МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Информатика и системы управления»

ОТЧЕТ

**Лабораторная работа №1**

По дисциплине: «Технологии разработки ПО»

Тема: «Работа над тестированиеми и дополнением компьютерного кода с использованием системы Git»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чернобаев И.Д.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дубков И.А.

Группа М23-ИВТ-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2024

Цель работы:

Файлы stack.c и stack.h являются простой реализацией стека, хранящего целые числа. В коде есть баги и ошибки.

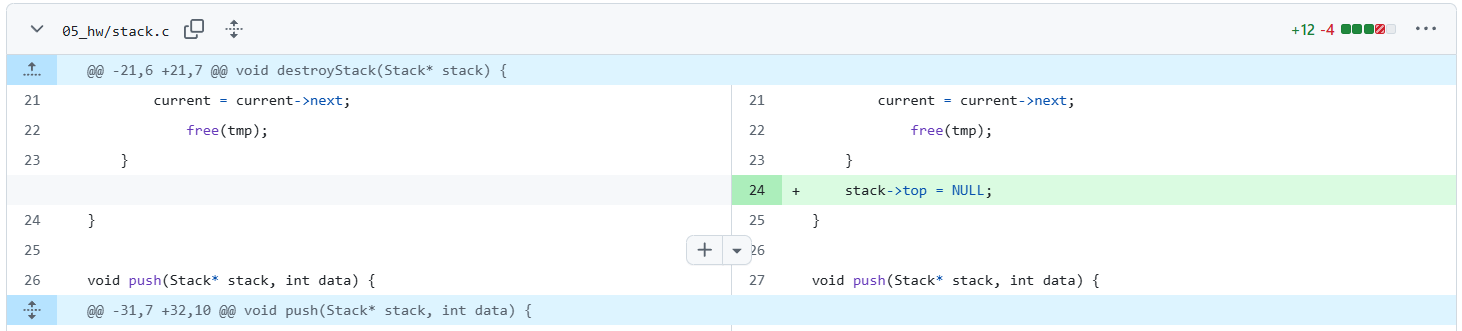
Необходимо привести код в порядок.

* Исправить ошибки;
* Покрыть код тестами;
* Проверить покрытие кода тестами (убедиться, что все функции и условия протестированы);
* Выбрать статический анализатор и запустить его (исправить ошибки, если будут);
* Запустить тесты с санитайзерами (исправить ошибки, если будут);
* Для базовых функций (push / pop) написать бенчмарки;
* Изучить возможности Github Actions и настроить CI для тестирования (\*).

Ход работы:

**Исправление ошибок**

В начале работы были исправлены ошибки в методах: pop (исправлена утечка памяти), isEmpty (убрали лишний код), searchbyValue (избегаем бесконечного цикла), destroyStaks (при удалении стека делаем новый топ NULL).



