## РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

### Кафедра прикладной информатики и теории вероятностей

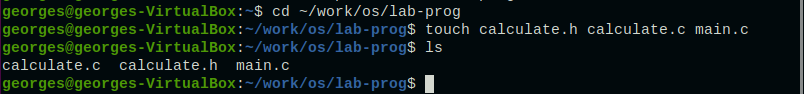
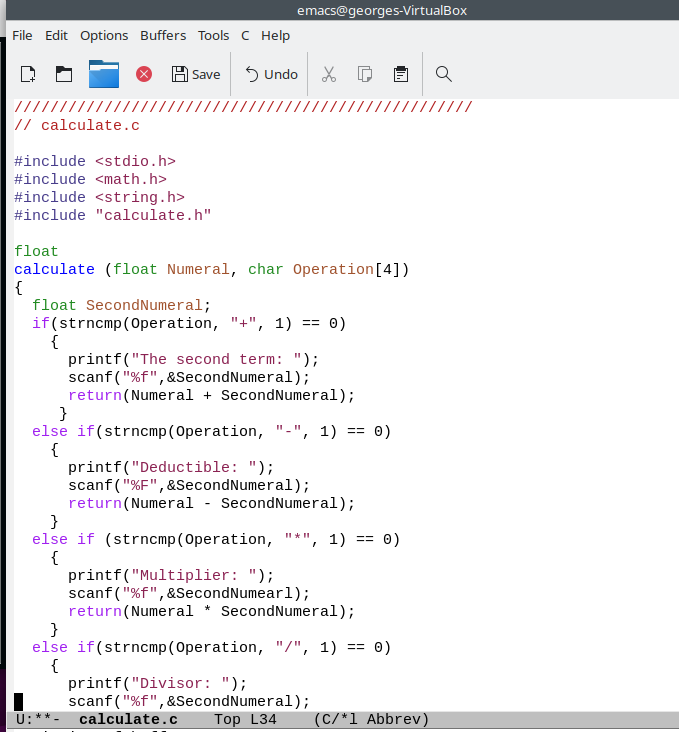
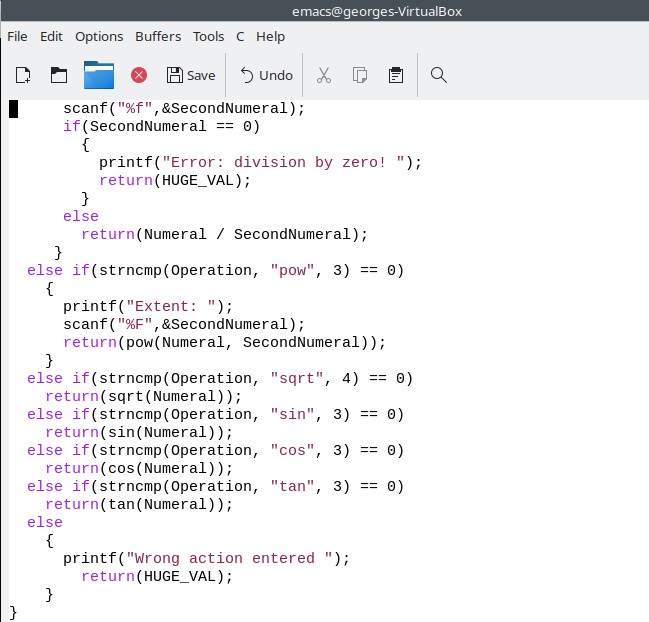
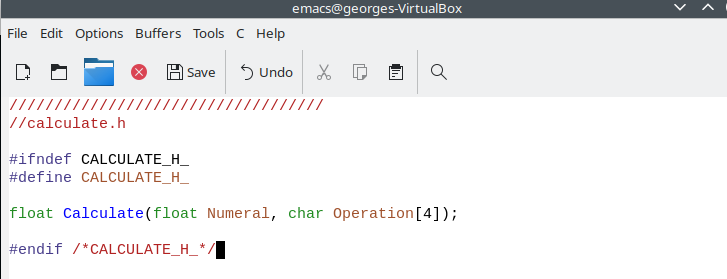
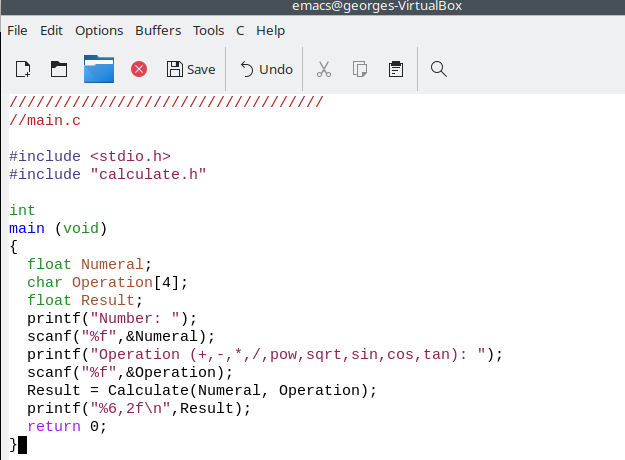
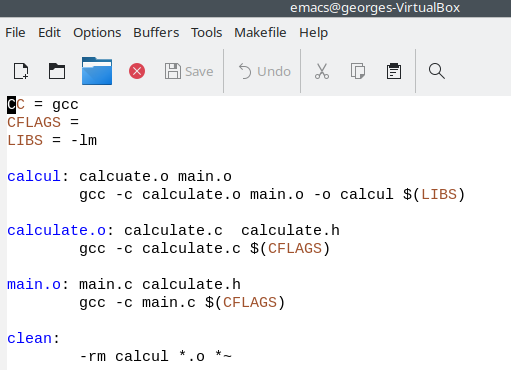
