РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Лабораторная работа No 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Абдуллина Ляйсан Раисовна НПИбд-01-21

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 Теоретическое введение

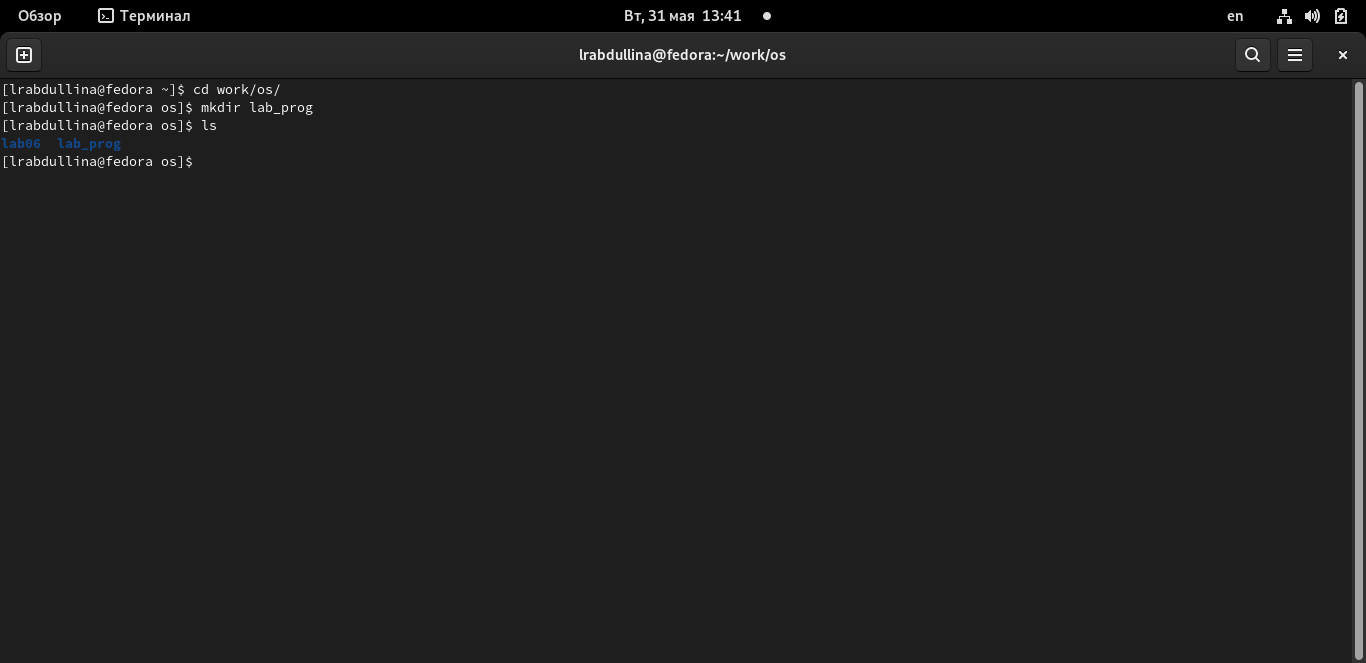
Процесс разработки программного обеспечения обычно разделяется на следующие этапы: - планирование,включающее сбор и анализтребований к функционалу и другим характеристикам разрабатываемого приложения;

* 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.

# 3 Выполнение лабораторной работы

## 3.1 1

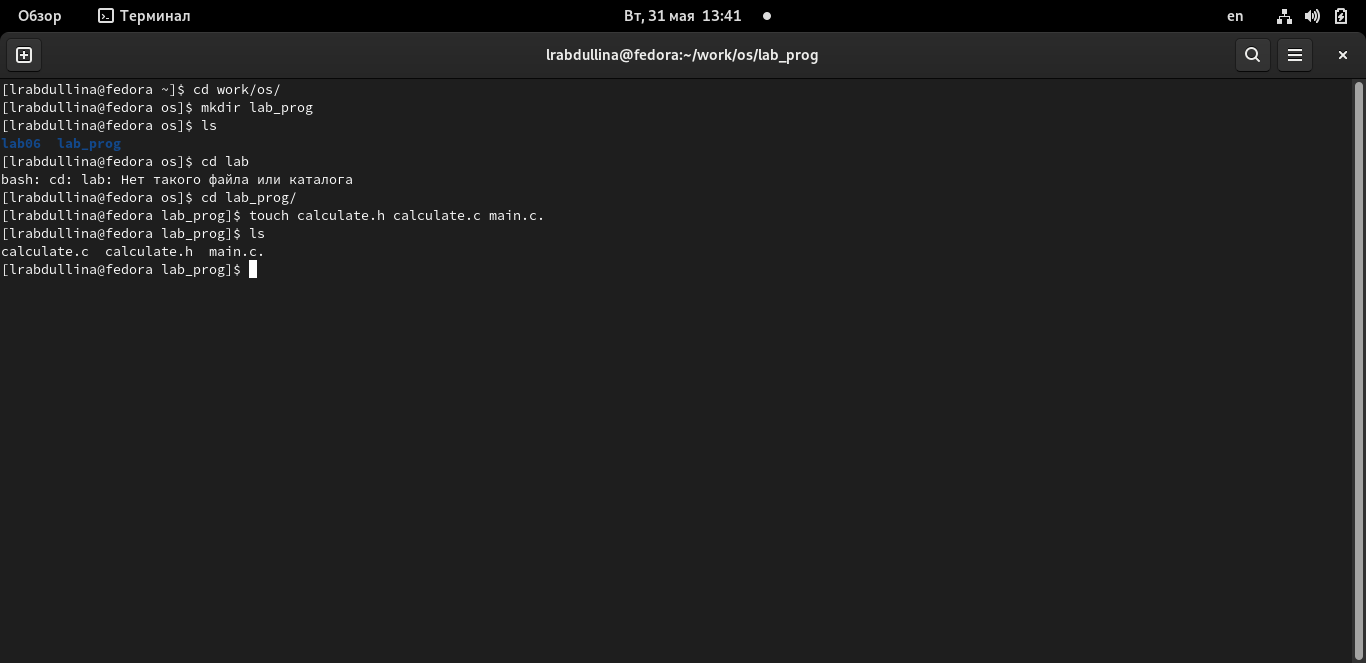
В домашнем каталоге создадим подкаталог ~/work/os/lab\_prog - через команду mkdir (скриншот 1)



Скриншот 1: Созданный каталог

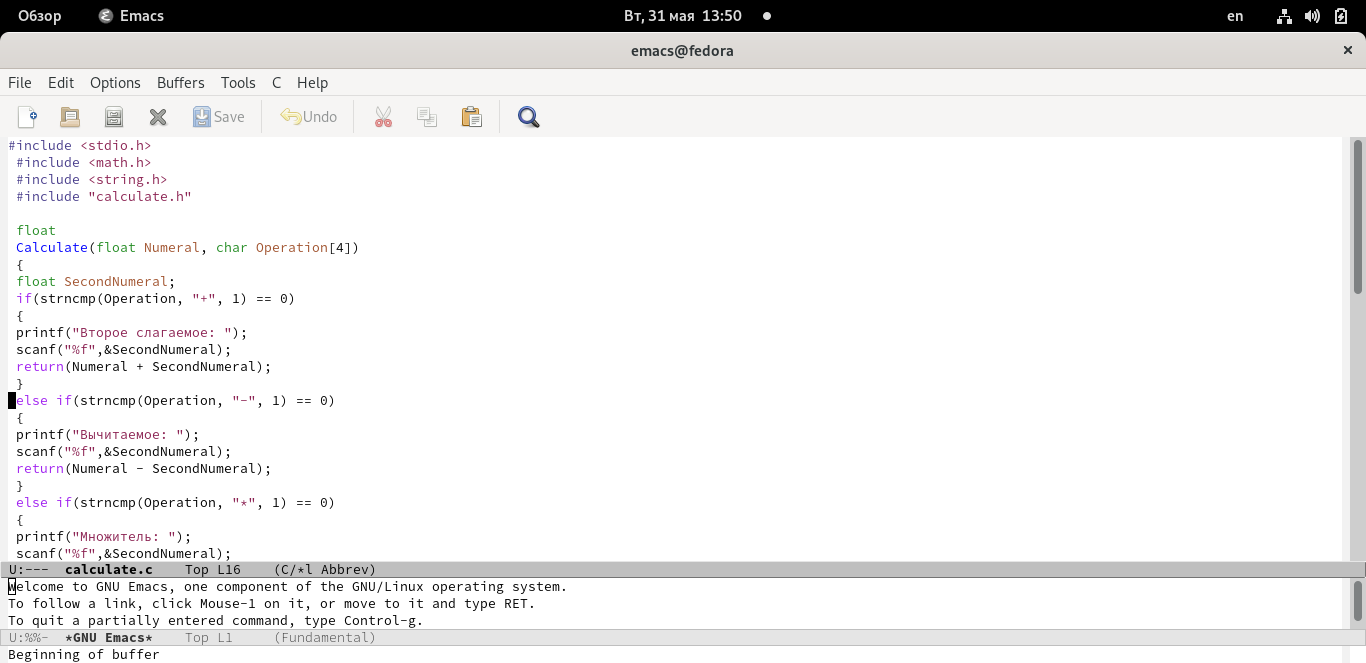
## 3.2 2

Cоздадим в нём файлы: calculate.h,calculate.c,main.c - через команду touch (скриншот 1)

