Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Выполнил студент группы КС-36 Акулинин Андрей Игоревич

Ссылка на репозиторий: <https://github.com/MUCTR-IKT-CPP/AkulininAI_36/tree/main/algorithms>

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 10.03.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

Целью лабораторной работы было изучение и реализация двух версий очереди: на основе односвязного списка и на основе двух стеков. Очередь — это структура данных, работающая по принципу FIFO (First In, First Out), где элементы добавляются в конец и извлекаются из начала.

Задачи работы:

1. Реализовать очередь на основе односвязного списка.
2. Реализовать очередь на основе двух стеков.
3. Обеспечить поддержку итераторов для обхода элементов очереди.
4. Провести тестирование реализованных структур:
   * Заполнение очереди целыми числами и вычисление их суммы, среднего, минимального и максимального значений.
   * Проверка работы со строковыми элементами.
   * Работа с пользовательскими структурами (фильтрация по возрасту).
   * Инверсия содержимого очереди.
5. Сравнить производительность двух реализаций на основе операций вставки и извлечения.

# Описание метода/модели.

**Очередь на основе односвязного списка**

Односвязный список — это структура данных, состоящая из узлов, каждый из которых содержит данные и указатель на следующий узел. Очередь на основе списка реализуется с использованием двух указателей: head (начало очереди) и tail (конец очереди).

* **Операции:**
  + push: Добавление элемента в конец очереди. Сложность: O(1).
  + pop: Удаление элемента из начала очереди. Сложность: O(1).
  + front: Получение элемента из начала очереди. Сложность: O(1).
  + isEmpty: Проверка на пустоту. Сложность: O(1).
  + Итератор: Позволяет обходить элементы очереди. Сложность обхода: O(n).
* **Преимущества:**
  + Простота реализации.
  + Эффективные операции добавления и удаления.
* **Недостатки:**
  + Требуется дополнительная память для хранения указателей.

**Очередь на основе двух стеков**

Очередь реализуется с использованием двух стеков: stack\_in для добавления элементов и stack\_out для извлечения. Когда stack\_out пуст, элементы из stack\_in переносятся в stack\_out.

* **Операции:**
  + push: Добавление элемента в stack\_in. Сложность: O(1).
  + pop: Если stack\_out пуст, перенос элементов из stack\_in в stack\_out (O(n) в худшем случае), затем удаление из stack\_out. Сложность: O(1) в среднем.
  + front: Аналогично pop. Сложность: O(1) в среднем.
  + isEmpty: Проверка на пустоту обоих стеков. Сложность: O(1).
  + Итератор: Реализован через копирование стеков. Сложность обхода: O(n).
* **Преимущества:**
  + Использование стандартной структуры данных (стек).
  + Эффективность операций в среднем случае.
* **Недостатки:**
  + В худшем случае операция pop требует O(n) времени.

# Выполнение задачи.

**Язык программирования и организация программы**

Для реализации был выбран язык C++. Программа состоит из следующих компонентов:

1. **Классы:**
   * Queue: Очередь на основе односвязного списка.
   * Queue2: Очередь на основе двух стеков.
   * Итераторы для обоих классов.
2. **Тесты:**
   * test1: Работа с целыми числами (заполнение, вычисление суммы, среднего, минимума и максимума).
   * test2: Работа со строками.
   * test3: Работа с пользовательскими структурами (фильтрация по возрасту).
   * inversion: Инверсия содержимого очереди.
   * compare\_performance: Сравнение производительности двух реализаций.

**Результаты тестирования**

1. **Тест 1 (целые числа):**
   * Обе реализации корректно работают с числами.
   * Результаты вычислений (сумма, среднее, минимум, максимум) совпадают для обеих очередей.
2. **Тест 2 (строки):**
   * Очереди корректно работают со строковыми элементами.
   * Операции push и pop выполняются без ошибок.
3. **Тест 3 (пользовательские структуры):**
   * Фильтрация по возрасту выполнена корректно.
   * Количество элементов, подходящих под условия, совпадает для обеих реализаций.
4. **Инверсия:**
   * Обеспечена корректная инверсия содержимого очереди без использования дополнительных операций перемещения.
5. **Сравнение производительности:**
   * Очередь на основе двух стеков показала себя быстрее в операциях push и pop на больших объемах данных (1000000 элементов).
   * Очередь на основе списка потребляет меньше памяти, но работает медленнее.

# Заключение.

В ходе выполнения лабораторной работы были изучены и реализованы две версии очереди: на основе односвязного списка и на основе двух стеков. Обе реализации показали свою применимость для различных задач, однако имеют свои преимущества и недостатки.

* **Очередь на основе списка** проще в реализации и требует меньше памяти, но менее эффективна при больших объемах данных.
* **Очередь на основе двух стеков** демонстрирует высокую производительность в среднем случае, но требует больше операций в худшем случае.

Реализация итераторов позволила обеспечить удобный обход элементов очереди и интеграцию с стандартными библиотеками. Тестирование подтвердило корректность работы обеих реализаций.

**Выводы:**

1. Очередь на основе списка подходит для задач с небольшим объемом данных.
2. Очередь на основе двух стеков предпочтительна для задач с большим количеством операций.
3. Реализация итераторов упрощает работу с очередью и делает код более универсальным.

Работа позволила глубже понять особенности реализации и применения очередей, а также оценить их производительность в различных условиях.