МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ЛАБОРАТОРНАЯ РАБОТА №3**

«Объектно-ориентированное программирование »

**по дисциплине: «Программирование»**

|  |  |
| --- | --- |
| Выполнил:  *Студент гр. «АБ-321», «АВТФ»*  *Морозов Даниил Иванович*  «23» декабря 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Старший преподаватель кафедры ЗИ*  *Киселев Антон Анатольевич*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

****Цели и задачи работы:** изучение основных принципов объектно- ориентированного программирования и основ юнит-тестирования.**

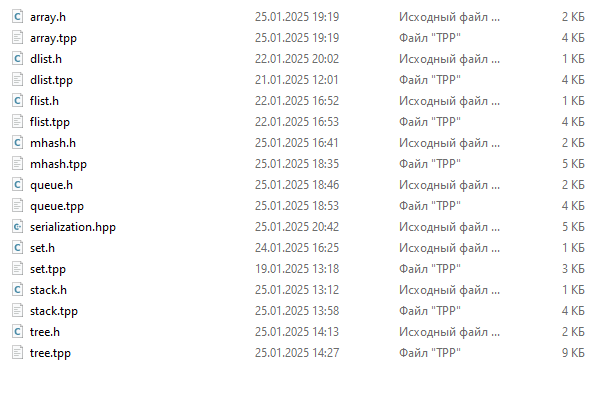
****Задание к работе:** Самостоятельно решить задачи в соответствии с индивидуальным вариантом.**

****Задание 1.****

**Реализовать классы с базовым набором операций (private, public) на основе лабораторной работы 1 на одном из множества языков X ∈ {Rust, Go, C + +}:**

1. **Массив. Операции: добавление элемента (по индексу и в конец массива), получение элемента по индексу, удаление элемента по индексу, замена элемента по индексу, длина массива, чтение.**
2. **Список (односвязный, двусвязный). Операции: добавление элемента в голову и в хвост, удаление элемента с головы и с хвоста, удаление элемента по значению, поиск элемента по значению, чтение.**
3. **Очередь. Операции: добавление и удаление элемента (push и pop), чтение.**
4. **Стек. Операции: добавление и удаление элемента (push и pop), чтение.**
5. **Хеш таблицы. Операции: предусмотреть работу с коллизиями добавление элемента (ключ-значение), получение значения элемента по ключу, удаление элемента по ключу.**
6. **Деревья (Вариант 2 Сбалансированное двоичное дерево (АВЛ-дерево), вариант 3 Full Binary Tree). Операции: добавление элемента, поиск элемента, удаление элемента с сохранением структуры дерева, чтение.**

**Данные структуры данных были реализованы мною на C++ в двух предыдущих работах. Структуры данных шаблонные, определяются в соответствующих .h файлах, реализованы в .tpp файлах, хранятся в папке libs.**

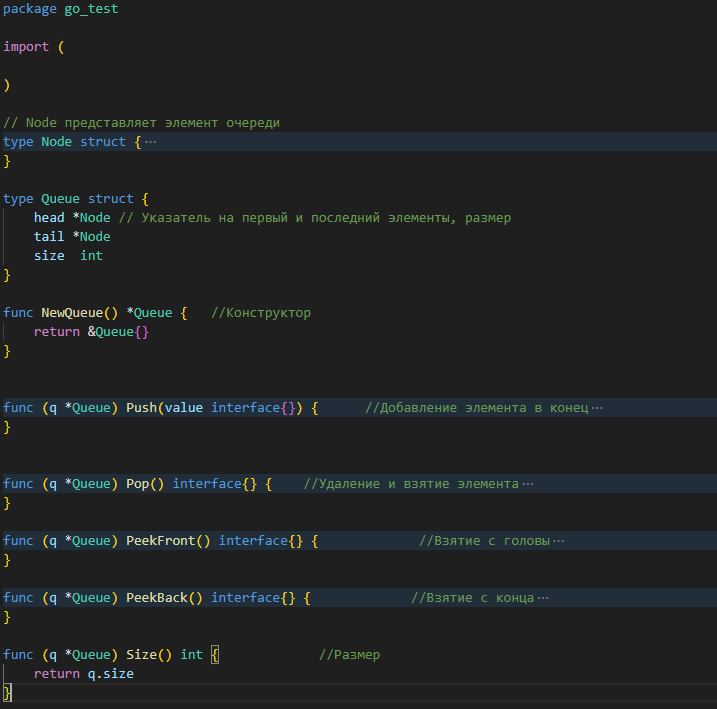
**Рис.1. Папка libs, реализации структур данных.**

****Задание 2.****

**Реализовать класс с базовым набором операций (private, public) на основе лабораторной работы 1 на одном из множества языков Y ∈ {Rust, Go, C +}/𝑋.**

**Вариант 2 – Очередь**

**Мною была реализована очередь на Go в самом обычном варианте, она содержит однонаправленные узлы, удаление идёт с головы, вставка идёт в конец очереди.**

Рис.2. Реализация очереди в go.

