Отчёт по лабораторной работе №13

Операционные системы

Самсонова Мария Ильинична

Содержание

[Цель работы 1](#_Toc105243921)

[Задание 1](#_Toc105243922)

[Выполнение лабораторной работы 2](#_Toc105243923)

[Вывод 15](#_Toc105243924)

[Ответы на контрольные вопросы 15](#_Toc105243925)

# Цель работы

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.

# Задание

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.
2. Создайте в нём файлы: calculate.h, calculate.c, main.c. 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.
3. Выполните компиляцию программы посредством gcc:

* gcc -c calculate.c
* gcc -c main.c
* gcc calculate.o main.o -o calcul -lm

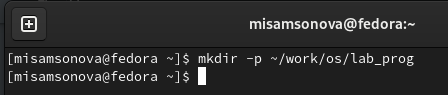
1. При необходимости исправьте синтаксические ошибки.
2. Создайте Makefile с данным в документе содержанием.
3. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

* Запустите отладчик GDB, загрузив в него программу для отладки:
  + gdb ./calcul
* Для запуска программы внутри отладчика введите команду run:
  + run
* Для постраничного (по 9 строк) просмотра исходного код используйте команду list:
  + list
* Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:
  + list 12,15
* Для просмотра определённых строк не основного файла используйте list с параметрами:
  + list calculate.c:20,29
* Установите точку останова в файле calculate.c на строке номер 21:
  + list calculate.c:20,27
  + break 21
* Выведите информацию об имеющихся в проекте точка останова:
  + info breakpoints
* Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:
  + run
  + 5
  + –
  + backtrace
* Отладчик выдаст следующую информацию:
  + #0 Calculate (Numeral=5, Operation=0x7fffffffd280 “-”)
  + at calculate.c:21
  + #1 0x0000000000400b2b in main () at main.c:17
* а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места.
* Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:
  + print Numeral На экран должно быть выведено число 5.
* Сравните с результатом вывода на экран после использования команды:
  + display Numeral
* Уберите точки останова:
  + info breakpoints
  + delete 1

1. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

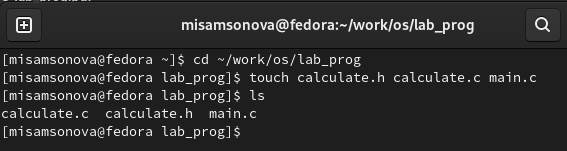
# Выполнение лабораторной работы

1. В домашнем каталоге создаём подкаталог ~/work/os/lab\_progс помощью команды «mkdir-p~/work/os/lab\_prog» (Рис. -@fig:001 ).



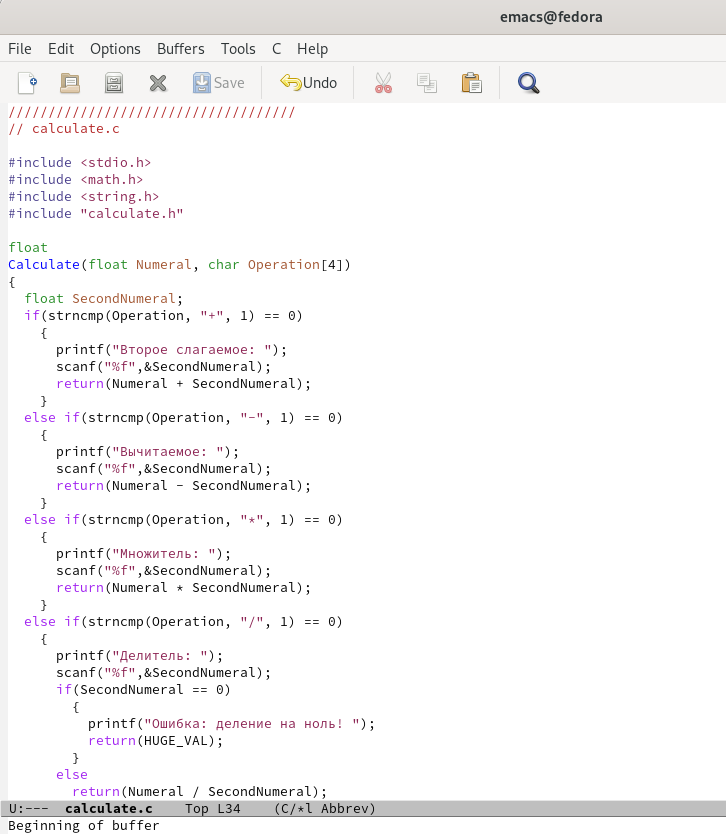
Создание подкаталога

1. Создали в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd ~/work/os/lab\_prog» и «touch calculate.h calculate.c main.c» (рис. -@fig:002 ).

