Отчёт по лабораторной работе №13

Операционные системы

Обрезкова Анастасия Владимировна

Содержание

# 1 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.

# 2 Выполнение лабораторной работы

1. В домашнем каталоге создаём подкаталог “~/work/os/lab\_prog” с помощью комнады “mkdir”.(рис. [1](#fig:001)).

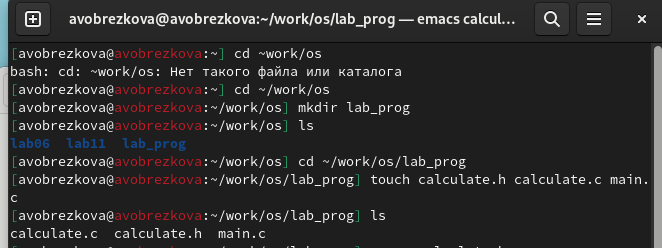


Figure 1: Создание подкаталога

1. Затем перейдём в каталог и создадим файлы: calculate.h, calculate.c, main.c. Делаю это с помощью команды “touch”.(рис. [2](#fig:002)).

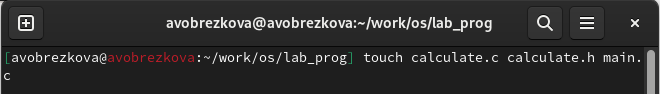


Figure 2: Создание подкаталога

Создадим примитинейший 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. Реализация функций калькулятора будет делать в файле calculate.c.(рис. [3](#fig:003)), (рис. [4](#fig:004)).

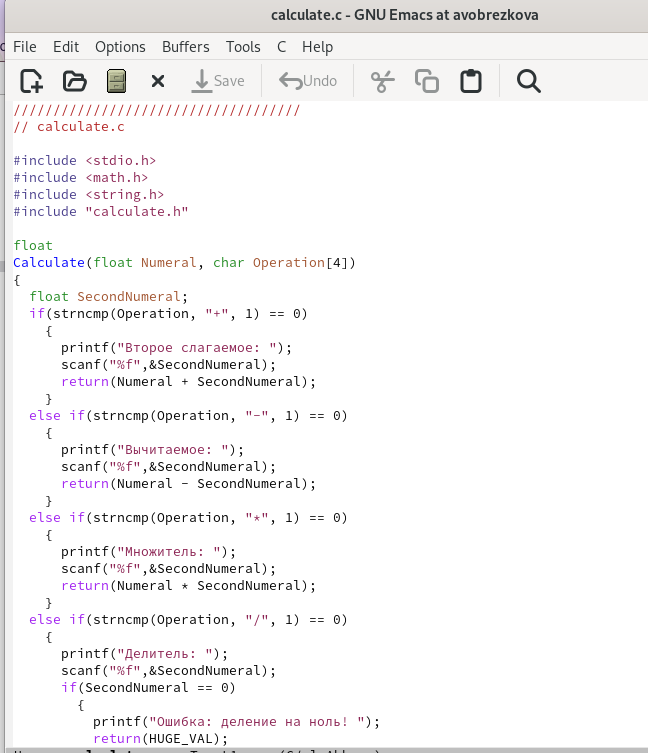


Figure 3: Файл calculate.c

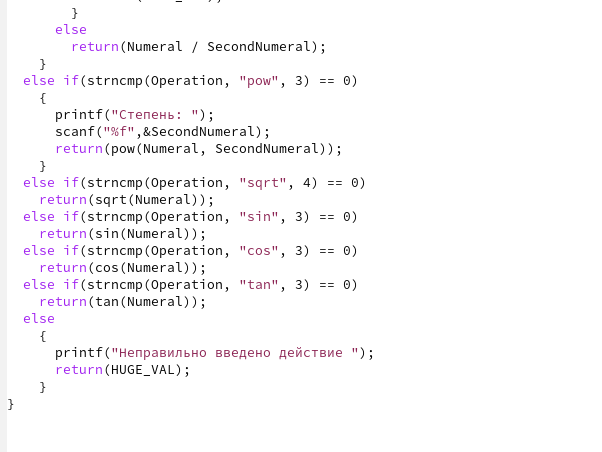


Figure 4: Файл calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора.(рис. [5](#fig:005)).

