Отчет по лабораторной работе №14

Дисциплина: Операционные системы

Тихонова Екатерина Андреевна

Содержание

# Цель работы

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

# Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог ~/work/os/lab\_progс помощью команды «mkdir-p~/work/os/lab\_prog»

Figure 1: Создаем подкаталог

Figure 1: Создаем подкаталог

Создала в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd~/work/os/lab\_prog»и «touchcalculate.hcalculate.cmain.c»

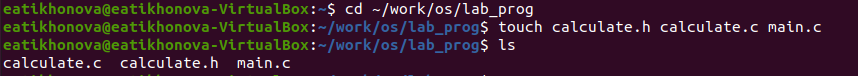


Figure 2: Создаем файлы

Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять 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

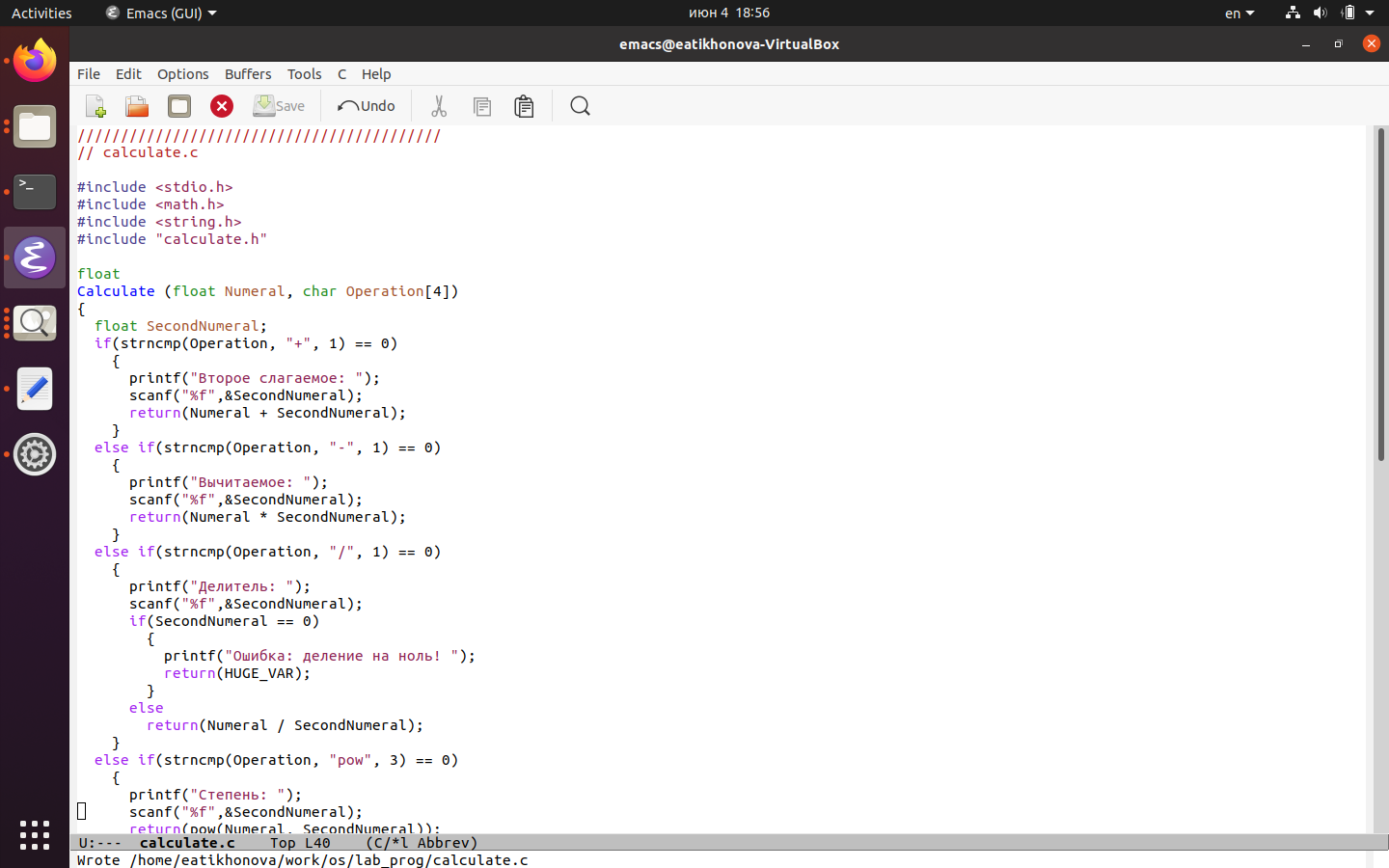


Figure 3: Редактируем файлы

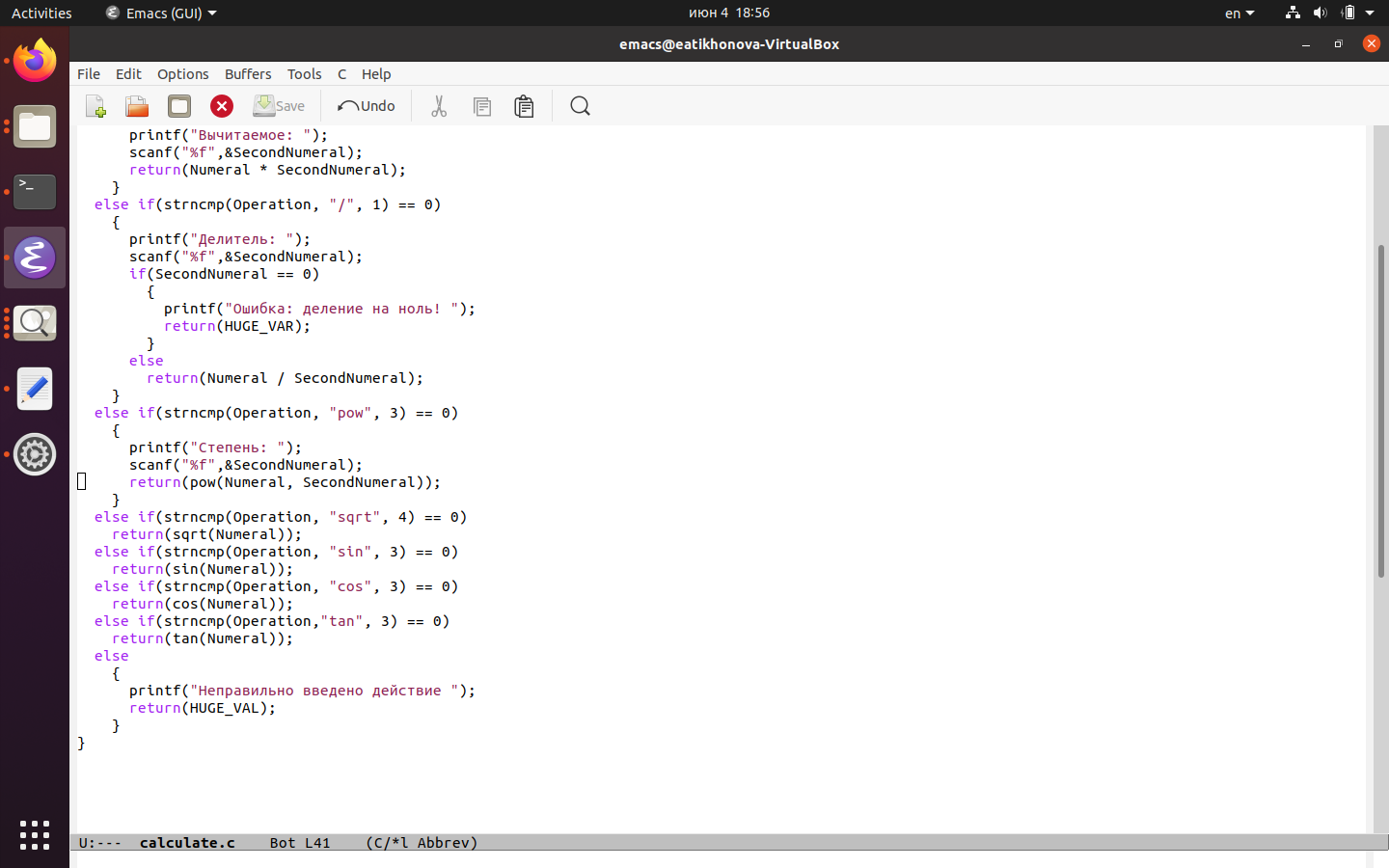


Figure 4: Редактируем файлы

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора

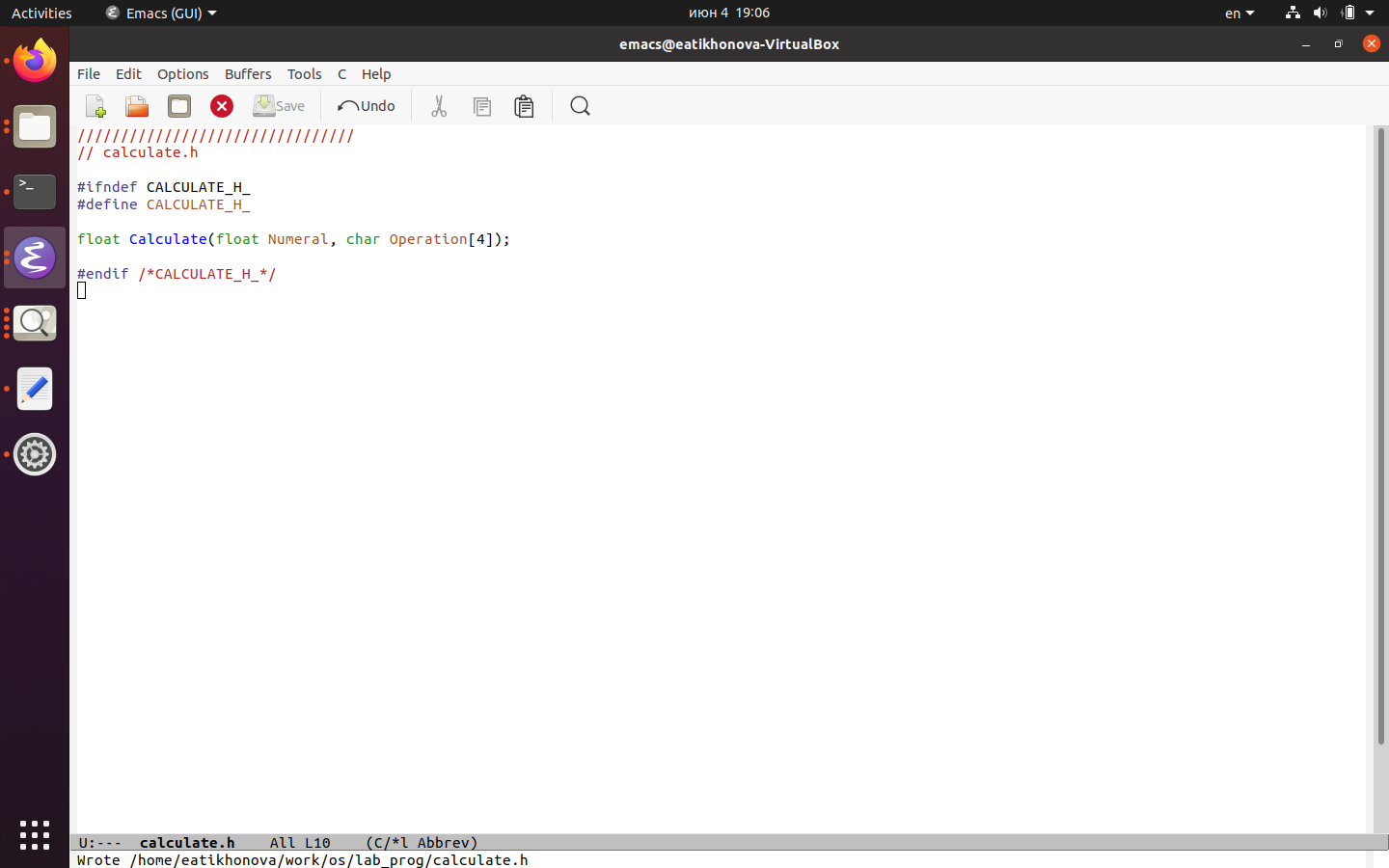


Figure 5: Файл

