Лабораторная работа 13

Отчет по лабораторной работе 13

Репина Ангелина Олеговна

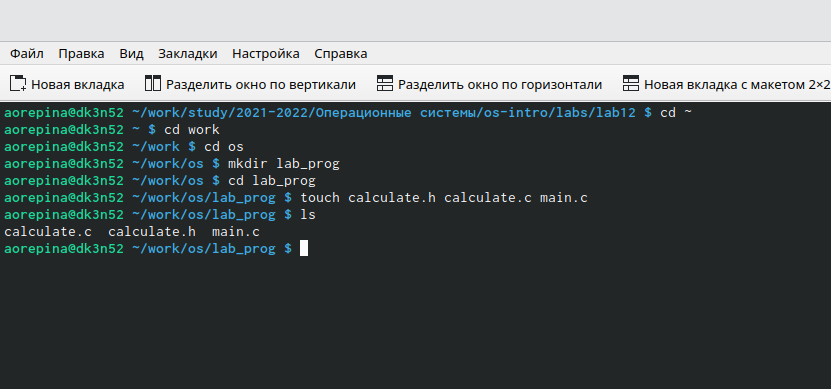
Содержание

# Цель работы

Цель данной лабораторной работы - приобретение простейших навыков разработки, анализа, тестирования и отладки приложений в ОС типа Linux на примере создания на языке программирования C калькулятора с простейшими функциями

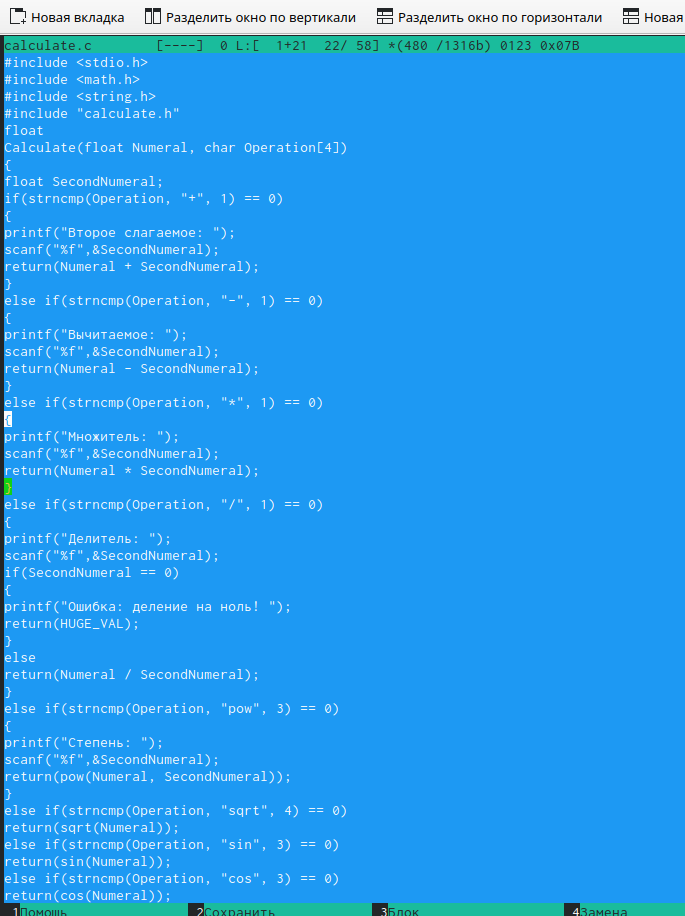
# Выполнение лабораторной работы

1. В домашнем каталоге создала подкаталог ~/work/os/lab\_prog и в нем уже создала три файла calculate.h, calculate.c, main.c (рис 1). 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.

