Отчёт по лабораторной работе № 13

*дисциплина:* Операционные системы

Андрианова Марина Георгиевна

Содержание

# 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.

# Выполнение лабораторной работы

1). В домашнем каталоге создадим подкаталог ~/work/os/lab\_prog(команда “mkdir ~/work/os/lab\_prog”)(рис.1).

Создание подкаталога

Рис.1: Создание подкаталога

2). Перейдём в созданный подкаталог(команда “cd ~/work/os/lab\_prog”) и создадим в нём файлы: calculate.h, calculate.c, main.c(команда “touch calculate.h calculate.c main.c”)(рис.2). Проверим созданные файлы командой “ls”(рис.3).

Создание файлов

Рис.2: Создание файлов

Проверяем созданные файлы

Рис.3: Проверяем созданные 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.

Запустила редактор Emacs в фоновом режиме(команда “emacs &”) и приступила к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с(рис.4):

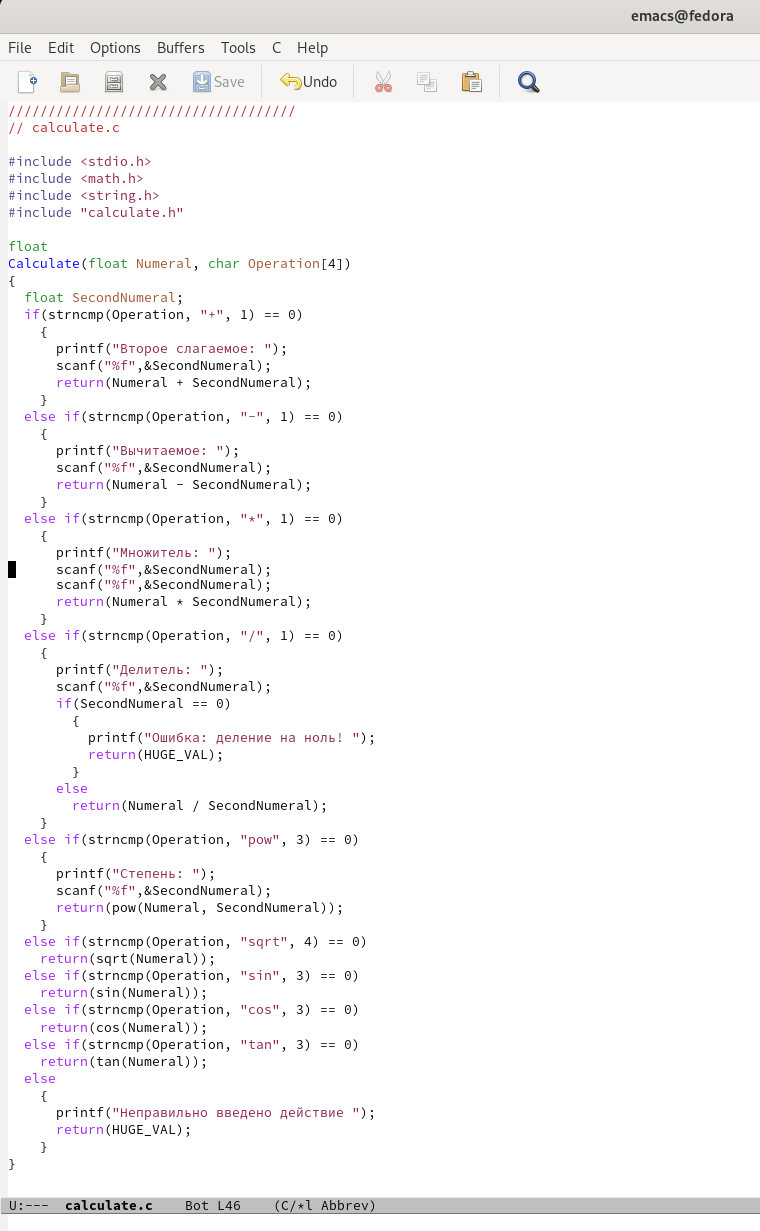


Рис.4: Программа в calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора(рис.5):

