Лабораторная работа №14

Дисциплина: Операционные системы

Колчева Юлия вячеславовна

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

В домашнем каталоге создаю подкаталог ~/work/os/lab prog. Создала в каталогеф файлы: calculate.h, calculate.c, main.c, используя команды «cd~/work/os/lab\_prog» и «touchc alculate.hcalculate.c main.c» (рис. 1)

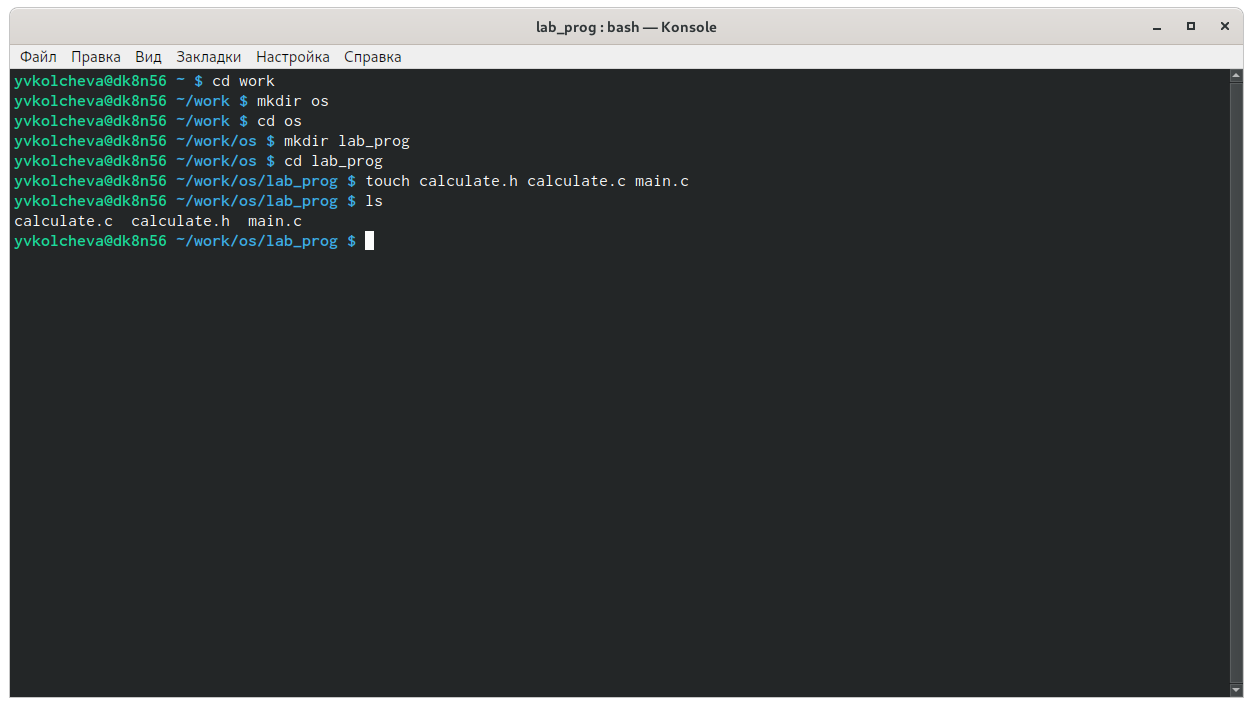


Figure 1: Создание

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 (рис. 2) (рис. 3)

