# Российский университет дружбы народов

### Факультет физико-математических и естественных наук

## Отчёт по лабораторной работе №14

### Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

**Дисциплина:** Операционные системы

**Студент:** Оразгелдиева Огулнур

**Группа:** НПИбд-02-20

**Студ. номер:** 1032205431

2021, Москва

**Лабораторная работа №14**

**Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux**

**Цель:**

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями

**Задачи:**

1. Ознакомиться с теоретическим материалом
2. Создать в каталоге ~/work/os/lab\_prog файлы calculate.h, calculate.c, main.c
3. Написать программу калькулятора, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan.
4. Выполнить компиляцию программы посредством gcc
5. Создать Makefile
6. Выполнить отладку программы calcul с помощью gdb
7. Проанализировать коды файлов calculate.c и main.c

**Теоретические сведения**

Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

– планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;

– проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;

– непосредственная разработка приложения:

– кодирование — по сути создание исходного текста программы (возможно в нескольких вариантах);

– анализ разработанного кода;

– сборка, компиляция и разработка исполняемого модуля;

– тестирование и отладка, сохранение произведённых изменений;

– документирование.

Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др.

После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (С, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C — как файлы на языке C++, а файлы c расширением .o считаются объектными. [[1]](https://esystem.rudn.ru/pluginfile.php/1142386/mod_resource/content/2/011-lab_prog.pdf)

Отладчиком называется программа, которая выполняет внутри себя другую программу. Основное назначение отладчика - дать возможность пользователю в определенной степени осуществлять контроль за выполняемой программой, то есть определять, что происходит в процессе ее выполнения. Наиболее известным отладчиком для Linux является программа GNU GDB. GDB содержит множество полезных возможностей, но для простой отладки достаточно использовать лишь некоторые из них. [[2]](https://www.opennet.ru/docs/RUS/linux_base/node199.html)

GDB — переносимый отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования, включая Си, C++, Free Pascal, FreeBASIC, Ada, Фортран, Python3, Swift, NASM и Rust.

Для того чтобы нам пройтись по такому файлу нам нужно скомпилировать его с помощью G++ с использованием флага -g (это действительно важно, без этого флага, программа не будет корректно работать в GDB). [[3]](https://habr.com/ru/post/491534/)

**Ход работы:**

1. В домашнем каталоге создадим подкаталог *~/work/os/lab\_prog*. (см. рис. 1)

Для этого используем команду создания каталогов *mkdir*

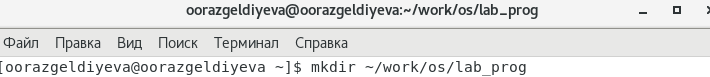


Рисунок 1. Создание каталога ~/work/os/lab\_prog

1. Перейдём в созданный подкаталог и создадим там файлы calculate.h, calculate.c, main.c. (см. рис. 2)

Чтобы перейти в каталог используем команду *cd*, и при помощи команды создания файлов *touch* создадим указанные файлы.

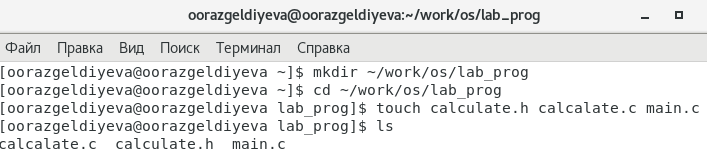


Рисунок 2. Создание файлов

1. В файле calculate.c пишем следующую программу: (см. рис. 3-5)