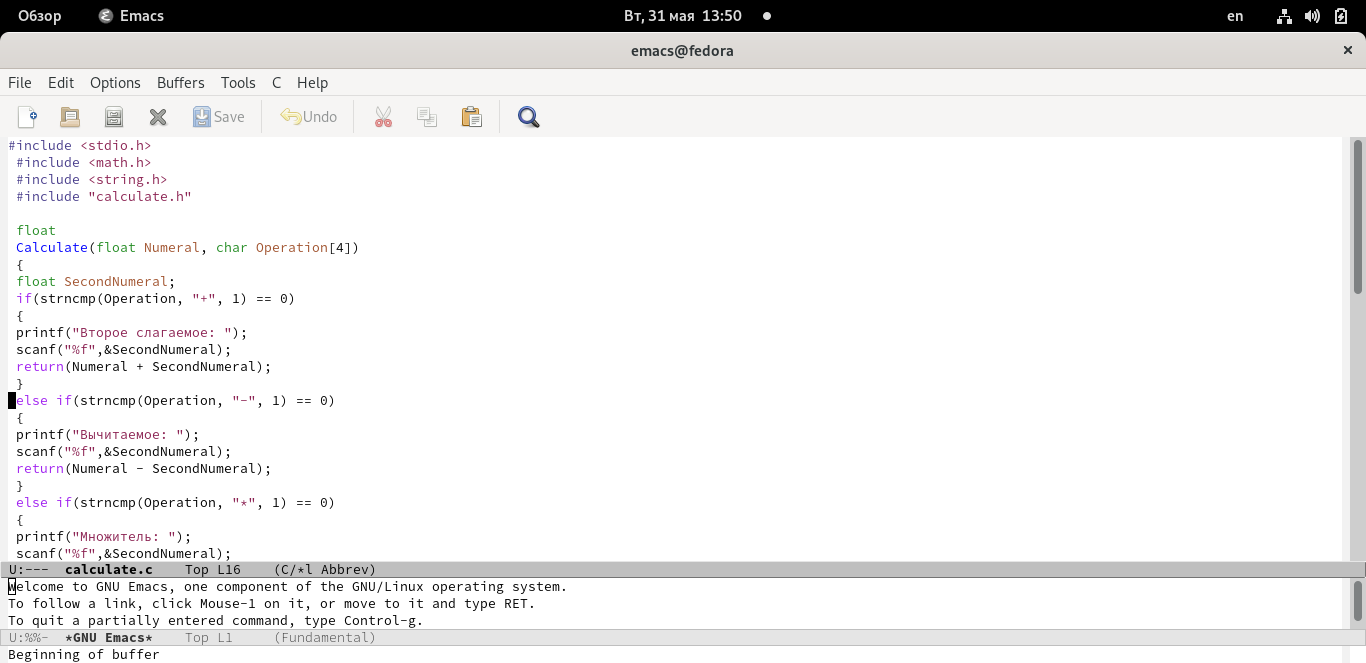


Скриншот 2: Созданные файлы

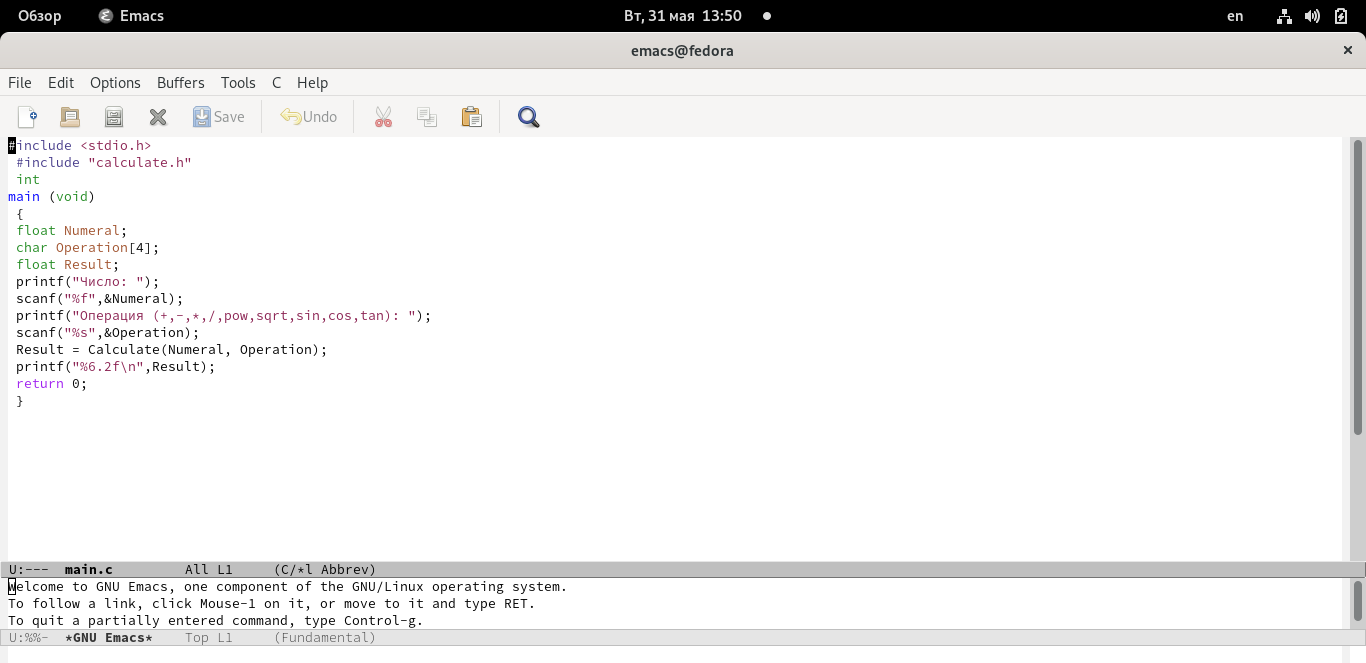
Это примитивнейший калькулятор,способный складывать,вычитать,умножать и делить, возводить число в степень,брать квадратный корень,вычислять sin,cos,tan. При запуске он запрашивает первое число,операцию,второе число. После этогопрограмма выведет результат и останавливается. (скриншоты 3,4,5)



Скриншот 3: Реализация функций калькулятора в файле calculate.с



Скриншот 4: Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора

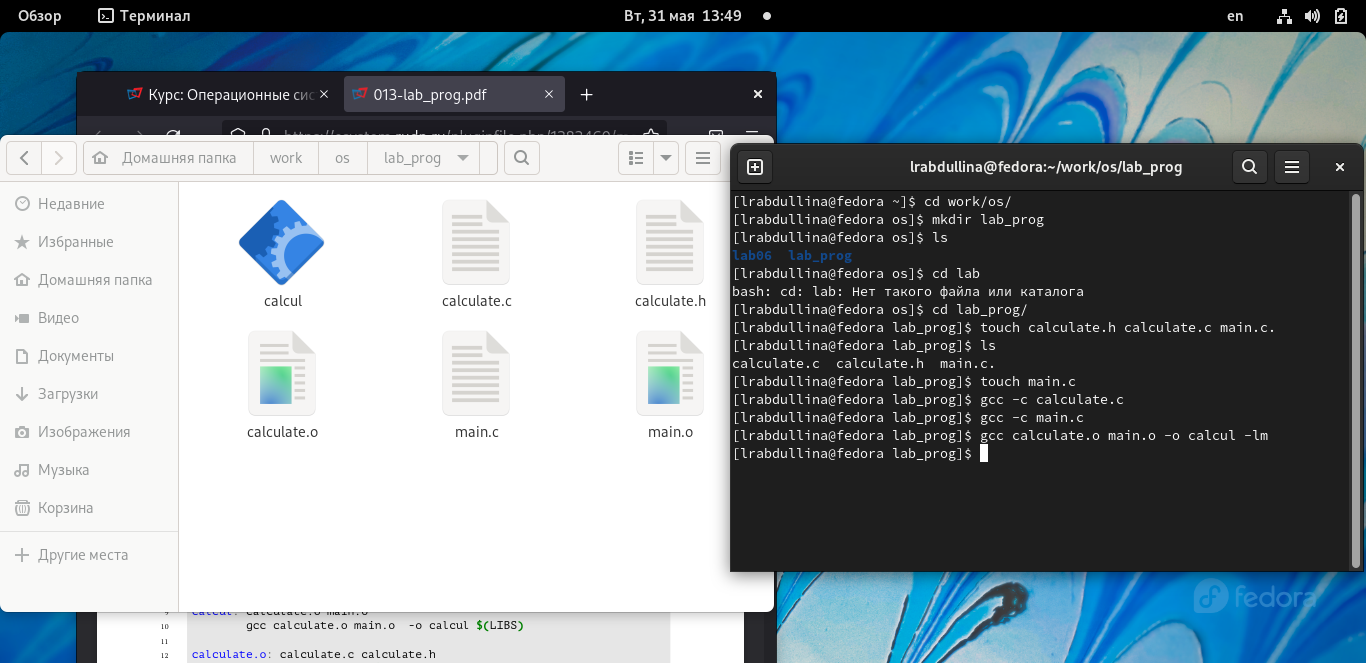


Скриншот 5: Основной файл main.c,реализующий интерфейс пользователя к калькулятору