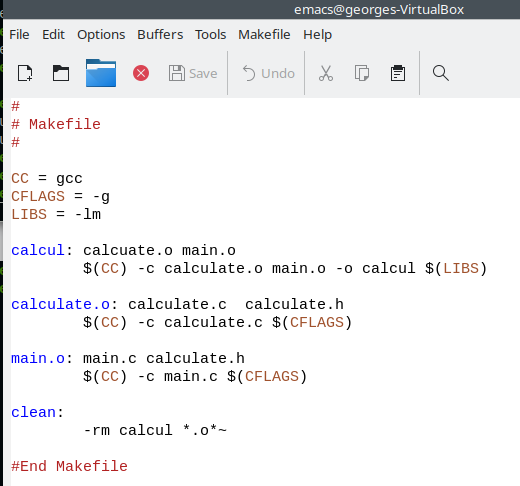
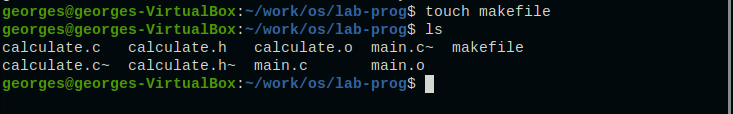
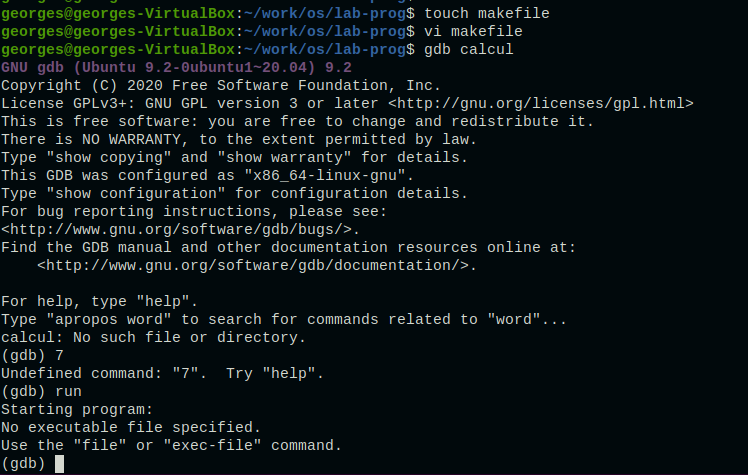
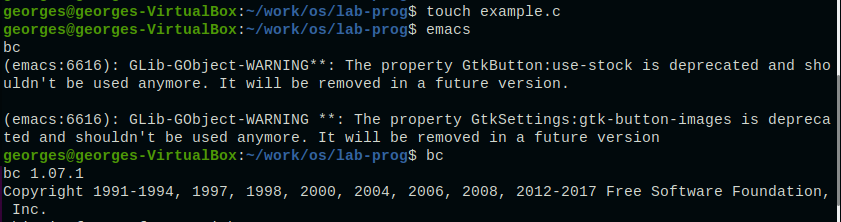
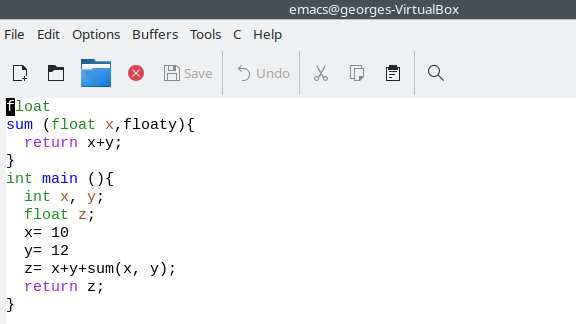
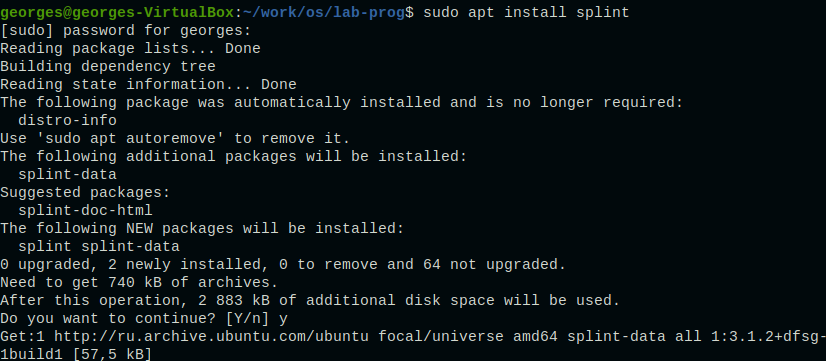
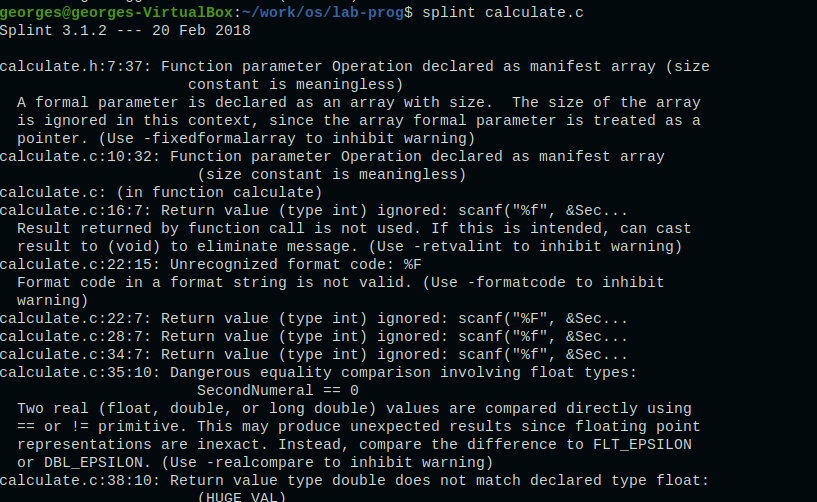
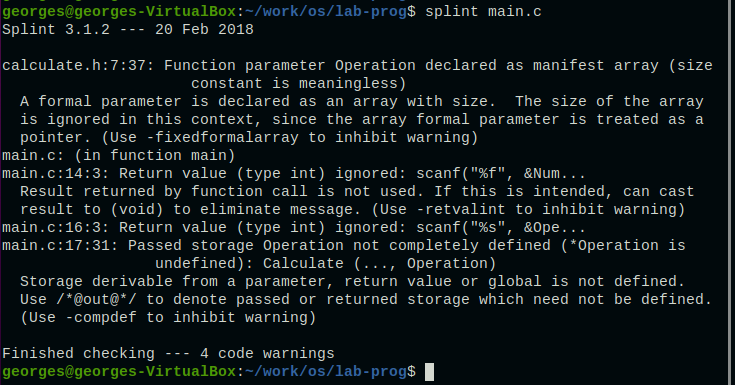
### ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