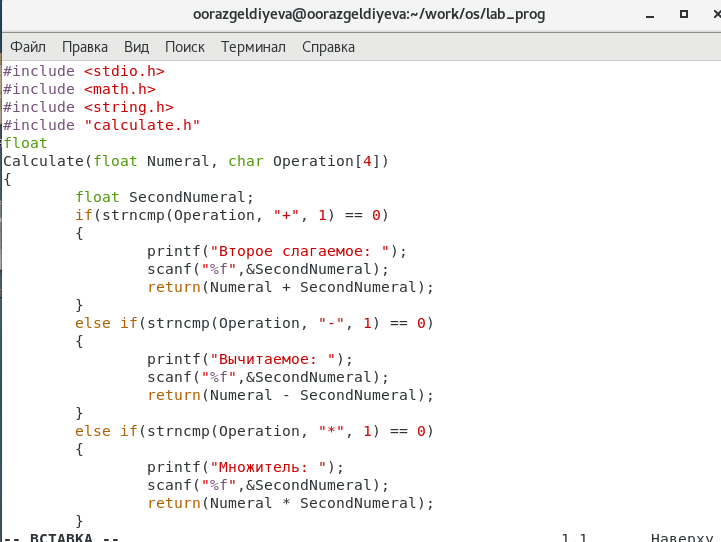


Рисунок 3. calculate.c

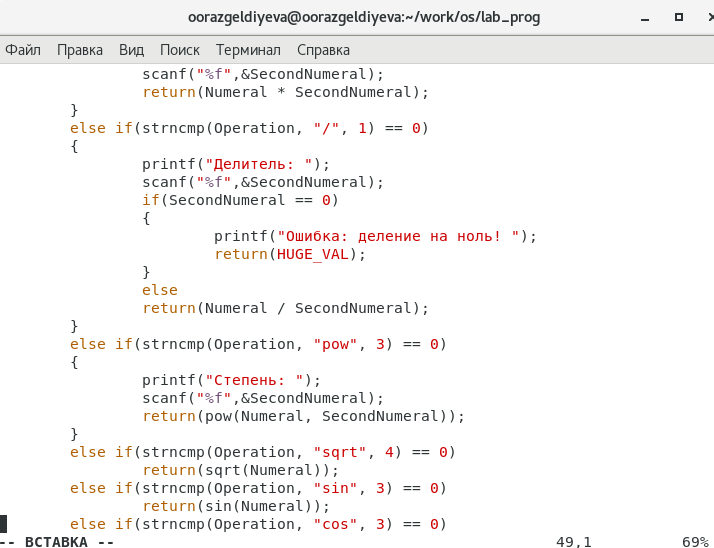


Рисунок 4. calculate.c

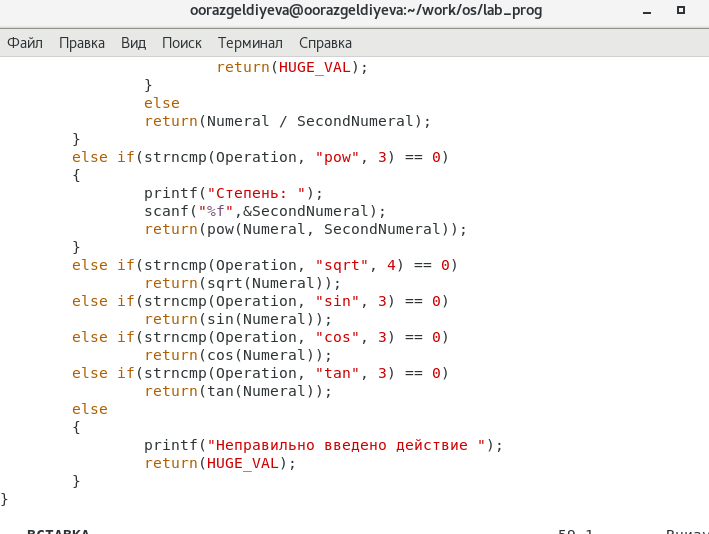


Рисунок 5. calculate.c

**Пояснения:** задается функция Calculate типа вещественный с вещественным аргументом Numeral и символьный аргумент размером 4 Operation. Эта функция работает следующим образом: при вводе операции + выводится запрос на ввод второго слагаемого, и функция выдаёт сумму введенных чисел. Аналогично с операциями -, \*, /, и тд.