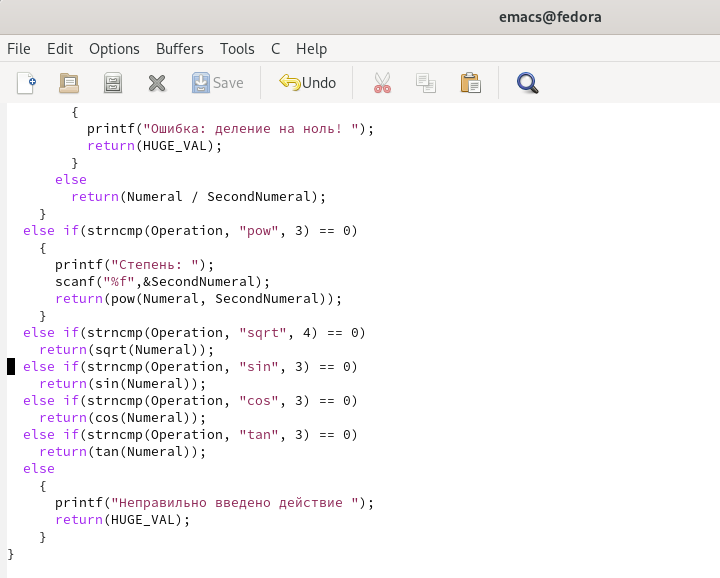


Создание файлов

* 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, приступили к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с рис. -@fig:003 , -@fig:004 ).

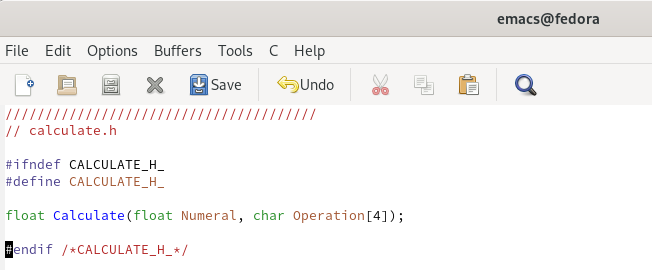


Программа в calculate.c



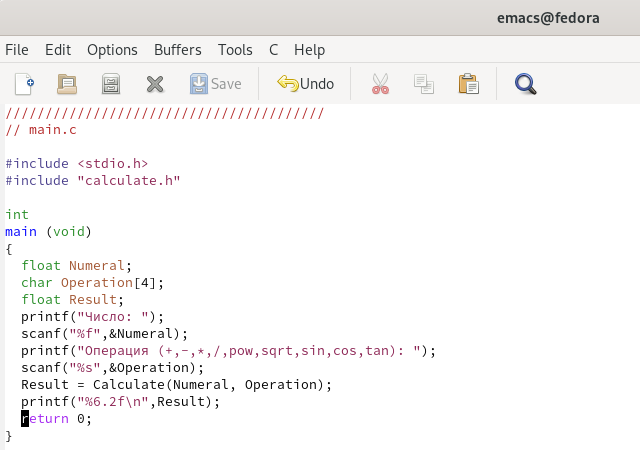
Программа в calculate.c

* Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис. -@fig:005 ).



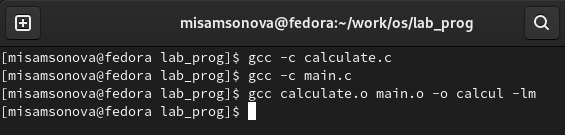
Программа в calculate.h

* Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. -@fig:006 ).



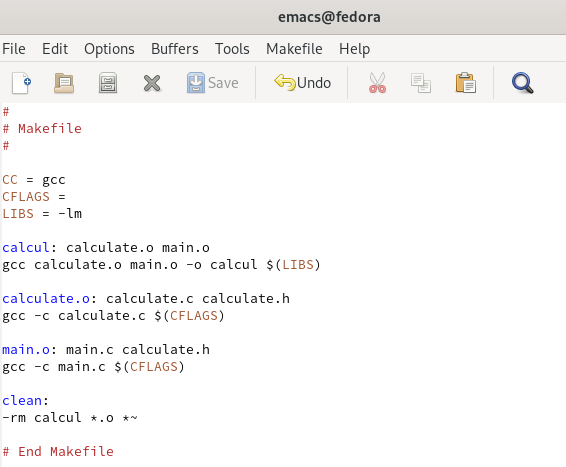
Программа в main.c

1. Выполнили 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» ( рис. -@fig:007 ).