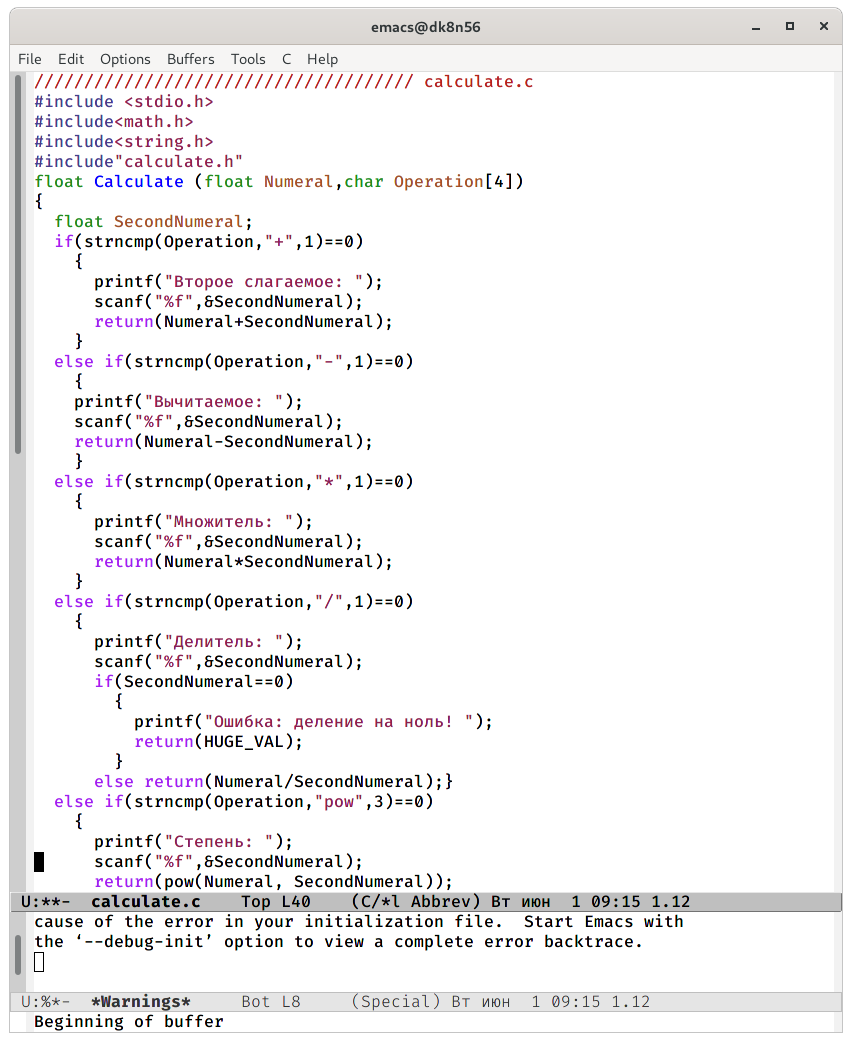


Figure 2: Первая часть

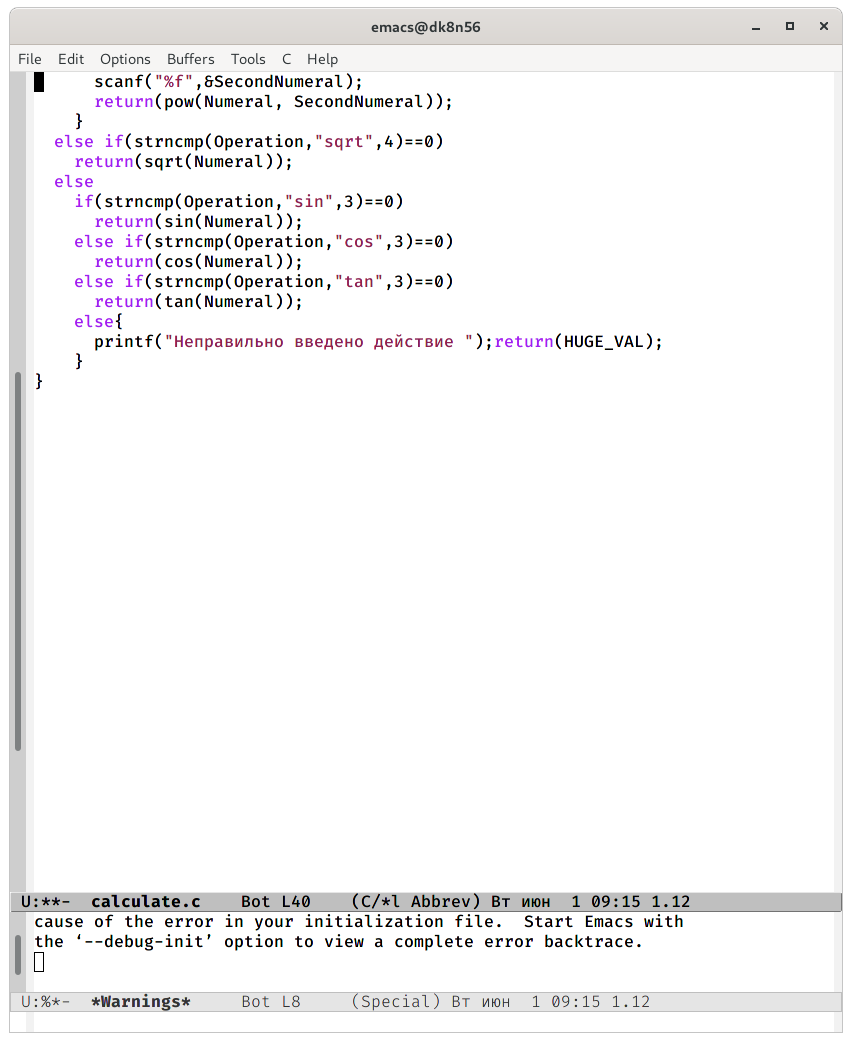


Figure 3: Вторая часть

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора(рис. 4)

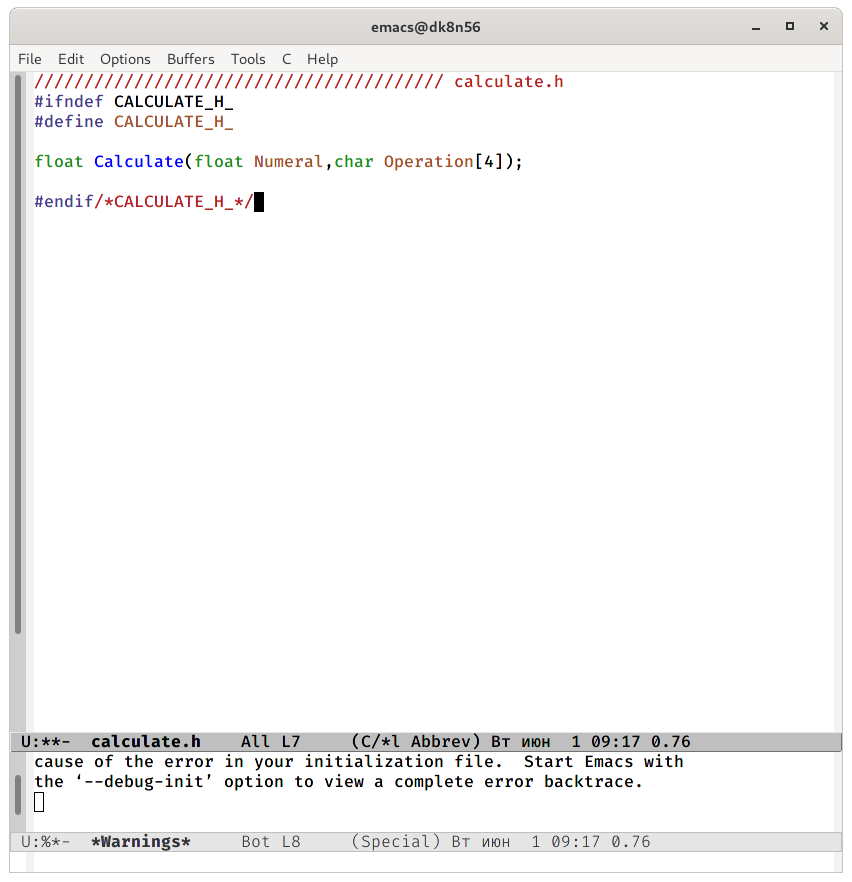


Figure 4: Файл1

Основной файл main.c,реализующий интерфейс пользователя калькулятору(рис. 5)