Внутри файла calculate.h пишем следующий текст: (см. рис. 6)

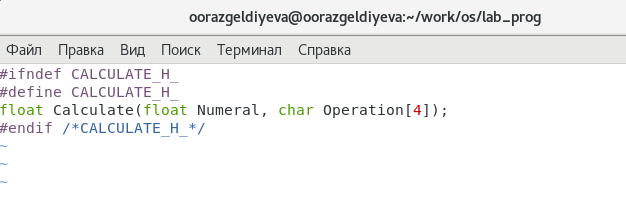


Рисунок 6. calculate.h

В main.c запишем основную функцию программы, (см. рис. 7) которая выводит сообщение о вводе числа и операции, читает введенные переменные и выполняет функцию, находящуюся в файле calculate.c.

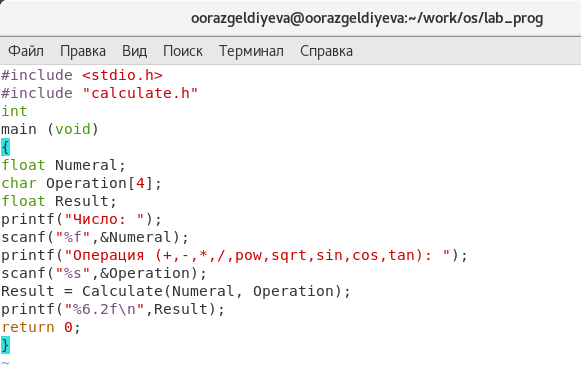


Рисунок 7. main.c

1. Выполнила компиляцию программы посредством *gcc*: (см. рис. 8)

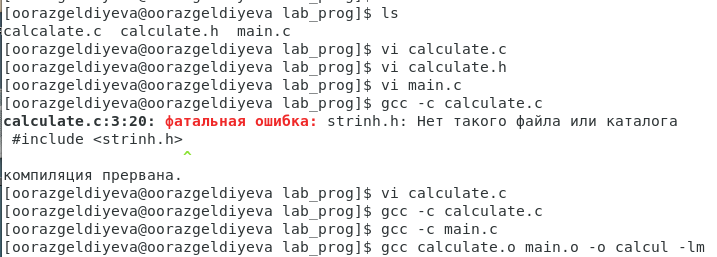


Рисунок 8. Компиляция программы

1. При помощи редактора vi создала и открыла файл Makefile. Записала в него следующий текст по образцу. (см. рис. 9)

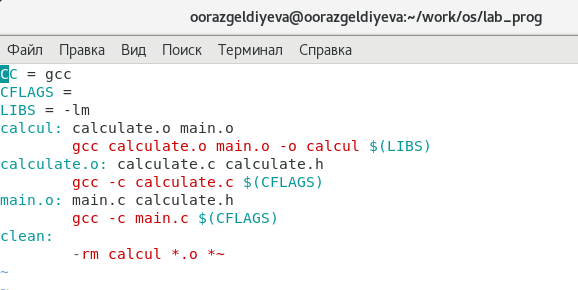


Рисунок 9. Makefile

Makefile содержит переменные СС, CFLAGS и LIBS, а также цели calcul, calculate.o, main.o и clean, которые в свою очередь содержать команды компиляции и удалении файлов.

Исправим содержимое Makefile. (см. рис. 10)

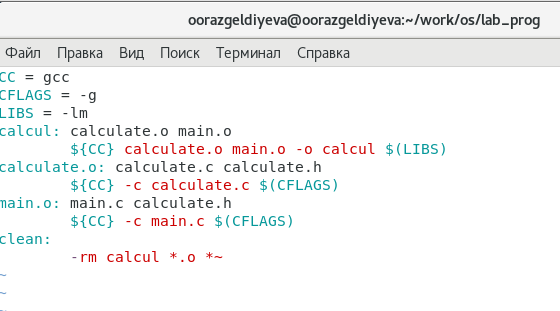


Рисунок 10. Makefile

Так как задана переменная СС, которая содержит gcc, можем поменять gcc на эту переменную. Чтобы отладчик работал исправно в переменную CFLAGS вносим -g.

