Лабораторная работа 13

Отчет по лабораторной работе 13

Куркина Евгения вячеславовна

Содержание

# 1 Цель работы

Здесь приводится формулировка цели лабораторной работы. Формулировки цели для каждой лабораторной работы приведены в методических указаниях.

Цель данной лабораторной 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

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

# 2 Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог ~/work/os/lab\_prog (рис. 1).

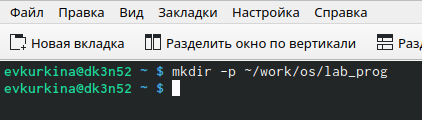


Рис. 1: Создание подкаталога

1. Создала файлы calculate.h, calculate.c, main.c. Написала текст примитивнейшего калькулятора, который 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 (рис. 2). Далее написала интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис. 3), а также текст основного файла main.c, реализующий интерфейс пользователя к калькулятору (рис. 4).

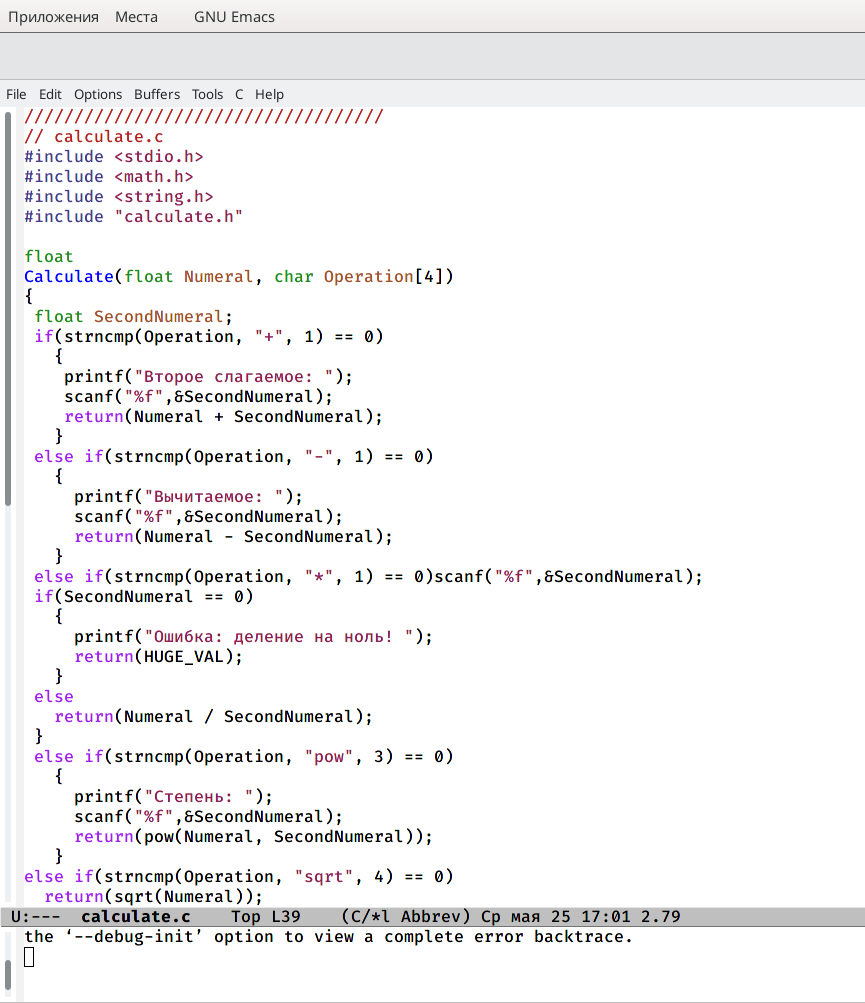


Рис. 2: Скрпит калькулятора

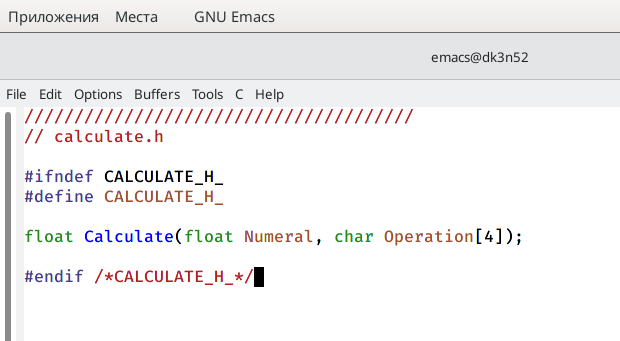


Рис. 3: Текст интерфейсного файла

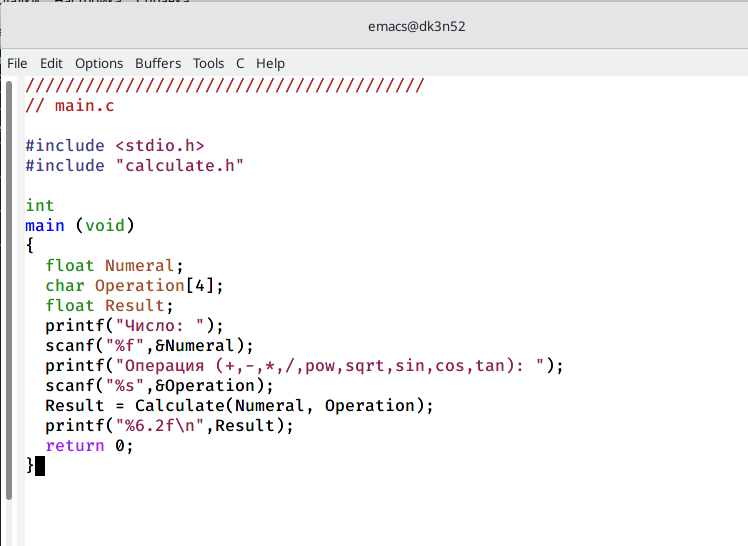


Рис. 4: Скрипт интерфейса

1. Выполнила компиляцию программы посредством gcc (рис. 5).

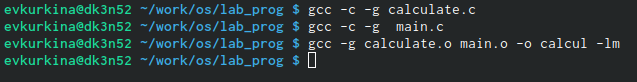


Рис. 5: Компиляция программы

1. Создала Makefile переписала в него данный текст (рис. 6), затем изменила его до рабочего состояния рис. 7). Данный файл необходим для автоматической компиляции файлов calculate.c. В переменную GLASS добавила опцию -g, утилита компиляции выбирается с помощью переменной СС. Далле я удалила файлы из каталога и выполнила компиляцию файлов (рис. 8)(рис. 9)