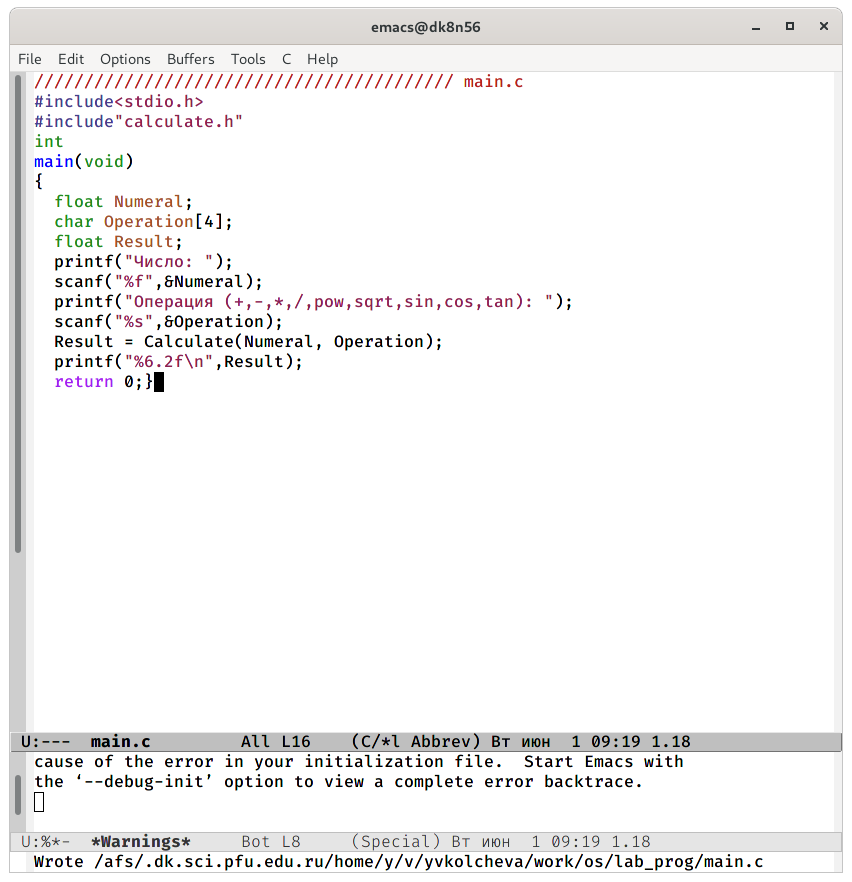


Figure 5: Файл2

Выполнила компиляцию программы посредством gcc ,используя команды «gcc-ccalculate.c», «gcc -c main.c» и « gcc calculate.o main.o -o calcul-lm» (рис. 6)

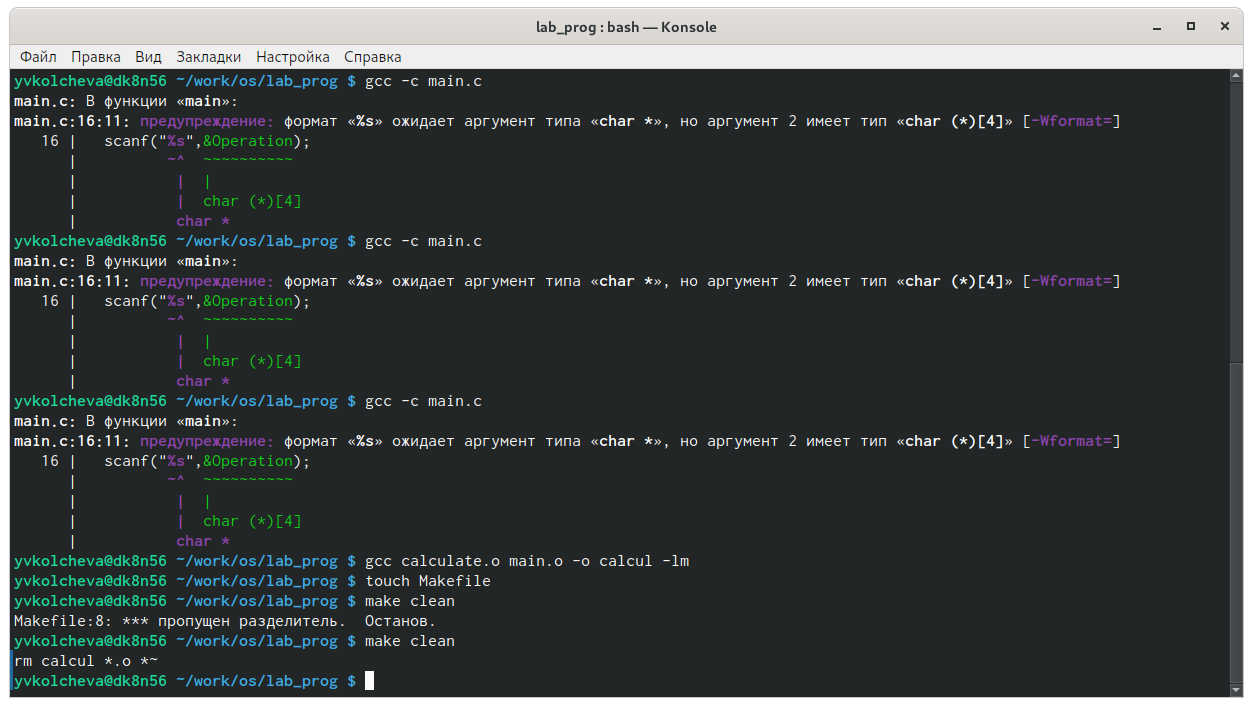


Figure 6: Компиляция

В ходе компиляции программы никаких ошибок не было, только предупреждение, не влияющее на работу программы. Создала Makefile (рис. 7)

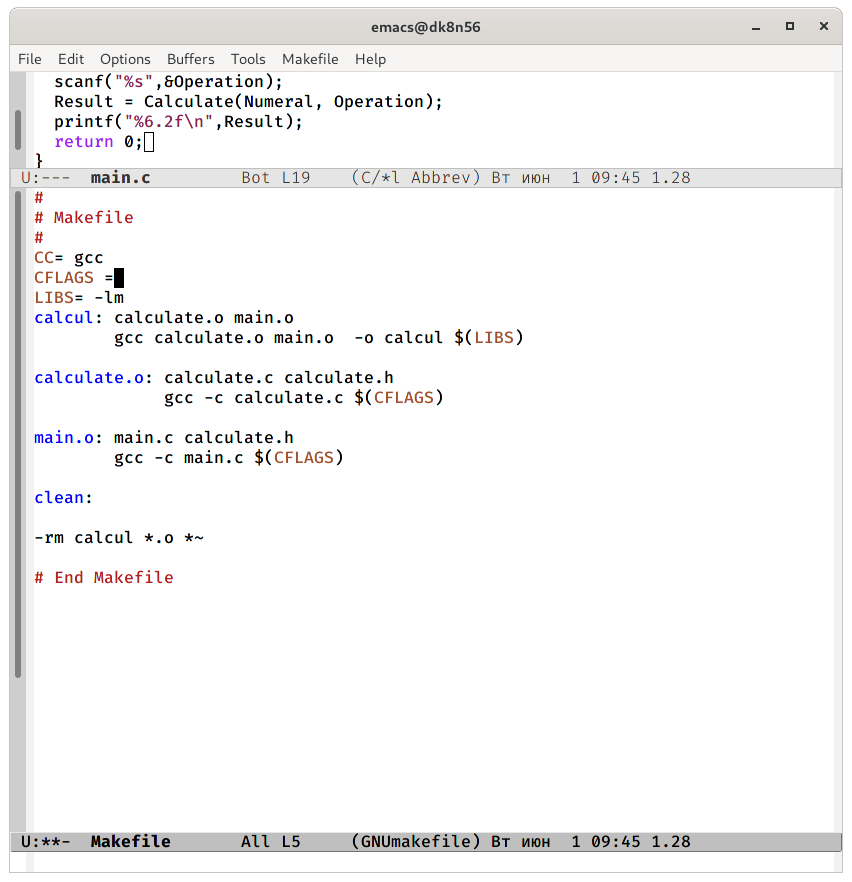


Figure 7: Makefile

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. Далее исправила Makefile (рис. 8)