С помощью команды make можем выполнить цели из Makefile. (см. рис. 11)

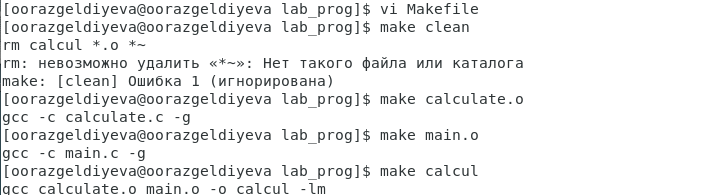


Рисунок 11. Выполнение Makefile

Таким образом, скомпилировали файлы.

1. С помощью gdb выполните отладку программы calcul.

* Для этого в командной строке вводим *gdb ./calcul*. (см. рис. 12)

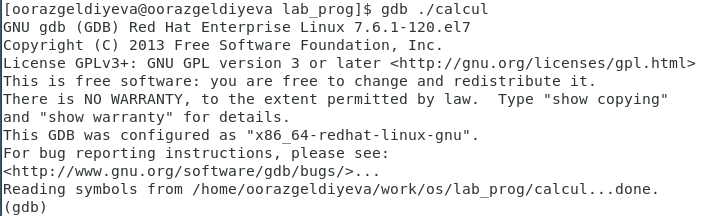


Рисунок 12. Отладчик gdb

В результате этого вывелось сообщение с информацией об отладчике.

* Для запуска программы внутри отладчика вводим команду run. (см. рис. 13)

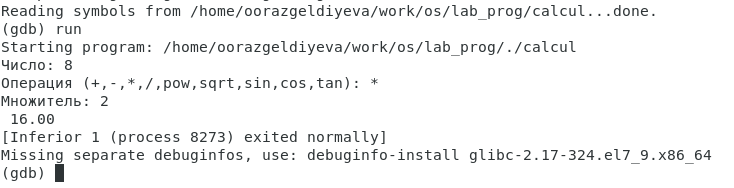


Рисунок 13. Запуск программы

Вводим для примера 8, \*, 2. В результате выполнилось умножение 8 на 2, ответ: 16.

* Для постраничного (по 9 строк) просмотра исходного кода используем команду list: (см. рис. 14)

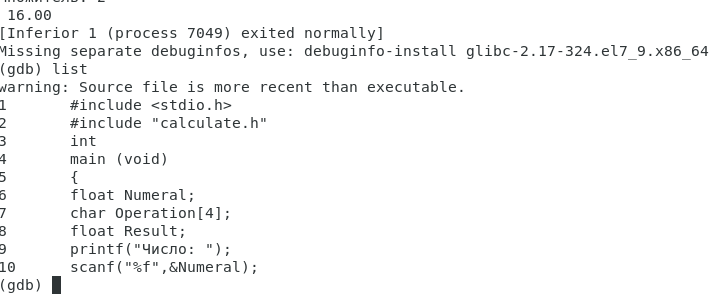


Рисунок 14. Постраничный просмотр кода

* Для просмотра строк с 12 по 15 основного файла используем list с параметрами: (см. рис. 15)

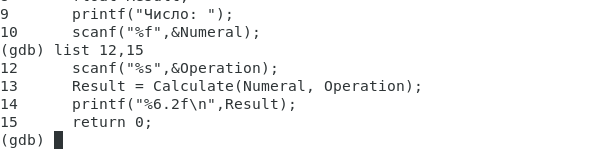


Рисунок 15. Просмотр строк с 12 по 15

* Для просмотра определённых строк не основного файла используем list с параметрами: (см. рис. 16)