## 3.3 3

Выполним компиляцию программы посредством gcc, используя команды (скриншот 6):

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



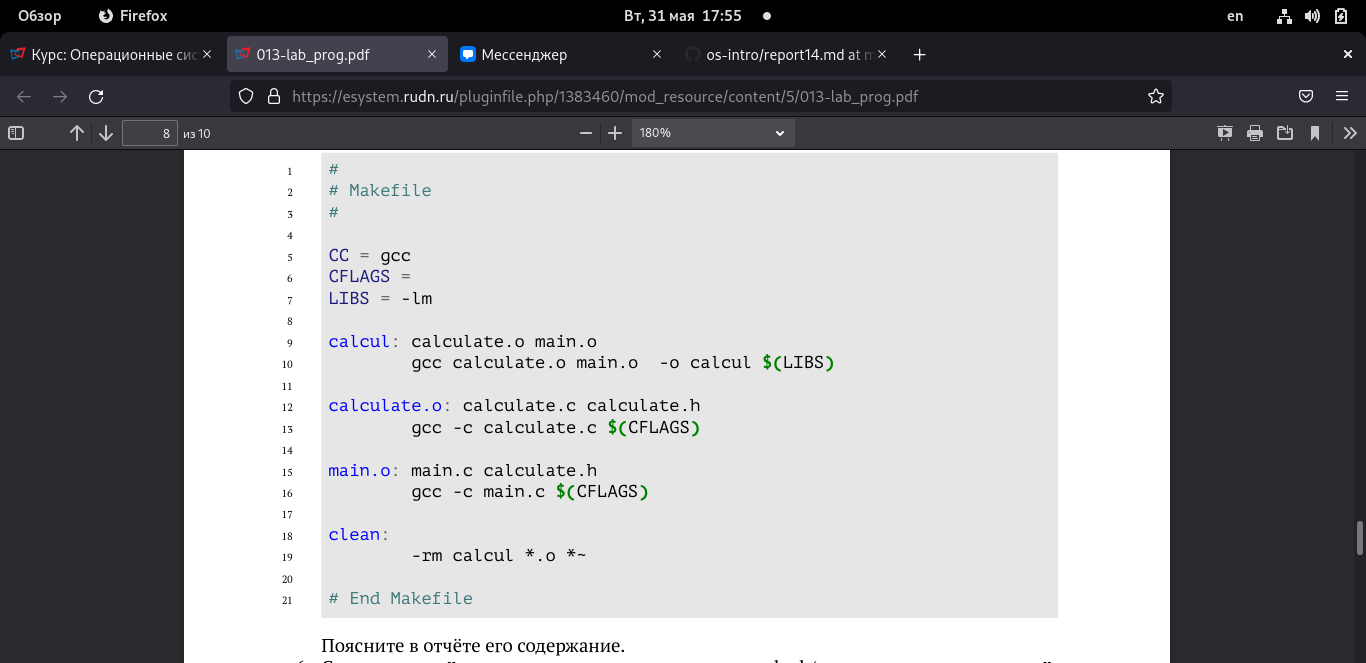
Скриншот 6: Созданные файлы

## 3.4 4

Необходимости исправлять синтаксические ошибки не было.

## 3.5 5

Cоздадим Makefile со следующим содержанием (скриншот 7):

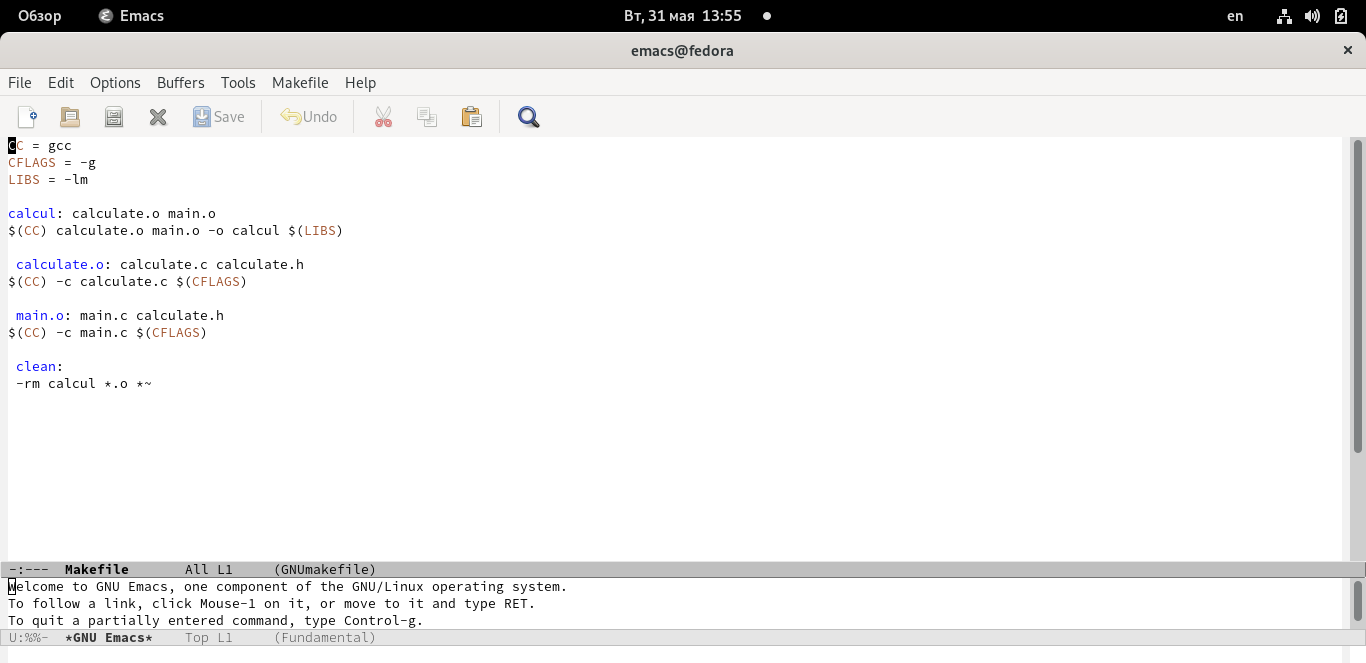


Скриншот 7: Созданный файл

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.

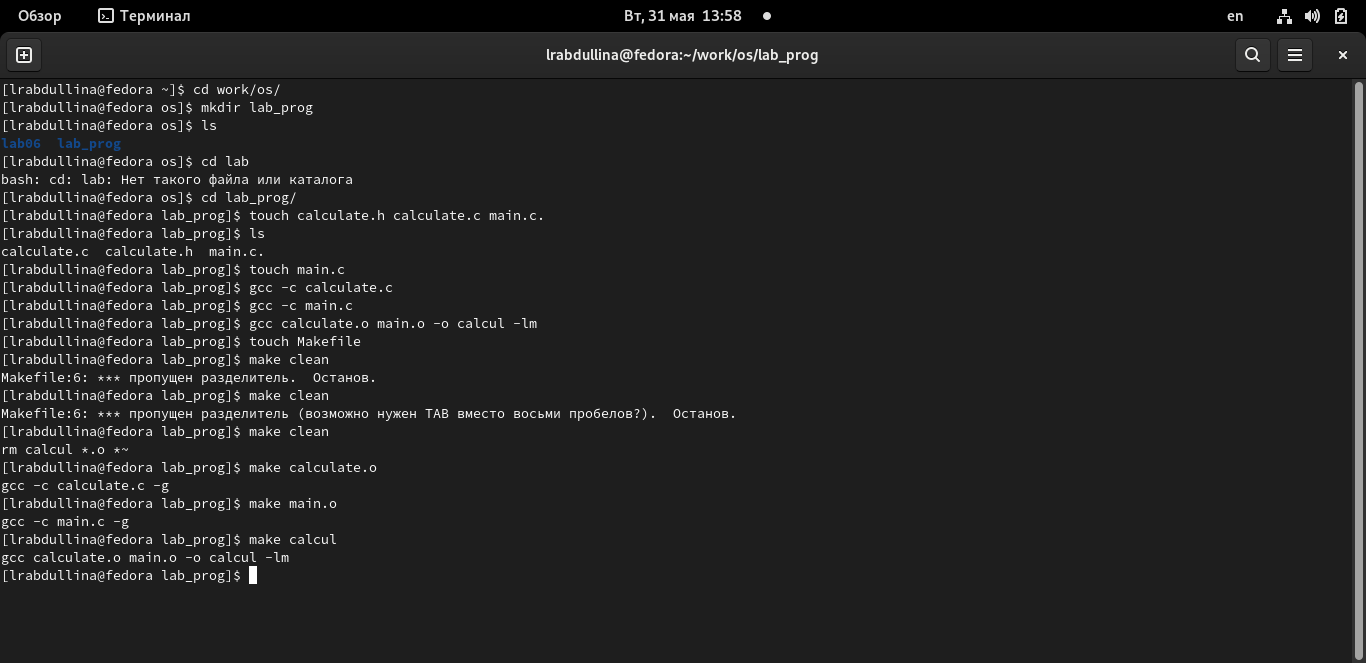
## 3.6 6

С помощью gdb выполним отладку программы calcul, но перед использованием gdb исправим Makefile. В переменную CFLAGS добавим опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделаем так, что утилита компиляции выбирается с помощью переменной CC. (скриншот 8)