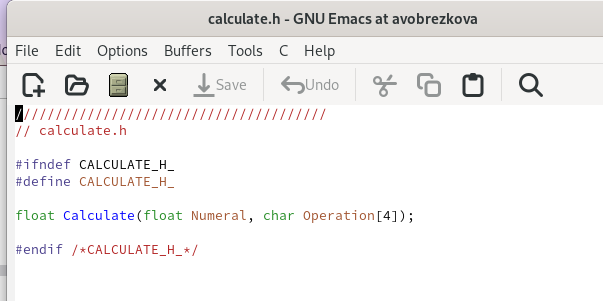


Figure 5: Файл calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору. (рис. [6](#fig:006)).

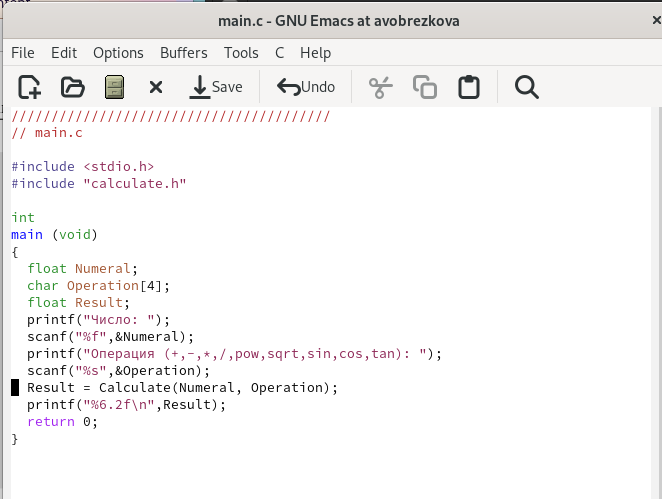


Figure 6: Файл main.c

1. Далее выполним компиляцию программы посредством gcc. (рис. [7](#fig:007)).

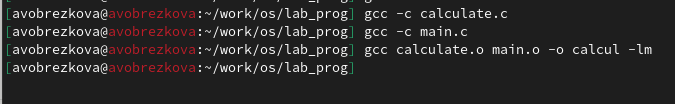


Figure 7: Компиляция файла

1. Ошибок не выявлено
2. Создадим Makefile c необходимым содержанием. Он необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а так же их объединения в один исполняемый файл calcul. Цель “clean” нужна доя 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.(рис. [8](#fig:008)).

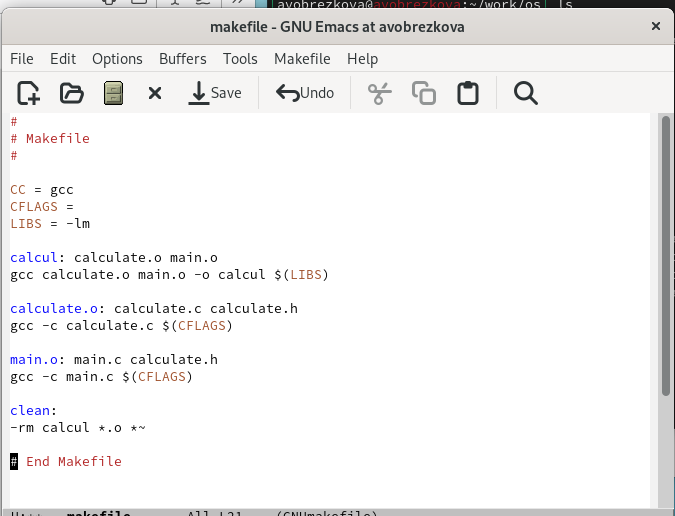


Figure 8: Makefile

1. Далее изменим файл. В переменную CFLAGS добавим “-g”, которая необходима для компиляции объектных файлов и их использования в программе отладчика GDB. Также, компиляция выбирается с помощью переменной СС.(рис. [9](#fig:009))