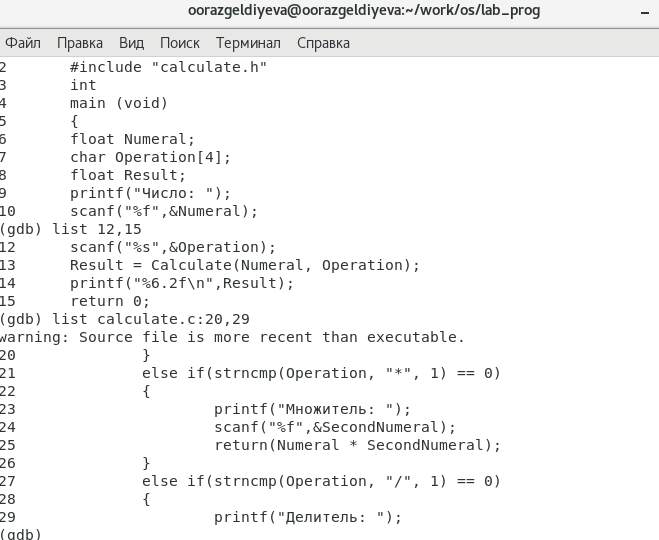


Рисунок 16. Просмотр определённых строк не основного файла

Таким образом, просмотрели строки с 20 по 29 файла calculate.c.

* Установим точку останова в файле calculate.c на строке номер 21.

Для этого используем команды list и break. (см. рис. 17)

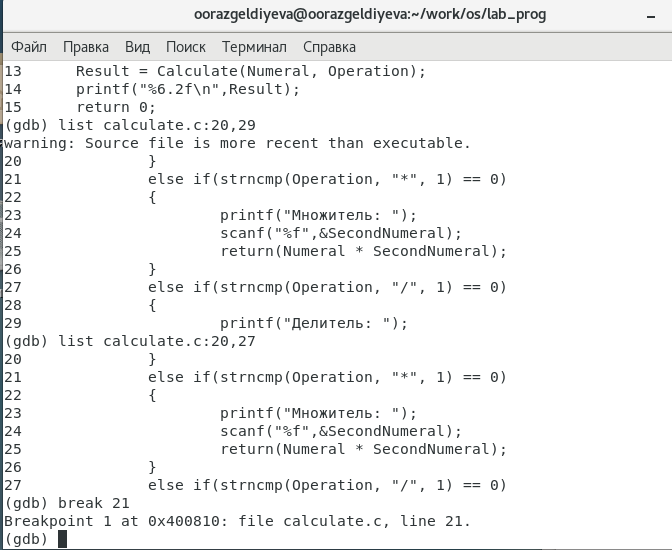


Рисунок 17. Установление точки останова

Как видим, 21 строка - это строка с операцией умножения.

* Выведем информацию об имеющихся в проекте точках останова, используя *info breakpoints*: (см. рис. 18)

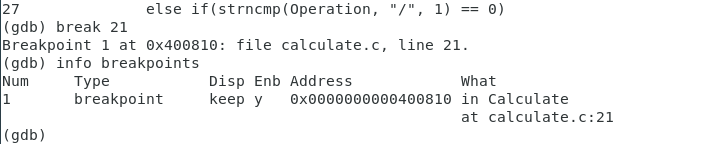


Рисунок 18. Точки останова

Как видно из рис. 18, точка останова на 21 строке файла calculate.c функции Calculate.

* Запустим программу внутри отладчика. используя команду run, и убедимся, что программа остановится в момент прохождения точки останова.

Чтобы сделать это, вводим, например. число 5 и операцию умножения, так как точка останова именно на этой строке.

Вывелось сообщение о том, что установлена точка останова. (см. рис. 19)

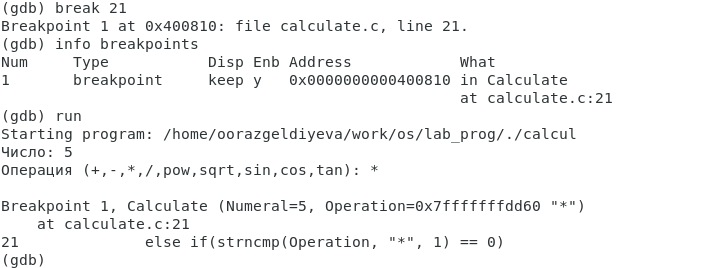


Рисунок 19. Запуск программы

Вводим backtrace, отладчик выводим следующее сообщение: (см. рис. 20)

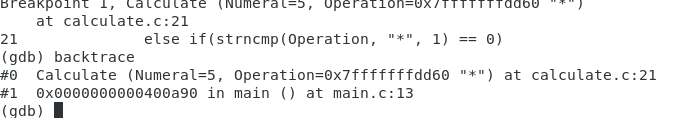


Рисунок 20. backtrace

Команда backtrace показывает весь стек вызываемых функций от начала программы до текущего места.