****Задание 3.****

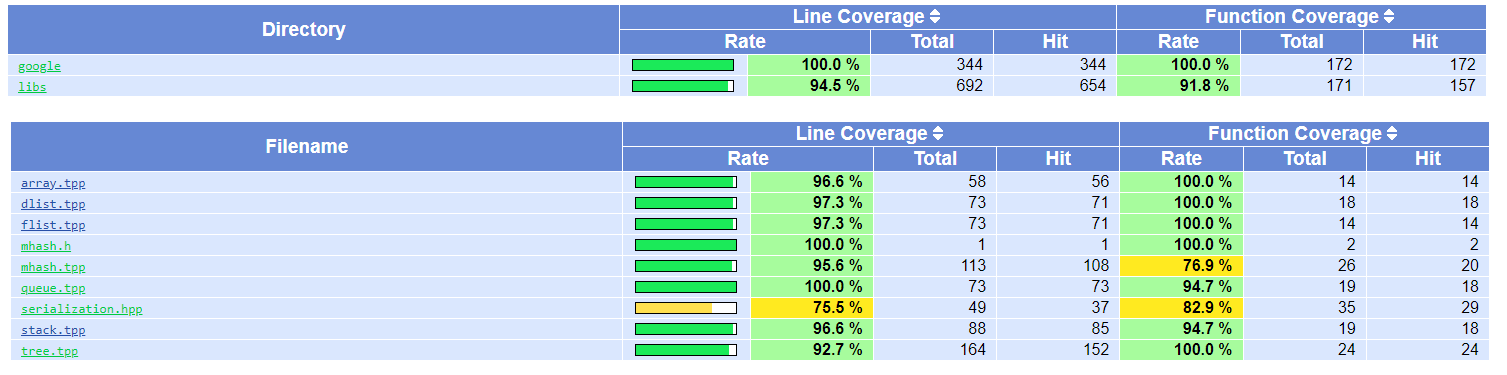
**Реализовать покрытие тестами (не менее 90%) в проектах 1 и 2. Benchmark использование обязательно.**

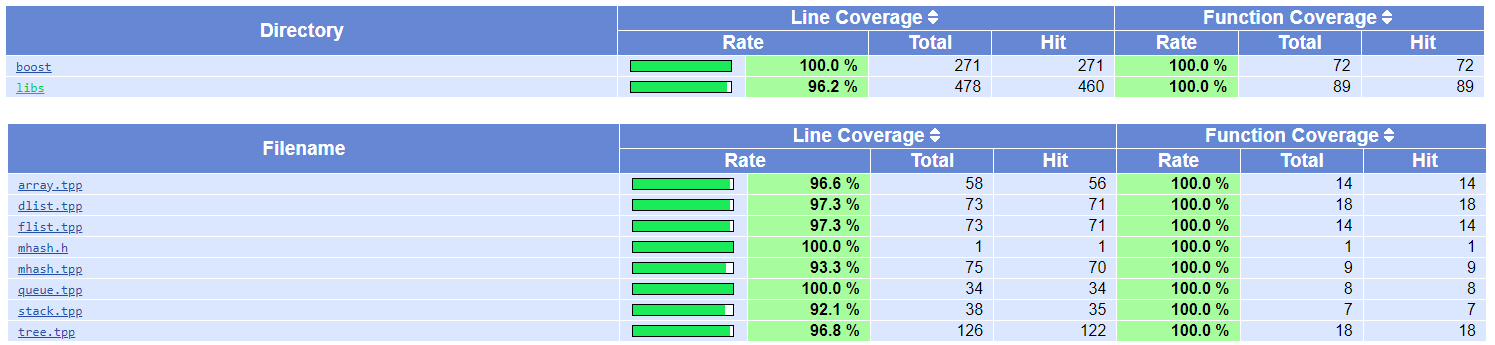
1. **С++: Boost Test Library && Google test. Создать HTML-отчет о покрытии, который предоставляет визуальный анализ того, как много кода было протестировано. Обязательно использовать make файл.**
2. **Go: testing && testify. Создать HTML-отчет о покрытии, который предоставляет визуальный анализ того, как много кода было протестировано.**