Компиляция программы

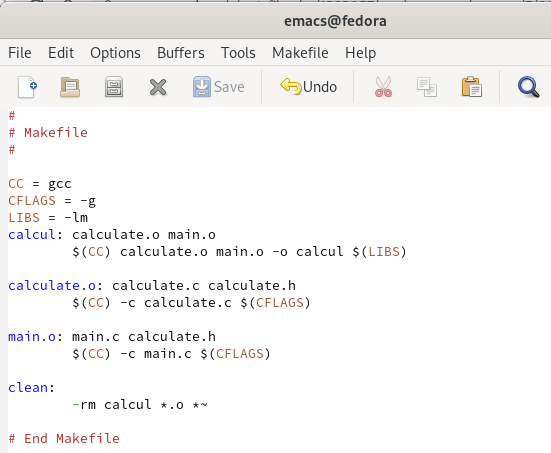
1. В ходе компиляции программы никаких ошибок выявлено не было.
2. Создали Makefile с необходимым содержанием (рис. -@fig:008 ).



Программа в Makefile

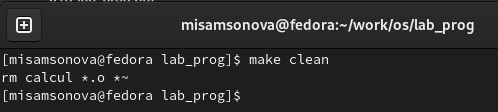
* Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цельmain.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS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

1. Далее исправили Makefile (рис. -@fig:009 ).

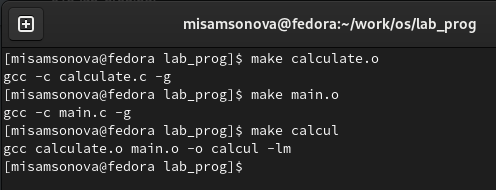


Программа в Makefile

* В переменную CFLAGS добавили опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделали так, что утилита компиляции выбирается с помощью переменной CC. После этого удалили исполняемые и объектные файлы из каталога с помощью команды «make clear» (рис. -@fig:010 ). Выполнили компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (рис. -@fig:011 ).

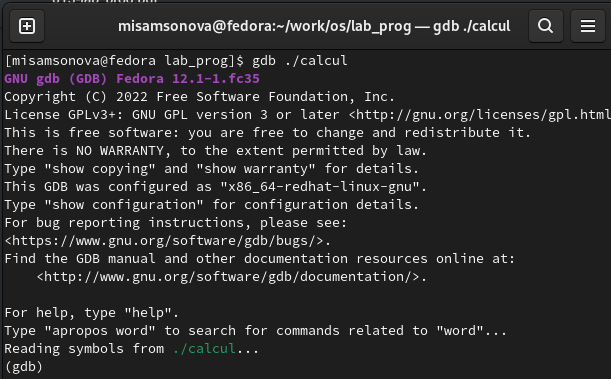


Удаление файлов



Компиляция файлов

* Далее с помощью gdb выполнили отладку программы calcul. Запустили отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul» (рис. -@fig:012 ).



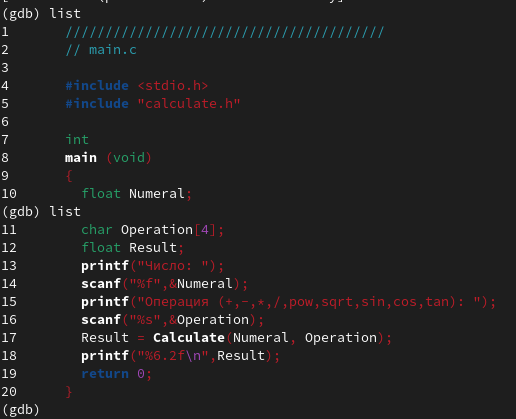
Работа с gdb

* Для запуска программы внутри отладчика ввели команду «run» (рис. -@fig:013 ).



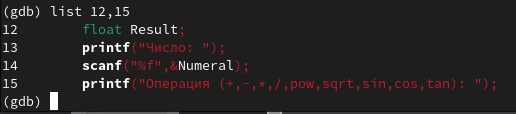
Работа с gdb - run

* Для постраничного (по10строк) просмотра исходного кода использовали команду «list» (рис. -@fig:014 ).