Посмотрим, чему равно на этом этапе значение переменной Numeral, введя: *print Numeral*. (см. рис. 21)

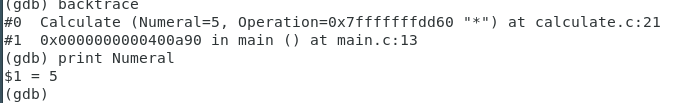


Рисунок 21. Значение переменной Numeral

Сравним с результатом вывода на экран после использования команды: *display Numeral*. (см. рис. 22)

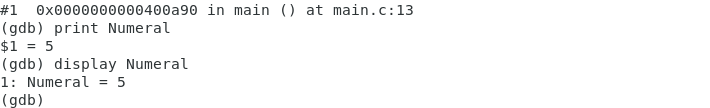


Рисунок 22. Значение переменной Numeral

В первом случае вывелось только значение переменной, а во втором случае можно увидеть и название самой переменной.

* Уберём точки останова,используя команду *delete*. (см. рис. 23)

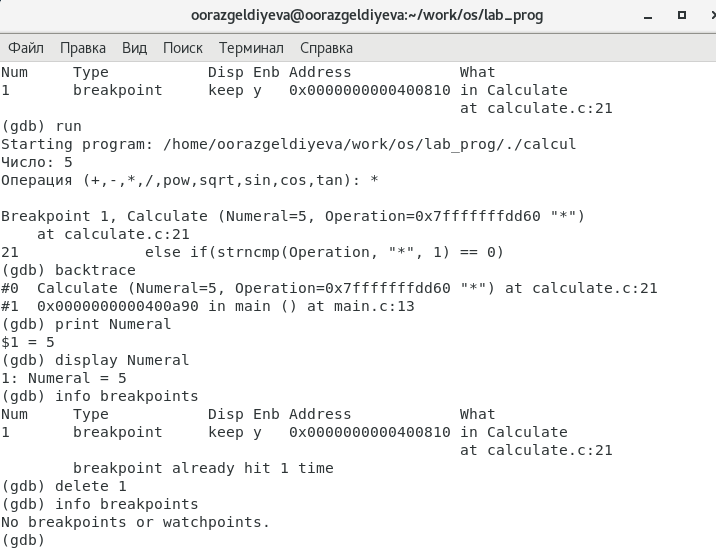


Рисунок 23. Удаление точек останова

Когда еще раз ввели команду *info breakpoints*, вывелось сообщение о том, что точек останова нет.

1. С помощью утилиты splint попробуем проанализировать коды файлов calculate.c. (см. рис. 24-25)