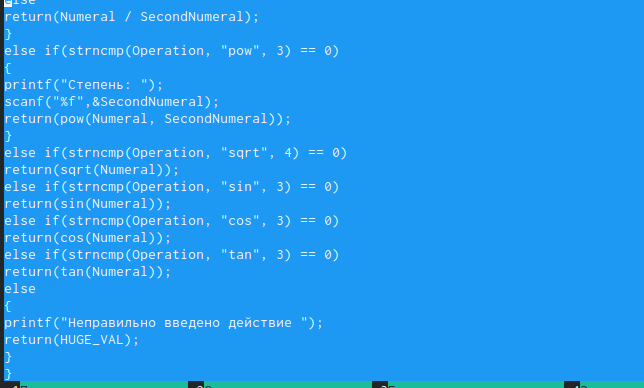


1: Создание каталогов и файлов

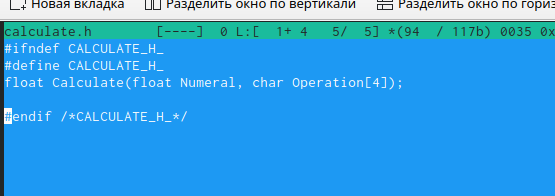
1. В созданных файлах написала программы для работы калькулятора, которые были предоставлены (рис. 2, 3, 4, 5)



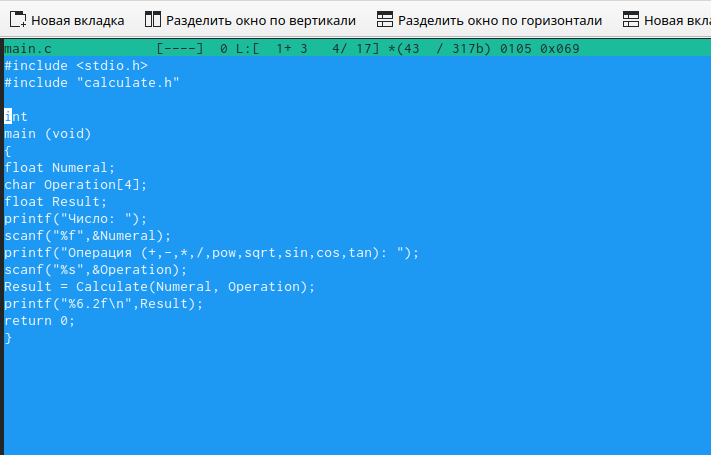
2: calculate.c



3: calculate.c

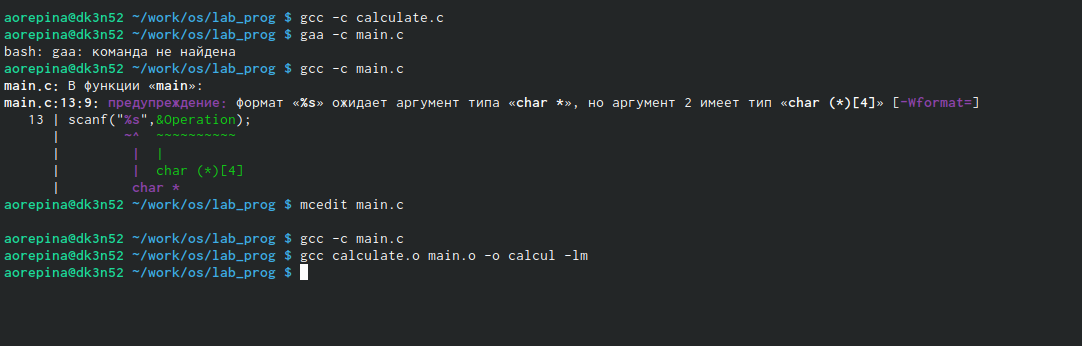


4: calculate.h



5: main.c

1. Выполнила компиляцию программы посредством gcc и при необходимости исправила синтаксис (рис 6)

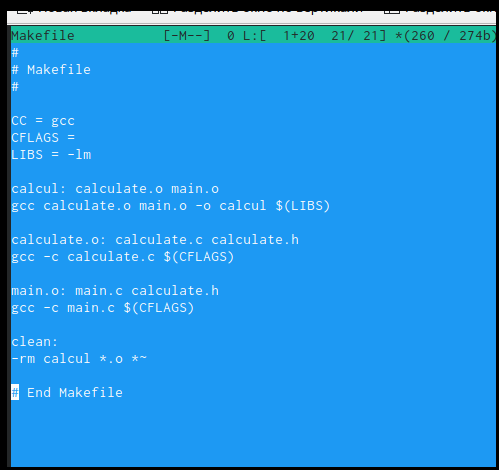


6: Компиляция

1. Создала Makefile (рис 7) и ввела в него предложенное содержимое (рис 8)