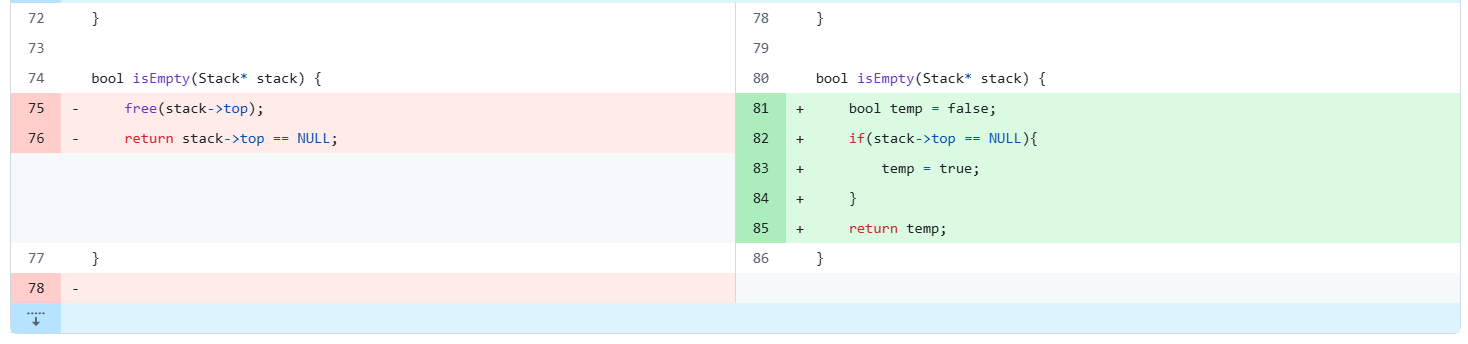
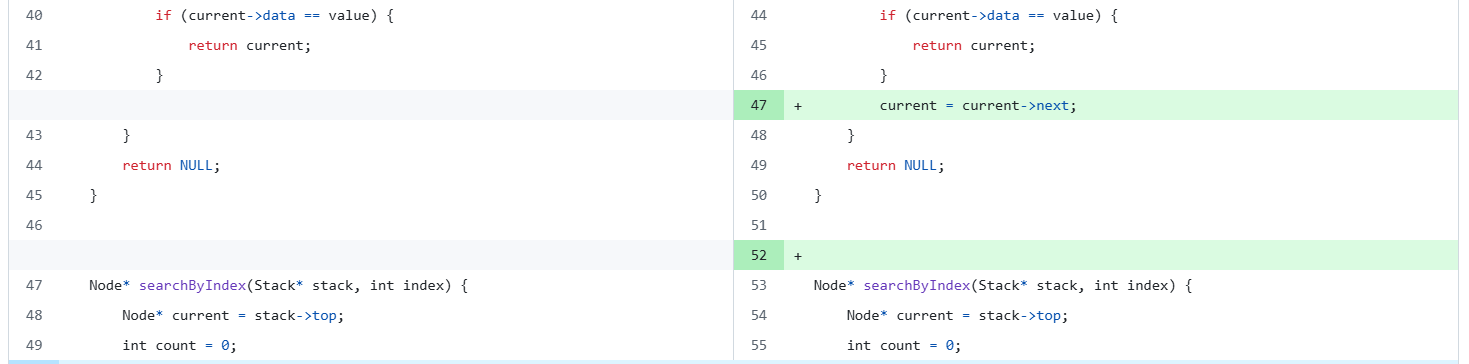
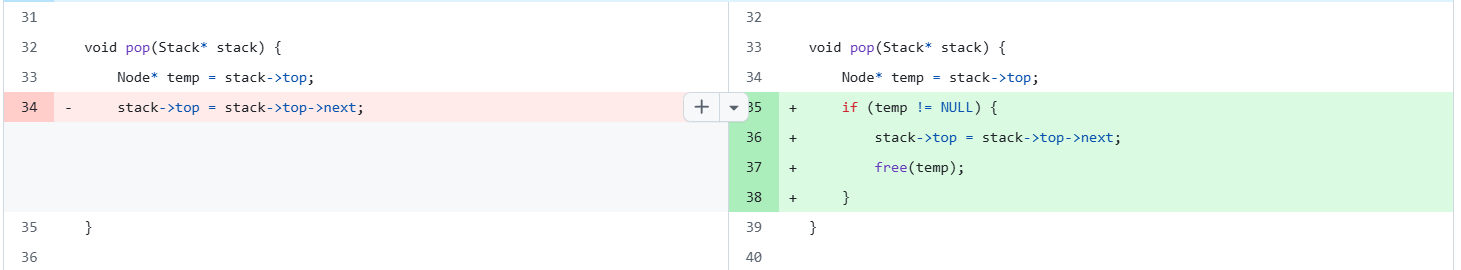


Рисунок 1 – сравнение изменений в файлах

**Написание и прохождение тестов**

* В ходе работы мы написали набор тестов для каждого метода класса Stack, где мы проверяли их работоспособность (см.Рисунок 2).