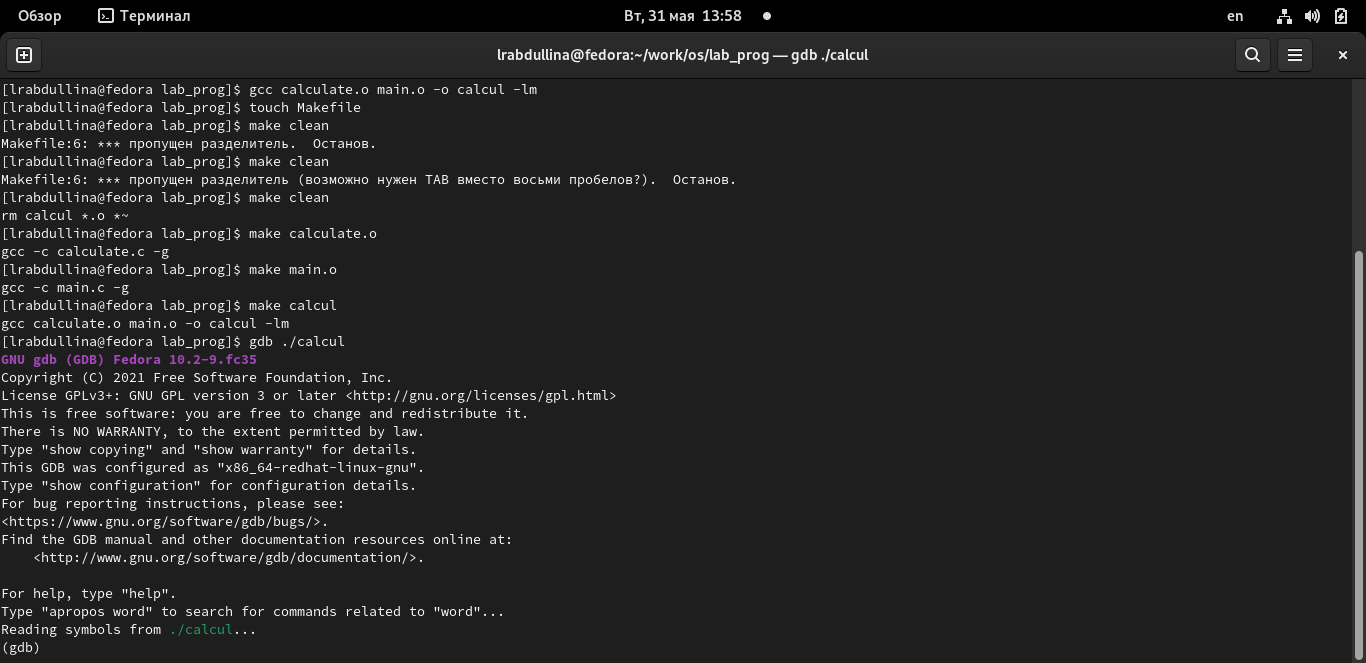
Скриншот 8: Исправленный файл

После этого удалим исполняемые и объектные файлы из каталога с помощью команды «make clean». Выполним компиляцию файлов, используя команды «make calculate.o», «make main.o», «male calcul» (скриншот 9)



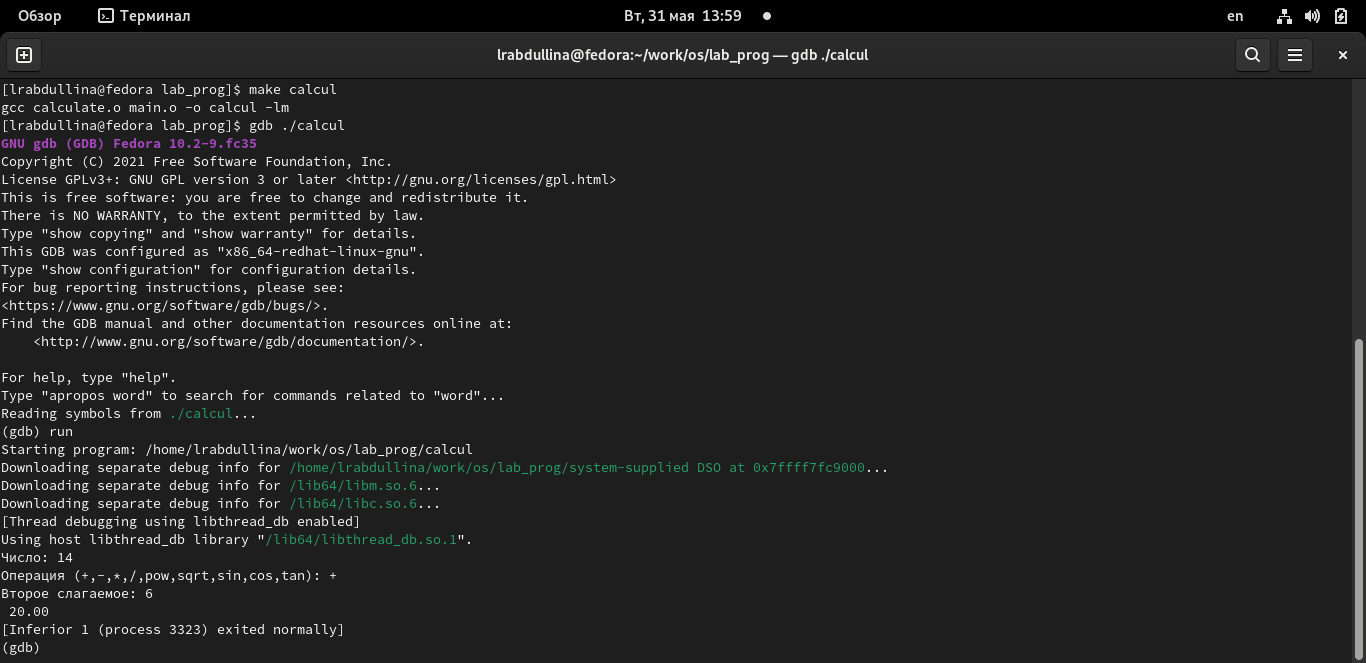
Скриншот 9: Компиляция файлов

* Запустим отладчик GDB, загрузив в него программу для отладки “gdb ./calcul” (скриншот 10)