7: Создала Makefile

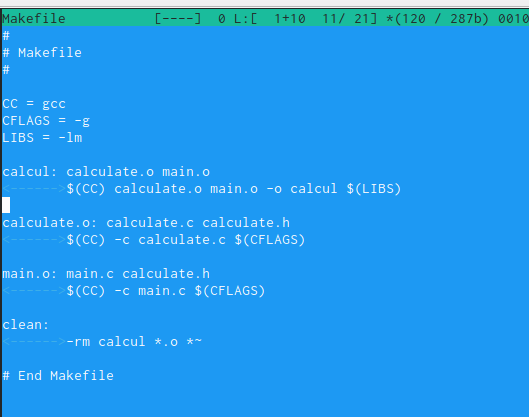
7: Создала Makefile



8: Makefile

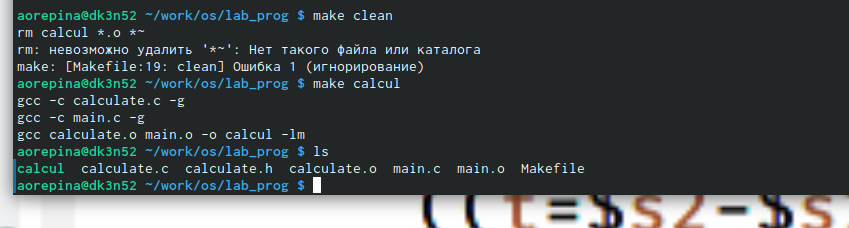
Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (main.o цель), а также их объединения в один исполняемый файл calcul. Claen нужно для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

1. Далее исправила Makefile (рис 9). В переменную CFLAGS добавила опцию -g необходимую для компиляции объектных файлов и их использования в программе отладчика GDВ. Сделаем так, что утилита компиляции выбирается с помощью перемнной CC



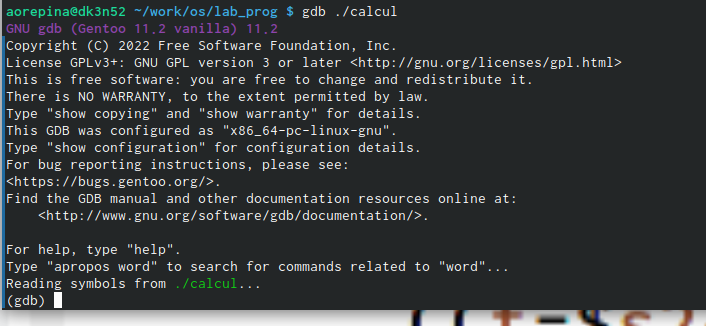
9: Makefile исправленный

После этого удалила исполняемые файлы (make clean) и выполнила компиляцию файлов, используя команды make calculate.o, make main.o, make calcul (рис 10)



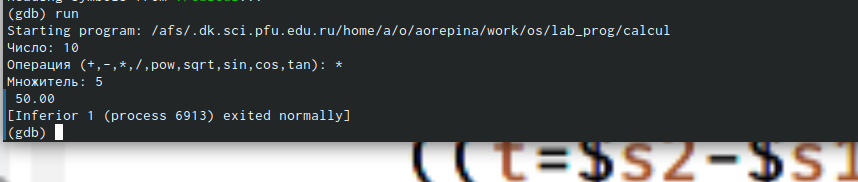
10: clean and make

1. Далее с помощью комианды gdb ./calcul запустила отладку программы (рис 11)



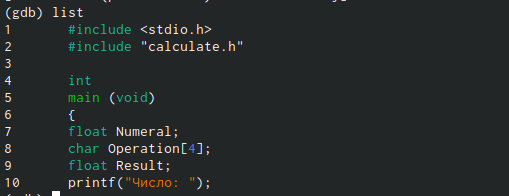
11: отладка

Для запуска программы внутри отладчика ввела команду run (рис 12)