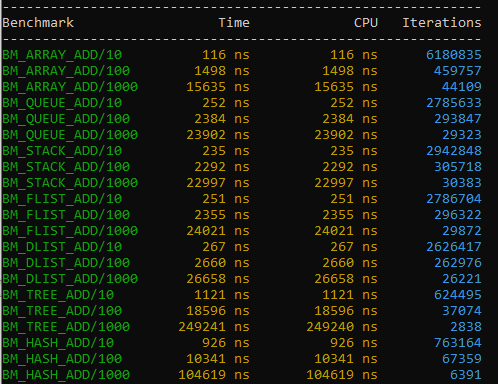
**В C++ очень просто подключить необходимые библиотеки при помощи cmake, просто нужно указать ссылки на github с необходимыми модулями связать их.**

**В директории C++ содержатся 3 папки: google, boost и benchmark, эти папки отвечают за гугл тесты, буст тесты и бенчмаркинг соответственно, в них расположены соответствующие cpp файлы тестов или замеров, CmakeLists.txt и папки build, в которых собирается проект.**

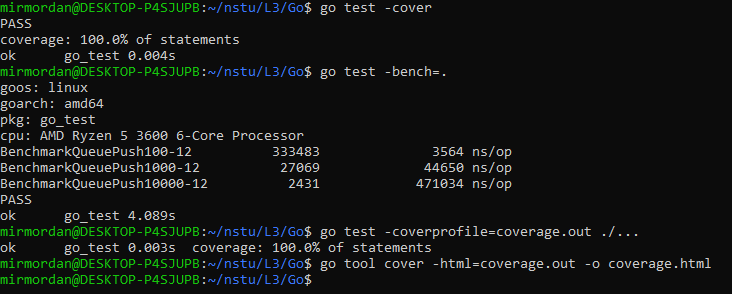
**Гугл тесты показались мне более удобными, потому что являются самостоятельной библиотекой без зависимостей, буст тесты же требуют рекурсивной установки зависимостей (я скачиваю boost целиком)**

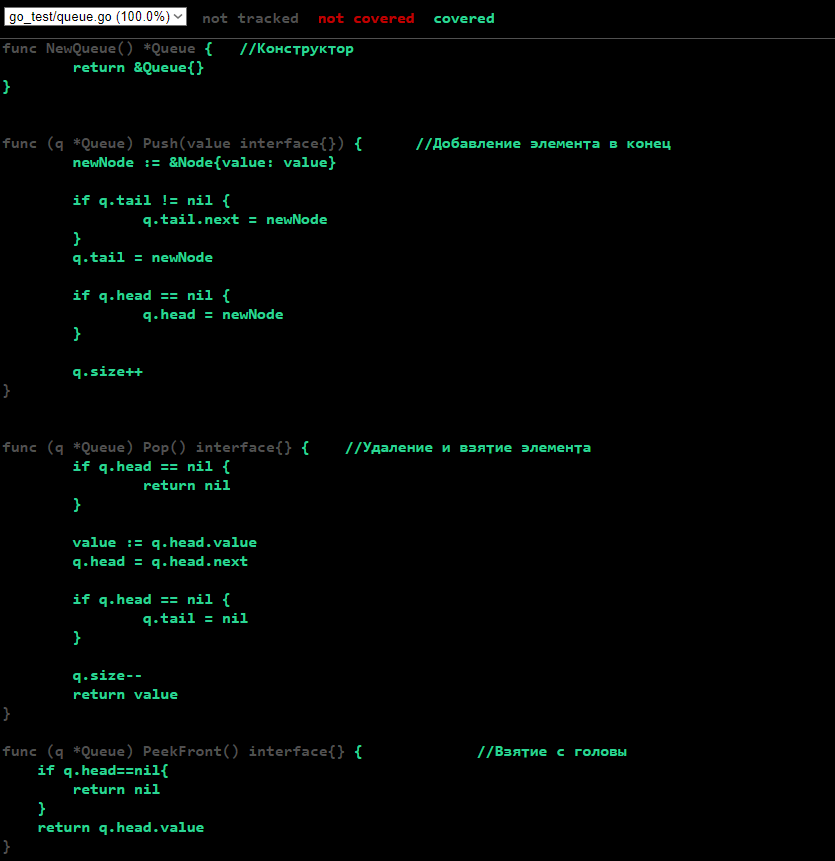
Рис.3. Покрытие тестами google test (serialization не в счёт)

Рис.4. Покрытие тестами boost test ( до внедрения сериализации)

Рис.5. Бенчмаркинг

В Go библиотека testing является встроенной, testify необходимо скачать с гитхаба. Testing обеспечивает минимальный функционал тестирования (функция вывода сообщения об ошибке, создание функций тестирования, бенчмаркинг), testify даёт более расширенный функционал тестирования, аналогичный тестам из C++, например утверждения и моки.