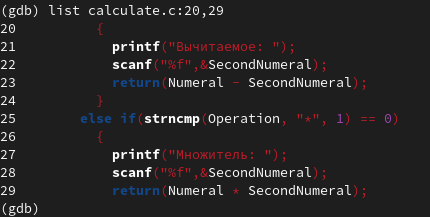
Работа с gdb - list

* Для просмотра строк с 12 по 15 основного файла использовали команду «list 12,15» (рис. -@fig:015 ).



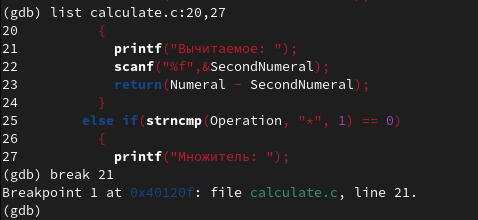
Работа с gdb - list 12,15

* Для просмотра определённых строк не основного файла использовали команду «list calculate.c:20,29» (рис. -@fig:016 ).



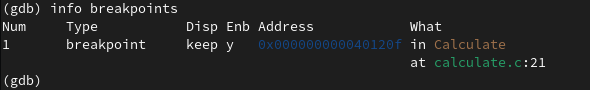
Работа с gdb - list calculate.c:20,29

* Установили точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (рис. -@fig:017 ).



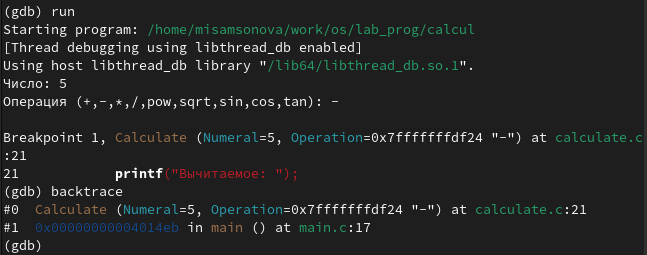
Работа с gdb - list calculate.c:20,27

* Вывели информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (рис. -@fig:018 ).



Работа с gdb - info breakpoints

* Запустили программу внутри отладчика и убедились, что программа остановилась в момент прохождения точки останова. Использовали команды «run», «5», «\*» и «backtrace» (рис. -@fig:019 ).



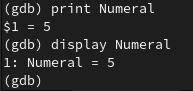
Работа с gdb - run

* Посмотрели, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (рис. -@fig:020 ).

Работа с gdb - print Numeral

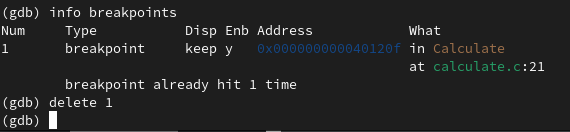
Работа с gdb - print Numeral

* Сравнили с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (рис. -@fig:021 ).



Работа с gdb - display Numeral

* Убрали точки останова с помощью команд «info breakpoints» и «delete1» (рис. -@fig:022 ).

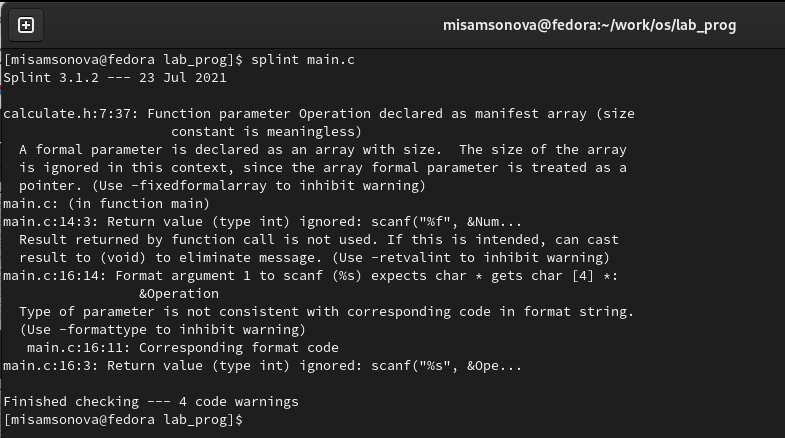


Работа с gdb - info breakpoints

1. Далее воспользовались командами «splint calculate.c» и «splint main.c» (рис. -@fig:023 , -@fig:024 ). 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.



Результат команды splint calculate.c



Результат конмады splint main.c

# Вывод

В процессе выполнения данной лабораторной работы мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# Ответы на 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.

1. 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».
2. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
3. 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.
4. 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.
5. 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
6. 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 (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);
* 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 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 (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.
3. 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.
4. 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.