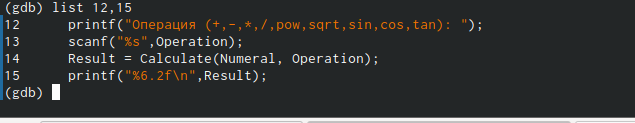
12: run

Для постраничного (по 9 строк) просмотра исходного кода использовала команду list (рис 13)



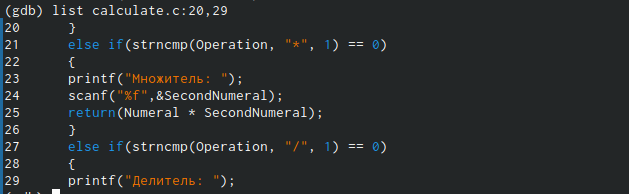
13: list

Для просмотра строк с 12 по 15 основного файла использовала list с параметрами (рис 14)



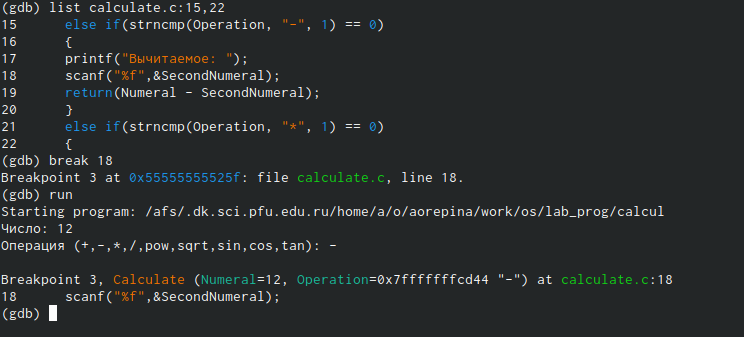
14: list с параметрами

Для просмотра опрееленных строк не основного файла использовала list с параметрами (рис 15)



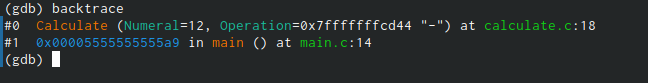
15

Установила точкеу останова в файле calculate.c на строке 18 и вывела информацию об имеющихся в проекте точках (рис 16)



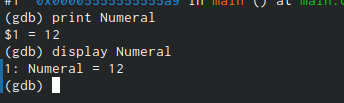
16: точки останова

Ввела команду backtrace которая показала весь стек вызываемых функций от начала программы до текущего места (рис 17)



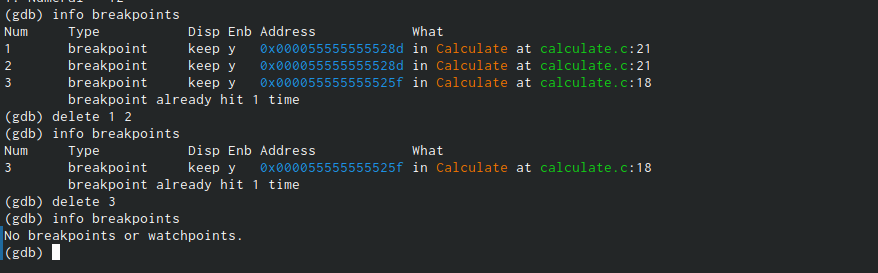
17: backtrace

Посмотрела чему равно на этом этапе значение перемнной Numeral введя команду print Numeral и сравнила с результатом команды display Numeral (рис 18)