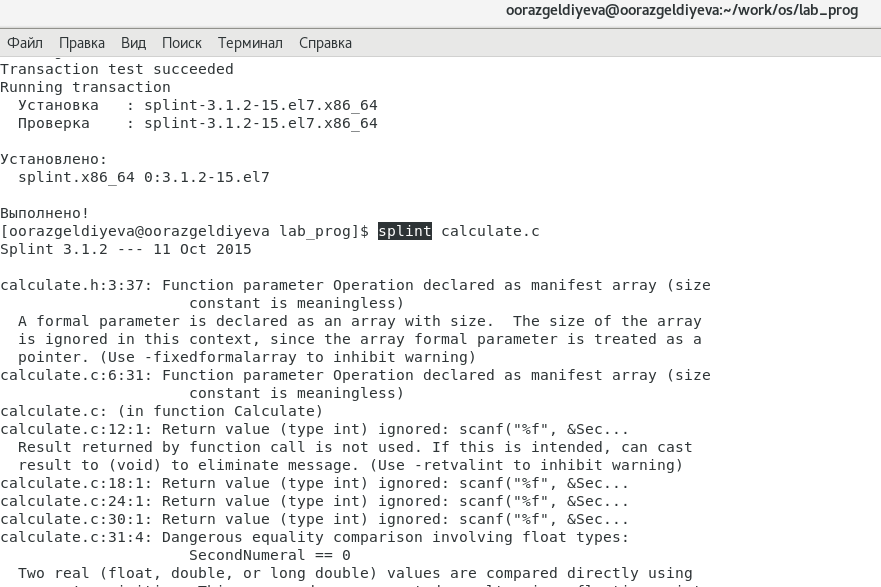


Рисунок 24. Анализ кода файла calculate.c

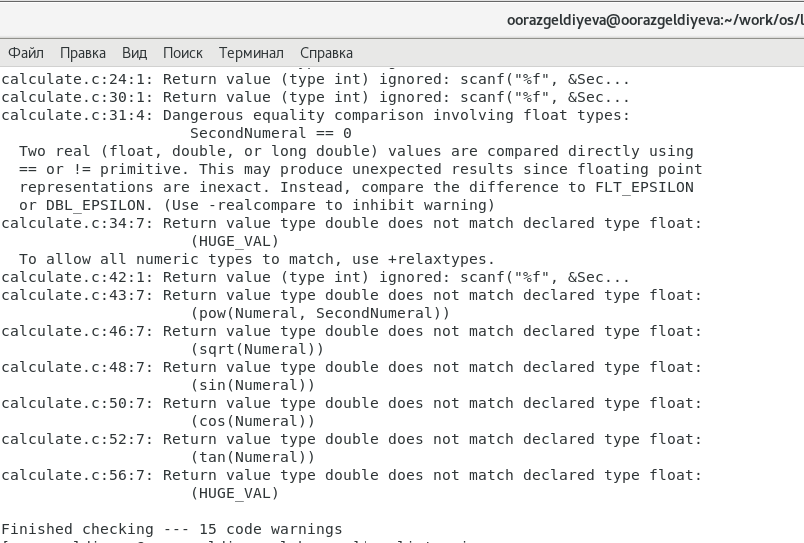


Рисунок 25. Анализ кода файла calculate.c

Как видим из рис. 24-25, в программе файла calculate.c 15 предупреждений: первые два предупреждения для параметра Operation - можно было не указывать размер аргумента; дальше 4 предупреждения для scanf; затем идет предупреждение для равенства двух вещественных чисел и тд.

Попробуем проанализировать main.c. (см. рис. 26)

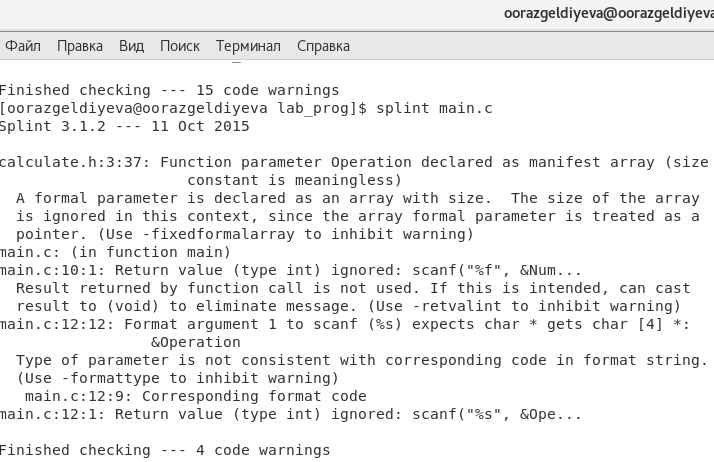


Рисунок 26. Анализ кода файла main.c

В этом файле всего 4 предупреждения: первое предупреждение как в предыдущем файле для параметра Operation; остальные для scanf.

**Вывод:** на лабораторной работе приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями

**Библиография**

[[1] - РУДН, Операционные системы, "Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux"](https://esystem.rudn.ru/pluginfile.php/1142386/mod_resource/content/2/011-lab_prog.pdf)

[[2] - Отладчик GDB](https://www.opennet.ru/docs/RUS/linux_base/node199.html)

[[3] - Краткая инструкция по использованию GDB](https://habr.com/ru/post/491534/)