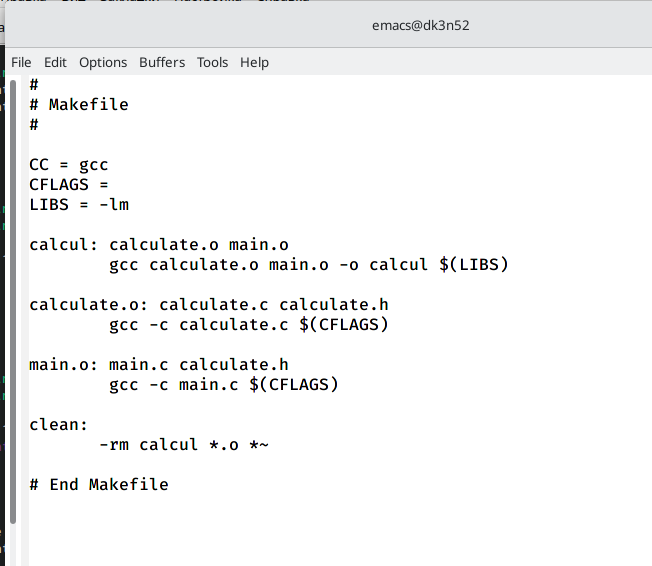


Рис. 6: Исходный текст

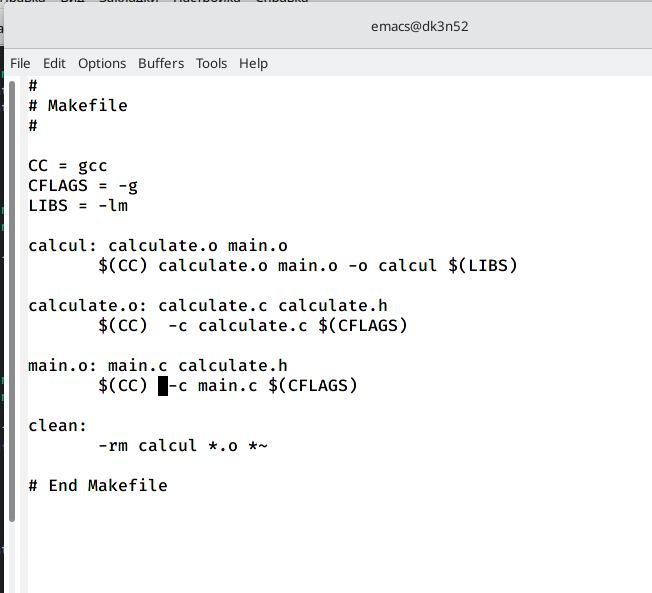


Рис. 7: Измененный текст

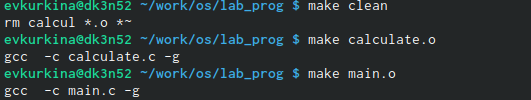


Рис. 8: удаление файлов и компиляция файлов

Рис. 9: Компиляция файлов

Рис. 9: Компиляция файлов

5)Запустила отладчик GDB, загрузив в него программу для отладки:gdb ./calcul(рис. 10).

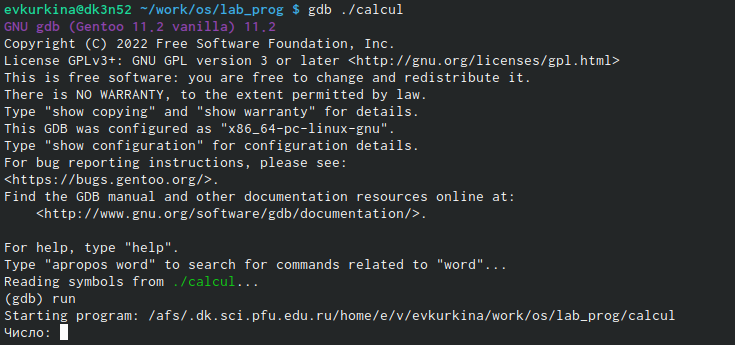


Рис. 10: Запуск отладчика

1. Запустила программу внутри отладчика командой run (рис. 11).

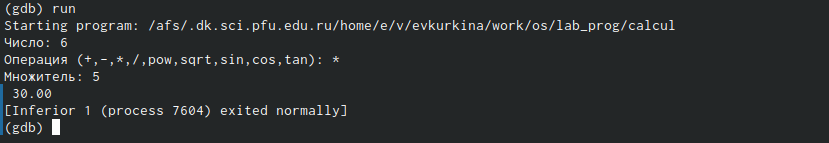


Рис. 11: Команда run

7)Для постраничного (по 9 строк) просмотра исходного код использовала команду list(рис. 12).



Рис. 12: Команда list

8)Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15(рис. 13). Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29(рис. 14).

