**Лабораторная работа №13**

### Дисциплина: Операционные системы

## Аксенова Анастасия

**Содержание**

1. [Цель работы](#_bookmark0) 5
2. [Задание](#_bookmark1) 6
3. [Выполнение лабораторной работы](#_bookmark2) 7
4. [Библиография](#_bookmark27) 24
5. [Выводы](#_bookmark28) 25

**List of Tables**

**List of Figures**

[3.1 Создание подкаталога](#_bookmark3) . . . . . . . . . . . . . . . . . . . . . . . . 7

[3.2 Создание файлов](#_bookmark4) . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

[3.3 Программа в calculate.c](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . 8

[3.4 Программа в calculate.c](#_bookmark6) . . . . . . . . . . . . . . . . . . . . . . . 8

[3.5 Программа в calculate.h](#_bookmark7) . . . . . . . . . . . . . . . . . . . . . . . 9

[3.6 Программа в main.c](#_bookmark8) . . . . . . . . . . . . . . . . . . . . . . . . . 9

* 1. [Компиляция программы](#_bookmark9) 10
  2. [Программа в Makefile](#_bookmark10) 10
  3. [Программа в Makefile](#_bookmark11) 11
  4. [Удаление файлов](#_bookmark12) 11
  5. [Компиляция файлов](#_bookmark13) 11
  6. [Работа с gdb](#_bookmark14) 12
  7. [Работа с gdb - run](#_bookmark15) 12
  8. [Работа с gdb - list](#_bookmark16) 13
  9. [Работа с gdb - list 12,15](#_bookmark17) 13
  10. [Работа с gdb - list calculate.c:20,29](#_bookmark18) 14
  11. [Работа с gdb - list calculate.c:20,27](#_bookmark19) 15
  12. [Работа с gdb - info breakpoints](#_bookmark20) 15
  13. [Работа с gdb - run](#_bookmark21) 16
  14. [Работа с gdb - print Numeral](#_bookmark22) 17
  15. [Работа с gdb - display Numeral](#_bookmark23) 17
  16. [Работа с gdb - info breakpoints](#_bookmark24) 18
  17. [Результат команды splint calculate.c](#_bookmark25) 19
  18. [Результат конмады splint main.c](#_bookmark26) 19

# Цель работы

Цель данной лабораторной работы — Приобрести простейшие навыки разра- ботки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# Задание

* 1. Сделать отчёт по лабораторной работе №14 в формате Markdown.
  2. создать на языке программирования С калькулятора с простейшими функ- циями.

# Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог ~/work/os/lab\_progс помощью коман- ды «mkdir-p~/work/os/lab\_prog» (Рисунок [3.1](#_bookmark3) ). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ).



Figure 3.1: Создание подкаталога

* 1. Создала в каталоге файлы: calculate.h, calculate.c, main.c, используя команды

«cd ~/work/os/lab\_prog» и «touch calculate.h calculate.c main.c» (алгоритм действий представлен на рис. [3.2](#_bookmark4) ).

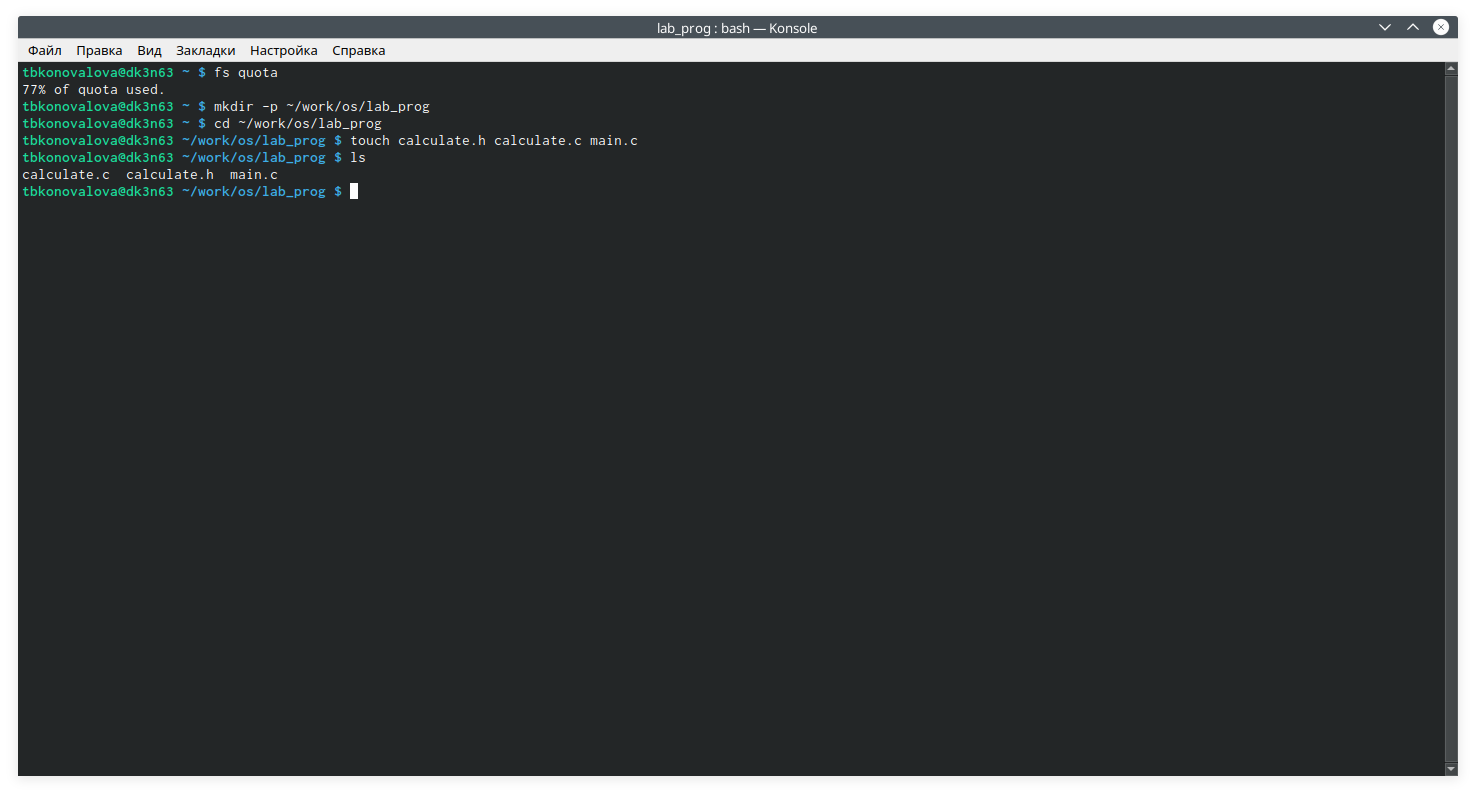


Figure 3.2: Создание файлов

Это будет примитивнейший калькулятор, способный складывать, вычитать,

умножать и делить, возводить число в степень, брать квадратный корень, вы- числять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится. Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с (Программа представлена на рис. [3.3](#_bookmark5) ,

[3.4](#_bookmark6) ). Вся необходимая информация про написания программ указана в следую- щем источнике: Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ).