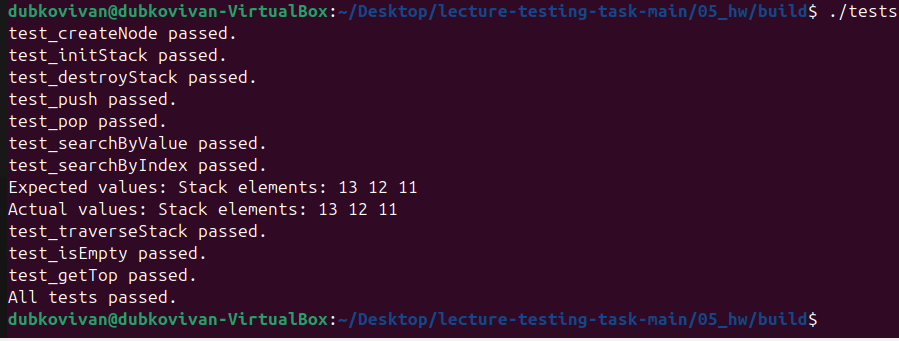


Рисунок 2 – результат тестирования

Далее было проверено покрытие код тестами

Для этого был выбран комплекс gcov, где мы снача встраиваем строки для запуска анализа в наш CMakeLists.txt, затем запускаем файлы, затем смотрим на собранную статистику (см. рисунки 3-4)

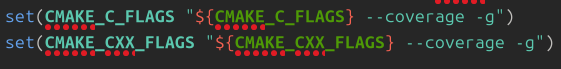
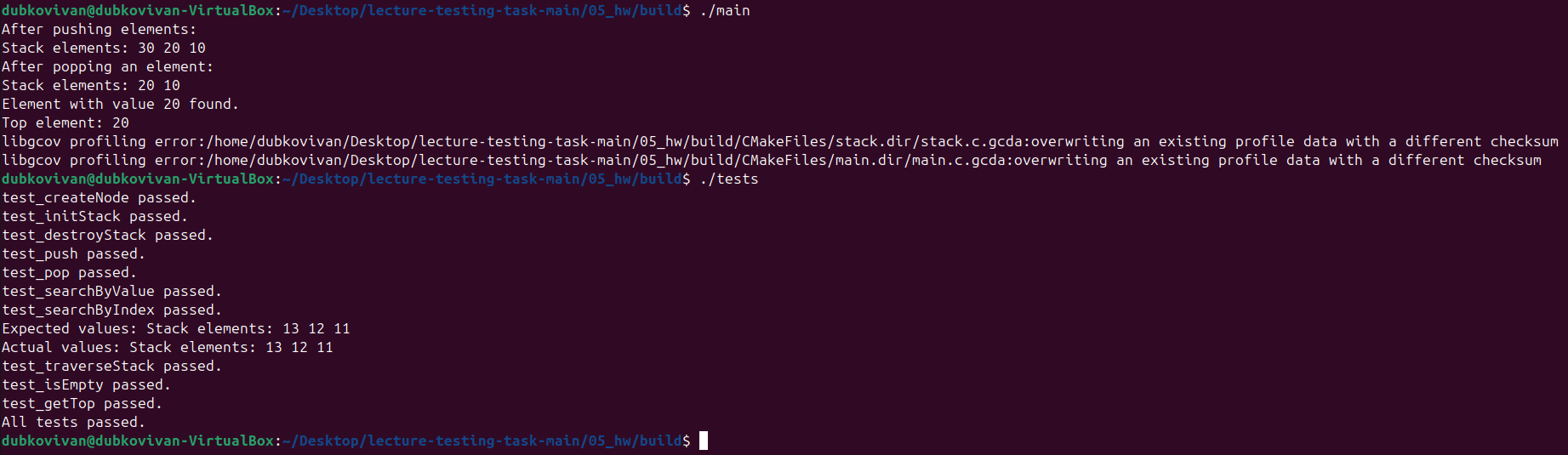
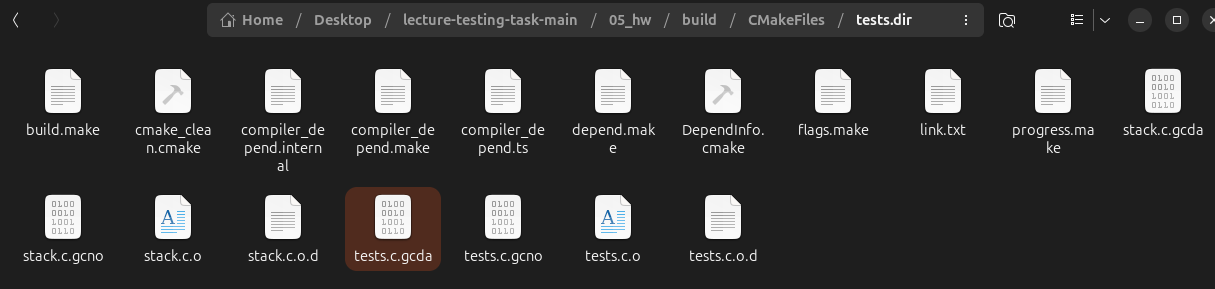