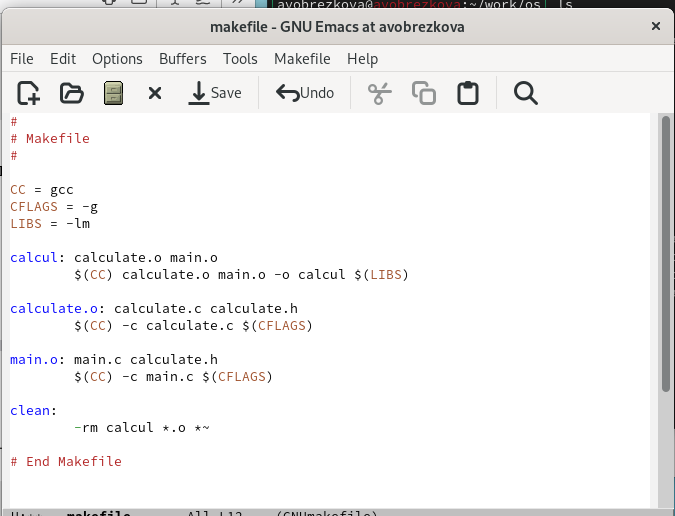


Figure 9: Изменения

После выполняем компиляцию файлов. (рис. [10](#fig:010))

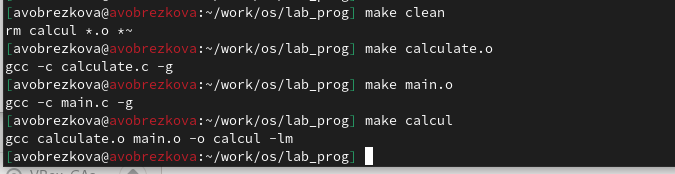


Figure 10: Компиляция

После этого выполняем gdb отладку программмы calcul. Запускаем GDB и загружаем в него программу для отладки, используя команду “gdb ./calcul”(рис. [11](#fig:011))

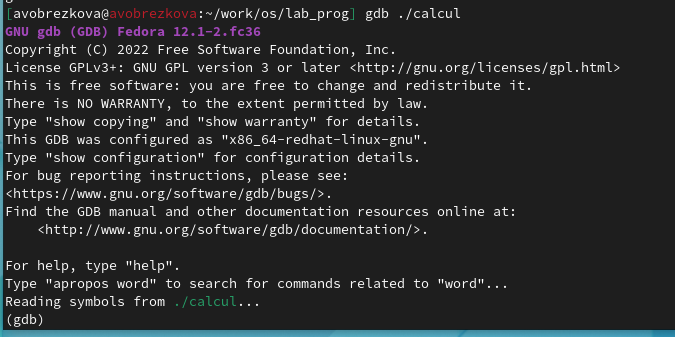


Figure 11: Отладчик

Далее вводим комнаду “run” для запуска программы внутри откадчика.(рис. [12](#fig:012))

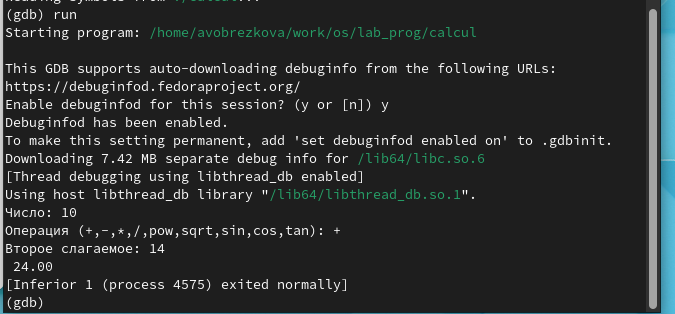


Figure 12: Запуск программы

Для постраничего просмотра исходного кода используем команду “list”.(рис. [13](#fig:013))

