* **=** **max(**self**.***max\_barrels***,** self**.***total\_barrels***)**  **def** unload**(**self**,** compartment**,** fuel\_type**):**  stack **=** self**.***compartments***[**compartment**]**  # Проверка: отсек не должен быть пуст  **if** stack**.***is\_empty***():**  self**.***error* **=** **True**  self**.***error\_message* **=** f"Ошибка: отсек {compartment} пуст, нельзя извлечь бочку."  **return**  # Проверка соответствия типа топлива  **if** stack**.***top***()** **!=** fuel\_type**:**  self**.***error* **=** **True**  self**.***error\_message* **=** **(**  f"Ошибка: ожидался вид топлива {fuel\_type}, но в отсеке {compartment} сверху {stack**.***top***()**}."  **)**  **return**  # Удаление бочки  stack**.***pop***()**  self**.***total\_barrels* **-=** 1  **def** process**(**self**,** operation**,** line\_num**=None):**  # Обработка одной строки операции  parts **=** operation**.***strip***().***split***()**  **if** **len(**parts**)** **!=** 3**:**  self**.***error* **=** **True**  self**.***error\_message* **=** f"Ошибка в строке {line\_num}: неправильный формат команды."  **return**  action**,** a\_str**,** b\_str **=** parts  **if** action **not** **in** **(**'+'**,** '-'**):**  self**.***error* **=** **True**  self**.***error\_message* **=** f"Ошибка в строке {line\_num}: неизвестное действие '{action}'."  **return**  **try:**  compartment **=** **int(**a\_str**)**  fuel\_type **=** **int(**b\_str**)**  **except** **ValueError:**  self**.***error* **=** **True**  self**.***error\_message* **=** f"Ошибка в строке {line\_num}: номер отсека и вид топлива должны быть целыми числами."  **return**  # Проверка допустимости номера отсека  **if** **not** **(**1 **<=** compartment **<** **len(**self**.***compartments***)):**  self**.***error* **=** **True**  self**.***error\_message* **=** f"Ошибка: номер отсека {compartment} вне допустимого диапазона."  **return**  # Выполнение действия  **if** action **==** '+'**:**  self**.***load***(**compartment**,** fuel\_type**)**  **else:**  self**.***unload***(**compartment**,** fuel\_type**)**  **def** is\_empty**(**self**):**  # Проверка: пуста ли баржа (все бочки выгружены)  **return** self**.***total\_barrels* **==** 0  **def** main**():**  # Приветствие и краткая инструкция  **print(**"🚢 Добро пожаловать в симулятор работы баржи."**)**  **print(**"Введите данные в следующем формате:"**)**  **print(**"На первой строке: N K P"**)**  **print(**"Где N — число операций, K — число отсеков, P — максимум бочек."**)**  **print(**"Далее N строк вида '+ A B' или '- A B'\n"**)**  **try:**  # Ввод первой строки с параметрами  n\_k\_p **=** **input(**"Введите значения N K P через пробел: "**).***strip***().***split***()**  **if** **len(**n\_k\_p**)** **!=** 3**:**  **print(**"❌ Ошибка: нужно ввести три целых числа через пробел."**)**  **return**  n**,** k**,** p **=** **map(int,** n\_k\_p**)**  **except** **ValueError:**  **print(**"❌ Ошибка: N, K и P должны быть целыми числами."**)**  **return**  # Проверка допустимых границ  **if** **not** **(**1 **<=** n **<=** 100\_000 **and** 1 **<=** k **<=** 100\_000 **and** 1 **<=** p **<=** 100\_000**):**  **print(**"❌ Ошибка: значения N, K и P должны быть от 1 до 100000."**)**  **return**  barge **=** Barge**(**k**,** p**)**  # Последовательная обработка всех операций  i **=** 0  **print(**f"\nВведите {n} операций:"**)**  **while** i **<** n**:**  op **=** **input(**f"[{i **+** 1}] >>> "**).***strip***()**  barge**.***process***(**op**,** i **+** 1**)**  **if** barge**.***error***:**  **print(**f"❌ {barge**.***error\_message*}"**)**  **print(**"🔁 Повторите ввод этой строки.\n"**)**  barge**.***error* **=** **False**  barge**.***error\_message* **=** ""  **continue** # не увеличиваем счётчик — повтор ввода  **else:**  i **+=** 1  # Проверка на пустоту после маршрута  **if** **not** barge**.***is\_empty***():**  **print(**"\n❌ Баржа не пуста после завершения всех операций."**)**  **return**  # Успешный результат  **print(**f"\n✅ Маршрут завершён без ошибок. Максимум бочек на борту: {barge**.***max\_barrels*}"**)**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  main**()** |