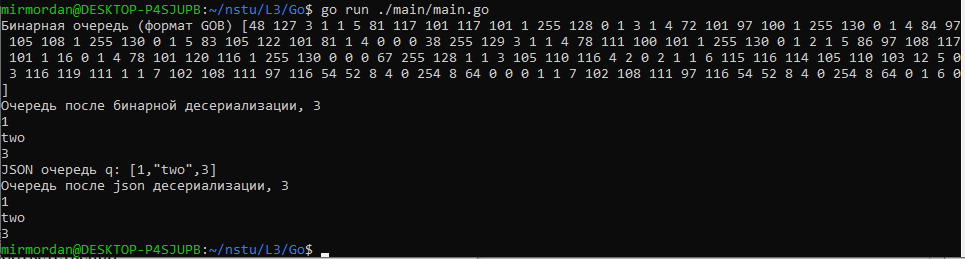
Рис.6. Покрытие и бенчмаркинг

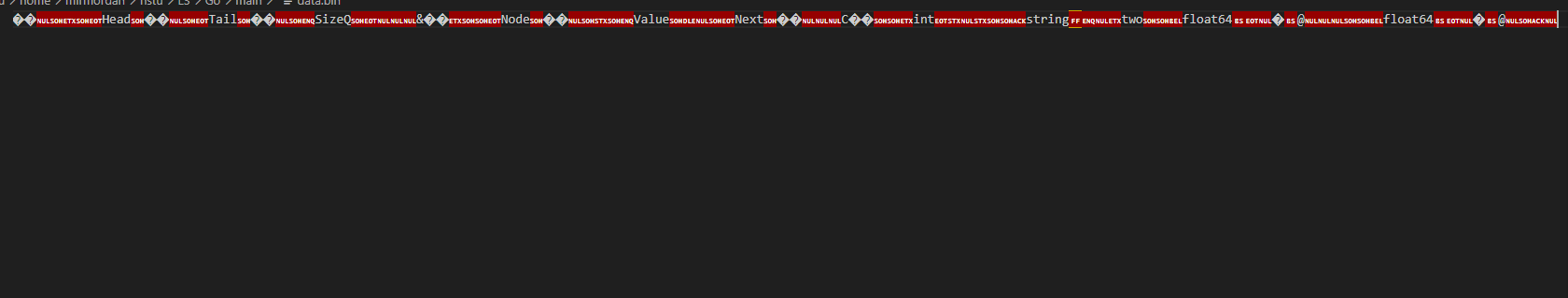
Рис.7. Отчёт о покрытии

****Задание 4.****

**Выполнить сериализацию и десериализацию данных бинарного и текстового форматов для проектов 1 и 2 соответственно.**

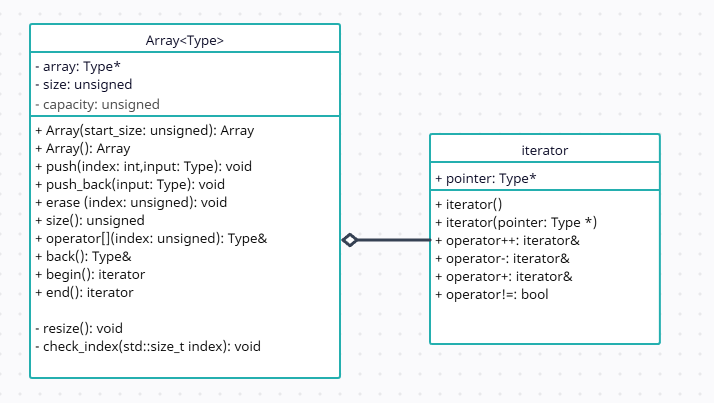
**В go сериализация упрощается использованием встроенных библиотек, json сереализация делается как оформление данных, когда как бинарная сериализация в GOB формат (специальный бинарный формат go) полностью записывает структуру.**