Основной файл main.c, реализующий интерфейс пользователя к калькулятору

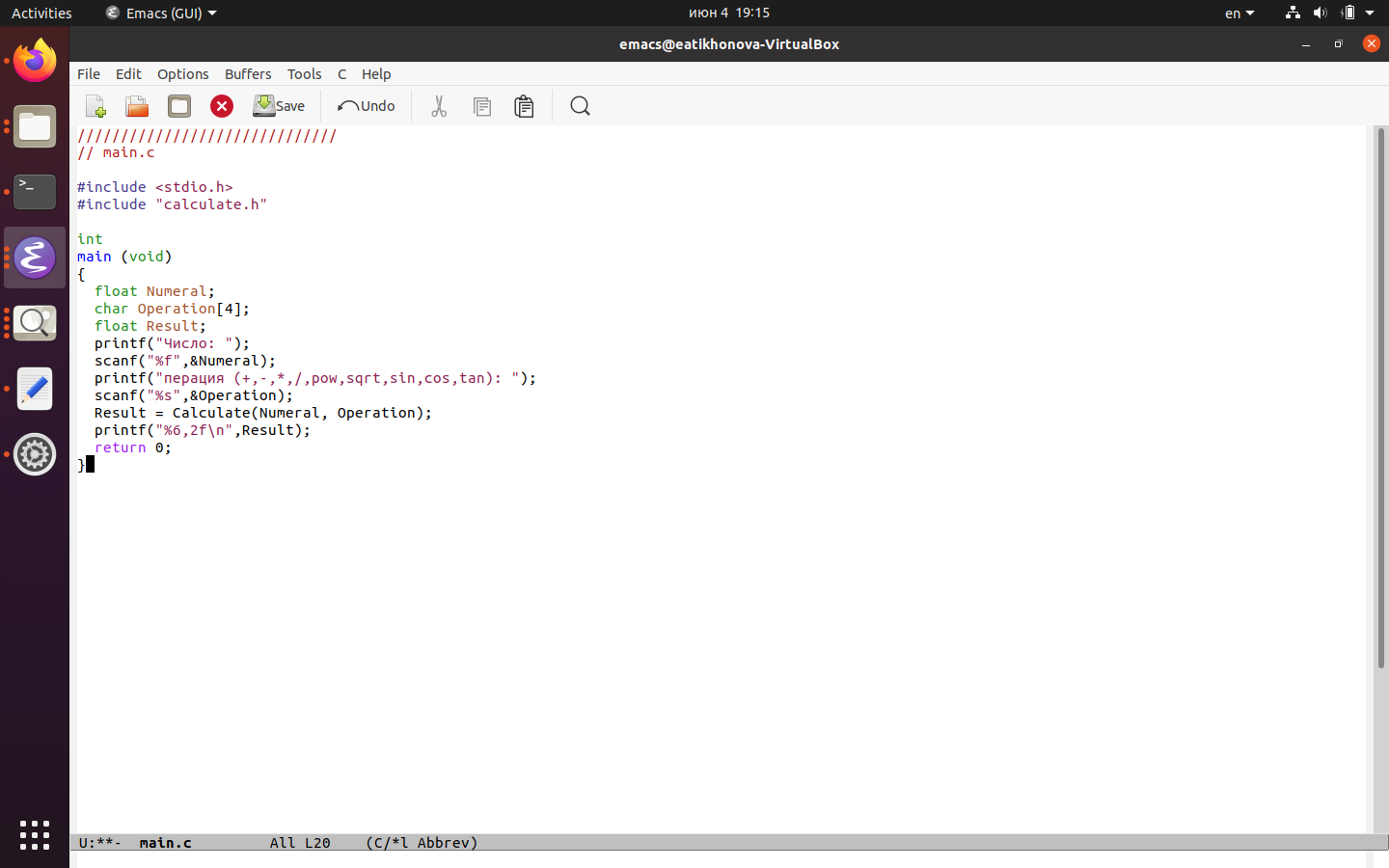


Figure 6: Файл

1. Выполнила компиляцию программы посредством gcc(версия компилятора: 8.3.0-19), используя команды «gcc-ccalculate.c», «gcc-cmain.c» и «gcc calculate.o main.o-o calcul-lm»

Figure 7: Компилируем

Figure 7: Компилируем

1. В ходе компиляции программы никаких ошибок выявлено не было.
2. Создала Makefile с необходимым содержанием.