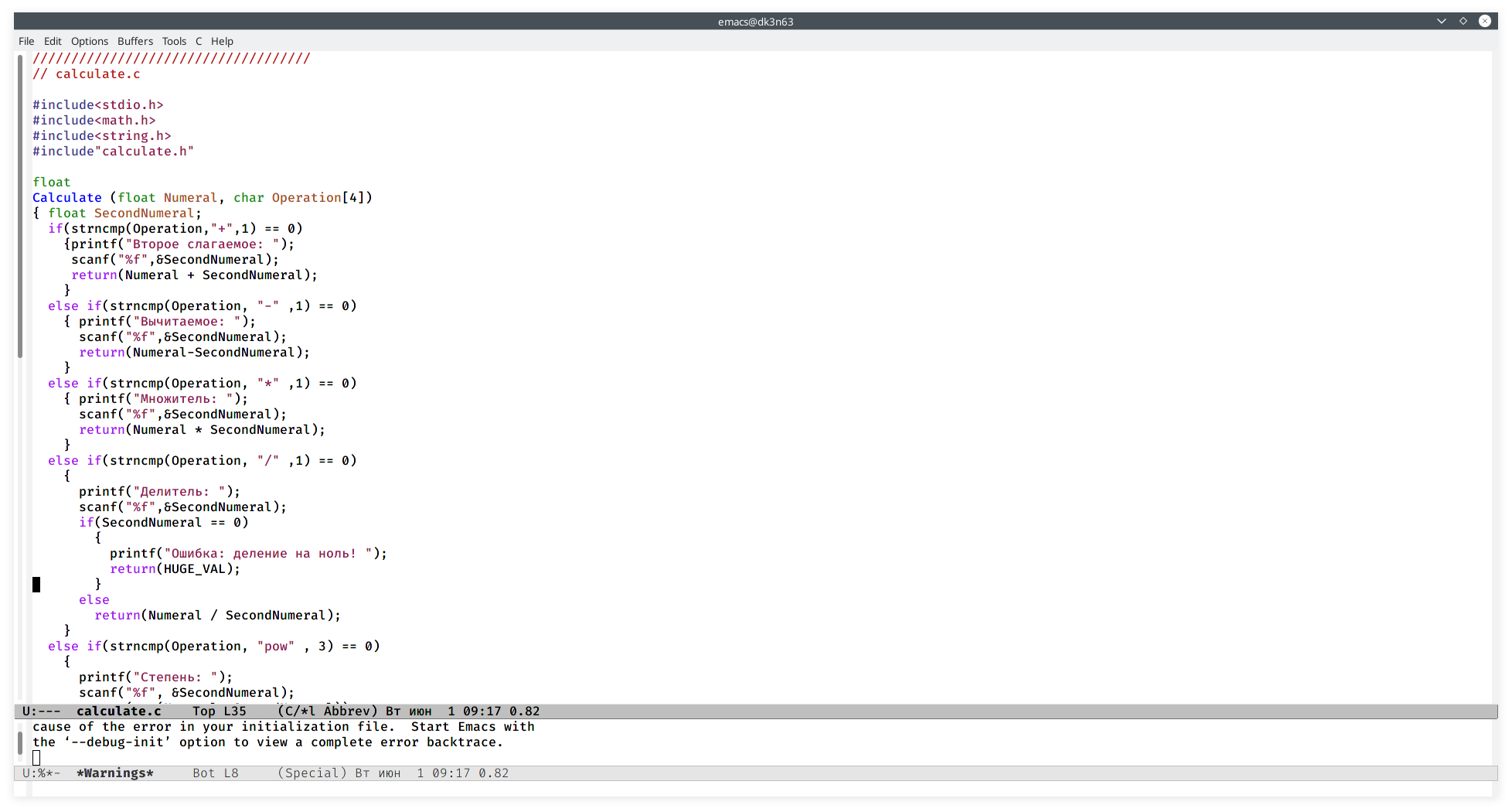


Figure 3.3: Программа в calculate.c

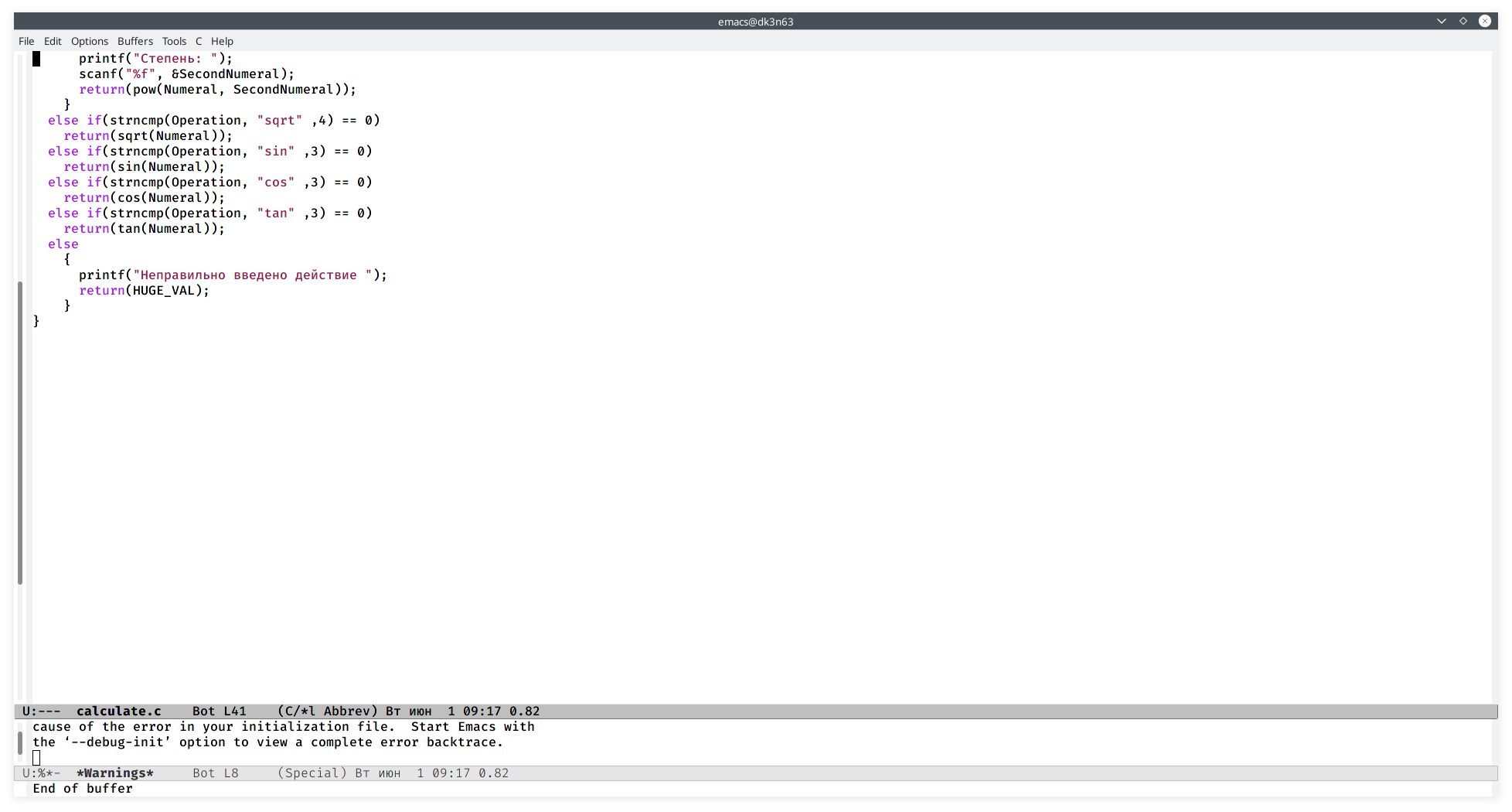


Figure 3.4: Программа в calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции каль-

кулятора (Скриншот [3.5](#_bookmark7) ). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: Электронный ресурс: https://it.wikireading.ru/34160

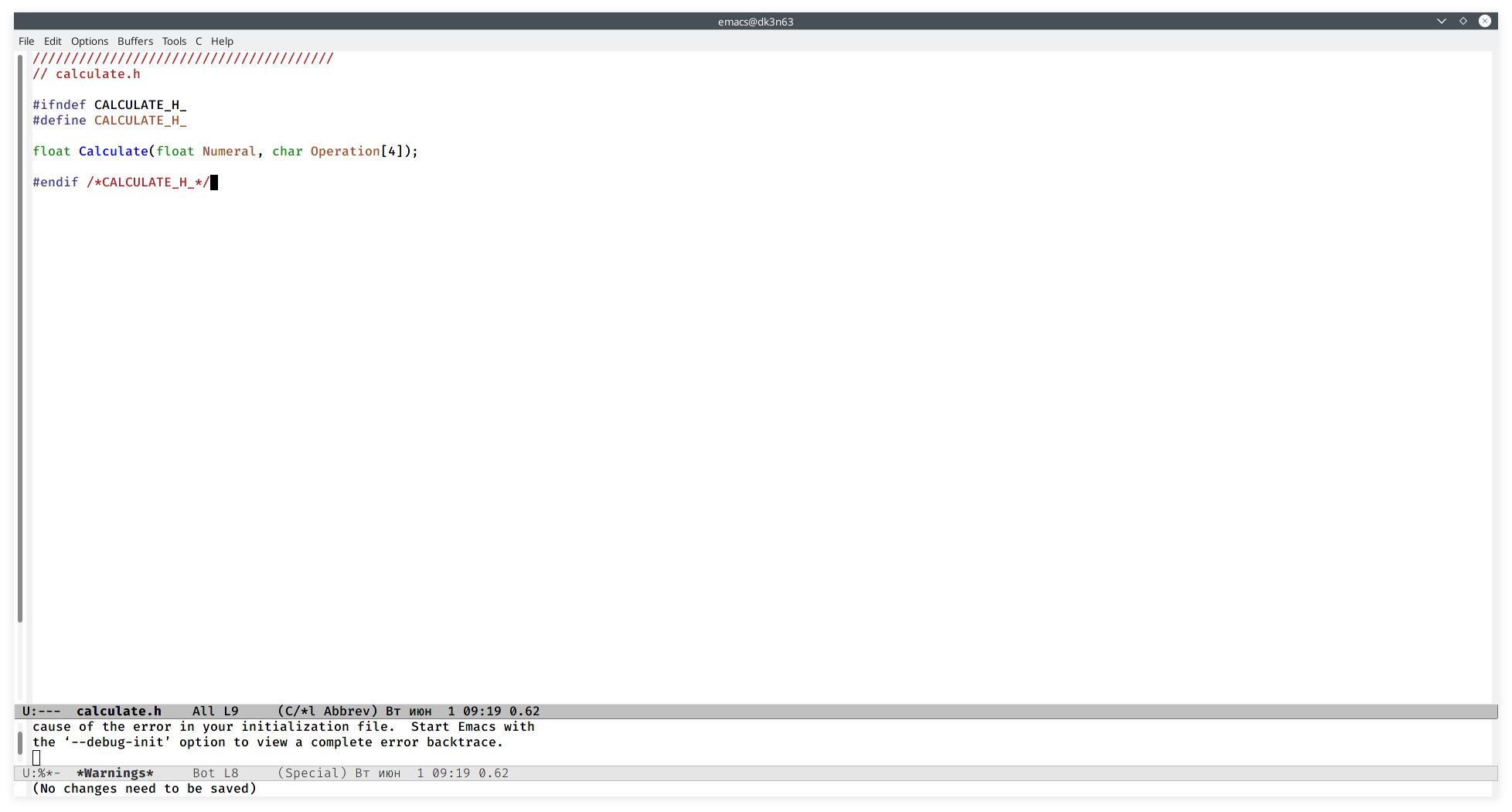


Figure 3.5: Программа в calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (Скриншот [3.6](#_bookmark8) ).