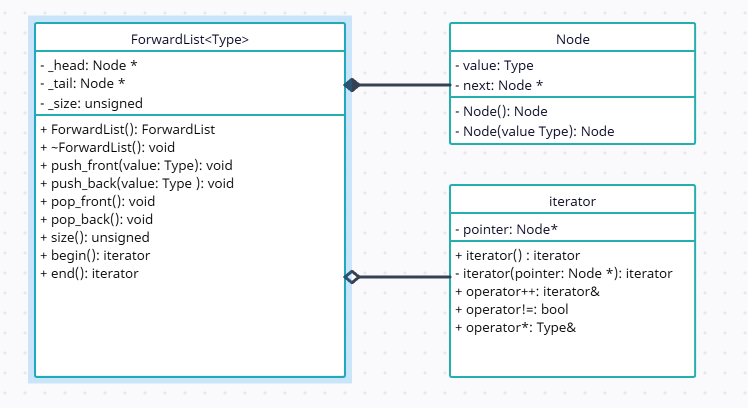
**Рис.10. Сереализация очереди.**

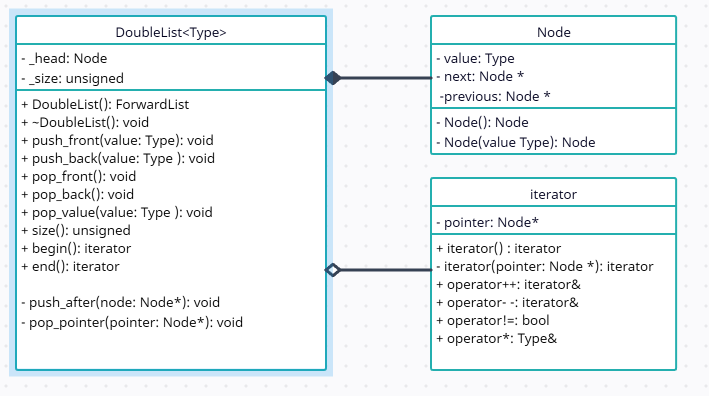
Рис.11. Очередь в GoB формате.

****Задание 5.****

**В отчете представить UML диаграммы.**

**Рис.12. UML диаграмма динамического массива.**

Рис.13. UML диаграмма односвязного списка

Рис.14. UML двусвязного списка

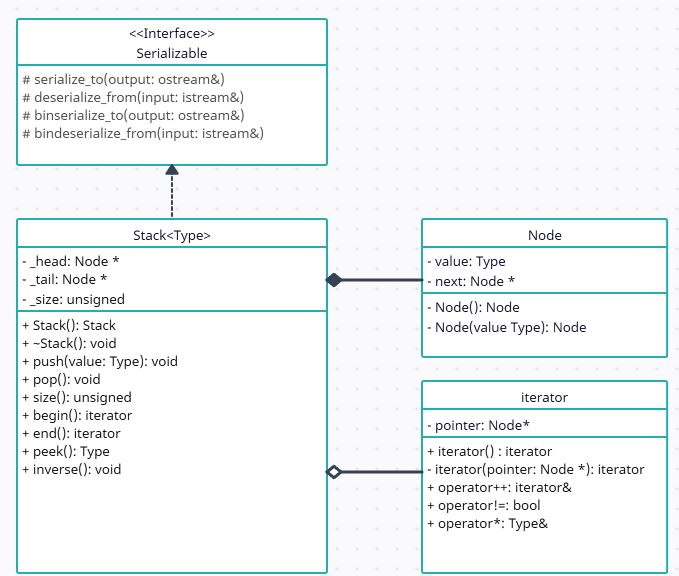
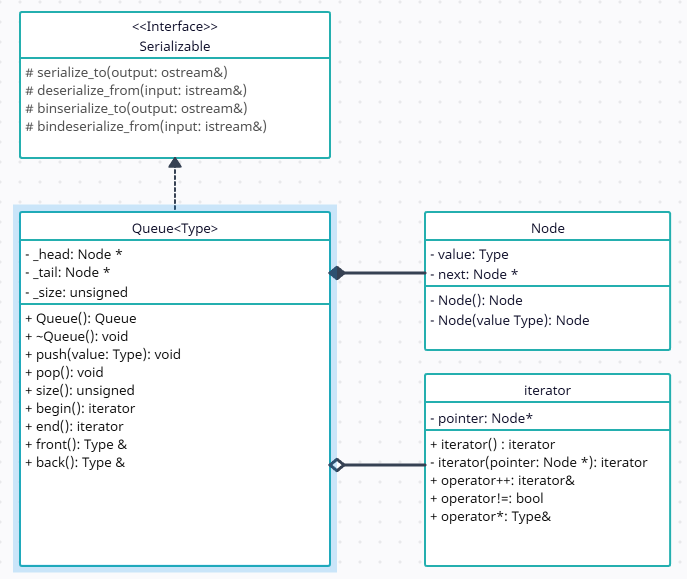