Рисунок 3 – команды для запуска анализа





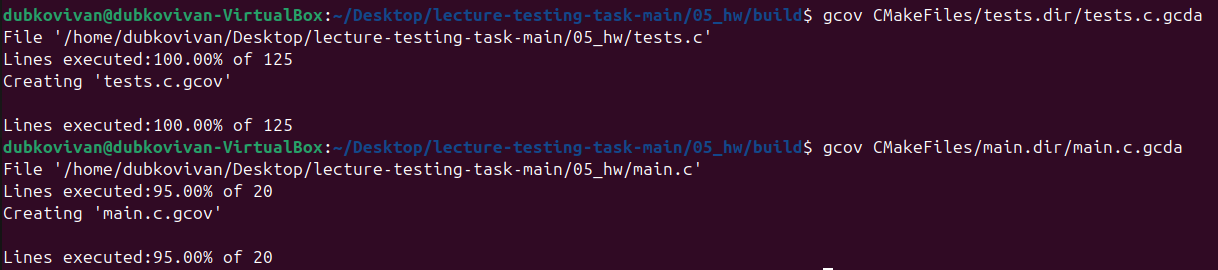
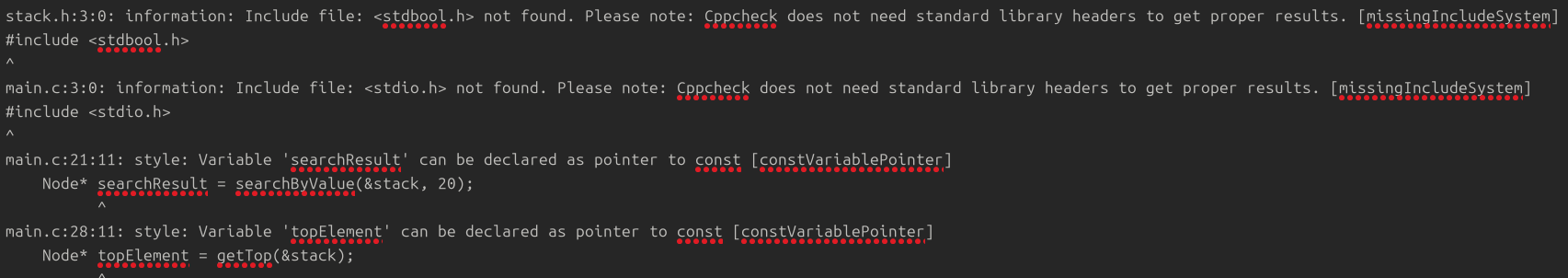