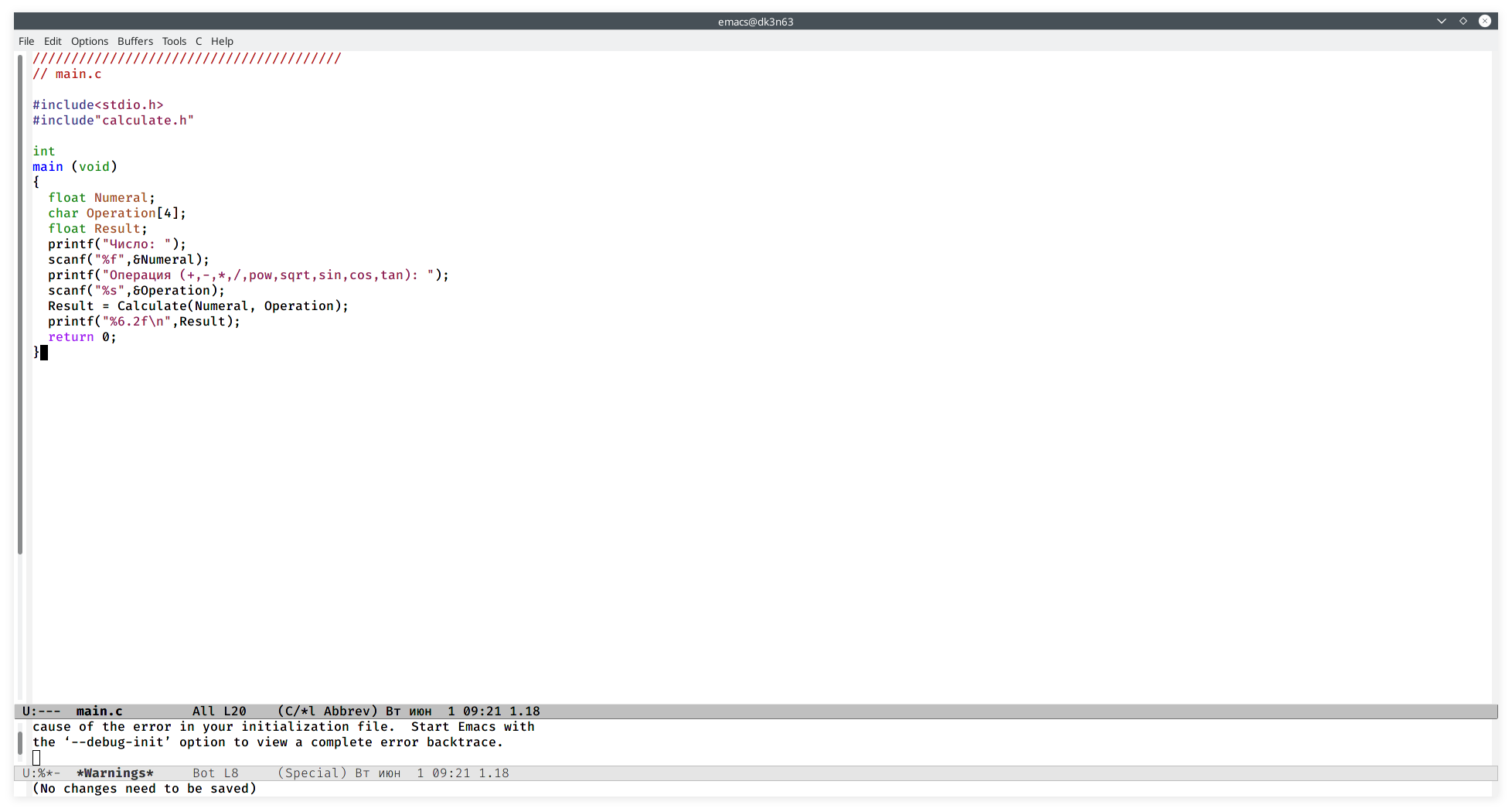


Figure 3.6: Программа в main.c

* 1. Выполнила компиляцию программы посредством gcc (версия компилятора

:8.3.0-19), используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (алгоритм действий представлен на рис. [3.7](#_bookmark9) ).



Figure 3.7: Компиляция программы

* 1. В ходе компиляции программы никаких ошибок выявлено не было. 5). Создала Makefile с необходимым содержанием (Рисунок [3.8](#_bookmark10) ).

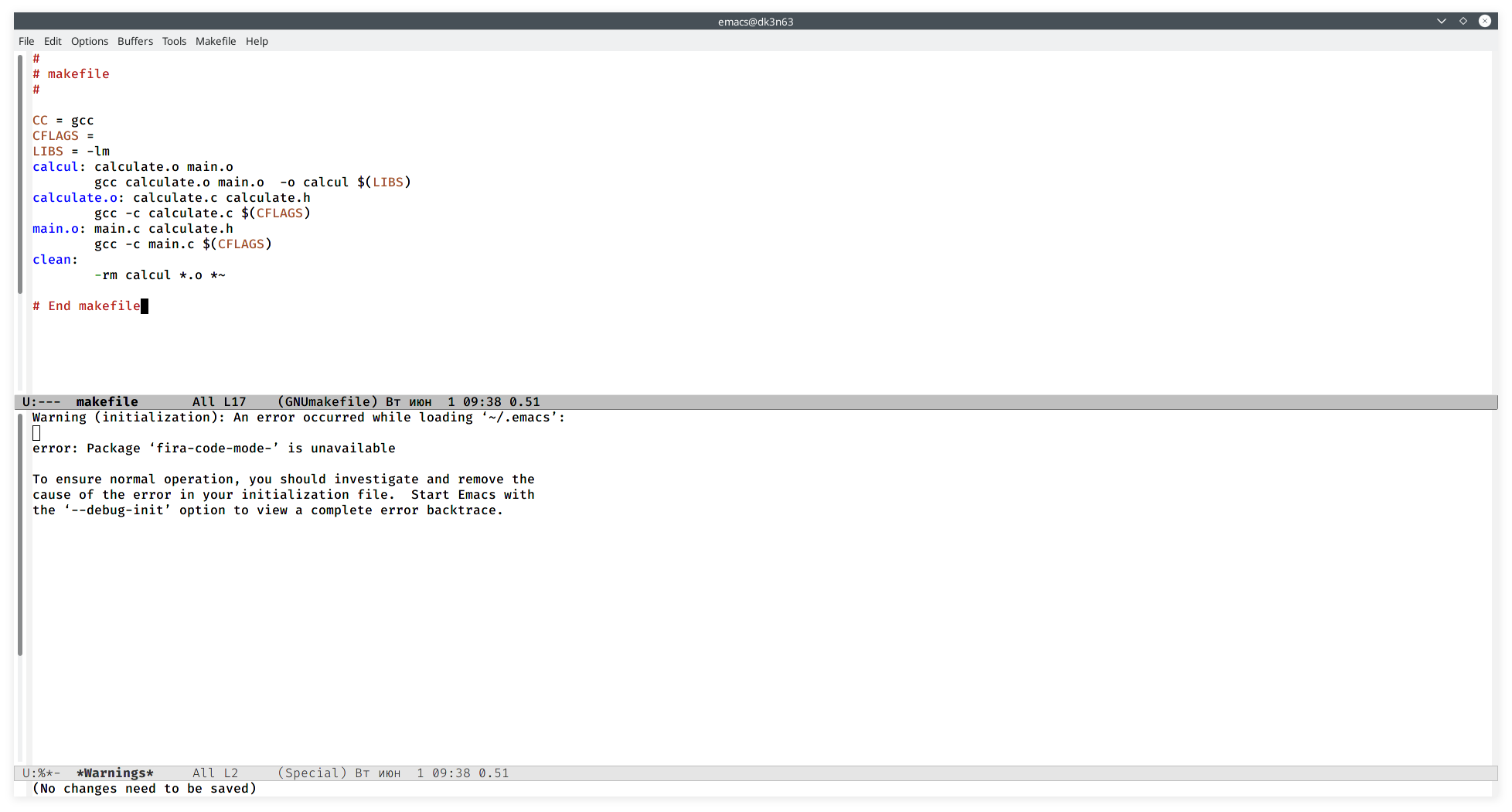


Figure 3.8: Программа в Makefile

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цельmain.o), а также их объединения в один испол- няемый файл calcul (цель calcul). Цель clean нужна для автоматического удале- ния файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

1. Далее исправила Makefile (Скриншот [3.9](#_bookmark11) ). Подробная информация о напи- сании программ в Linux указана в следующем источнике: Электронный ресурс: https://vunivere.ru/work23597