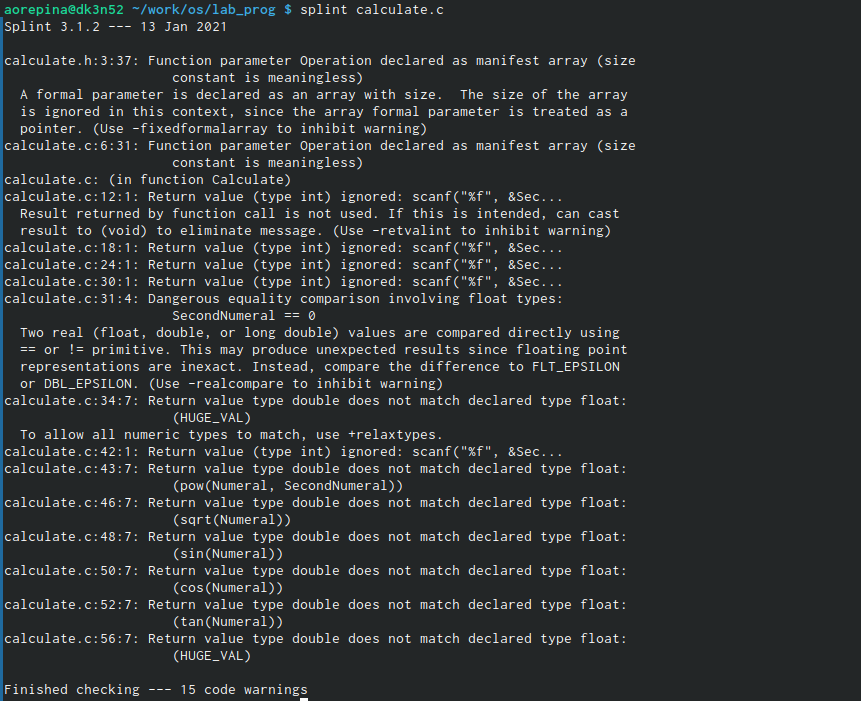
18: Numeral

Убрала точки останова (рис 19)

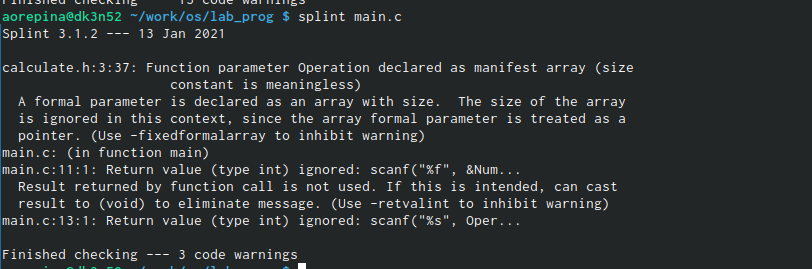


19: убрала точки останова

1. С помощью утилиты splint проанализировала коды файлов calculate.c main.c . Воспользовалась командлй splint calculate.c и splint main.c (рис 20, 21). С помощью этой команды выяснилось. что в файлах присутствует функция чтения, возвращающая целое число, но эти числа не используются и нигде не сохраняются. Утилита вывела предепреждение о том, что в файлепроисходит сравнение вещественного числа с нулем. Также возвращаемые значения в фугкциях записываются в переменную, что свидетельствует о потери данных



20: splint calculate.c



21: splint main.c

Итоговые вопросы:

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: 1. планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения; 2. проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования; 3. 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; 4. 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). Для имени 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-ohellomaiВ 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.n.c». 4). Основное назначение компилятора языка Си в UNIX заключается 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 Makefile имеет 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.c. В этом примере в начале файла заданы три переменные: CC и 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 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 строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdbfile.o 8). 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 строки или название функции); 3. clear − удалить все точки останова в функции; 4. continue − продолжить выполнение программы; 5. delete − удалить точку останова; 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 список 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 − выполнить программу пошагово, но без выполнения вызываемых в программе функций; 12. print − вывести значение указываемого в качестве параметра выражения; 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 программа 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. 9). Cхема отладки программы показана в 6 пункте лабораторной работы. 10). При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена 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,int −критическая проверка программ, написанных на языке Си. 12). Утилита splint анализирует 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 работt программы, переменные с некорректно заданными значениямии типами и многое другое.

# Выводы

В ходе данной лабораторной работы я приобрела навыки разработки, анализа, тестирования и отладки приложений в ОС на примере создания на языке программирования С калькулятора с простейшими функциями