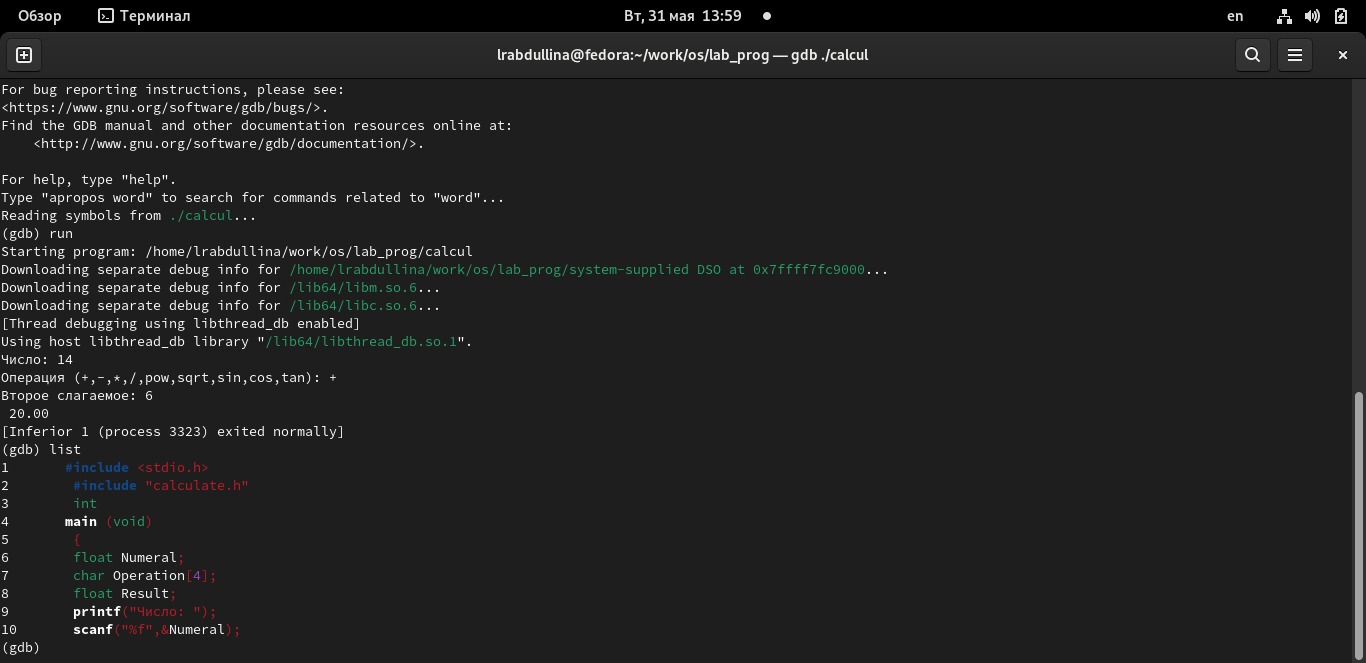
Скриншот 10: Запуск отладчика

* Для запуска программы внутри отладчика введем команду “run” и произведем арифметическую операцию (скриншот 11)



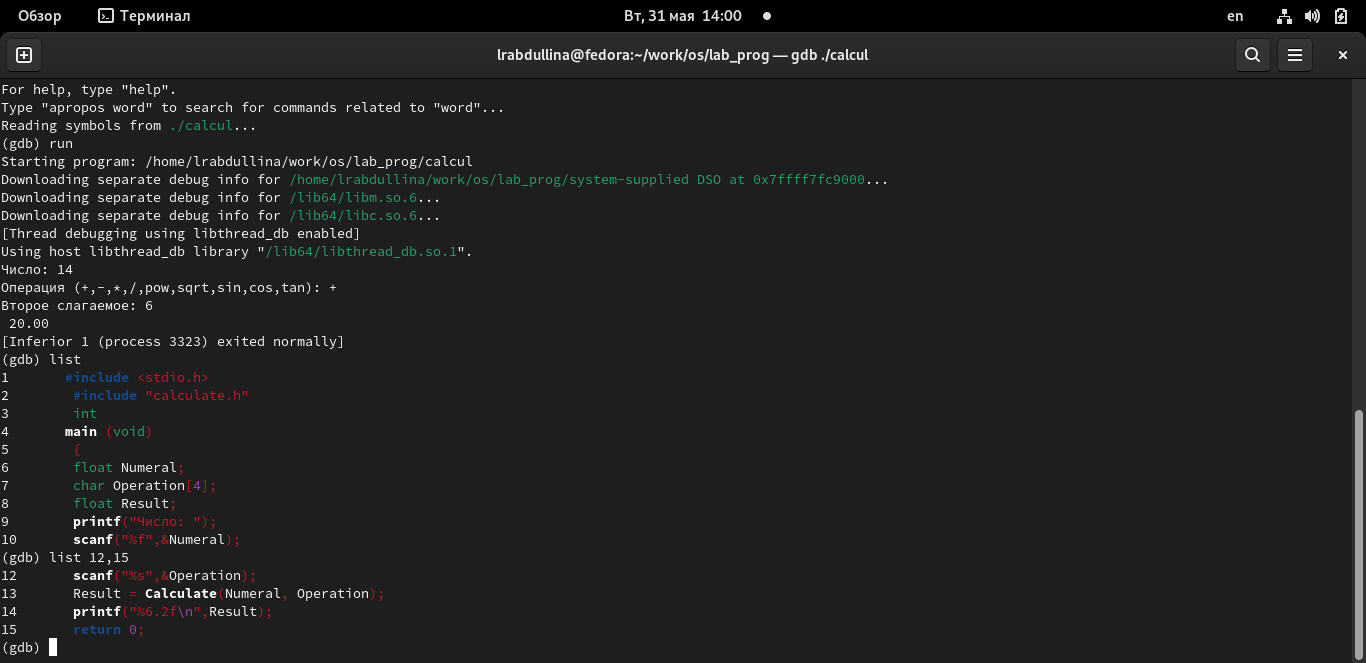
Скриншот 11: Запуск программы внутри отладчика

* Для постраничного (по 9 строк) просмотра исходного код используем команду “list”(скриншот 12)



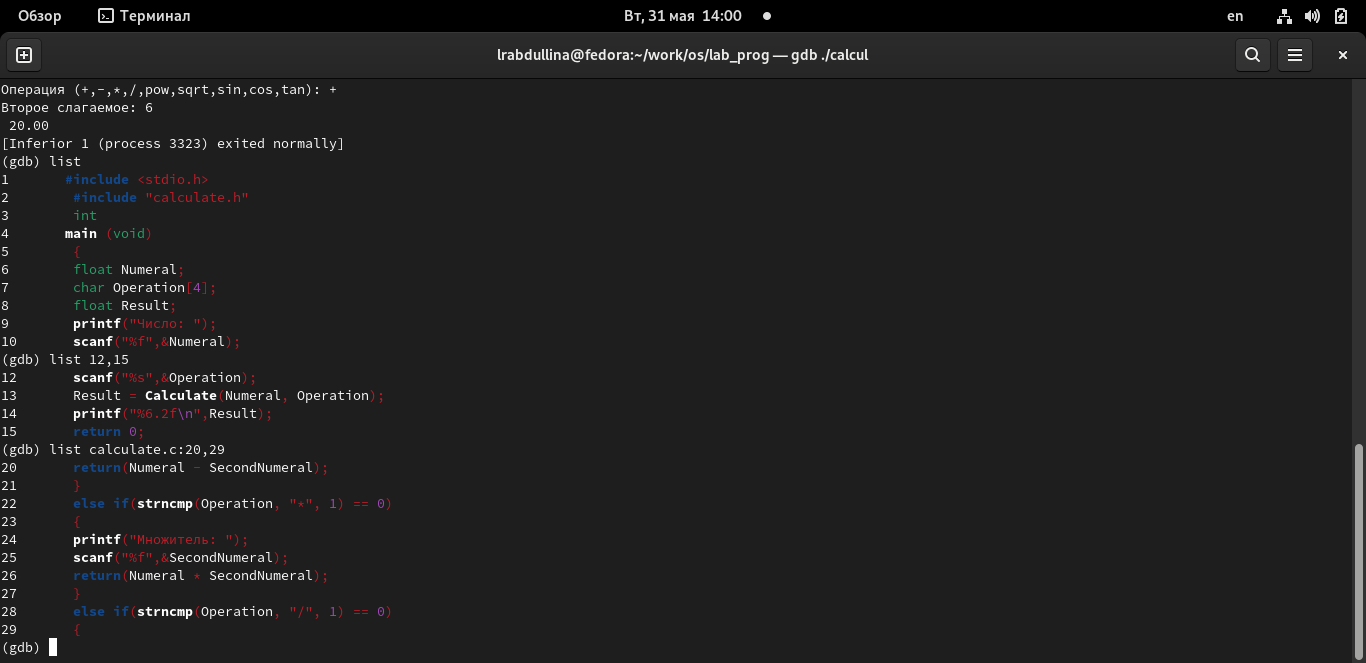
Скриншот 12: Просмотр кода (постранично)

* Для просмотра строк с 12 по 15 основного файла используем “list 12,15” (скриншот 13)