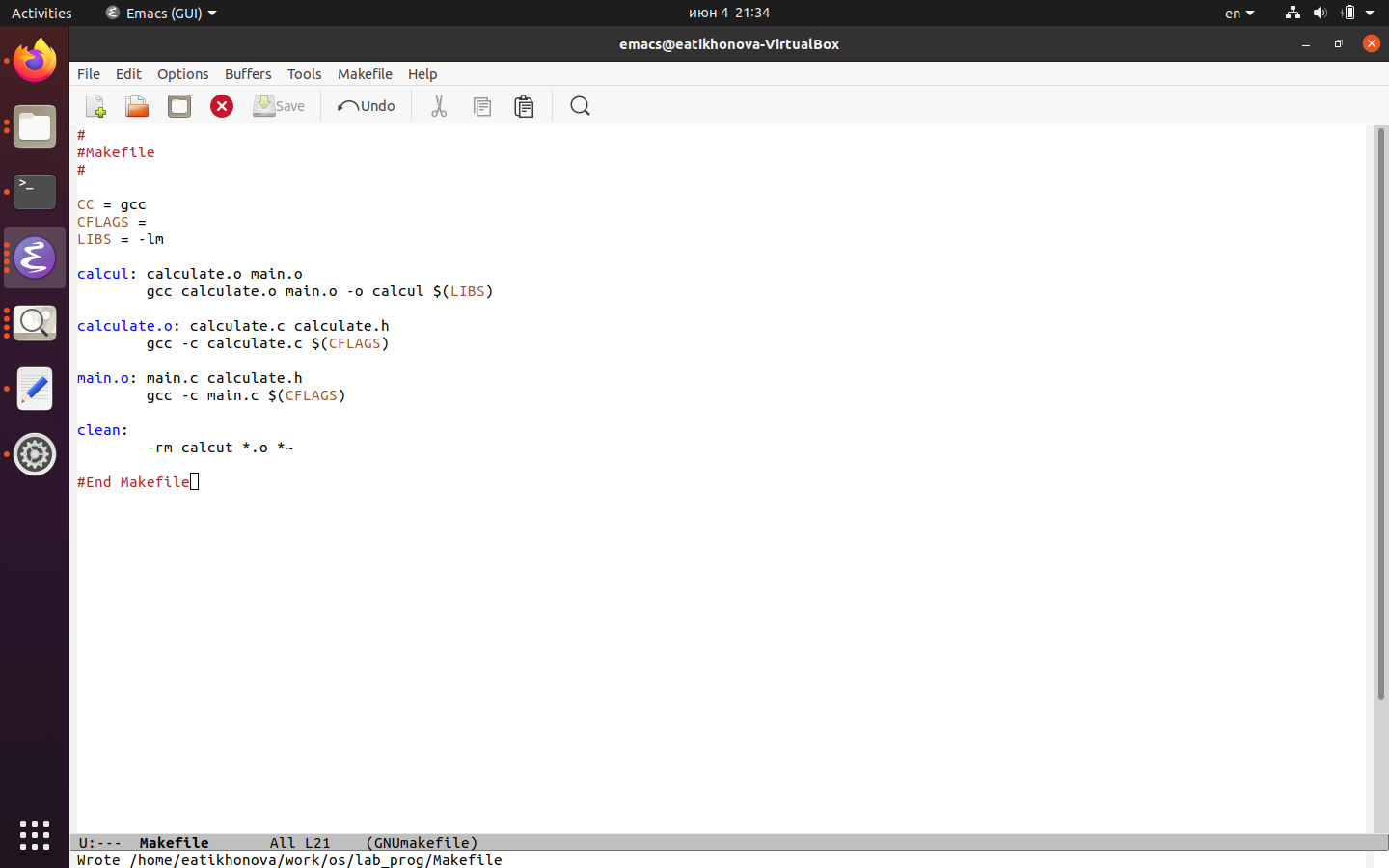


Figure 8: Проверяем работу

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul(цель calcul). Цель clean нужна для автоматического удаления 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. 6. Далее исправила Makefile

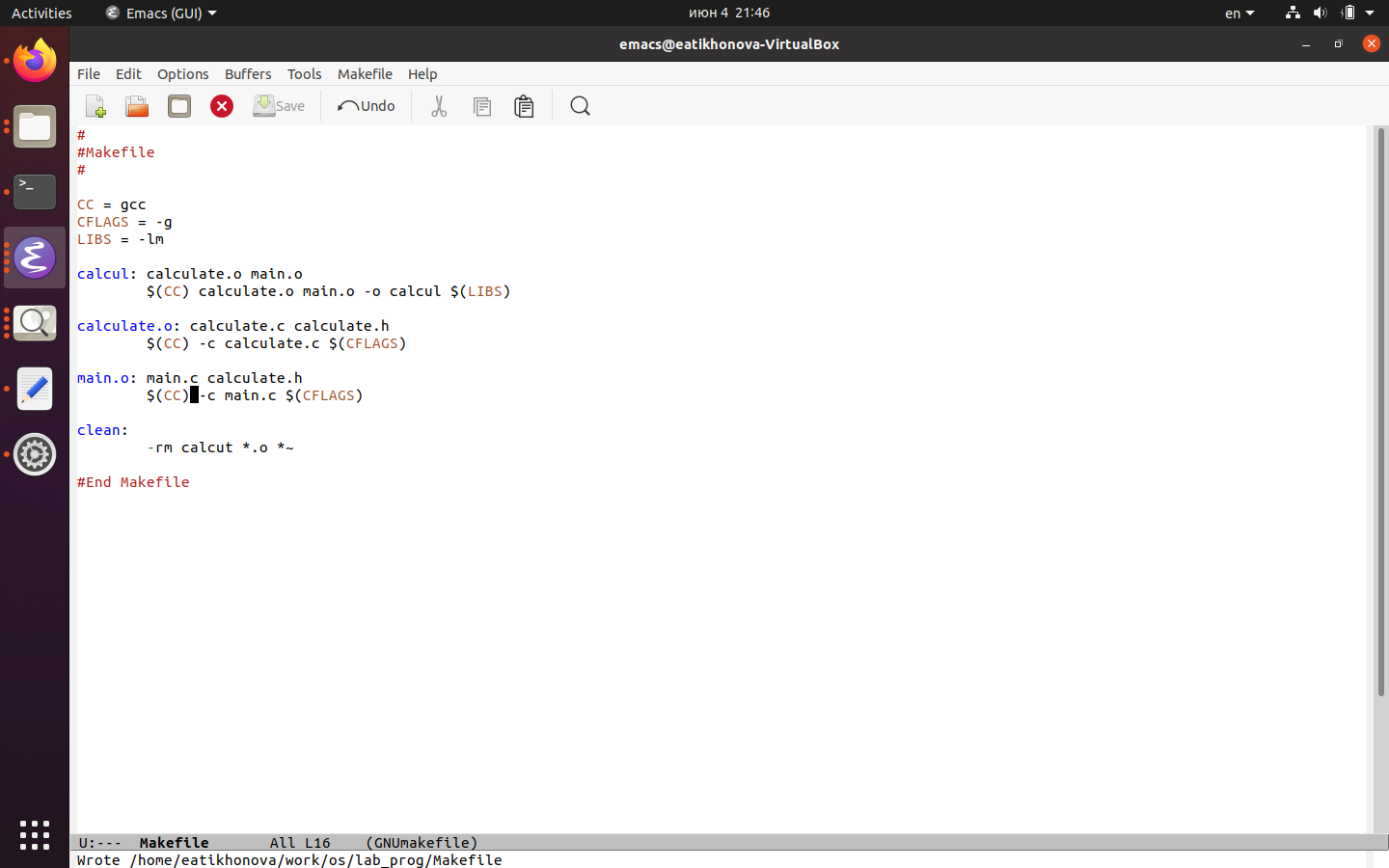


Figure 9: Исправляем файл

Впеременную 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 «makeclear». Выполнила компиляцию файлов, используя команды «makecalculate.o», «makemain.o», «malecalcul»

Figure 10: Выполниляем компиляцию

Figure 10: Выполниляем компиляцию

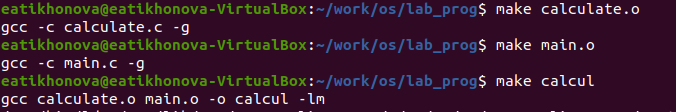


Figure 11: Выполниляем компиляцию

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду:«gdb./calcul»

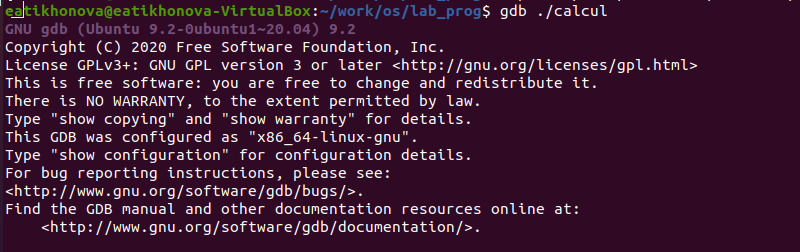


Figure 12: Выполняем отладку

Для запуска программы внутри отладчика ввела команду «run»