*дисциплина: Операционные системы*

**Студент: ГЕОРГЕС Гедеон** **Группа: НПМбд-02-20** **Студ. бил.:№ 1032204593**

МОСКВА 2021 г.

# 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. # Ход работы: 1. В домашнем каталоге создаю подкаталог ~/work/os/lab\_prog с помощью команды «mkdir -p ~/work/os/lab\_prog» (Рисунок 1).  (Рисунок 1) 1. 1.Создал в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd ~/work/os/lab\_prog» и «touch calculate.h calculate.c main.c» (Рисунок 2)  (Рисунок 2) 1. 2.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, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с (Рисунки 3, 4)  (Рисунок 3)  (Рисунок 4) 1. 3.Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (Рисунок 5)  (Рисунок 5) 1. 4.Основной файл main.c, реализующий интерфейс пользователя к калькулятору (Рисунок 6)  (Рисунок 6) 2. Выполнил 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» (Рисунок 7)  (Рисунок 7) 3. Создал Makefile с необходимым содержанием (Рисунок 8)  (Рисунок 8) 3. 1.Далее исправила Makefile (Рисунок 9).  (Рисунок 9) 3. 2.Создали Makefile со следующим содержанием:  (Рисунок 10) 4. Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB загрузив в него программу для отладки используя команду: «gdb ./calcul» (Рисунок 11)  (Рисунок 11) 5. Создали файл example.c, записали программу в данный файл (Рисунок 12).  (Рисунок 12)  (Рисунок 12) С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Предварительно я установила данную утилиту с помощью команд «sudo apt update» и «sudo apt install splint» (Рисунок 13)  (Рисунок 13) 7. С помощью утилиты splint проанализировали коды файлов calculate.c и main.c. (Рисунок 14)  (Рисунок 14)  (Рисунок 15) # Вывод: В ходе выполнения данной лабораторной работы 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. ### Ответы на 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 к функционалу и другим характеристикам разрабатываемого приложения;  проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;  непосредственная разработка приложения: o кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; o сборка, компиляция и разработка исполняемого модуля; o 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. 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 main.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 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. 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 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 8) 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. 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 массива символов уже является указателем на первый элемент этого массива. 11) 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. 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.