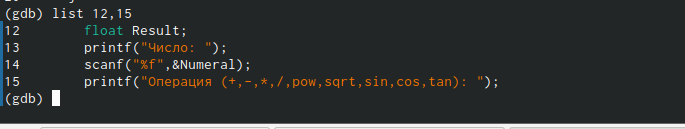


Рис. 13: Команда просмотра с 12 по 15 строк

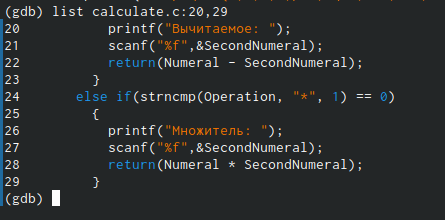


Рис. 14: Команда просмотра определенных строк

1. Установила точку отановки на 21 строке:list calculate.c:20,27 break 21 (рис. 15).

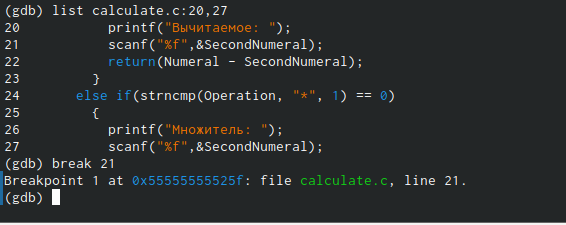


Рис. 15: Установка точки остановки

10)Вывела информацию об имеющихся точках останова:info breakpoints (рис. 16).

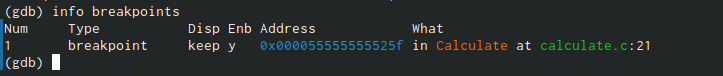


Рис. 16: Информация о точках останова

11)Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова: run 5 - backtrace Отладчик выдал следующую информацию: 1 #0 Calculate (Numeral=5, Operation=0x7fffffffd280 “-”) 2 at calculate.c:21 3 #1 0x0000000000400b2b in main () at main.c:17(рис. 17).