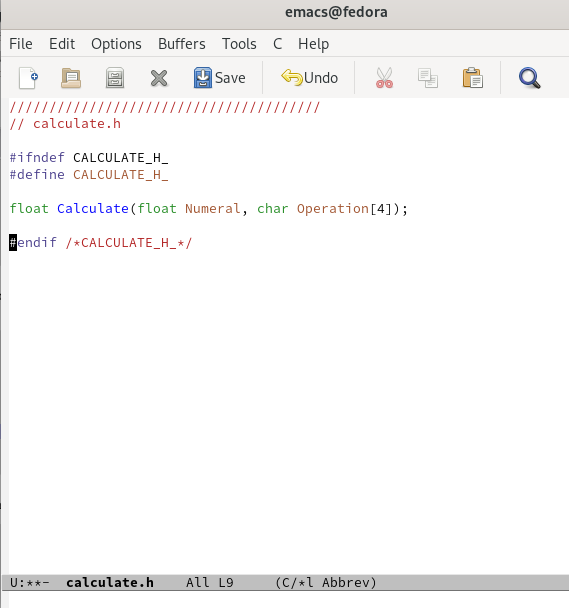


Рис.5: Программа в calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору(рис.6):

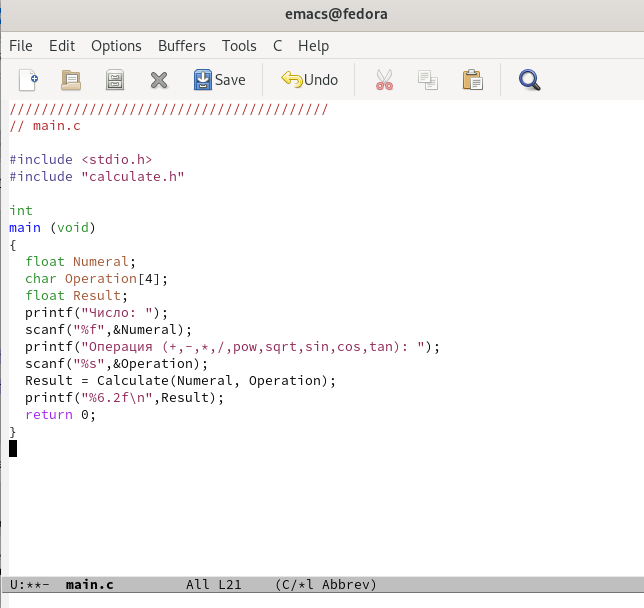


Рис.6: Программа в main.c

3). Выполним компиляцию программы посредством gcc, используя команды “gcc -c calculate.c”, “gcc -c main.c” и “gcc calculate.o main.o 0o calcul -lm”(рис.7).

Компиляция программы

Рис.7: Компиляция программы

4). В ходе компиляции программы никаких синтаксических ошибок выявлено не было. 5). Создала Makefile с необходимым содержанием(рис.8).

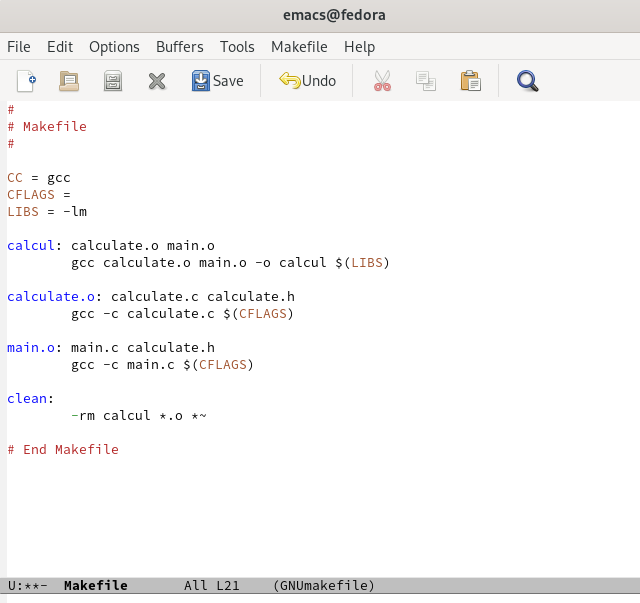


Рис.8: Программа в Makefile

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.

6). Далее исправила Makefile (рис.9).

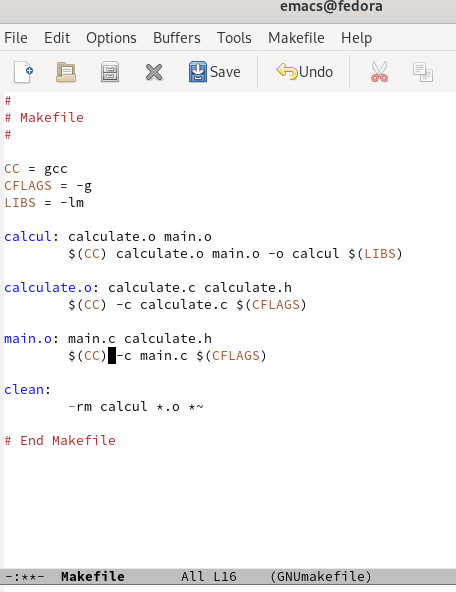


Рис.9: Изменённая программа в Makefile

В переменную 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 “make clean” (рис.10 ). Выполнила компиляцию файлов, используя команды “make calculate.o”, “make main.o”, “make calcul”(рис.11).