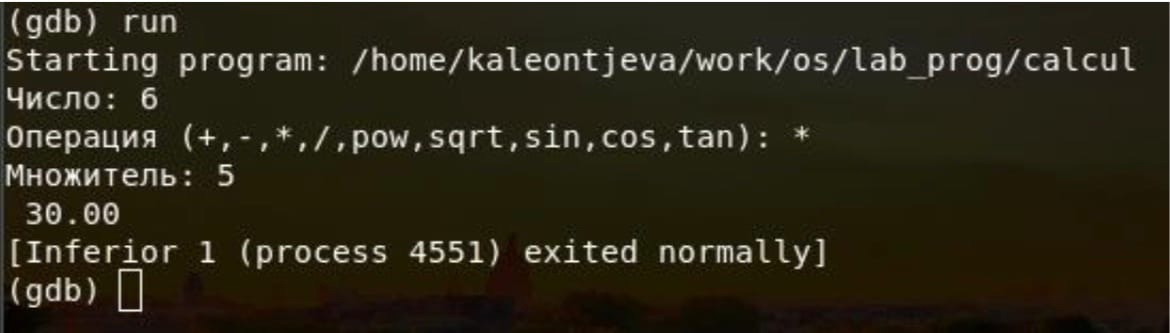


Figure 13: Ввела команду

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list»

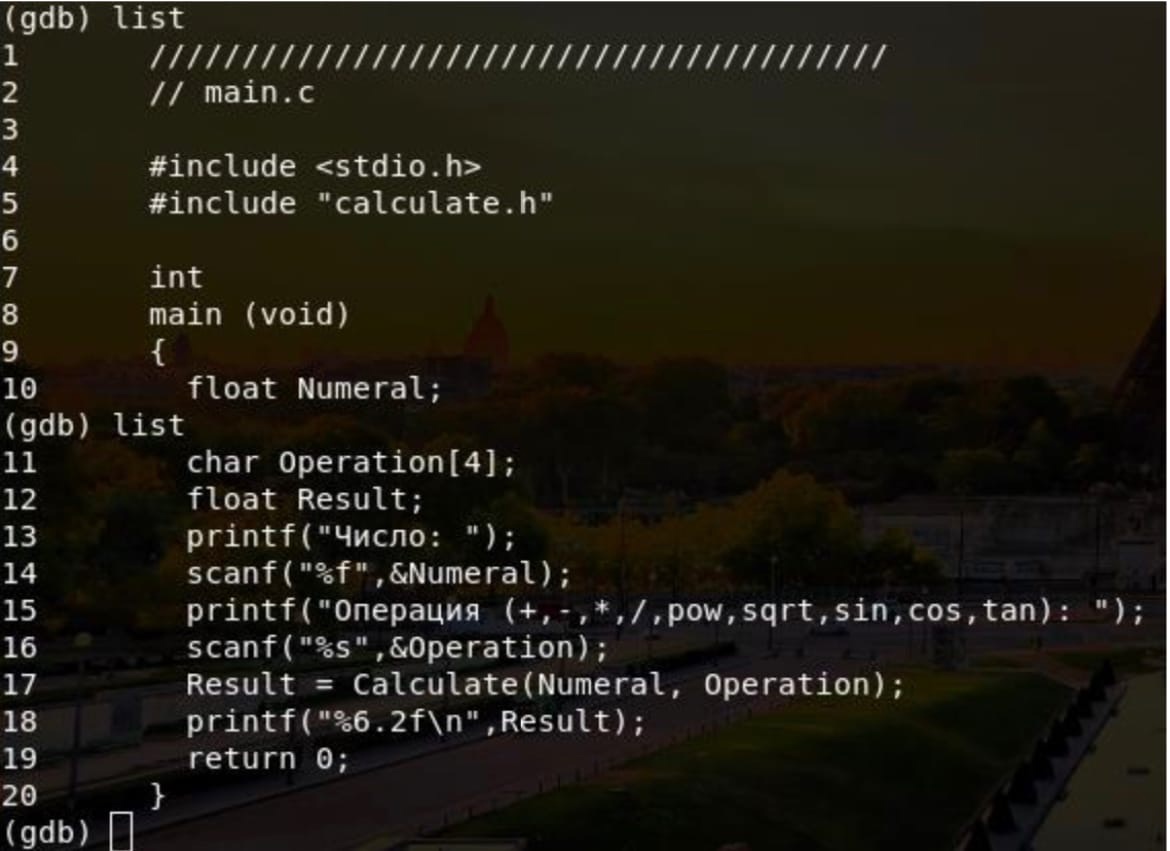


Figure 14: Используем команду

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12, 15»

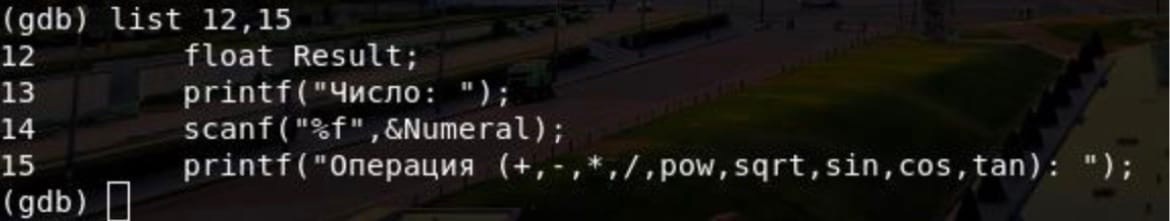


Figure 15: Используем команду

Для просмотра определённых строк неосновного файла использовала команду «listcalculate.c: 20, 29»