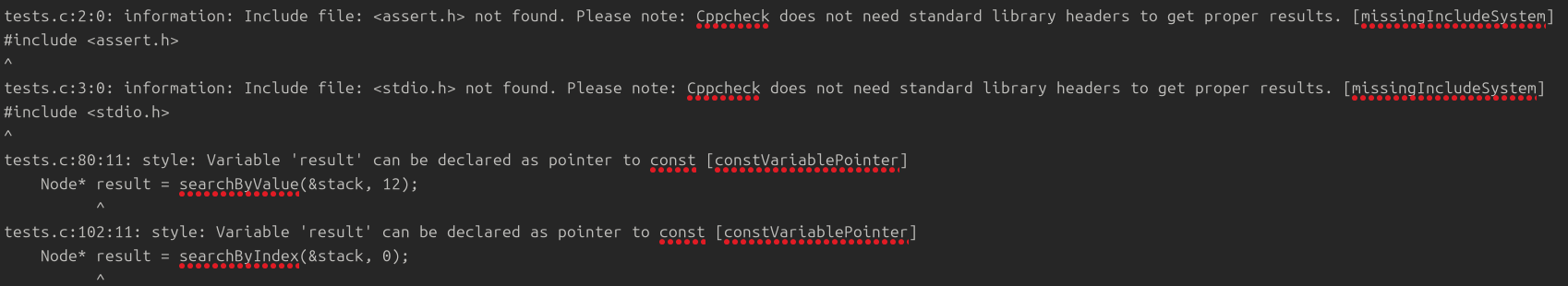
Рисунок 4 – запуск анализа и полученные результаты

Как видим, в процессе работы программы строки кода в файле tests.c были использованы на 100%, что означает, что все функции стека были успешно протестированы.

**Проверка статическим анализатором**

* Для проверки был использован анализатор Cppcheck. Он был запущен на файлах main.c, stack.c и tests.c. В результате его работы были получены следующие данные (см. Рисунок 5):