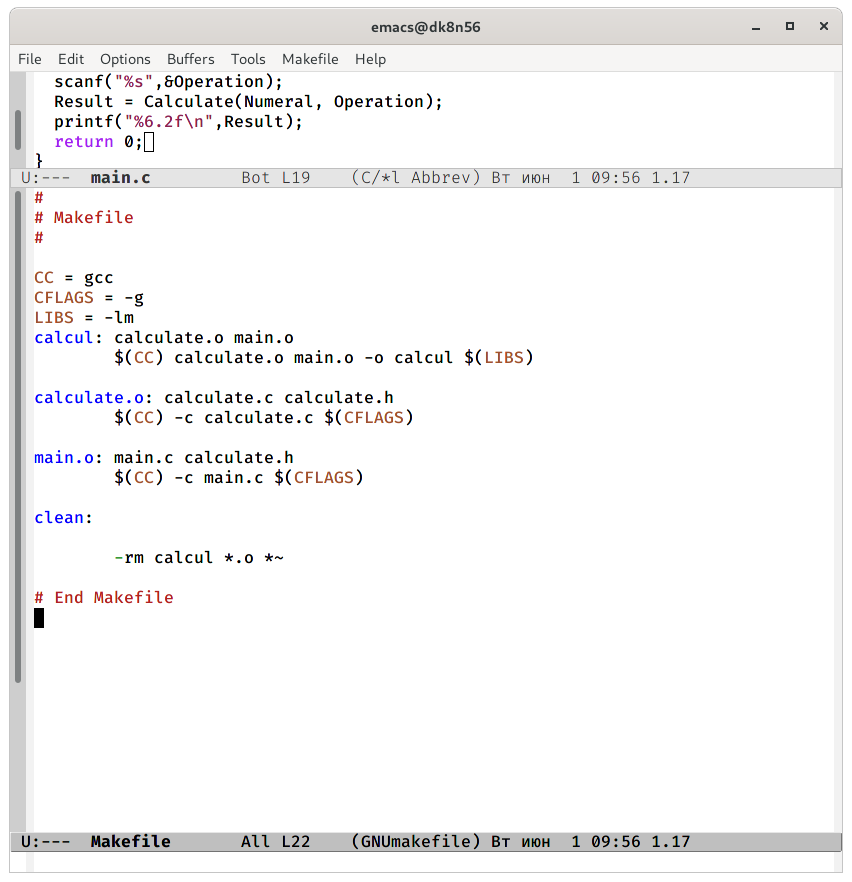


Figure 8: Makefile2

В переменную CFLAGS добавила опцию -g,необходимую для ккомпиляции 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».Выполнила компиляцию файлов,используя команды «make calculate.o»,«make main.o»,«make calcul» (рис. 9)

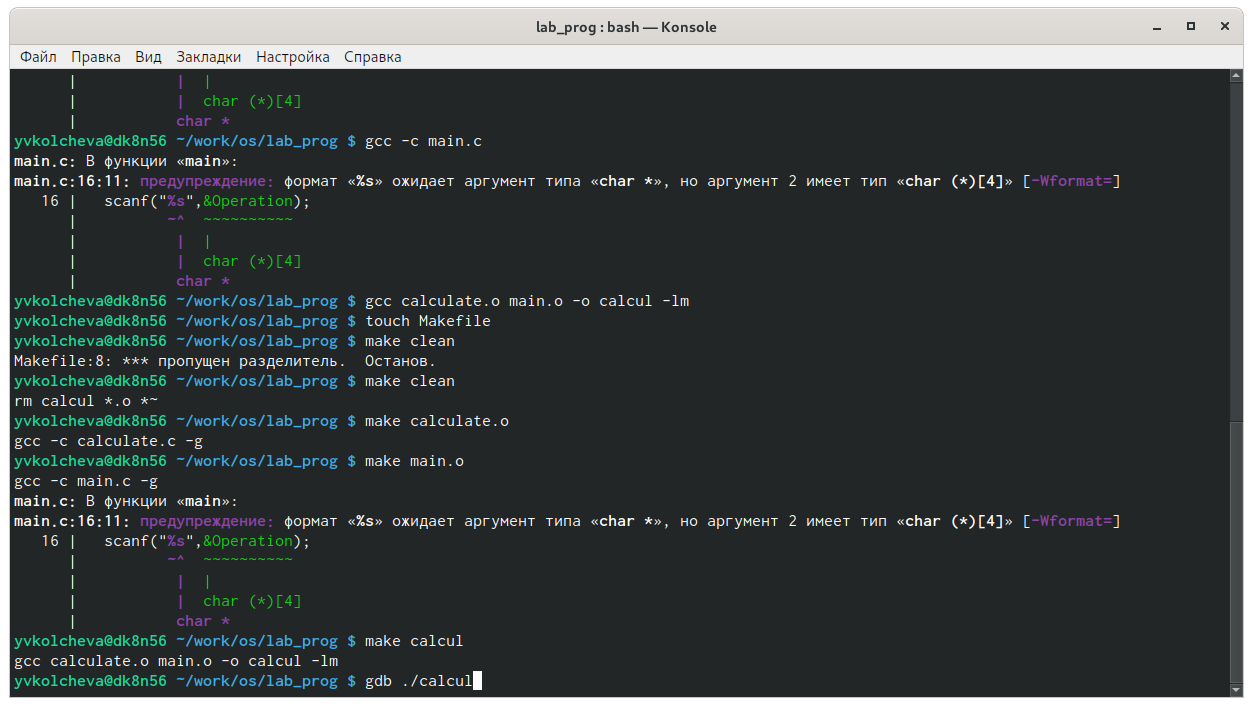


Figure 9: Работа в консоли

Далее с помощью gdb выполнила отладку программы calcul.Запустила отладчик GDB,загрузив в него программу для отладки,используя команду:«gdb./calcul».(рис. 10)