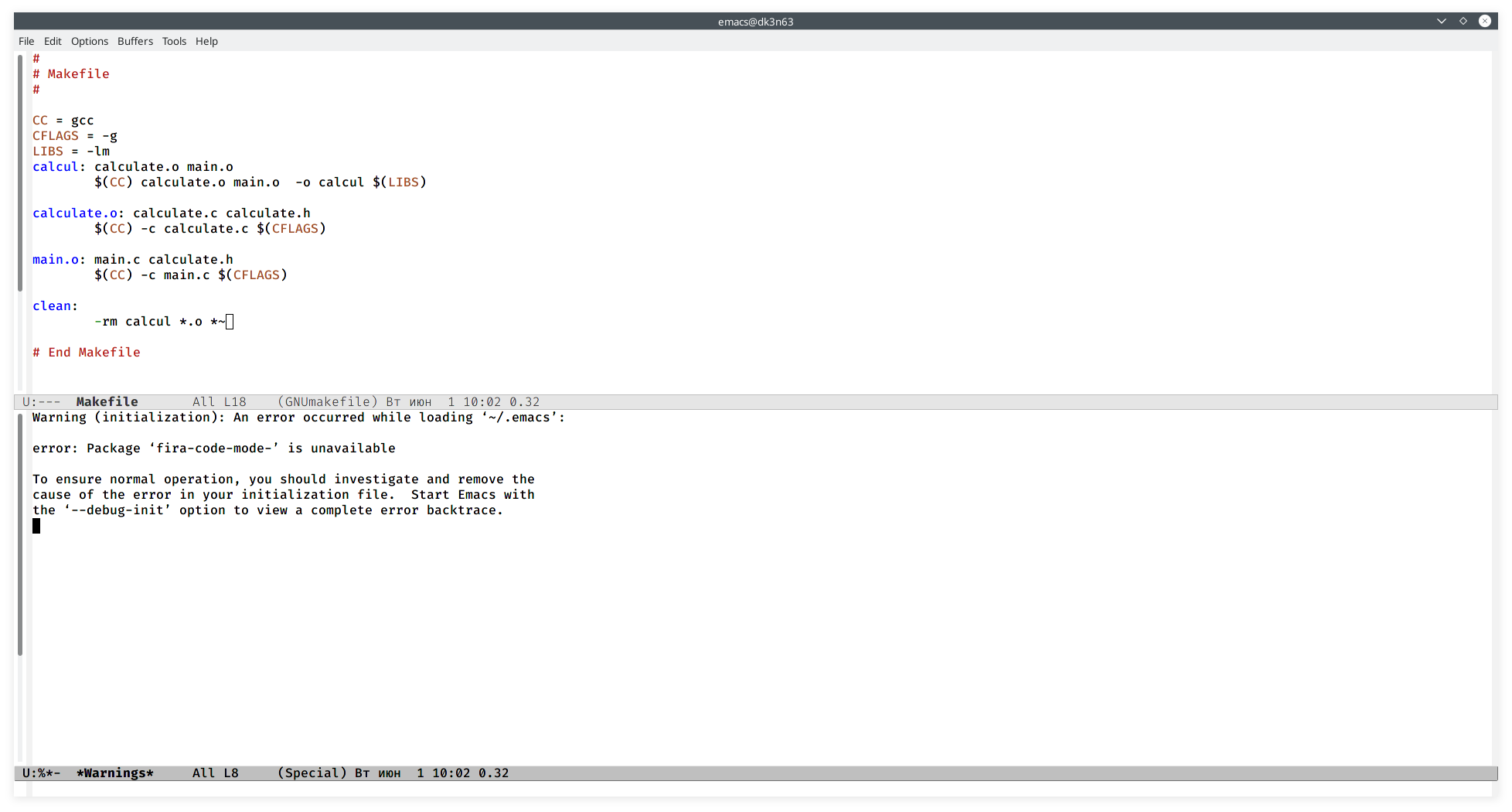


Figure 3.9: Программа в Makefile

В переменную CFLAGS добавила опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделала так, что утилита компиляции выбирается с помощью переменной CC. После этого я удалила исполняемые и объектные файлы из каталога с помощью команды

«make clear» (Рисунок [3.10](#_bookmark12) ). Выполнила компиляцию файлов, используя команды

«make calculate.o», «make main.o», «make calcul» (Рисунок [3.11](#_bookmark13) ).



Figure 3.10: Удаление файлов

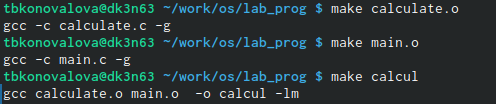


Figure 3.11: Компиляция файлов

Далее с помощью gdb выполнила отладку программы calcul. Запустила отлад- чик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul» (алгоритм действий представлен на рис. [3.12](#_bookmark14) ).

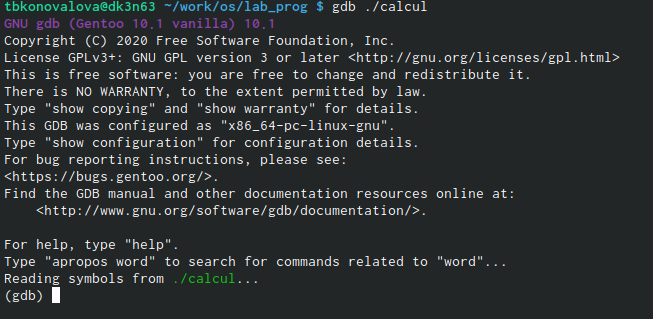


Figure 3.12: Работа с gdb

Для запуска программы внутри отладчика ввела команду «run» (Рисунок [3.13](#_bookmark15) ).

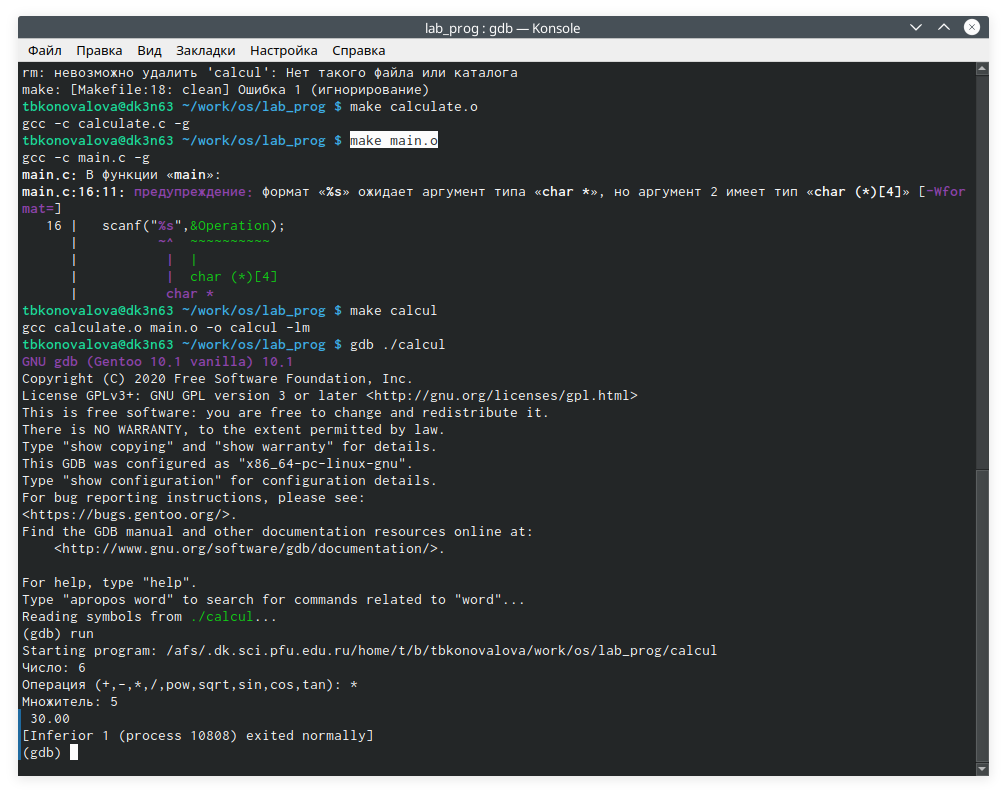


Figure 3.13: Работа с gdb - run

Для постраничного (по10строк) просмотра исходного кода использовала ко- манду «list» (Рисунок [3.14](#_bookmark16) ).