Удаление файлов

Рис.10: Удаление файлов

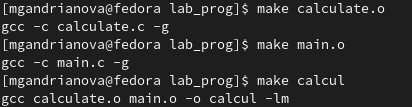


Рис.11: Компиляция файлов

С помощью gdb выполнила отладку программы calcul: – Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: “gdb./calcul”(рис.12).

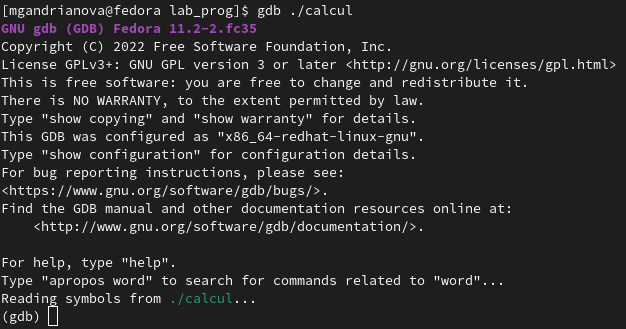


Рис.12: Работа с gdb

Для запуска программы внутри отладчика ввела команду “run”(рис.13).

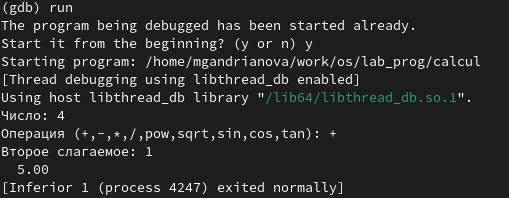


Рис.13: Работа с gdb - run

Для постраничного (по 10 строк) просмотра исходного кода использовала команду “list”(рис.14).

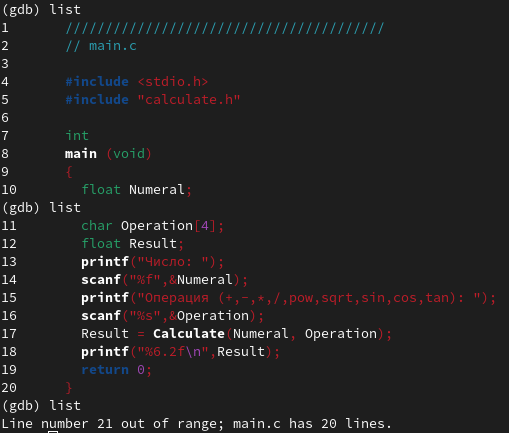


Рис.14: Работа с gdb - list

Для просмотра строк с 12 по 15 основного файла использовала команду “list 12,15”(рис.15).

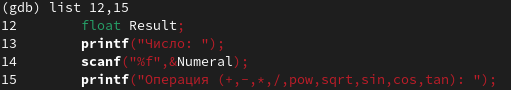


Рис.15: Работа с gdb - list 12,15

Для просмотра определённых строк не основного файла использовала команду “list calculate.c:20,29”(рис.16).

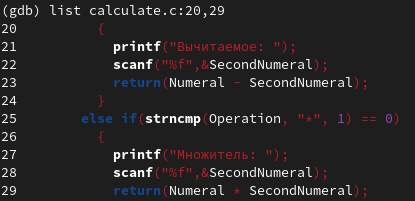


Рис.16: Работа с gdb - list calculate.c:20,29

Установила точку останова в файле calculate.c на строке номер 21, используя команды “list calculate.c:20,27” и “break 21”(рис.17).

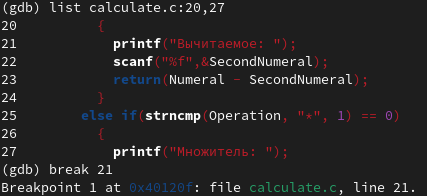


Рис.17: Работа с gdb - list calculate.c:20,27 и break 21

Вывела информацию об имеющихся в проекте точках останова с помощью команды “info breakpoints”(рис.18).