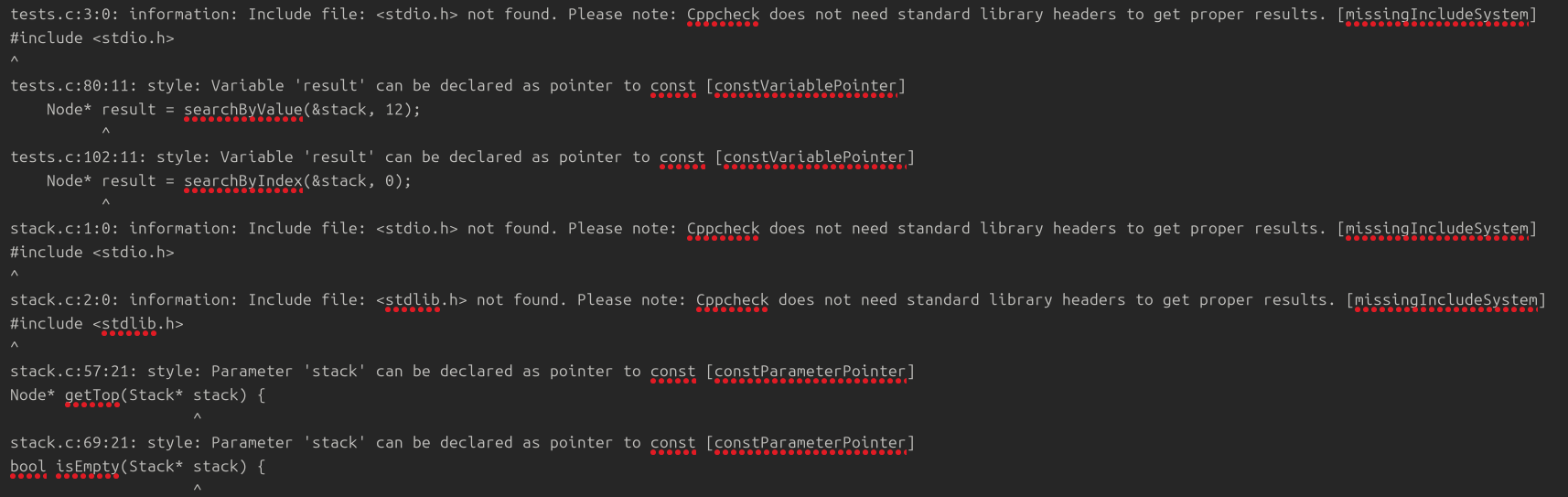


Рисунок 5 – результат работы статического анализатора

Все выявленные ошибки были связаны либо с обращением к библиотекам, что cppcheck не понимает, либо объявлением указателей, что было решено проигнорировать как некритичное уточнение.

Таким образом, код был успешно проверен при помощи статического анализатора.

**Запуск теста с санитайзером**

* Для проверки был использован санитайзер address sanitizer. Он был запущен на файлах main.c, stack.c и tests.c. В результате его работы были получены следующие данные (см. Рисунок 6):

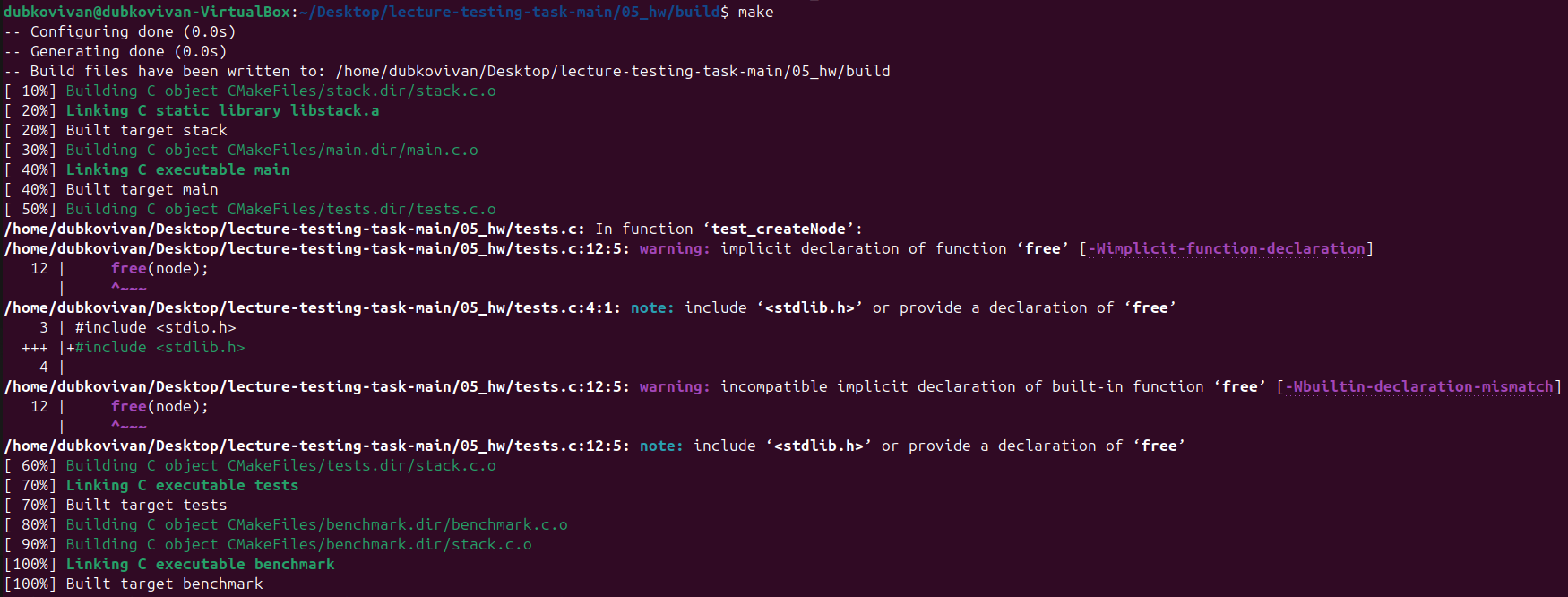


Рисунок 6 – результат работы address sanitizer.