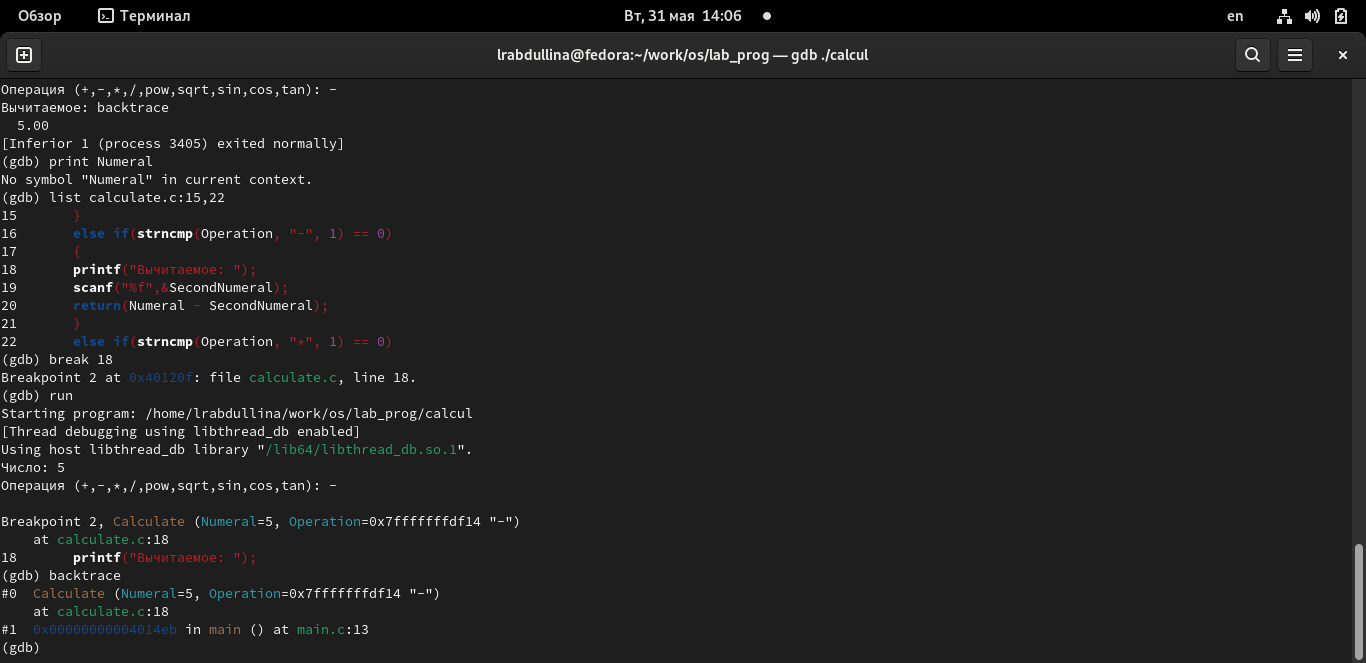
Скриншот 13: Просмотр строк 12-15

* Для просмотра определённых строк не основного файла используем “list calculate.c:20,29” (скриншот 14)



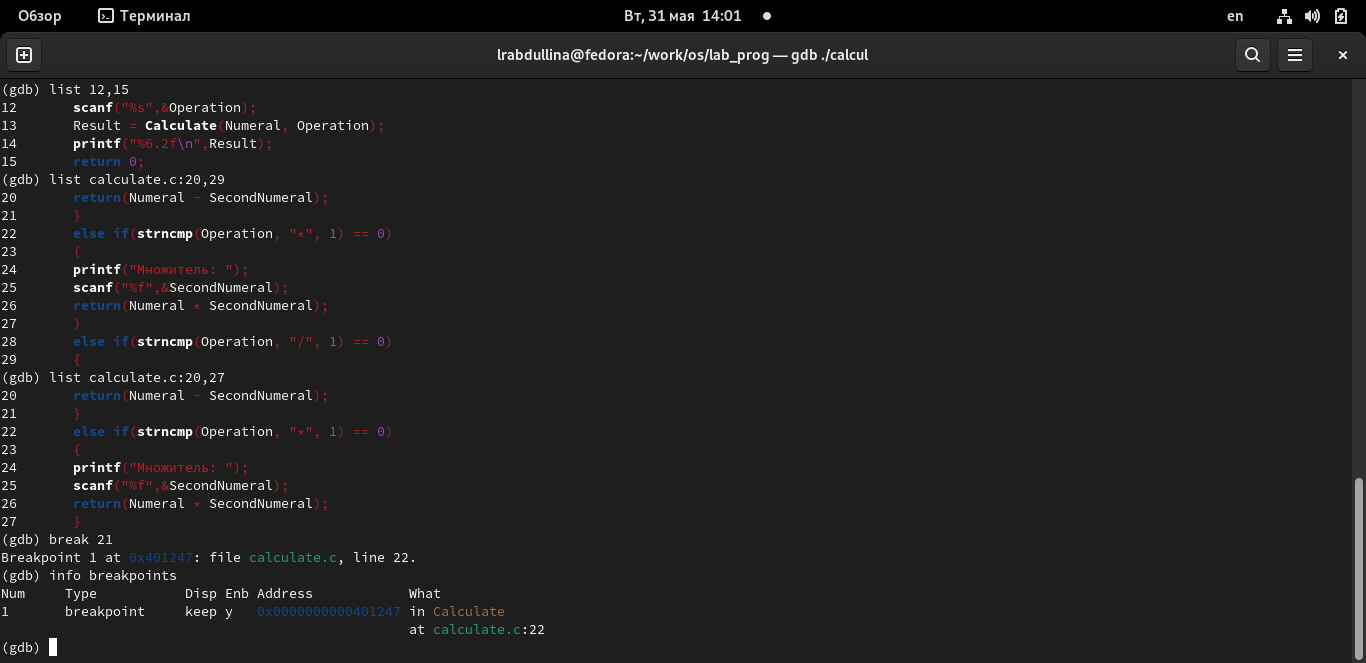
Скриншот 14: Просмотр строк не основного файла

* Установим точку останова в файле calculate.c на строке номер 18 с командами “list calculate.c:20,27”, “break 18” (скриншот 15)



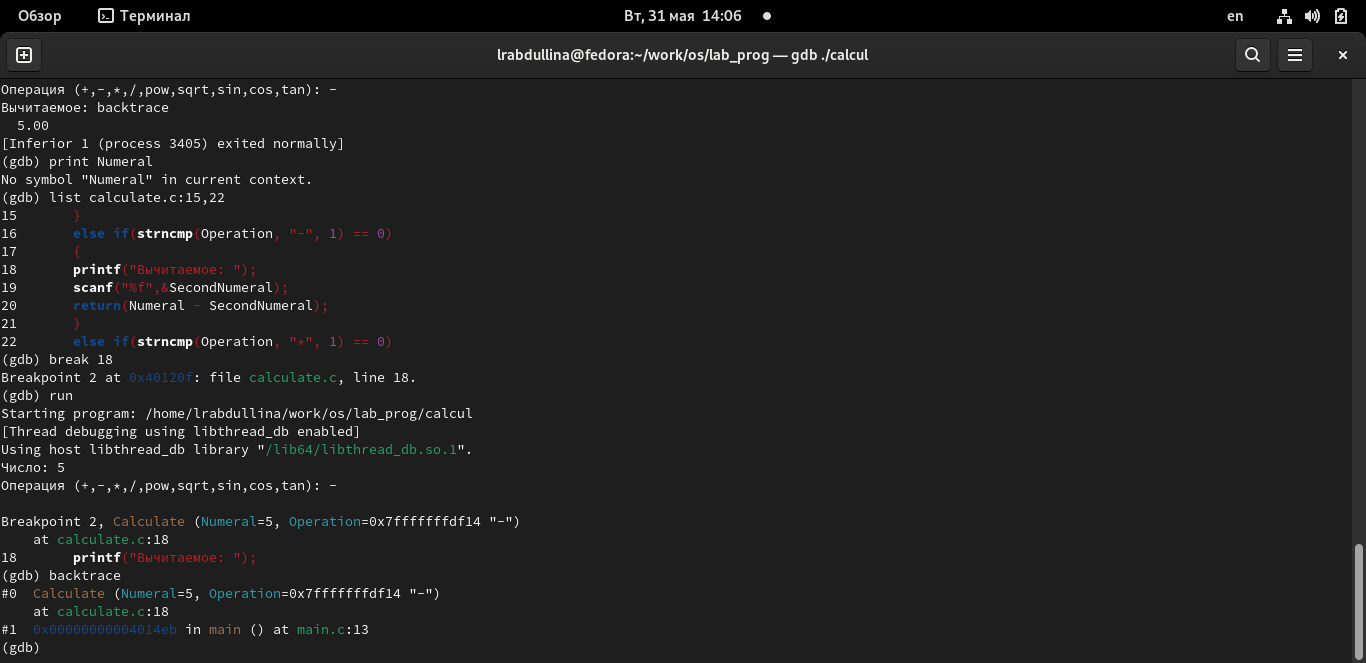
Скриншот 15: Установка точки останова

* Выведим информацию об имеющихся в проекте точек останова “info breakpoints” (скриншот 16)



Скриншот 16: Информация о точках останова

* Запустите программу внутри отладчика и убедимся,что программа остановится в момент прохождени точки останова. (скриншот 17)