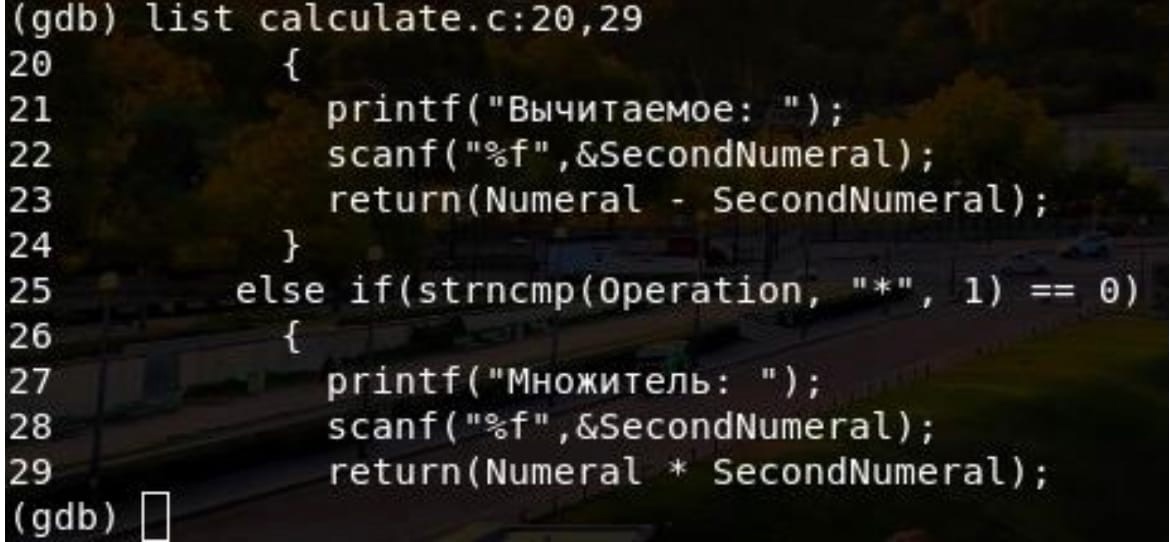


Figure 16: Используем команду

Установила точку останова в файле calculate.c на строке номер 21, используя команды «listcalculate.c: 20, 27» и «break 21»

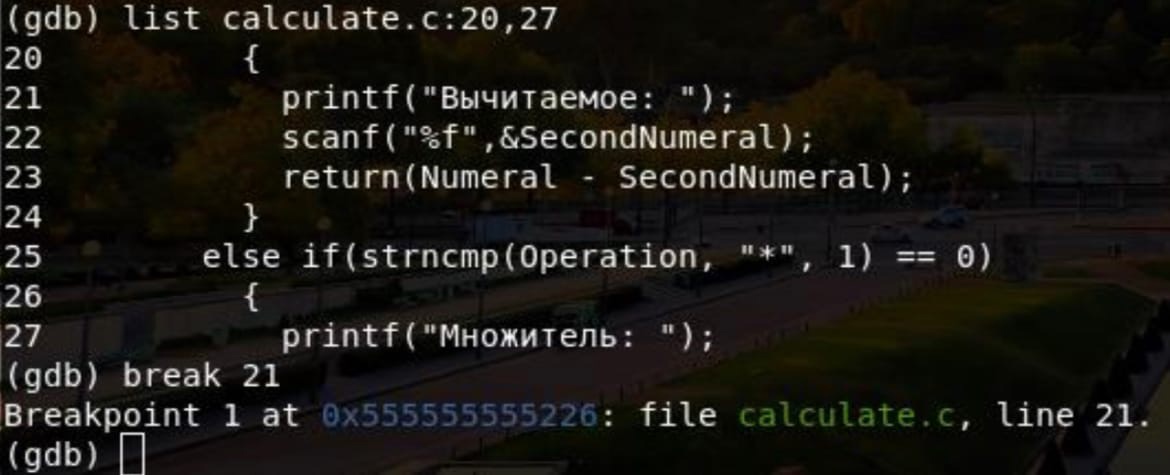


Figure 17: Используем команду

Вывела информацию об имеющихся в проекте точках останова с помощью команды «infobreakpoints»

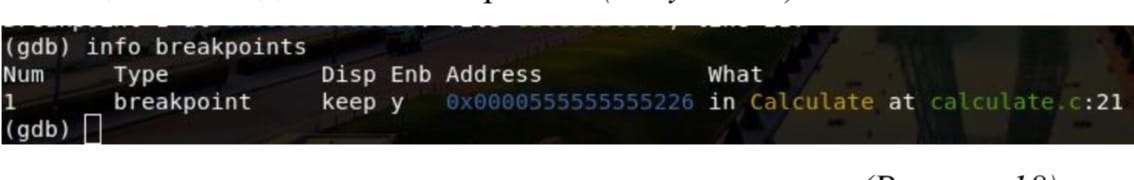


Figure 18: Используем команду

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run»,«5»,«−» и «backtrace»

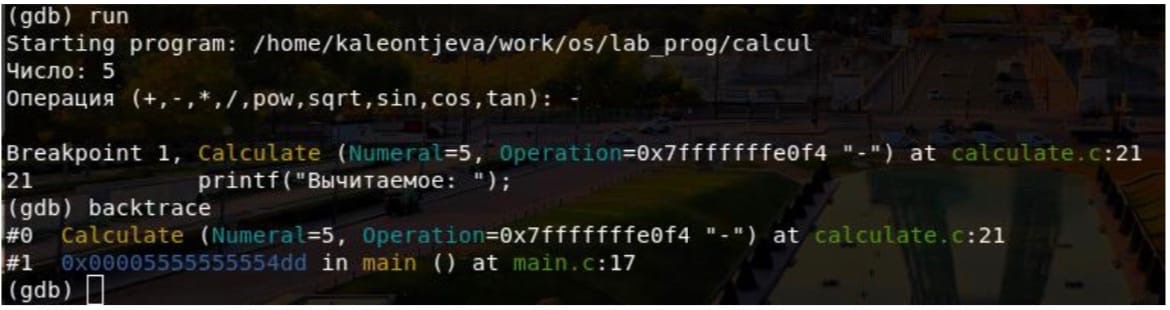


Figure 19: Используем команду

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «printNumeral»

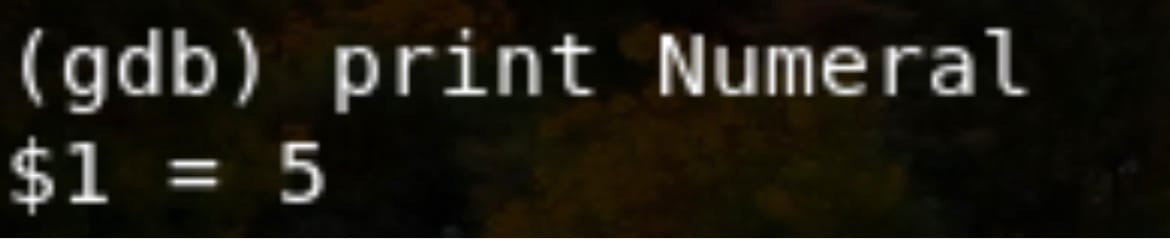


Figure 20: Используем команду

Сравнила с результатом вывода на экран после использования команды «displayNumeral». Значения совпадают