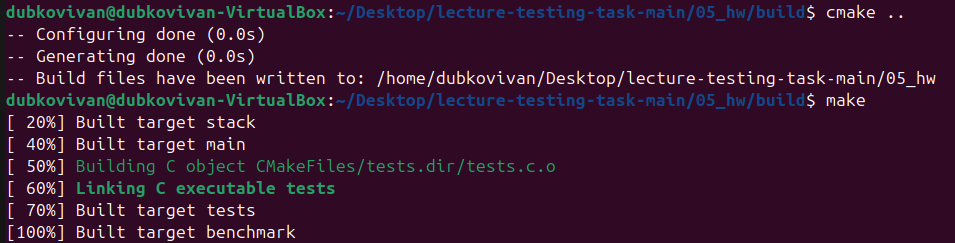
Оказалось, что встроенная функция free() требовала объявления или подключения библиотеки stdlib.h. После добавления данной библиотеки в файле stack.c проверка санитайзера выглядела следующим образом:  


Рисунок 7 – результат работы address sanitizer после исправлений.

Как мы можем наблюдать, использование санитайзера позволило исправить помарку при работе программы.

**Написание бенчмарков для pop и push**

* Были реализованы тесты замера скорости выполнения функций поп и пуш, и получены замеры на количество выполненных итераций функцией (см. Рисунок 8):

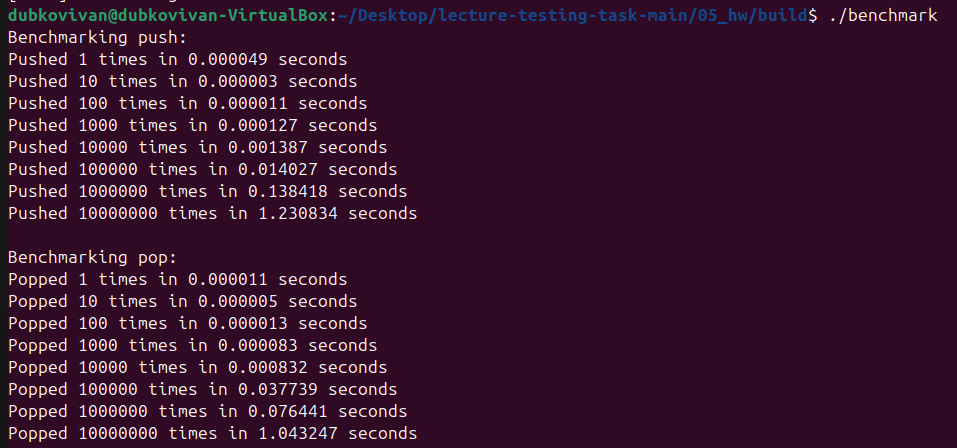


Рисунок 8 – результат работы benchmark алгоритмов.

Как мы можем заметить, Push функция возрастает линейно по затратам времени на количество операций. Функция Pop менее линейно, но все еще с заметной тенденцией к линейному возрастанию. Заметим, что в экстремальных случаях система ведет себя странно, что и следовало ожидать ввиду принципов работы системы ПК и ее внутренних оптимизаций.

Вывод:

Написание данной программы помогло мне понять основные принципы и направления работы с тестированием программ на языках высокого уровня. Был получен опыт использования статических анализаторов и санитайзеров. Были получены и освежены навыки работы с репозиториями git и github.