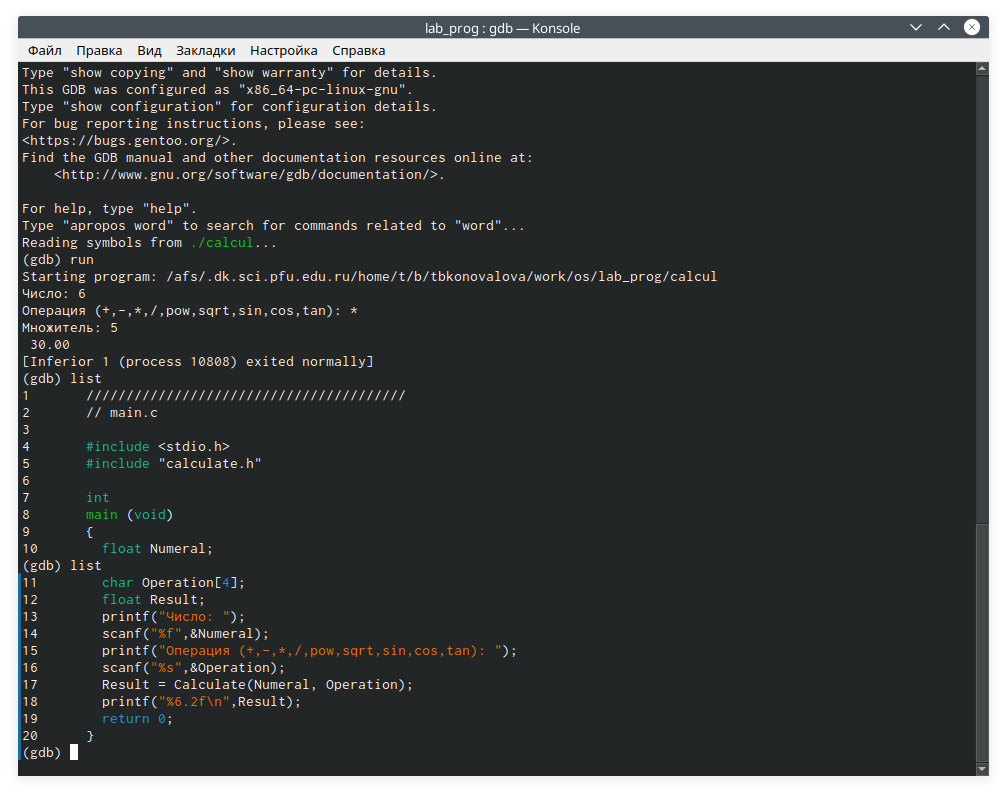


Figure 3.14: Работа с gdb - list

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12,15» (Рисунок [3.15](#_bookmark17) ).

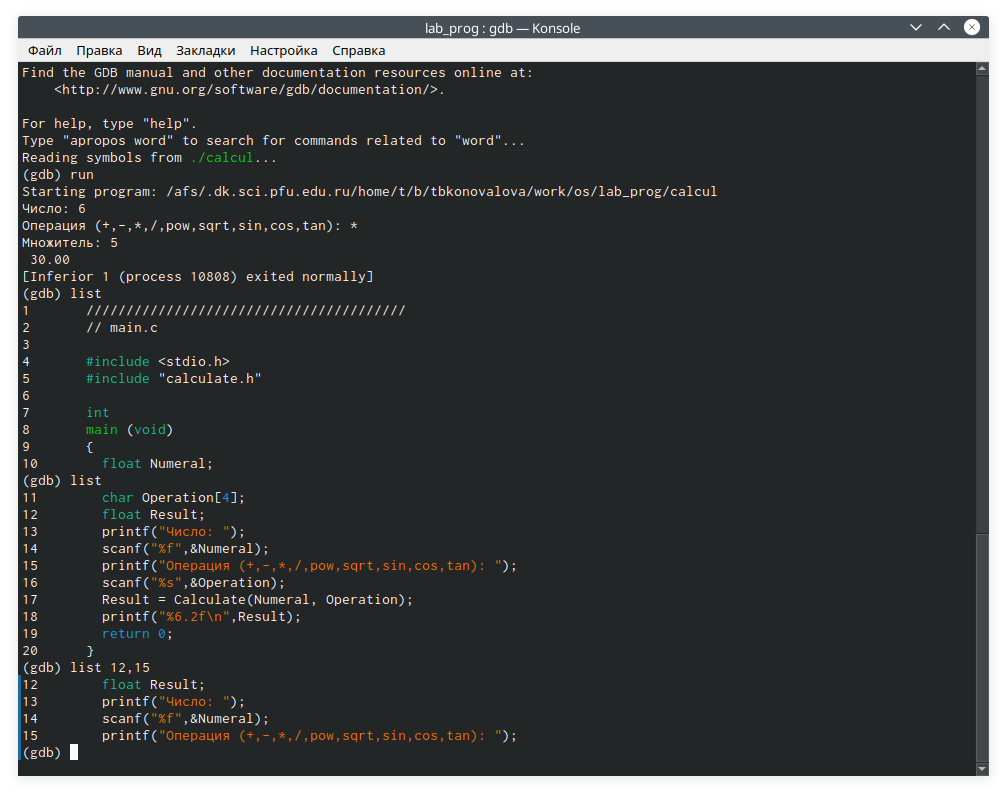


Figure 3.15: Работа с gdb - list 12,15

Для просмотра определённых строк не основного файла использовала команду

«list calculate.c:20,29» (Рисунок [3.16](#_bookmark18) ).

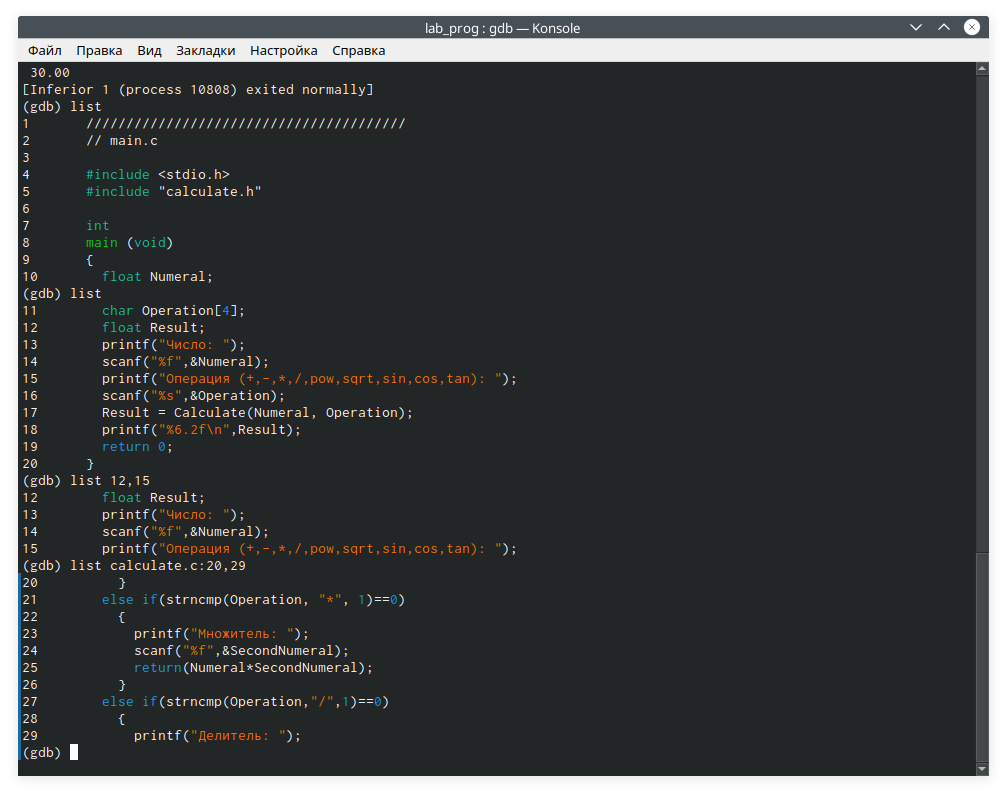


Figure 3.16: Работа с gdb - list calculate.c:20,29

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (Рисунок [3.17](#_bookmark19) ).

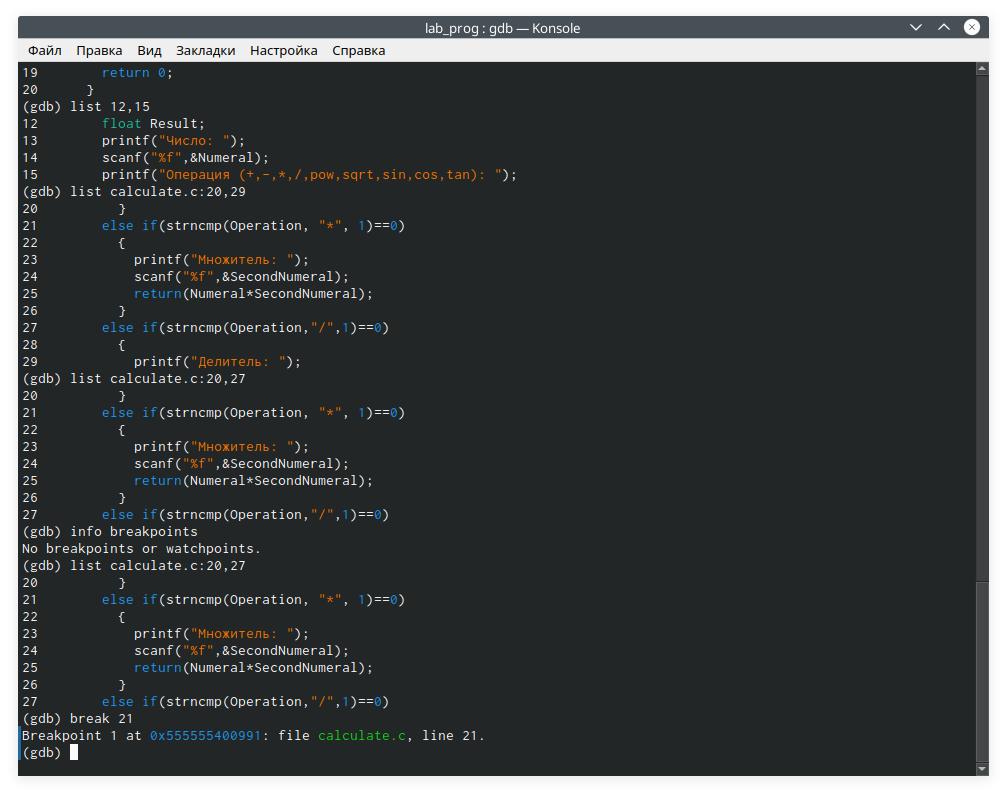


Figure 3.17: Работа с gdb - list calculate.c:20,27

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (Скриншот [3.18](#_bookmark20) ).