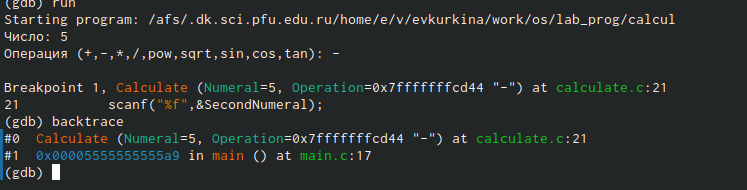


Рис. 17: Проверка точки остановки

1. Просмотрела, чему равно на этом этапе значение переменно Numeral: print Numeral (рис. 18).

Рис. 18: Значение переменной

Рис. 18: Значение переменной

13)Сравнила с результатом вывода на экран после использования команды: display Numeral (рис. 19).

Рис. 19: Сравнение результатов

Рис. 19: Сравнение результатов

14)Убрала точку останова info breakpoints delete (рис. 20).

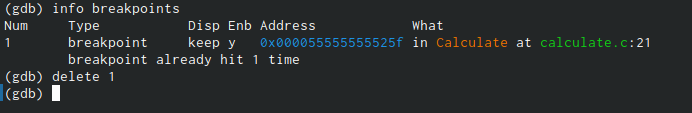


Рис. 20: Удаление точки останова

1. С помощью утилиты splint проанализировала коды файлов calculate.c и main.c (рис. 21).(рис. 22).

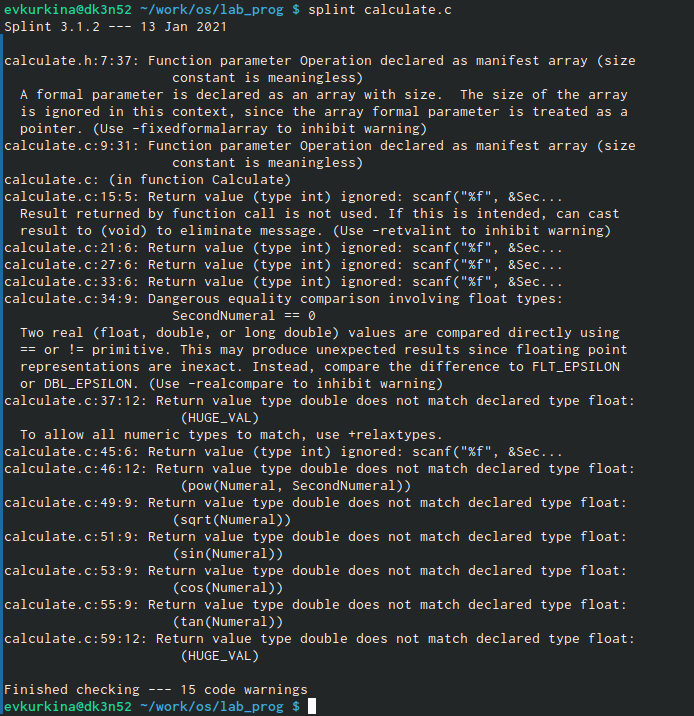


Рис. 21: Анализ кода файла calcilate.c

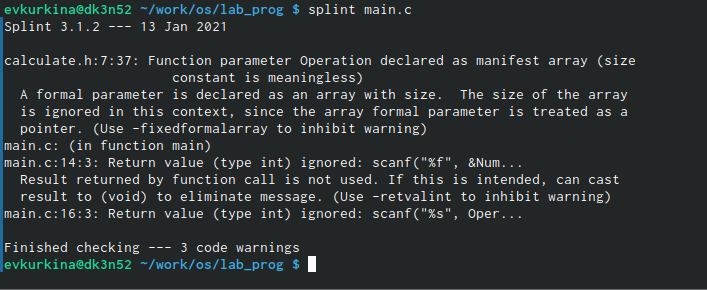


Рис. 22: Анализ кода файла main.c

16)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.

3). 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».

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 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

(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 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

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-hи mangdb.

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,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 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. # Выводы

Во время выполнения данной лабораторной работы, я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений OC UNIX/LINUX на примере создания на языке программирования С калькулятора с простейшими функциями.

# Список литературы