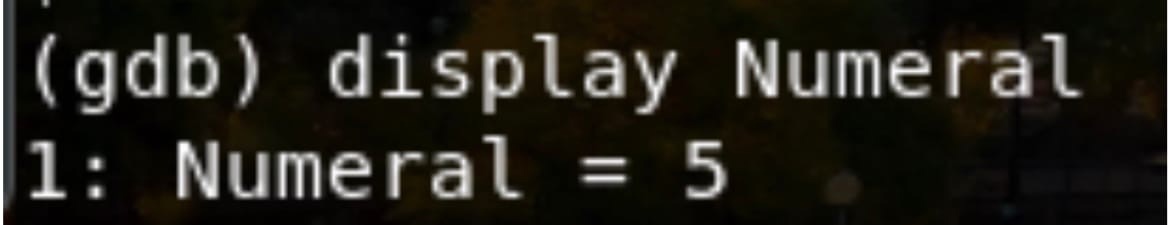


Figure 21: Cравниваем

Убрала точки останова с помощью команд «infobreakpoints» и «delete1»

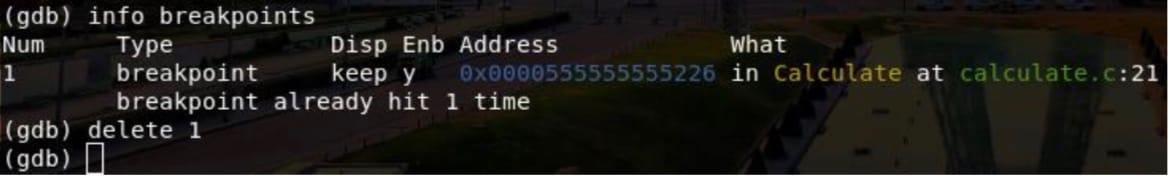


Figure 22: Убрала точки

1. С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Предварительно я установила данную утилиту с помощью команд «sudoaptupdate» и «sudoaptinstallsplint»

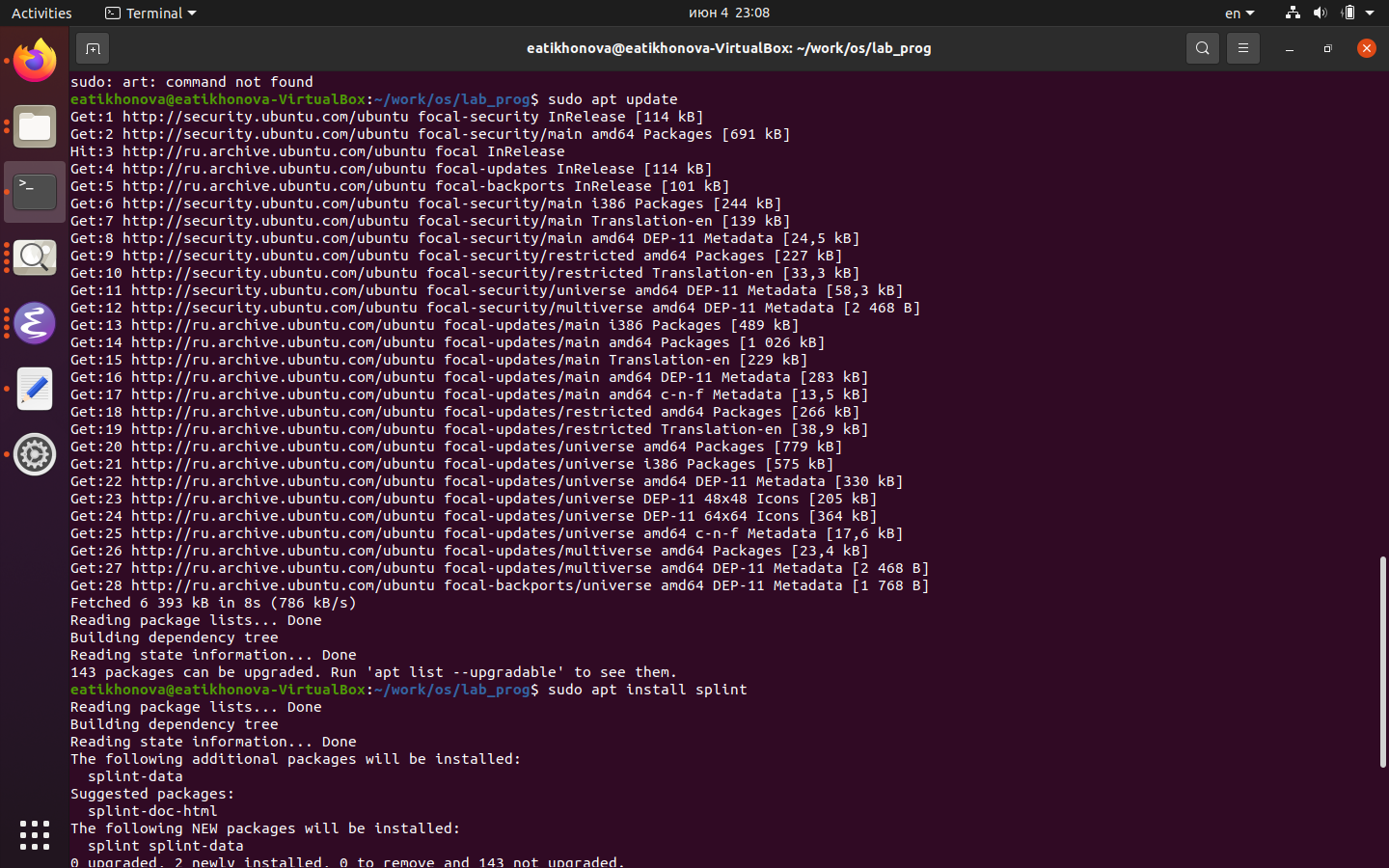


Figure 23: Убрала точки

Далее воспользовалась командами «splintcalculate.c» и «splintmain.c». 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 происходит 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.