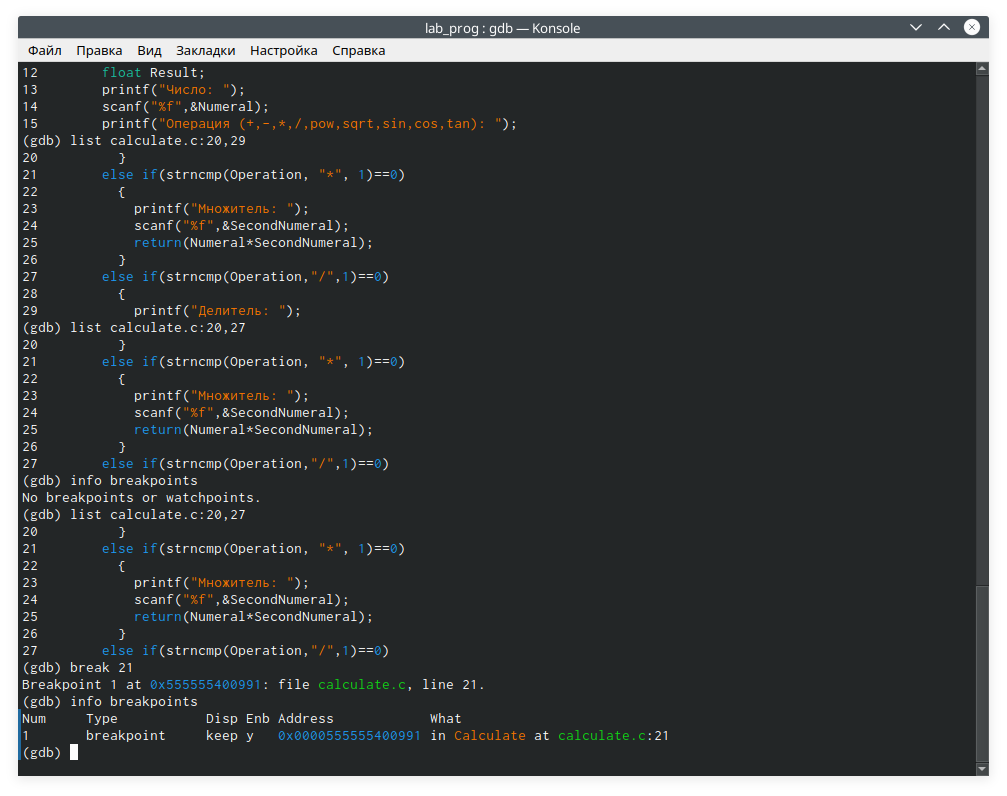


Figure 3.18: Работа с gdb - info breakpoints

Запустила программу внутри отладчика и убедилась, что программа останови- лась в момент прохождения точки останова. Использовала команды «run», «5»,

« » и «backtrace» (Скриншот [3.19](#_bookmark21) ).

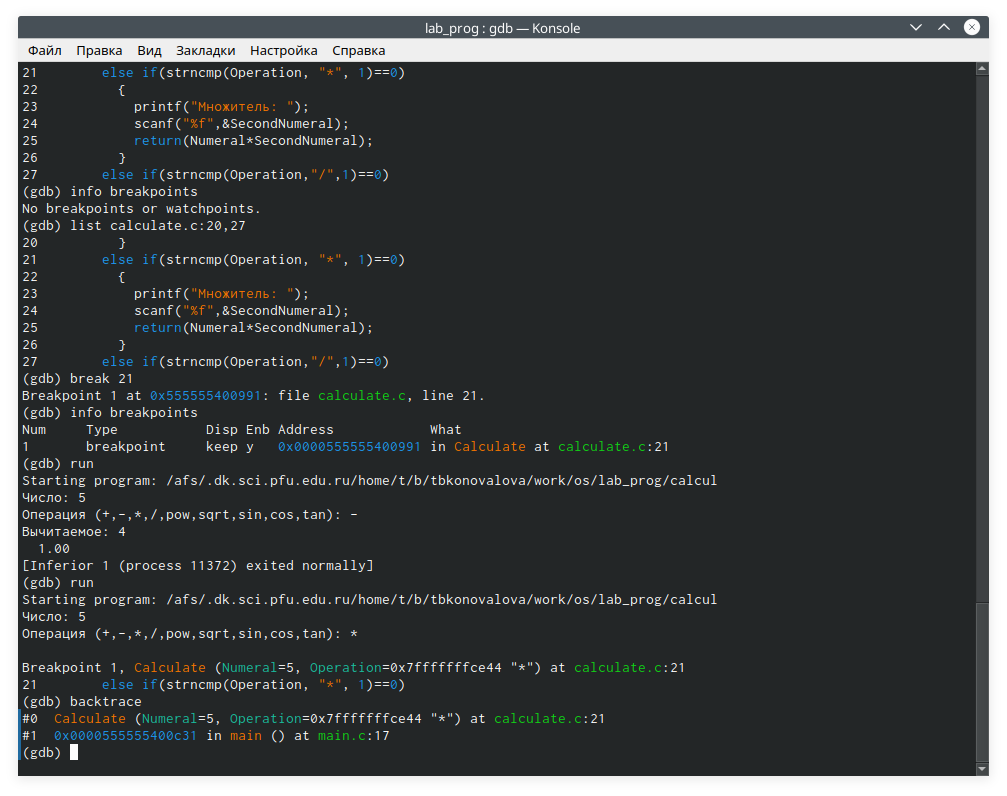


Figure 3.19: Работа с gdb - run

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (Скриншот [3.20](#_bookmark22) ).

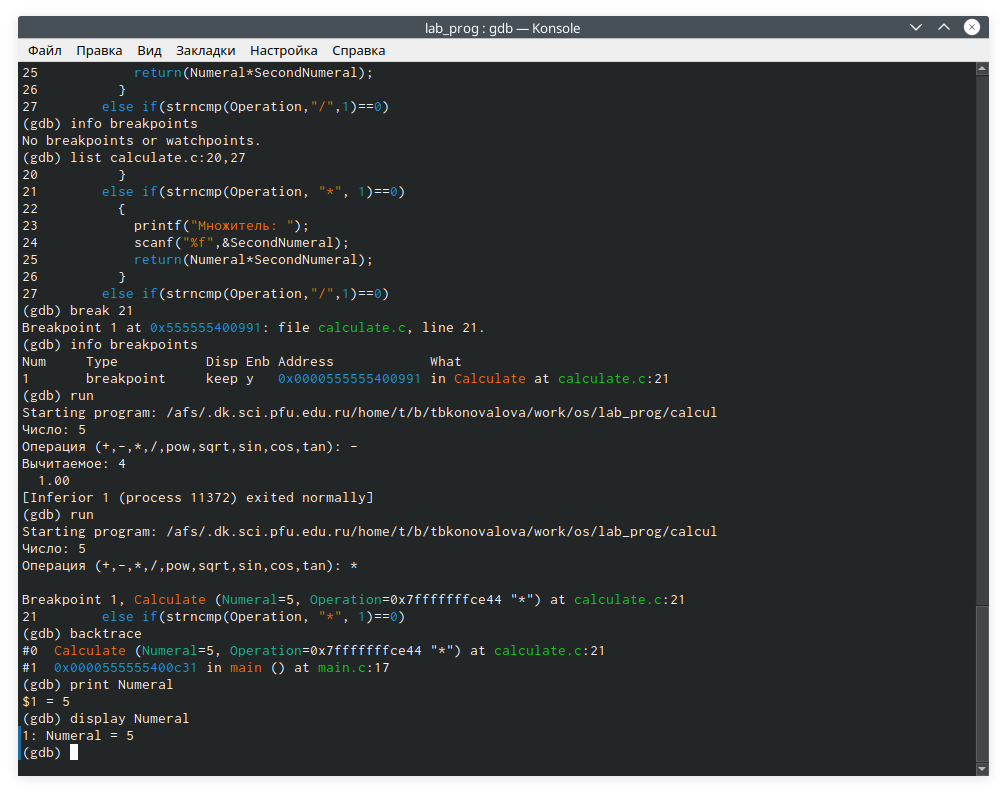


Figure 3.20: Работа с gdb - print Numeral

Сравнила с результатом вывода на экран после использования команды

«display Numeral». Значения совпадают (Скриншот [3.21](#_bookmark23) ).