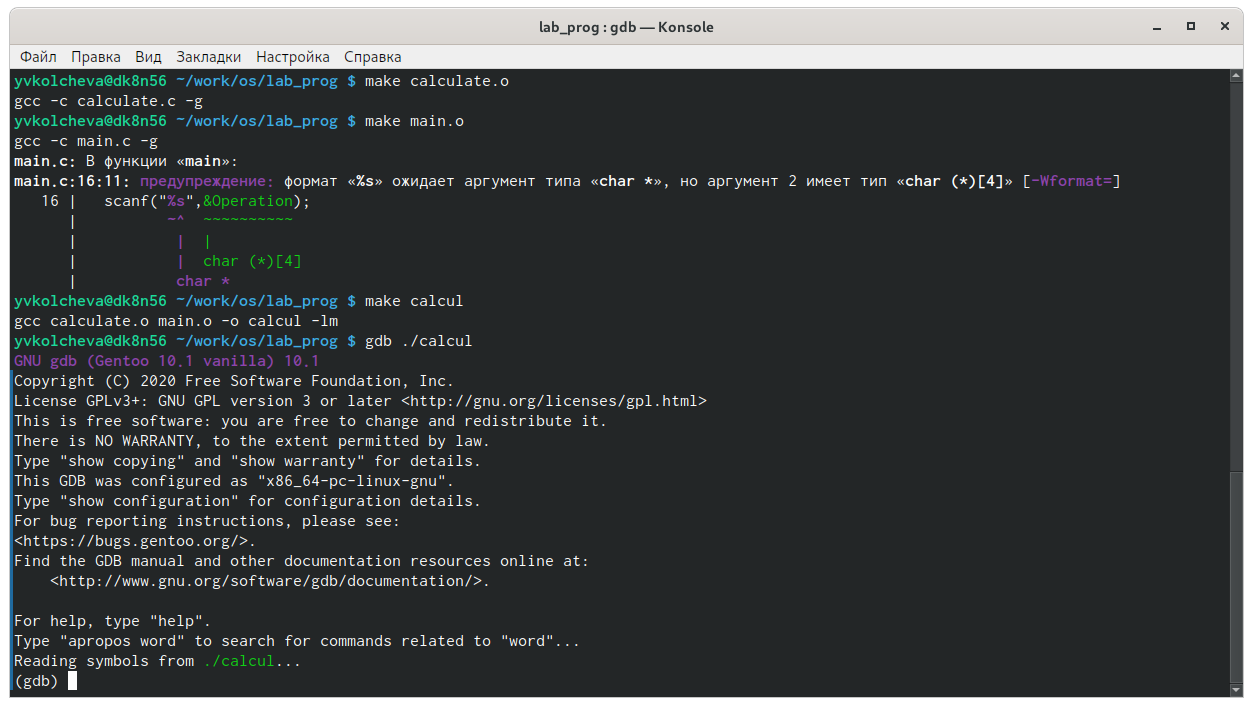


Figure 10: отладчик

Для запуска программы внутри отладчика ввела команду «run».(рис. 11)

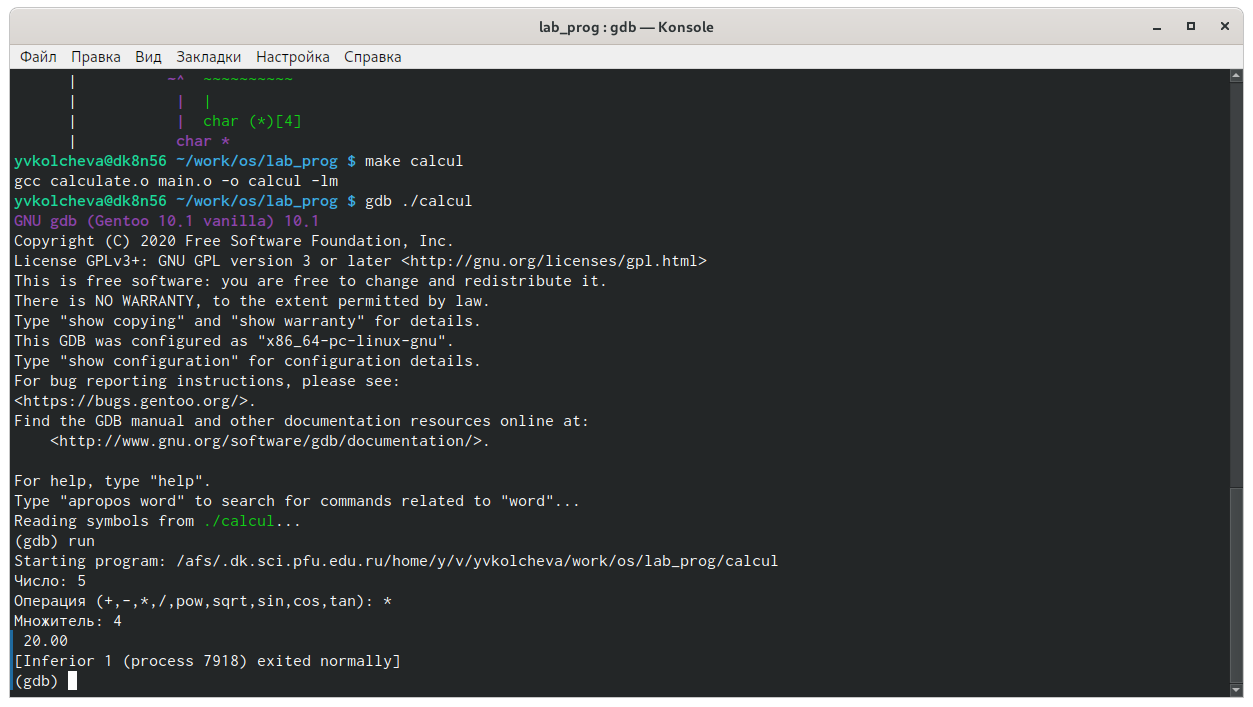


Figure 11: Работа программы

Для постраничного (по 10 строк)просмотра исходного кода использовала команду «list» (рис. 12)