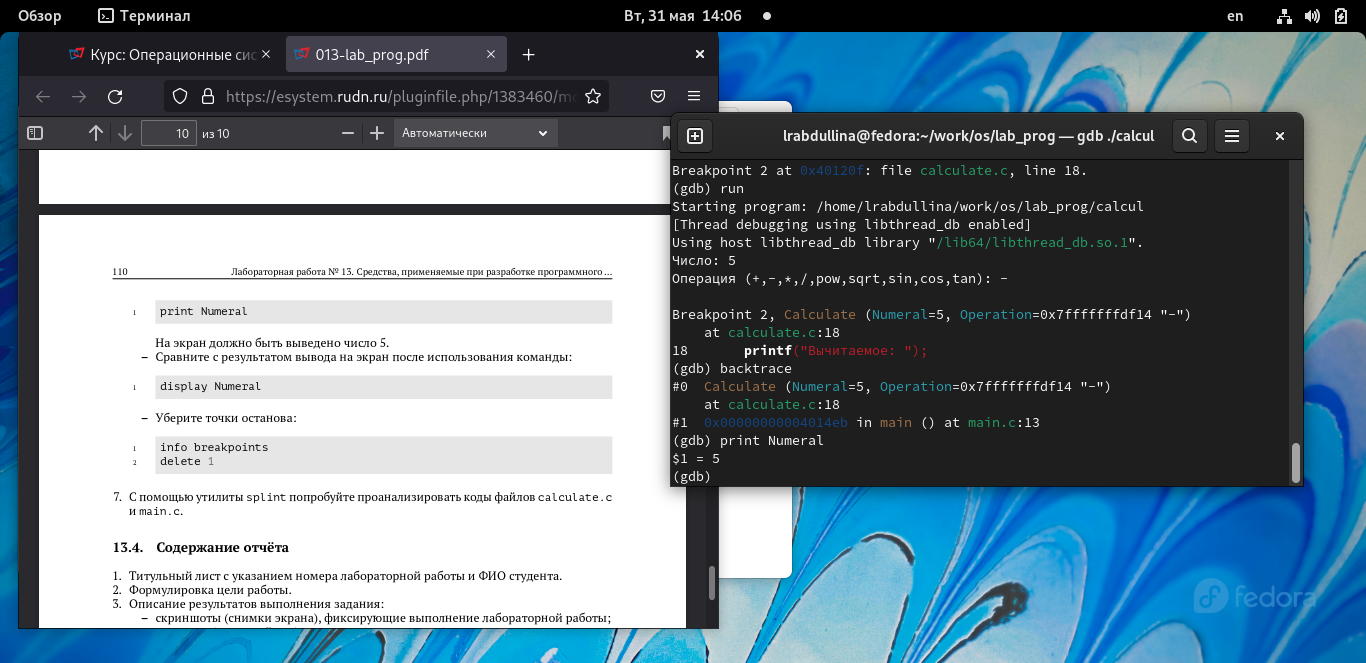


Скриншот 17: Оставка у точки

* Отладчик выдал следующую информацию: #0 Calculate (Numeral=5, Operation=0x7fffffffd280 “-”) at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17

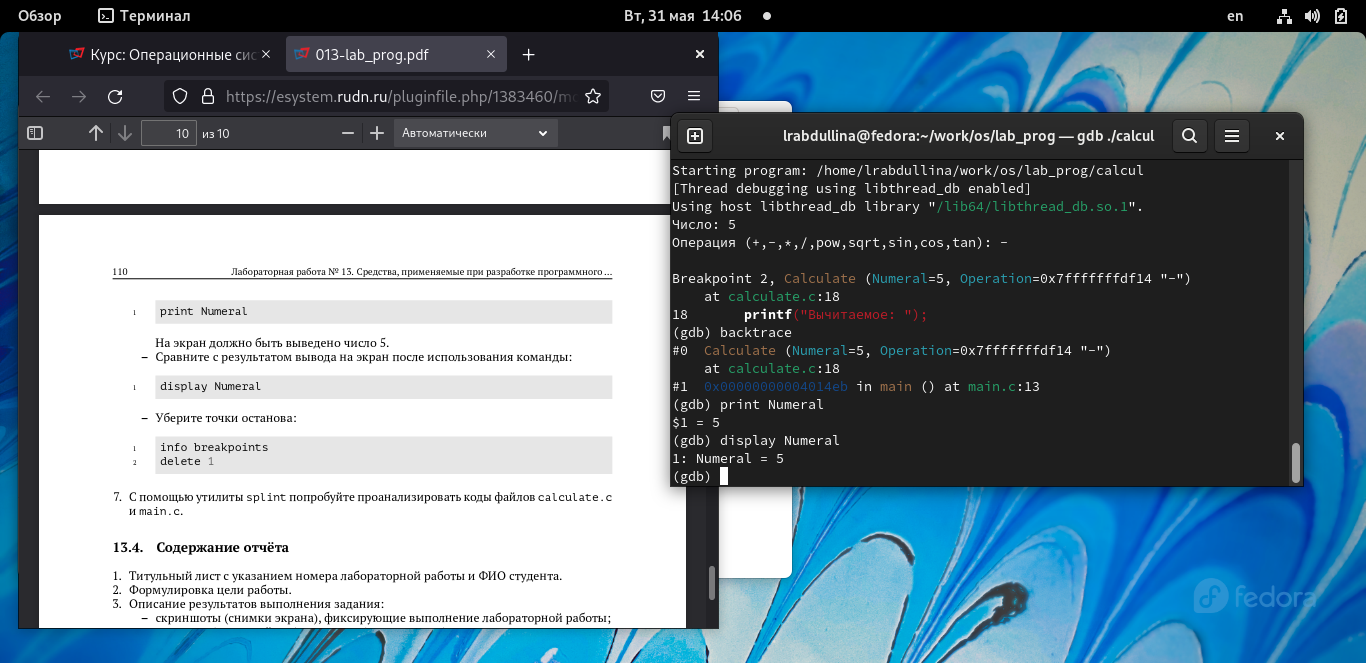
а команда “backtrace” показала весь стек вызываемых функций от начала программы до текущего места.

* Посмотрим,чему равно на этом этапе значение переменной Numeral,введя “print Numeral”. На экране выведено число 5. (скриншот 18)



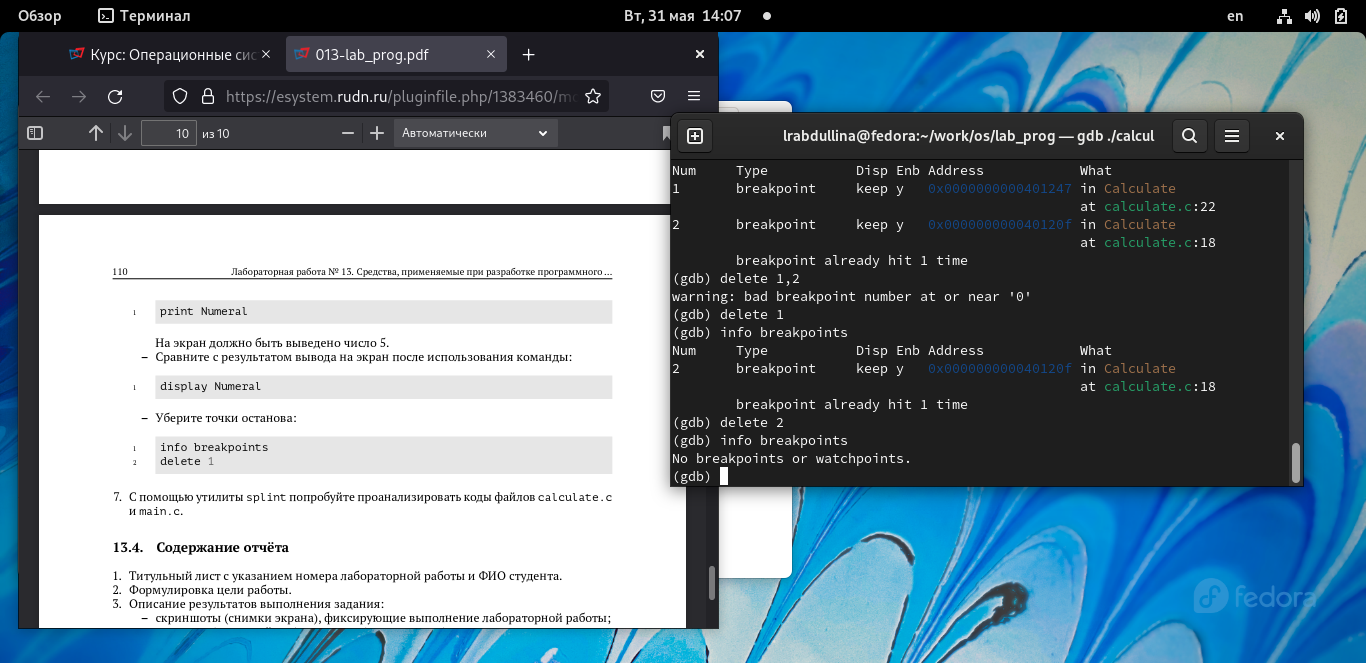
Скриншот 18: Значение Numeral

* Сравним с результатом вывода на экран после использования команды “display Numeral”. *Значения совпадают* (скриншот 19)