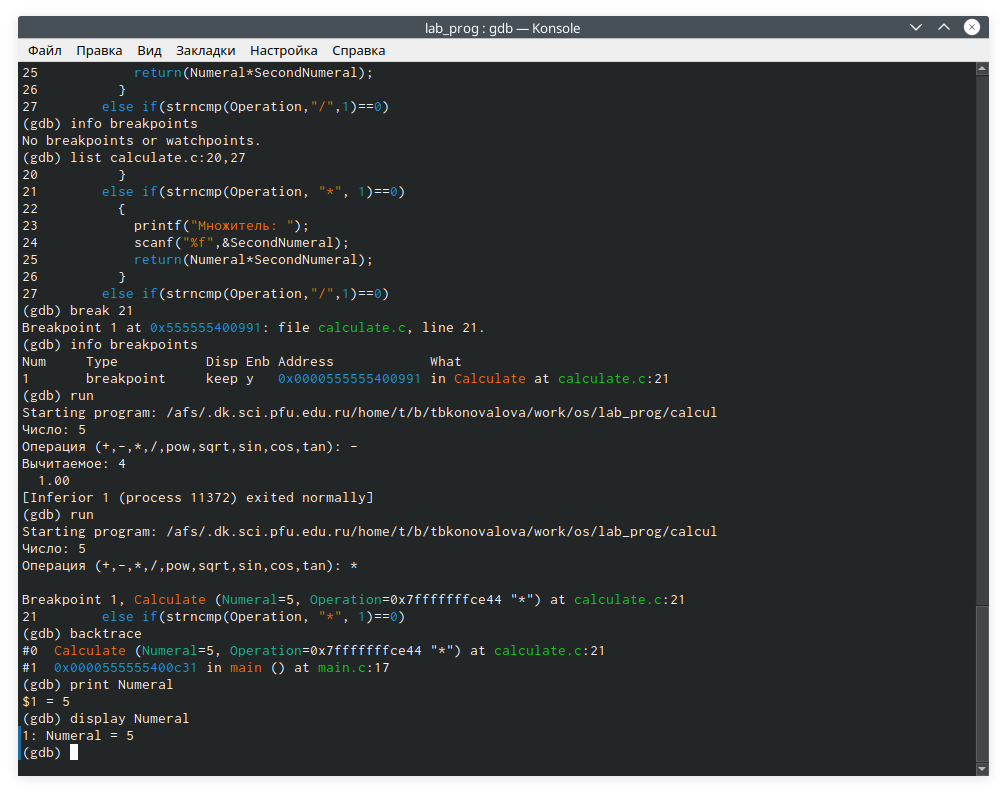


Figure 3.21: Работа с gdb - display Numeral

Убрала точки останова с помощью команд «info breakpoints» и «delete1» (алго- ритм действий представлен на рис. [3.22](#_bookmark24) ).

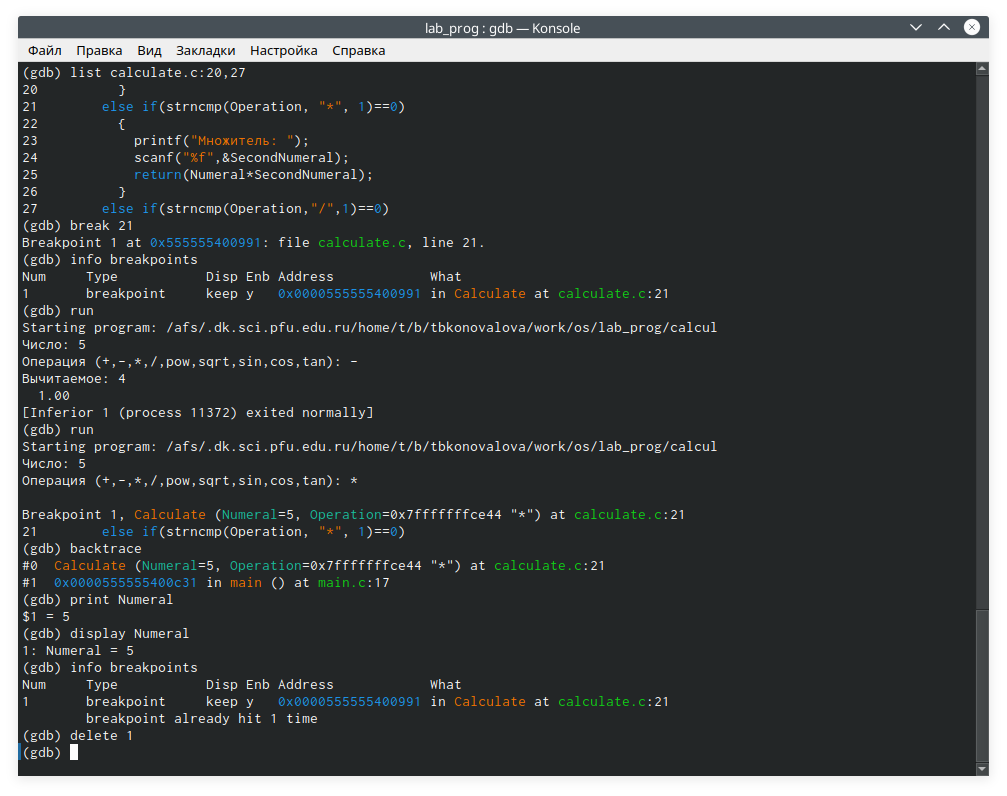


Figure 3.22: Работа с gdb - info breakpoints

1. Далее воспользовалась командами «splint calculate.c» и «splint main.c» (ал- горитм действий представлен на рис. [3.23](#_bookmark25) , [3.24](#_bookmark26) ).C помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохранятся. Утилита вывела предупреждение о том, что в файле calculate.c про- исходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потери данных.