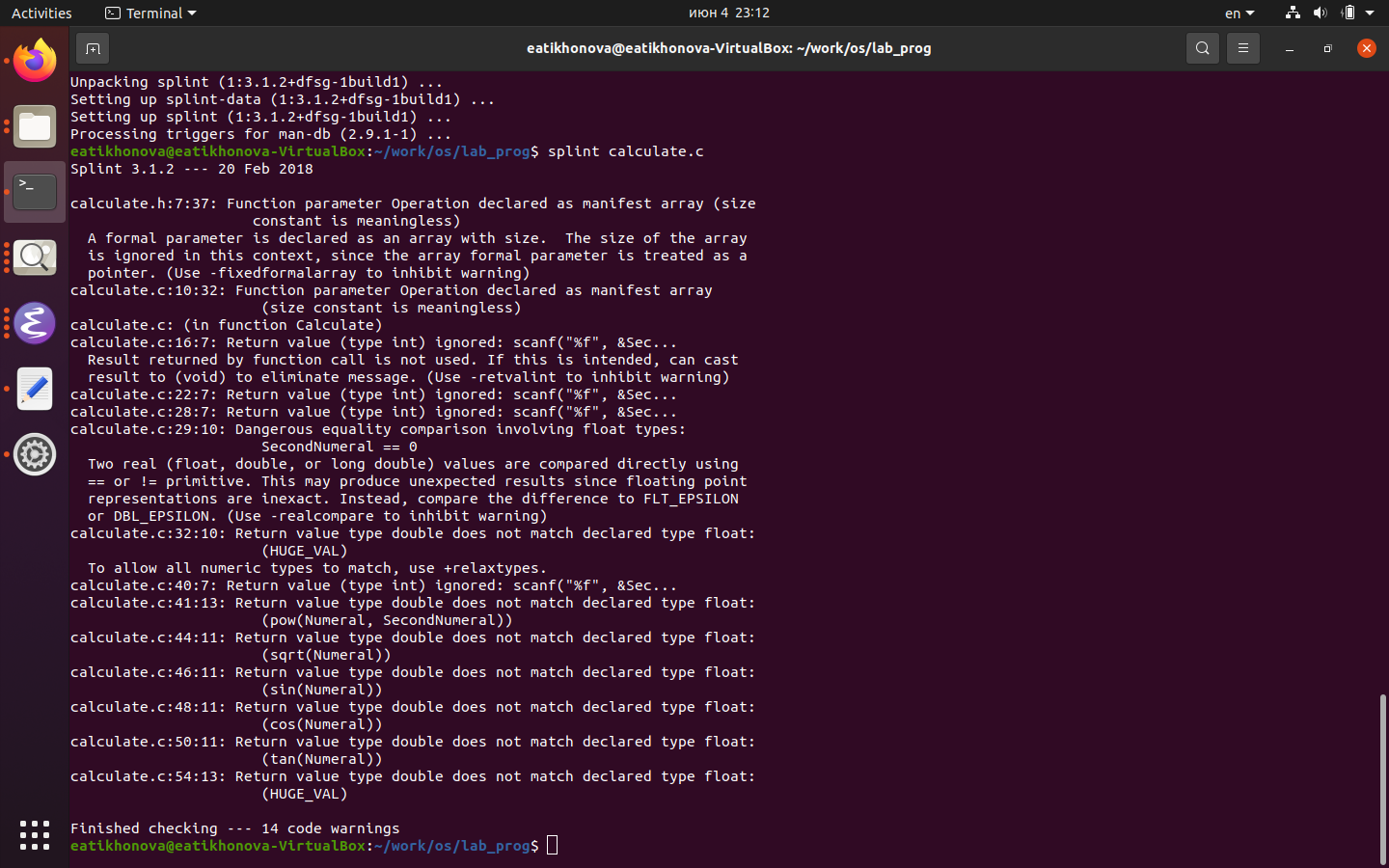


Figure 24: Используем команду

Figure 25: Используем команду

Figure 25: Используем команду

# Контрольные вопросы:

1)Чтобы получить информацию о возможностях программ gcc, make, gdbи др.нужно воспользоваться командой manили опцией -help(-h)для каждой команды. 2)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: oкодирование −по сути создание исходного текста программы (возможно в нескольких вариантах); –анализ разработанного кода; oсборка, компиляция и разработка исполняемого модуля; oтестирование и отладка, сохранение произведённых изменений; 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.3)Дляимени входного файла суффикс определяет какая компиляция требуется. 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.Например, 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-ohellomain.c».4)Основное назначение компилятора языка Си в UNIXзаключается в компиляции всей программы и получении исполняемого файла/модуля.5)Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.6)Для работы с утилитой makeнеобходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefileили Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefileимеет следующий синтаксис: … : …<команда 1>…Сначала 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. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды − собственно действия, которые необходимо выполнить для достижения цели.Общий синтаксис 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 обрабатываться. 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 последовательных 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.cВ этом примере в начале файла заданы три переменные: CCи CFLAGS. Затем указаны цели, их зависимости 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.7)Во время работы над кодом программы программист неизбежно сталкивается 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 использования GDBнеобходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -gкомпилятора gcc: gcc-cfile.c-gПосле этого для начала работы с gdbнеобходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdbfile.o8)Основные команды отладчика gdb:backtrace−вывод на экран пути к текущей точке останова (по сутивывод − названий всех функций)break −установить точку останова (в качестве параметра можетбыть указан номер строки или название функции)clear −удалить все точки останова в функцииcontinue −продолжить выполнение программыdelete −удалить точку остановаdisplay −добавить выражение в список выражений, значения которых отображаются при достижении точки останова программыfinish −выполнить программу до момента выхода из функцииinfo breakpoints −вывести на экран список используемых точек остановаinfo watchpoints −вывести на экран список используемых контрольных выраженийlist −вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальнойи конечной строк)next −выполнить программу пошагово, но без выполнения вызываемых в программе функций print −вывести значение указываемого в качестве параметра выраженияrun −запуск программы на выполнениеset −установить новое значение переменнойstep −пошаговое выполнение программыwatch −установить контрольное выражение, при изменении значения которого программа будет остановленаДля выхода из gdbможно воспользоваться командой quit(или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdbможно получить с помощью команд gdb-hи mangdb. 9)Cхема отладки программы показана в 6 пункте лабораторной работы.10)При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.cдопущена ошибка, которую компилятор мог пропустить (возможно, 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. 11)Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:cscope −исследование функций, содержащихся в программе,lint −критическая проверка программ, написанных на языке Си.12)Утилита splintанализирует программный код, проверяет корректность задания 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 значения не используются в работе программы, переменные с некорректно заданными значениямии типами и многое другое.4

# Выводы

В 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.