Скриншот 19: Значение Numeral

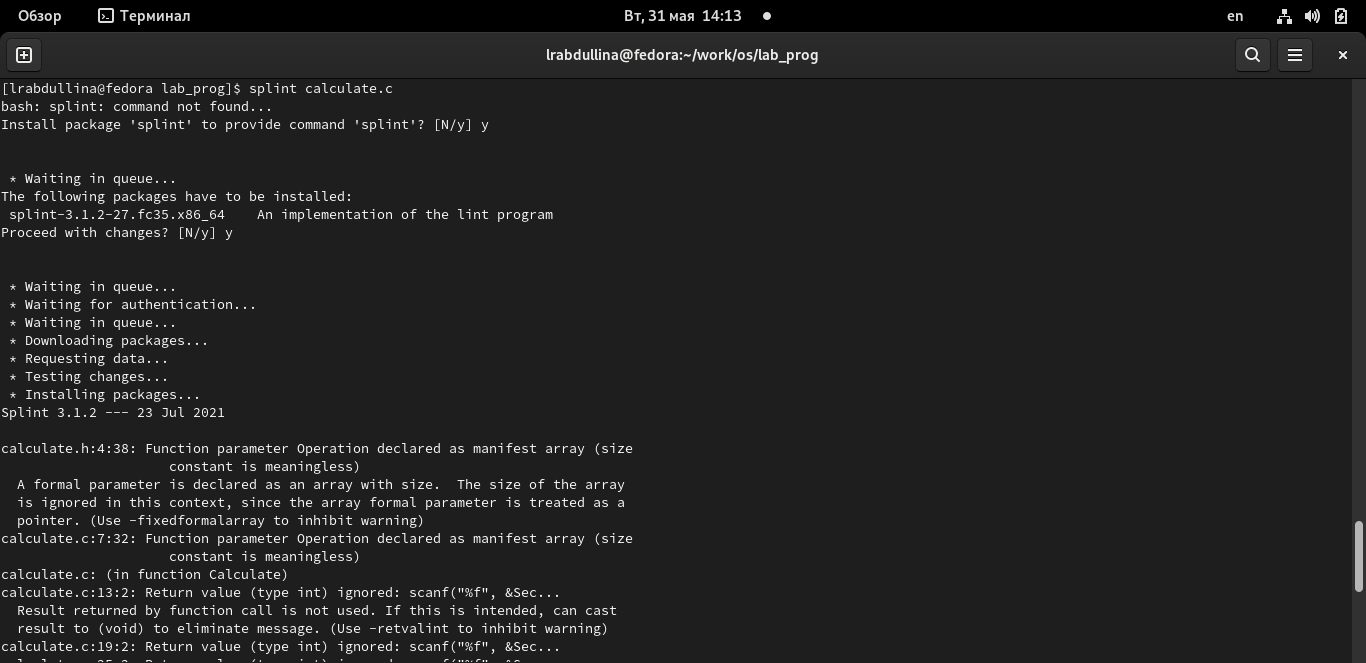
* Уберем точки останова “info breakpoints”, “delete 1” (скриншот 20)



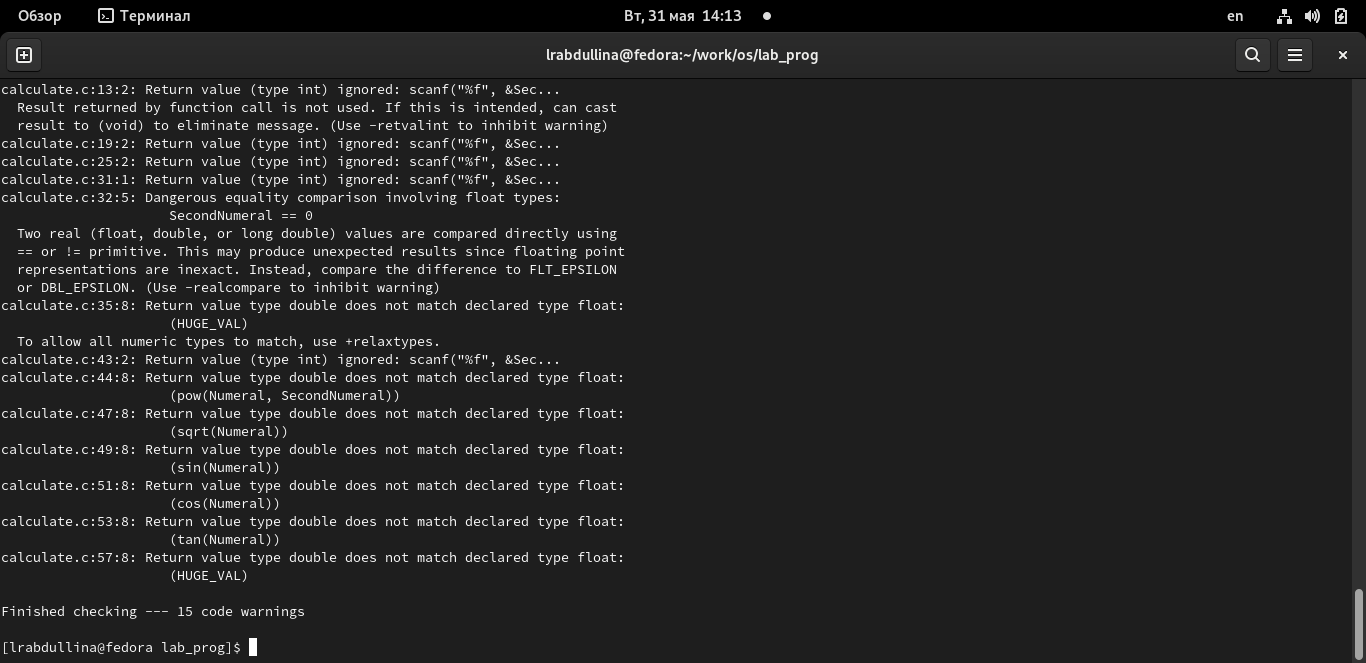
Скриншот 20: Убираем точки останова

## 3.7 7

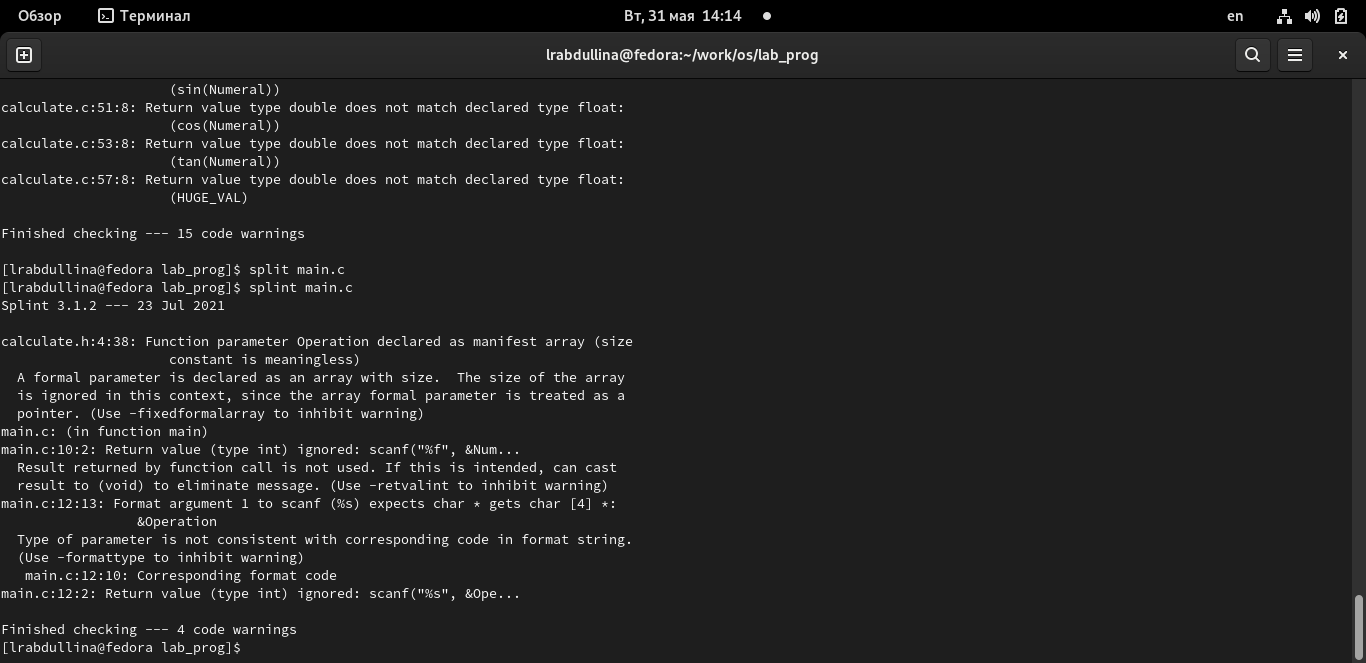
С помощью утилиты splint попробуем проанализировать коды файлов calculate.c и main.c. (скриншот 21,22, 23)



Скриншот 21: Загрузка утилиты



Скриншот 22: Информация о calculate.c



Скриншот 23: Информация о main.c

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.

# 4 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:

* планирование, включающее сбор и анализ требований кфункционалу 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. 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».
2. Основное назначение компилятора языка Си в UNIX заключается вкомпиляции 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 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 водной 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.

1. 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 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
2. 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 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 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. 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 массива символов уже является указателем на первый элемент этого 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.

1. 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 работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

# 5 Выводы

В ходе 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.

# 6 Список литературы

https://esystem.rudn.ru/course/view.php?id=5790 :::