Работа с gdb - info breakpoints

Рис.18: Работа с gdb - info breakpoints

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды “run”, “5”,“-” и “backtrace”(рис.19).Отладчик выдал следующую информацию(рис.20).

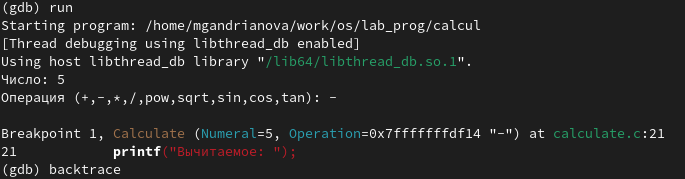


Рис.19: Работа с gdb - run

Результат после команды “backtrace”

Рис.20: Результат после команды “backtrace”

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду “print Numeral”(рис.21).

Работа с gdb - print Numeral

Рис.21: Работа с gdb - print Numeral

Сравнила с результатом вывода на экран после использования команды “display Numeral”(рис.22). Значения совпадают.

Работа с gdb - display Numeral

Рис.22: Работа с gdb - display Numeral

Убрала точки останова с помощью команд “info breakpoints” и “delete 1”(рис.23).

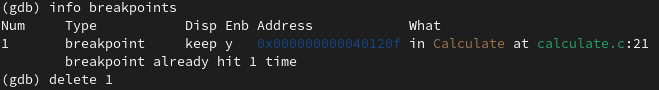


Рис.23: Работа с gdb - info breakpoints

7).Далее воспользовалась командами “splint calculate.c” и “splint main.c” (рис.24,рис.25).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.

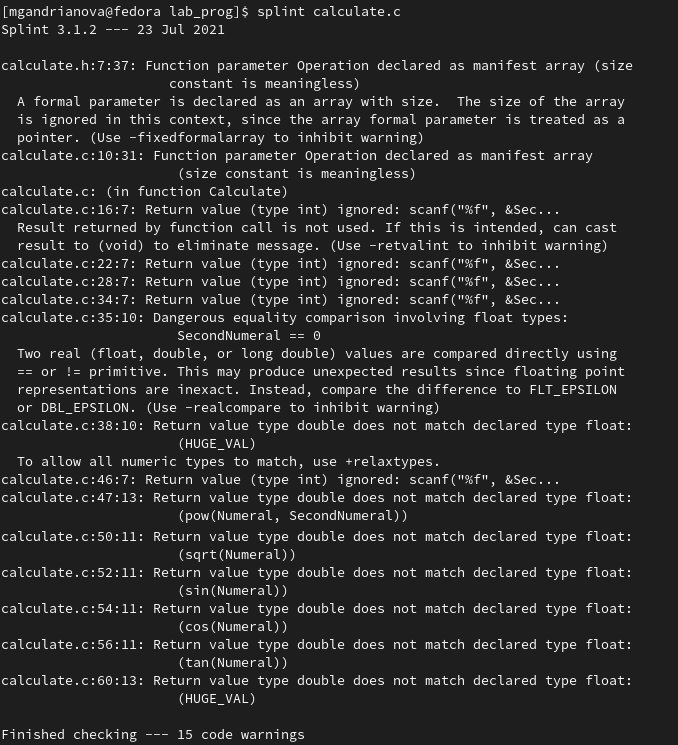


Рис.24: Результат команды splint calculate.c

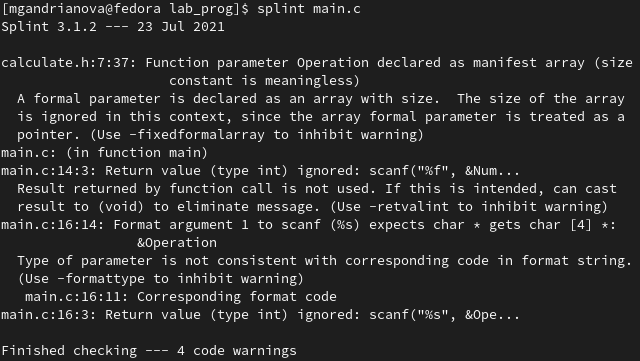


Рис.25: Результат команды splint main.c

# Выводы

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

# Контрольные вопросы

1). Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или 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, geany и 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).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.

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 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 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

(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 UNIX входит 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 -g компилятора gcc: gcc-cfile.c-g. После этого для начала работы с gdb необходимо 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 (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-h и man gdb.

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,splint −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 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 работе программы, переменные с некорректно заданными значениями и типами и многое другое.