Рис.15.UML Стека

Рис.16.UML очереди

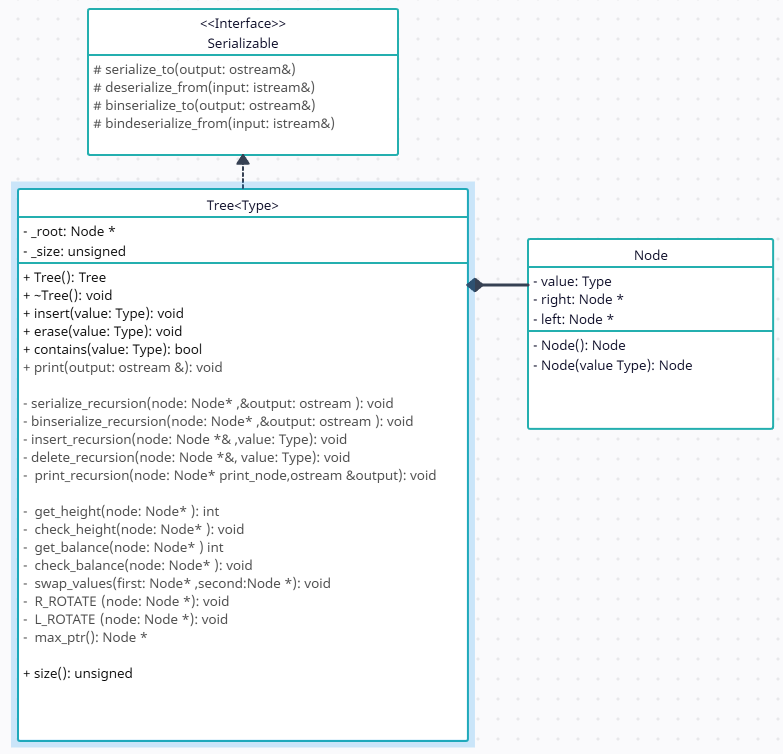
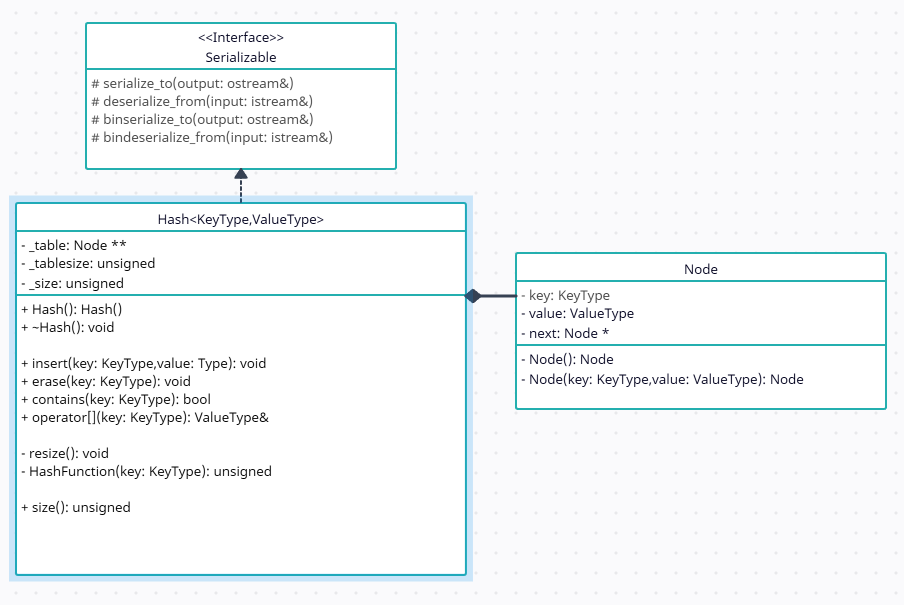


Рис.17. UML дерева

Рис.18. UML хеш-таблицы

**Задание 6.**

фвфвф