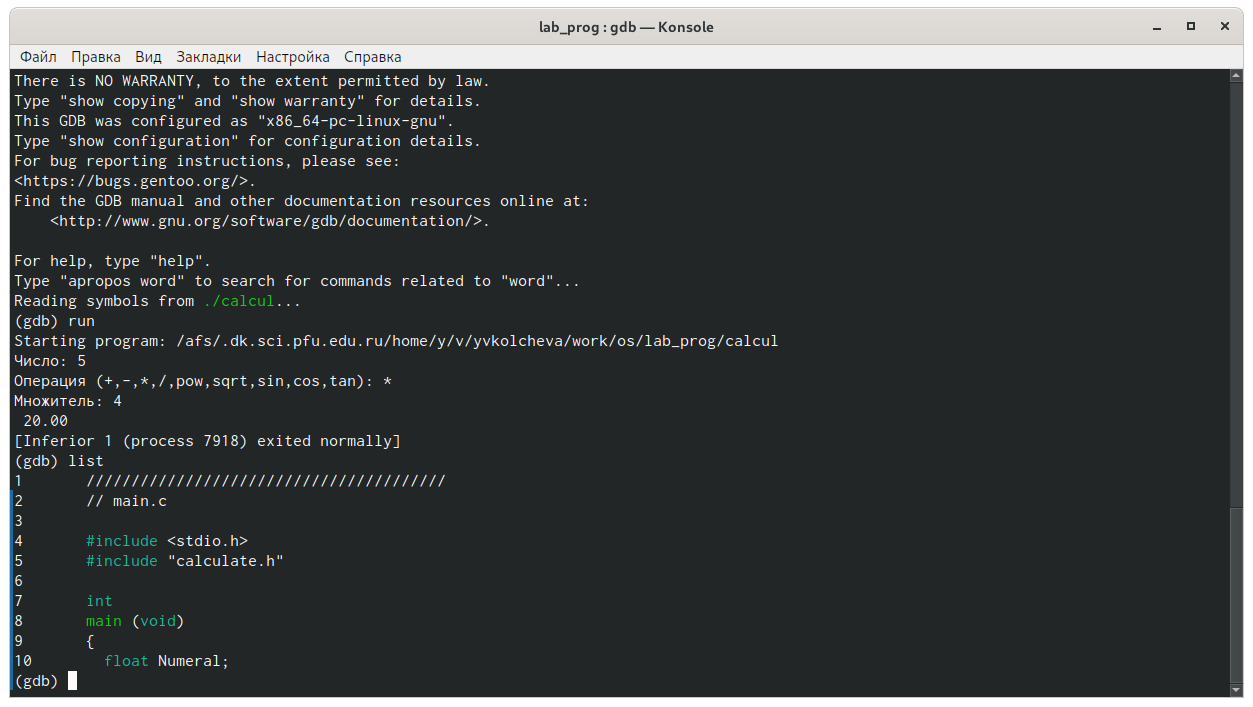


Figure 12: Команда list

Для просмотра строк с 13 по 15 основного файла использовала команду «list 13,15» (рис. 13)

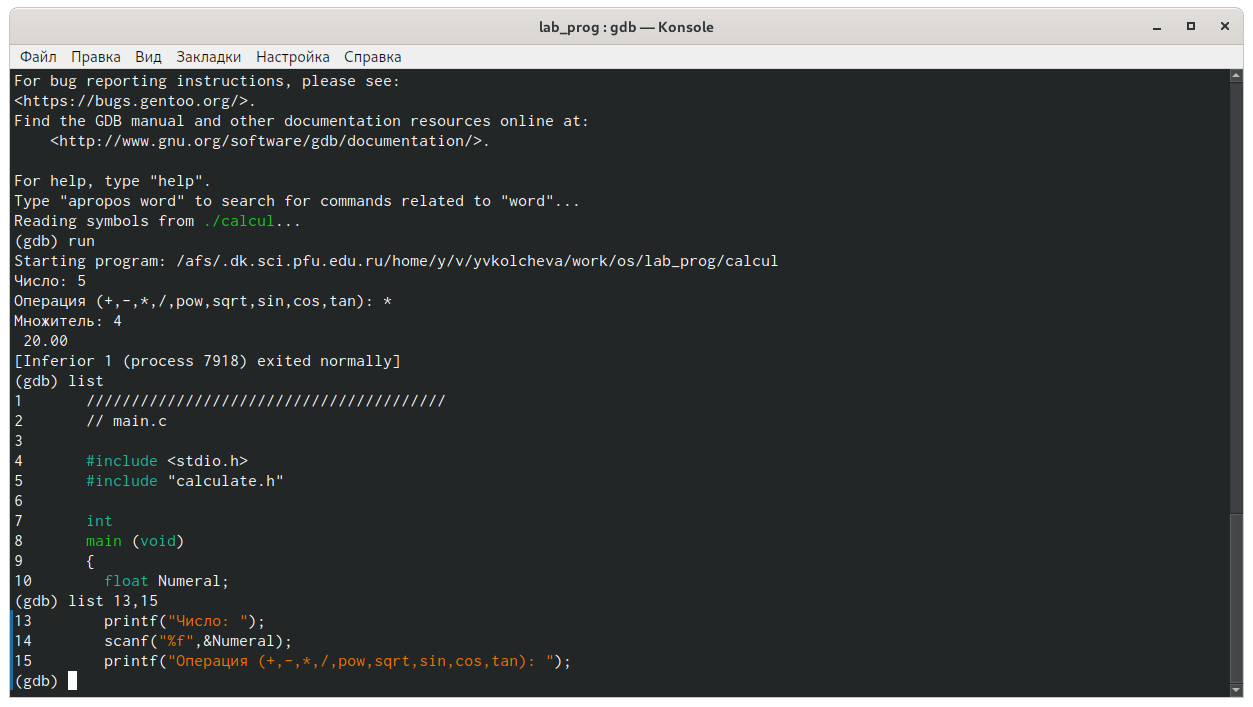


Figure 13: Команда list2

Для просмотра определённых строк неосновного файла использовала команду «listcalculate.c:2,6» (рис. 14)

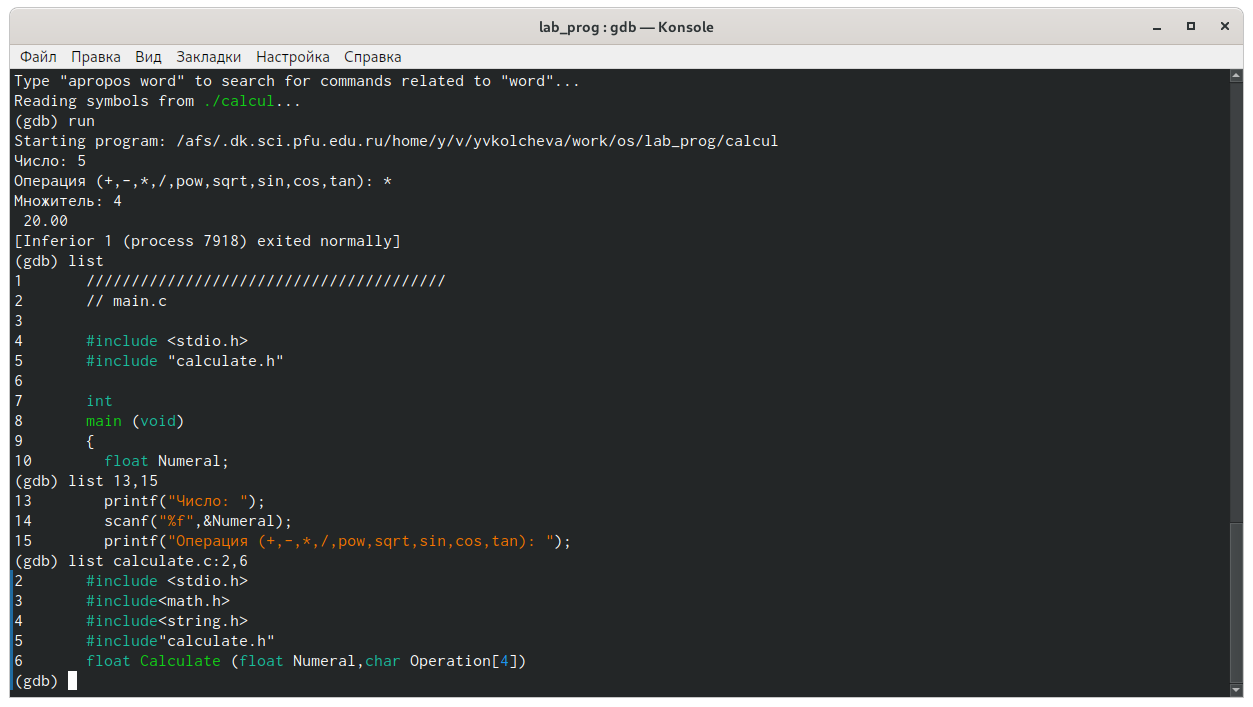


Figure 14: Команда list3

Установила точку останова в файле calculate.c на строке номер 8, (рис. 15)