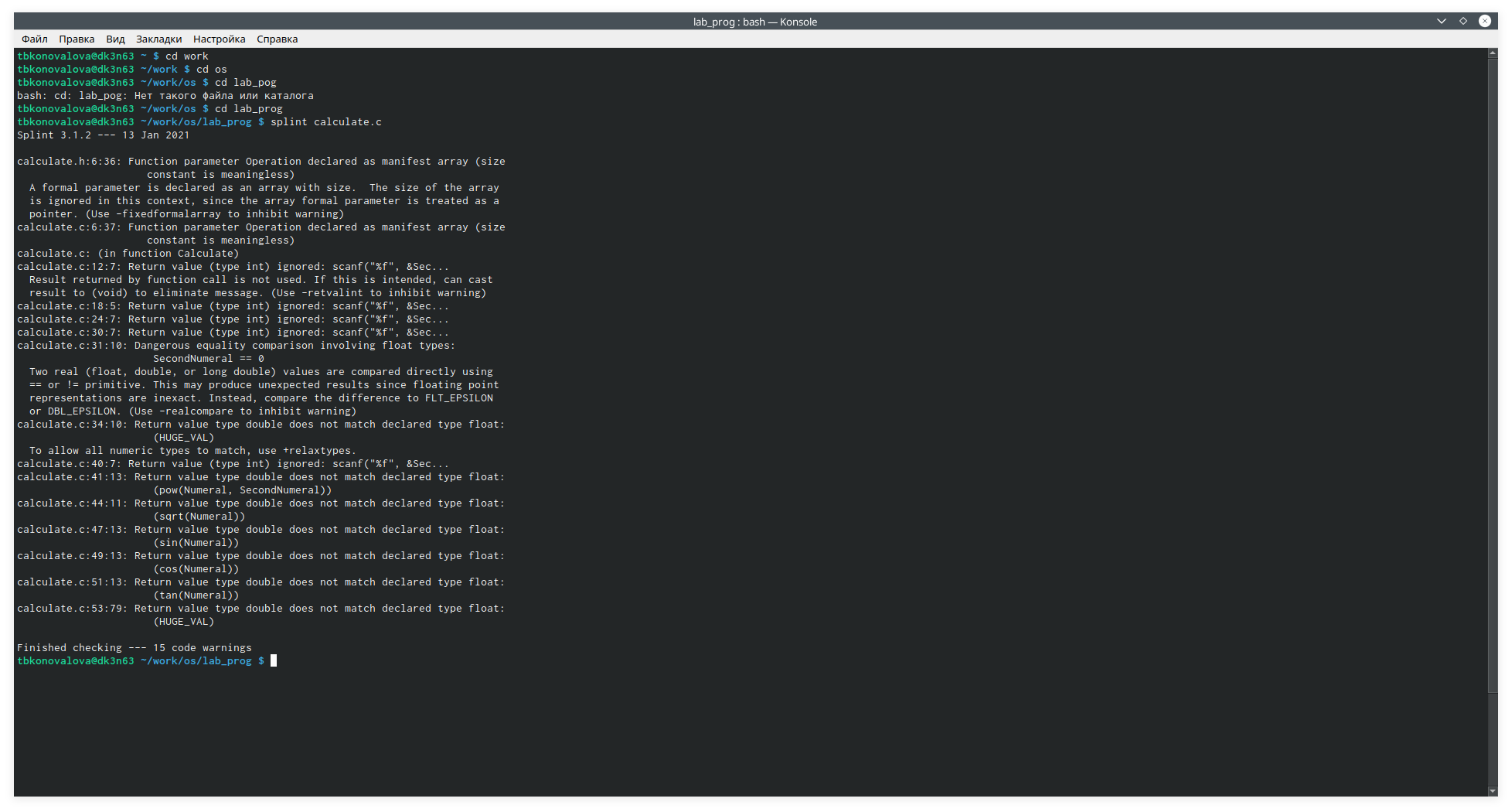


Figure 3.23: Результат команды splint calculate.c

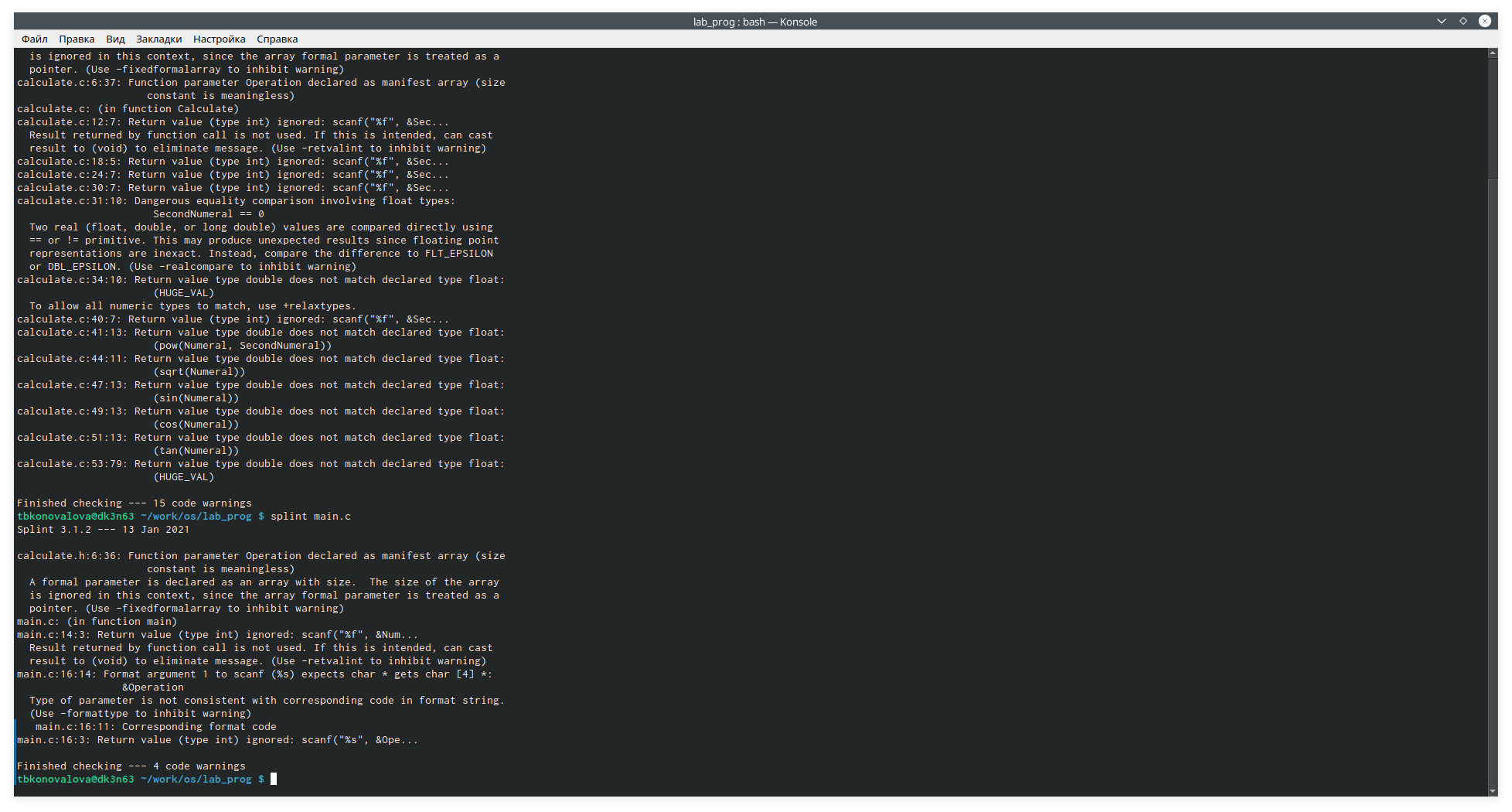


Figure 3.24: Результат конмады splint main.c

Контрольные вопросы:

1. Чтобы получить информацию о возможностях программ gcc, make, gdbи др.нужно воспользоваться командой manили опцией -help(-h)для каждой коман- ды.
2. Процесс разработки программного обеспечения обычно разделяется на следующие этапы:
3. планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
4. проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
5. непосредственная разработка приложения: oкодирование −по сути создание исходного текста программы (возможно в нескольких вариантах);

–анализ разработанного кода; oсборка, компиляция и разработка испол- няемого модуля; oтестирование и отладка, сохранение произведённых изменений;

1. документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geanyи др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.
2. Для имени входного файла суффикс определяет какая компиляция требу- ется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом)

.cвоспринимаются gccкак программы на языке С, файлы с расширением .ccили

.C−как файлы на языке C++, а файлы cрасширением .oсчитаются объектными.На- пример, в команде «gcc-cmain.c»:gccпо расширению (суффиксу) .cраспознает тип файла для компиляции и формирует объектный модуль −файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -oи в качестве параметра задать имя создаваемого файла: «gcc-ohellomaiВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке про- граммирования С калькулятора с простейшими функциями.n.c».

1. Основное назначение компилятора языка Си в UNIX заключается в компи- ляции всей программы и получении исполняемого файла/модуля.
2. Для сборки разрабатываемого приложения и собственно компиляции по- лезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает

взаимосвязи между файлами.

1. Для работы с утилитой makeнеобходимо в корне рабочего каталога с Ва- шим проектом создать файл с названием makefileили Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefile имеет следующий синтаксис: … : …<команда 1>…Снача- ла задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefileможет выступать имя файла или название какого-то действия. Зависи- мость задаёт исходные параметры (условия) для достижения указанной цели. За- висимость также может быть названием какого-то действия. Команды − собствен- но действия, которые необходимо выполнить для достижения цели.Общий син- таксис Makefileимеет вид: target1 [target2…]:[:] [dependment1…][(tab)commands] [#commentary][(tab)commands] [#commentary]. Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабаты- ваться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких после- довательных строках.Пример более сложного синтаксиса Makefile:## Makefile for abcd.c#CC = gccCFLAGS =# Compile abcd.c normalyabcd: abcd.c$(CC) -o abcd

$(CFLAGS) abcd.cclean:-rm abcd *.o* ~# EndMakefileforabcd.c. В этом примере в на-

чале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их за- висимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем cleanпроизводит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

1. Во время работы над кодом программы программист неизбежно сталкивает- ся с появлением ошибок в ней. Использование отладчика для поиска и устранения

ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNUдля ОС типа UNIXвходит отладчик GDB(GNUDebugger). Для ис- пользования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -gкомпилятора gcc: gcc-cfile.c-g. После этого для начала работы с gdbнеобходимо в командной стро- ке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdbfile.o

1. Основные команды отладчика gdb: 1. backtrace − вывод на экран пути к текущей точке останова (по сутивывод − названий всех функций); 2. break − установить точку останова (в качестве параметра можетбыть указан номер стро- ки или название функции); 3. clear − удалить все точки останова в функции; 4. continue − продолжить выполнение программы; 5. delete − удалить точку оста- нова; 6. display − добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы; 7. finish − выполнить программу до момента выхода из функции; 8. info breakpoints −вывести на экран список используемых точек останова; 9. info watchpoints −вывести на экран спи- сок используемых контрольных выражений; 10. list − вывести на экран исходный код (вВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями. качестве параметра может быть указано название файла и через двоеточие номера начальнойи конечной строк); 11. next − вы- полнить программу пошагово, но без выполнения вызываемых в программе функций; 12. print − вывести значение указываемого в качестве параметра выра- жения; 13. run − запуск программы на выполнение; 14. set − установить новое значение переменной; 15. step − пошаговое выполнение программы; 16. watch

– установить контрольное выражение, при изменении значения которого про- грамма будет остановлена. Для выхода из gdbможно воспользоваться командой

quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно получить с помощью команд gdb-hи mangdb.

1. Cхема отладки программы показана в 6 пункте лабораторной работы.
2. При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена ошибка, которую компилятор мог пропустить (воз- можно, из-за версии 8.3.0-19): в строке scanf(“%s”, &Operation); нужно убрать знак &, потому что имя массивасимволов уже является указателемна первый элементэтого массива.
3. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:�cscope −исследова- ние функций, содержащихся в программе,�lint −критическая проверка программ, написанных на языке Си.
4. Утилита splint анализирует программный код, проверяет корректность за- дания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора Cанализатор splintгенерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значе- ния не используются в работt программы, переменные с некорректно заданными значениямии типами и многое другое.

# Библиография

* 1. Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ)
  2. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
  3. Программное обеспечение GNU/Linux. Лекция 11. udev, DBus, PolicyKit (Г. Курячий, МГУ)
  4. Электронный ресурс: https://vunivere.ru/work23597
  5. Электронный ресурс: https://it.wikireading.ru/34160

# Выводы

В ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux- на примере создания на языке программирования С калькулятора с простейши- ми функциями.