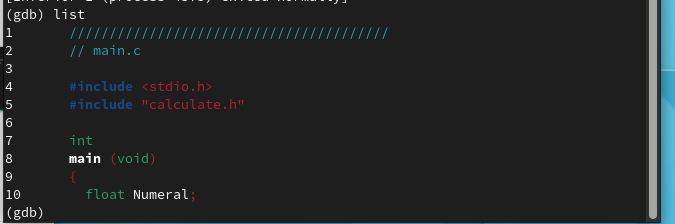


Figure 13: Просмотр кода

Для просмотра строк с 12 по 15 основного файла используем команду “list 12,15”.(рис. [14](#fig:014))

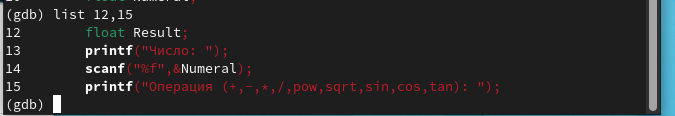


Figure 14: Просмотр строк

Для просмотра определённых строк не основного файла используем команду “list calculate.c:20,29”.(рис. [15](#fig:015))

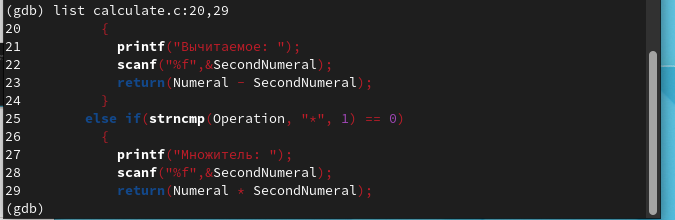


Figure 15: Просмотр строк

Для установки точки в файле “calculate.c” на строке 21 используем команды “list calculate.c:20,27” и “break 21”.(рис. [16](#fig:016))

Figure 16: Установка точки

Figure 16: Установка точки

Чтобы вывесни информацию об имеющихся точках останова используем команду “info breakpoint”(рис. [17](#fig:017))

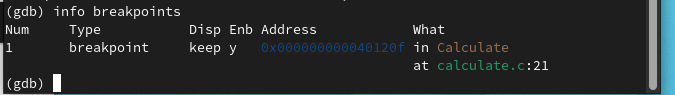


Figure 17: Информация о точках

Запустим программу внутри отладчика и убедимся, что программа остановилась в момент прохождения точки останова.(рис. [18](#fig:018))

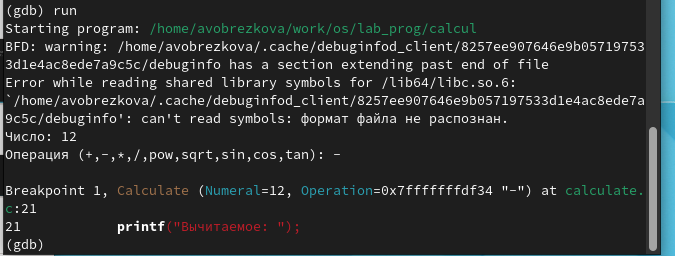


Figure 18: Остановка программы

Посмотрим, чему на этом этапе равно значение переменной Numeral, с помощью команды “print Numeral” и сравним его с результатом вывода на экарн после использования команды “display Numeral”. Значения совпадают.(рис. [19](#fig:019))

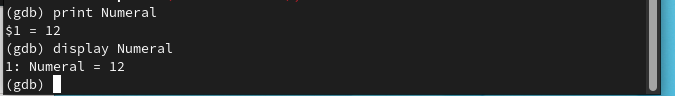


Figure 19: Просмотр значения

Уберём точки останова с помощью команды “info breakpoints”(рис. [20](#fig:020))

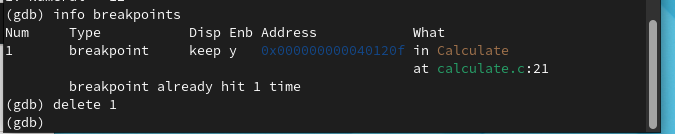


Figure 20: Удаление точки останова

1. С помощью утилиты splint проанализировала коды файлов calculate.c и main.c.(рис. [21](#fig:021)), (рис. [22](#fig:022))