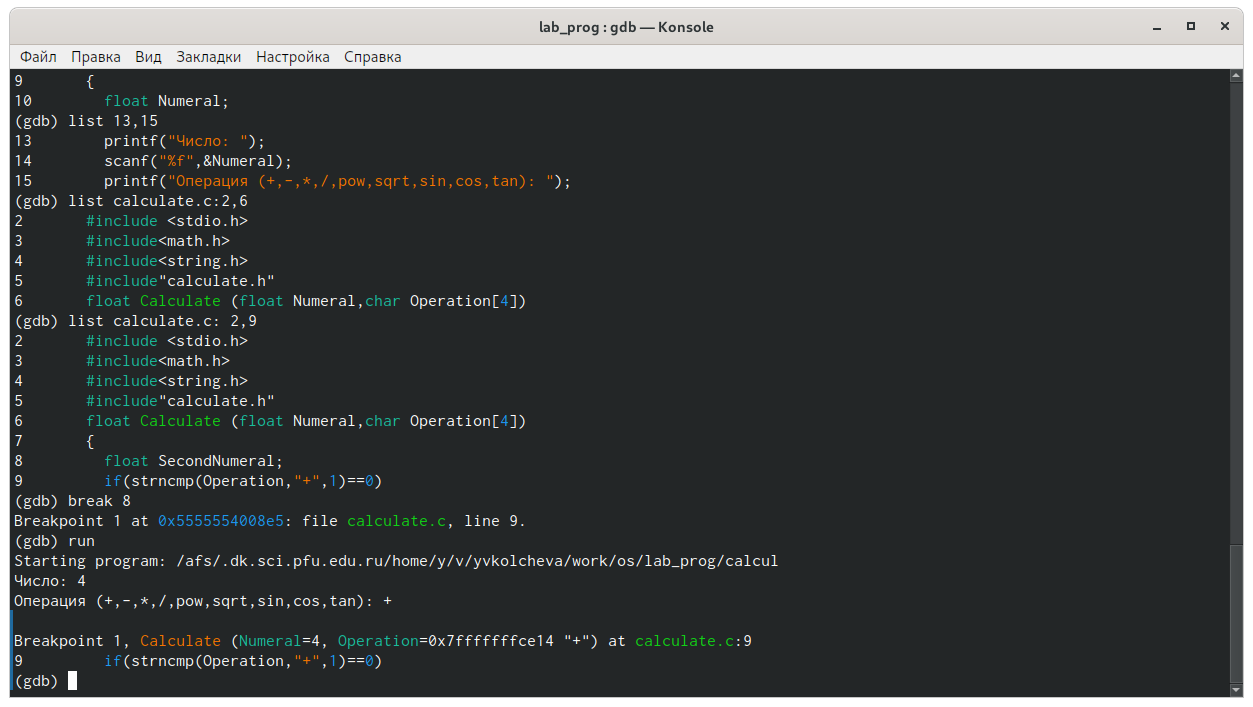


Figure 15: Точка останова

Запустила программу внутри отладчика и убедилась,что программа остановилась в момент прохождения точки останова.Посмотрела,чему равно на этом этапе значение переменной Numeral,введя команду «printNumeral» Сравнила с результатом вывода на экран после использования команды «display Numeral».Значения совпадают. Убрала точки останова с помощью команд «info breakpoints»и «delete 1»(рис. 16)

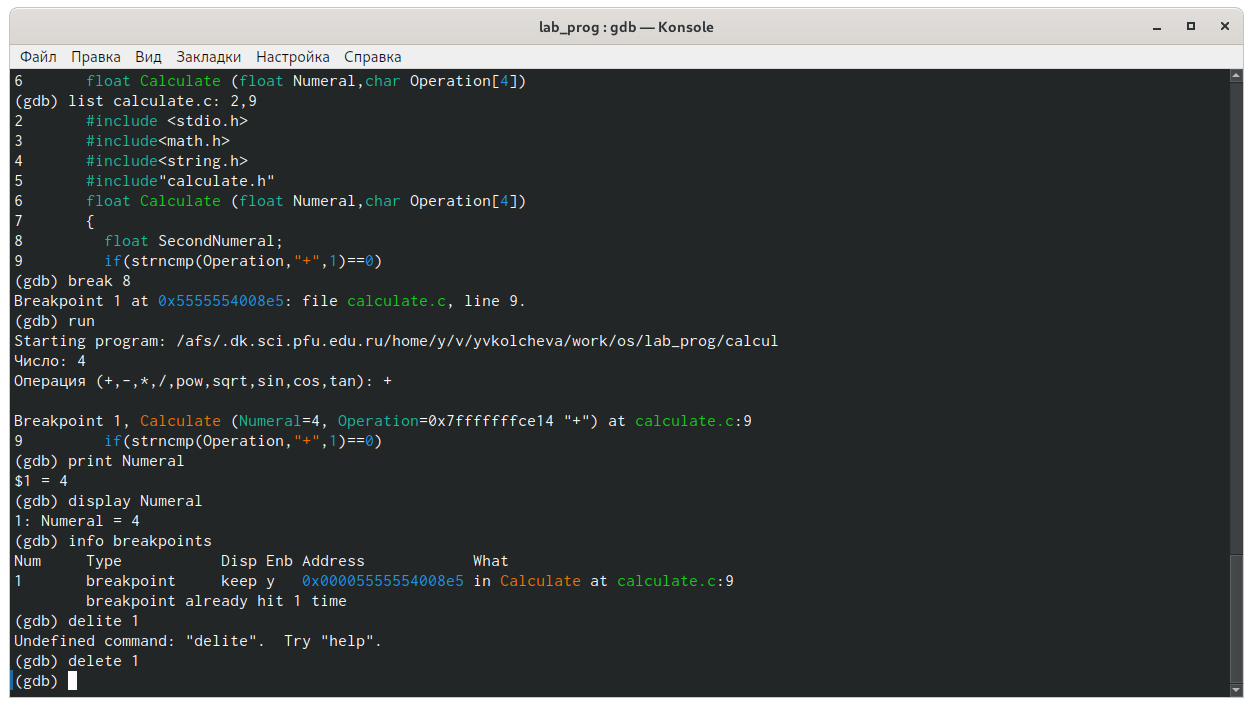


Figure 16: Запуск

Далее воспользовалась командами «splint calculate.c» и «splint main.c» (рис. 17) (рис. 18)

