|  |  |  |
| --- | --- | --- |
| 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 «Языки программирования»  Вариант 10 | | |
|  | | Работу выполнил  студент группы ПМИ-10,2023-2024 1 курса  Казаков Н.С Фамилия И.О.  «16» июня 2024 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

Постановка задачи………………………………………………………………………..3

[Алгоритм решения 3](#_Toc153130028)

Тестирование…………………………………………………………………………….5

Код программы…………………………………………………………………………..5

# Постановка задачи

Очередь. Реализуйте структуру данных "очередь". Напишите программу, содержащую описание очереди и моделирующую работу очереди, 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.

# Алгоритм решения

## Инициализация очереди

* Создаем класс Queue, который будет представлять собой очередь.
* В конструкторе класса (\_\_init\_\_) инициализируем пустой список self.queue, который будет использоваться для хранения элементов очереди.
* Список выбран в качестве структуры данных для очереди, так как он позволяет легко добавлять элементы в конец и удалять из начала с приемлемой производительностью.

Операции с очередью

* Метод push: добавляет элемент в конец очереди.
  + Принимает параметр n (элемент, который нужно добавить) и добавляет его в конец списка self.queue.
  + Возвращает строку "ок" для подтверждения успешного добавления.
* Метод pop: удаляет и возвращает первый элемент очереди.
  + Проверяет, пуста ли очередь, с помощью метода is\_empty.
  + Если очередь пуста, возвращает "ошибка".
  + Иначе удаляет и возвращает первый элемент списка self.queue.
* Метод front: возвращает первый элемент очереди без удаления.
  + Проверяет, пуста ли очередь, с помощью метода is\_empty.
  + Если очередь пуста, возвращает "ошибка".
  + Иначе возвращает первый элемент списка self.queue.
* Метод size: возвращает количество элементов в очереди.
  + Возвращает длину списка self.queue.
* Метод clear: очищает очередь.
  + Очищает список self.queue.
  + Возвращает "ок" для подтверждения успешной очистки.
* Метод exit: завершает работу программы.
  + Возвращает строку "пока" для подтверждения завершения.
* Метод is\_empty: проверяет, пуста ли очередь.
  + Возвращает True, если длина списка self.queue равна 0, иначе False.

## Обработка команд

* Создаем функцию process\_command, которая принимает на вход объект очереди queue и строку команды command.
* Функция обрабатывает различные команды:
  + push: запрашивает у пользователя ввод числа для добавления в очередь, вызывает метод push и возвращает его результат.
  + pop: вызывает метод pop и возвращает его результат.
  + front: вызывает метод front и возвращает его результат.
  + size: вызывает метод size и возвращает его результат.
  + clear: вызывает метод clear и возвращает его результат.
  + exit: вызывает метод exit и возвращает его результат.
  + Если команда не распознана, возвращает "неизвестная команда".

## Главная функция

* Создаем главную функцию main, которая:
  + Выводит информацию о доступных командах.
  + В бесконечном цикле запрашивает у пользователя ввод команды.
  + Вызывает функцию process\_command для обработки команды и выводит результат.
  + Завершает цикл при вводе команды exit.

## Точка входа в программу

* Определяем блок if \_\_name\_\_ == "\_\_main\_\_", чтобы запускать главную функцию main при выполнении скрипта.

# Тестирование

# Изображение выглядит как текст, снимок экрана, Шрифт, документ Автоматически созданное описание

# Код программы

class Queue:

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

# Инициализация очереди

self.queue = []

def push(self, n):

# Добавление элемента в очередь

self.queue.append(n)

return "ok"

def pop(self):

# Удаление первого элемента из очереди

if self.is\_empty():

return "error"

return self.queue.pop(0)

def front(self):

# Получение первого элемента очереди без удаления

if self.is\_empty():

return "error"

return self.queue[0]

def size(self):

# Возвращение размера очереди

return len(self.queue)

def clear(self):

# Очистка очереди

self.queue.clear()

return "ok"

def exit(self):

# Завершение работы программы

return "bye"

def is\_empty(self):

# Проверка, пуста ли очередь

return len(self.queue) == 0

def process\_command(queue, command):

# Обработка команд

if command == "push":

# Обработка команды push

try:

n = int(input("Введите число для добавления в очередь: "))

return queue.push(n)

except ValueError:

return "ошибка ввода"

elif command == "pop":

# Обработка команды pop

return queue.pop()

elif command == "front":

# Обработка команды front

return queue.front()

elif command == "size":

# Обработка команды size

return queue.size()

elif command == "clear":

# Обработка команды clear

return queue.clear()

elif command == "exit":

# Обработка команды exit

return queue.exit()

else:

# Неизвестная команда

return "неизвестная команда"

def main():

# Вывод информации о командах

print("Доступные команды:")

print("push - добавить элемент в очередь. После ввода этой команды введите число для добавления.")

print("pop - удалить первый элемент из очереди.")

print("front - вывести первый элемент очереди.")

print("size - вывести размер очереди.")

print("clear - очистить очередь.")

print("exit - завершить выполнение программы.")

print()

# Главная функция для запуска программы

queue = Queue()

while True:

# Запрос команды от пользователя

command = input("Введите команду (push, pop, front, size, clear, exit): ")

result = process\_command(queue, command)

print(result)

if command == "exit":

break

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

# Точка входа в программу

main()