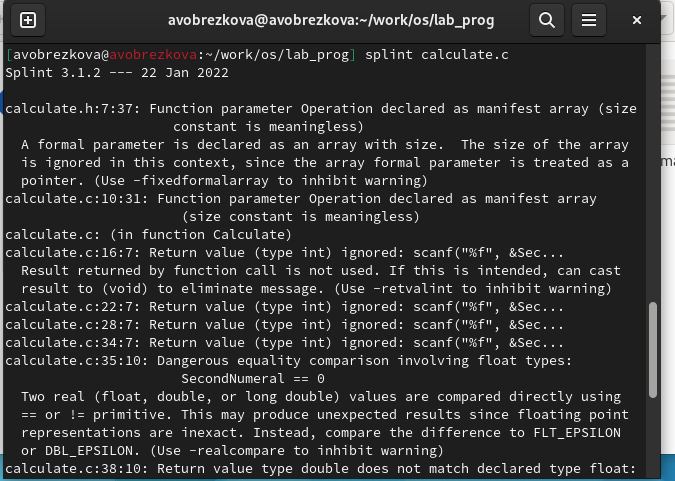


Figure 21: Анализ файла 1

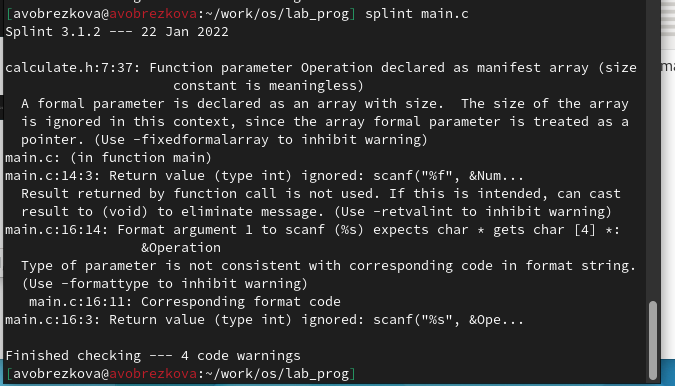


Figure 22: Анализ файла 2

Данные изменения можно проверить по ссылке: <https://github.com/avobrezkova/study_2022-2023_os-intro/tree/master/labs/lab13>

# 3 Выводы

В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в OC типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# 4 Ответы на контрольные вопросы

1. ак получить информацию о возможностях программ gcc, make, gdb и др.?

Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.

1. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

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:

– кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода;

– сборка, компиляция и разработка исполняемого модуля;

– тестирование и отладка, сохранение произведённых изменений;

• документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

1. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом).c воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C − как файлы на языке C++, а файлы c расширением .o считаются объектными. Например, в команде «gcc -c main.c»: gcc по расширению (суффиксу) .c распознает 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 -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».

1. Каково основное назначение компилятора языка С в UNIX?

Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

1. ля чего предназначена утилита make?

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.

1. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или 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 Makefile может 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 Makefile имеет 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 = gcc CFLAGS = #Compile abcd.c normaly abcd: abcd.c

$(CC) -o abcd $(CFLAGS) abcd.c clean: -rm abcd .o ~ #End Makefile for abcd.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 clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

1. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска иустранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc: gcc -c file.c -g После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdb file.o

1. Назовите и дайте основную характеристику основным командам отладчика gdb.

Основные команды отладчика gdb: - backtrace − вывод на экран пути к текущей точке останова (по сути вывод −названий всех функций) - break − установить точку останова (в качестве параметра может быть указанномер 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 начальной и 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 gdb можно 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 -h и man gdb.

1. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

Cхема отладки программы показана в 6 пункте лабораторной работы.

1. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

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); нужноубрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.

1. Назовите основные средства, повышающие понимание исходного кода программы.

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.

1. Каковы основные задачи, решаемые программой splint?

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 C анализатор splint генерирует 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 работ программы, переменные с некорректно заданными значениями и типами и многое другое