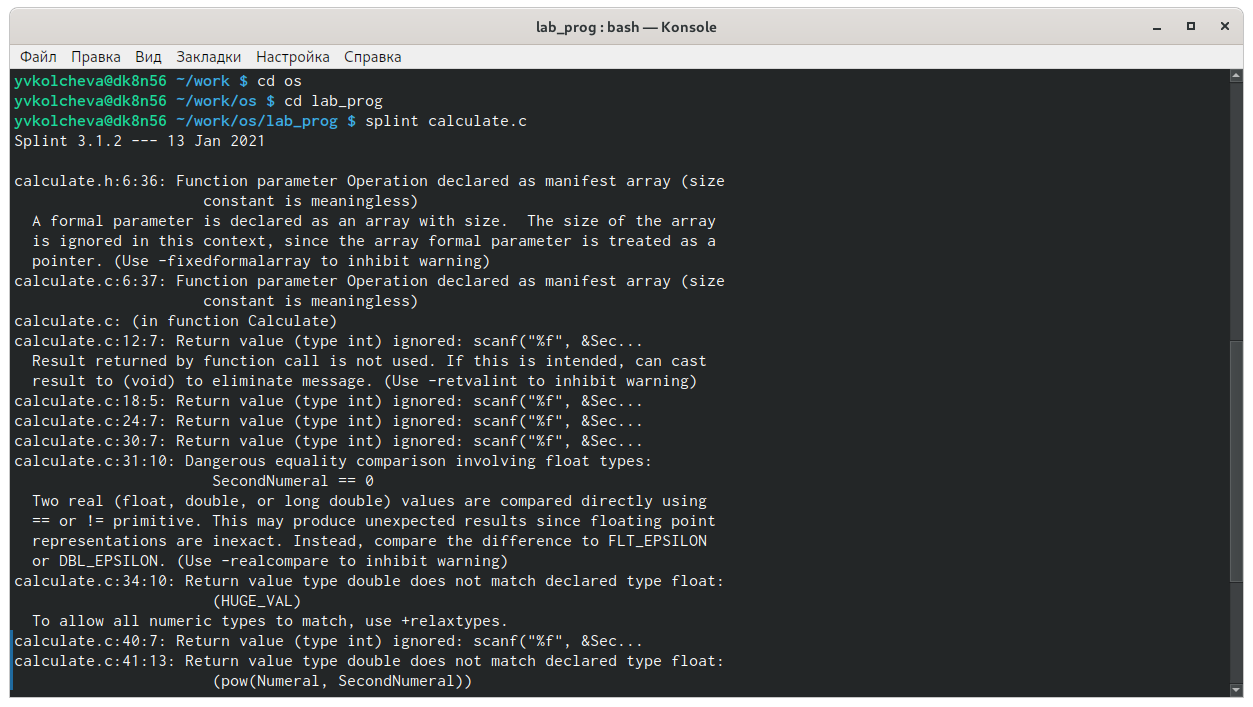


Figure 17: splint calculate.c

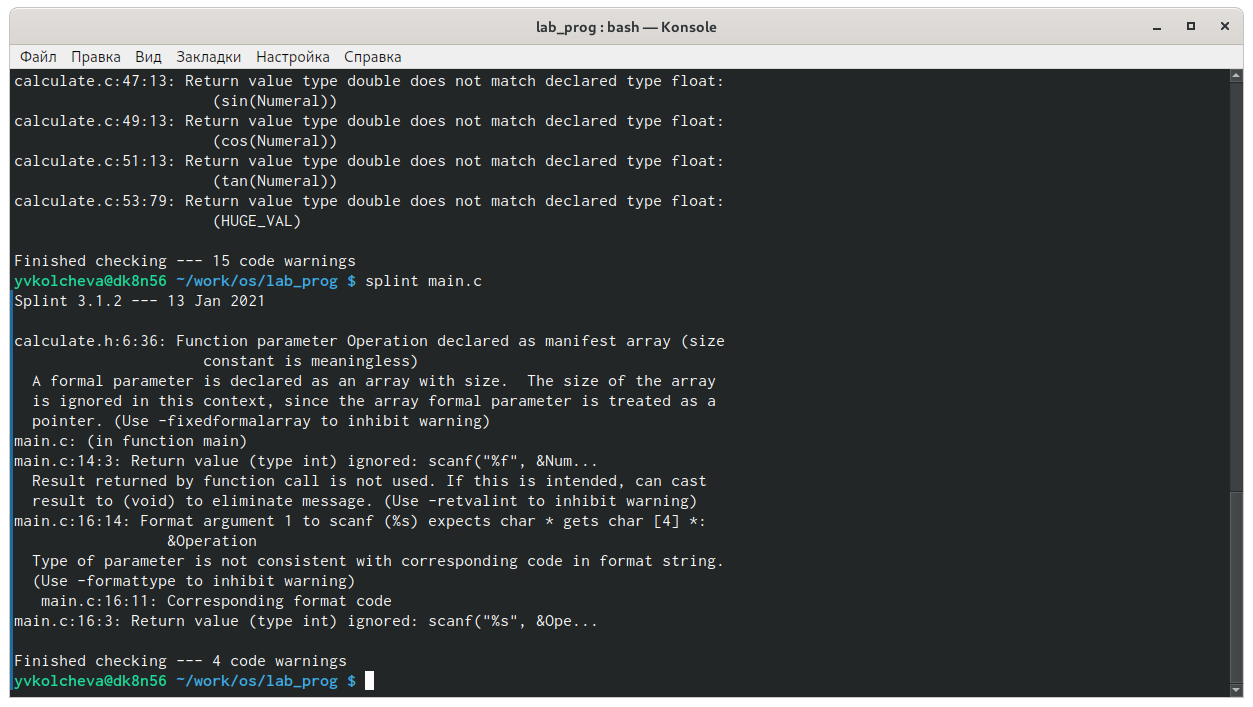


Figure 18: splint main.c

# 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.

# 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.o 8)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 друг

# Библиография

Лекция Кудрявцева https://esystem.rudn.ru/mod/url/view.php?id=718563

Лабораторная работа №14 https://esystem.rudn.ru/mod/resource/view.php?id=718613