# 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: Ким Реачна Группа: НПИбд-02-20

Москва 2021г.

### 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.

### Теоретичекое введение:

**Этапы разработки приложений:**

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.

**Компиляция исходного текста и построение исполняемого файла:**

Для компиляции файла main.c, достаточно в командной строке ввести:

gcc -c main.c

Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла:

gcc -c hello main.c

**Тестирование и отладка:**

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

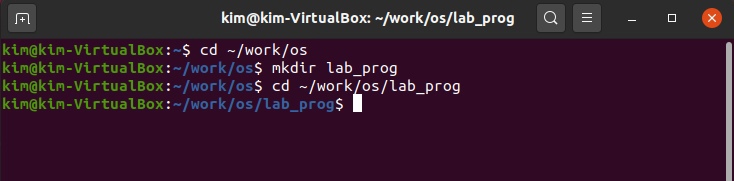
**Анализ исходного текста программы:**

splint <file.c>

### Выполнение работы:

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog *(Рисунок 1)*

*Рисунок 1: создайте подкаталог ~/work/os/lab\_prog*

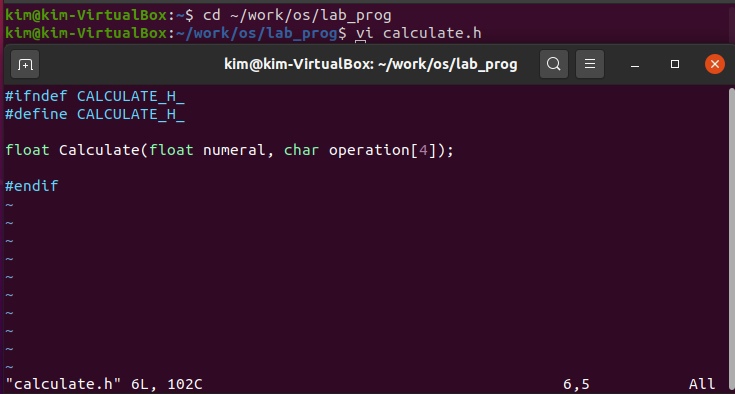


Сначала мы меняем наш каталог на ~/work/os, а затем создаем файл lab\_prog с помощью команды mkdir.

1. . Создайте в нём файлы: calculate.h, calculate.c, main.c. 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.

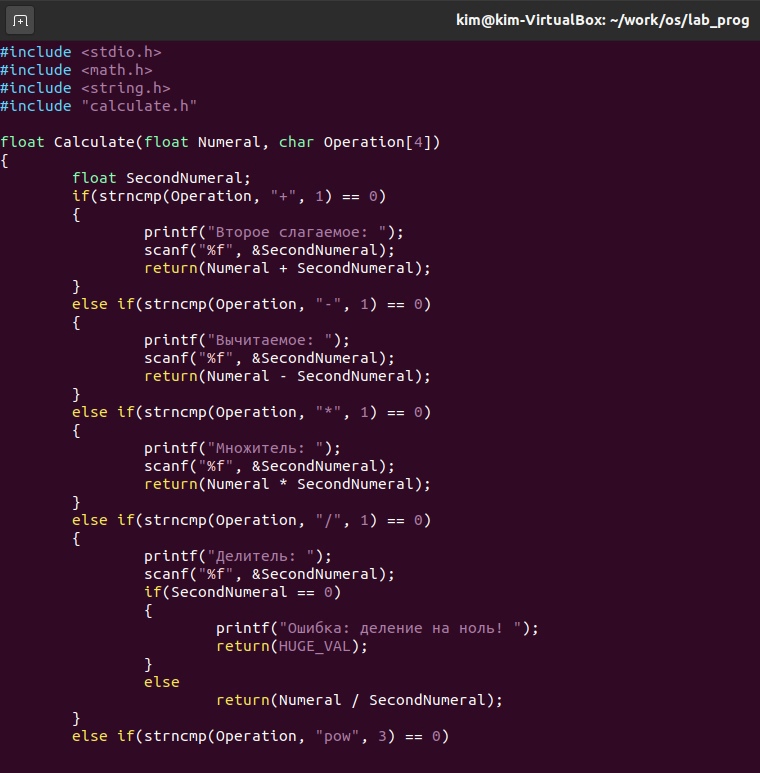
2.1. Сначала давайте создадим файл calculate.h использование редактора команд vi calculate.h *(Рисунок 2)*

*Рисунок 2: создадим файл calculate.h*



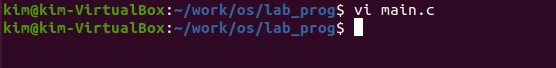
2.2. Затем создайте файл calculate.c с помощью редактора vi снова vi calculate.c*(Рисунок 3)*

*Рисунок 3: создайте файл calculate.c*

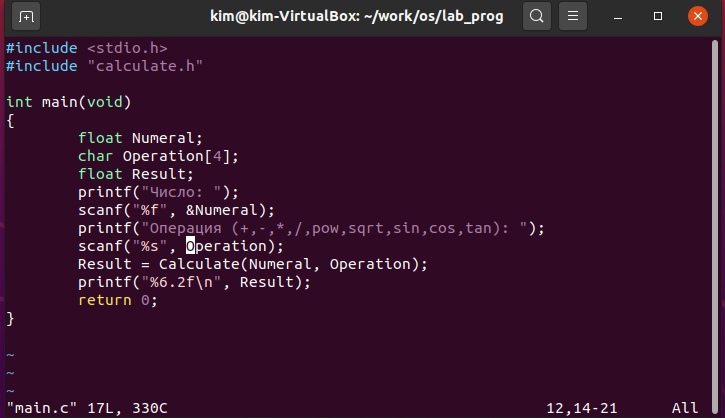


2.3. Теперь создайте командный файл main.c *(Рисунок 4-5)*

*Рисунок 4: создайте командный файл main.c*



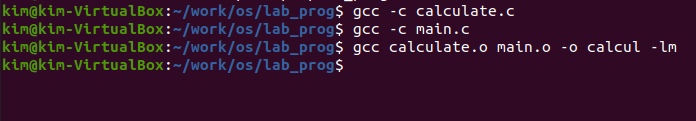
*Рисунок 5: создайте командный файл main.c*



1. Выполните компиляцию программы посредством gcc: *(Рисунок 6)*

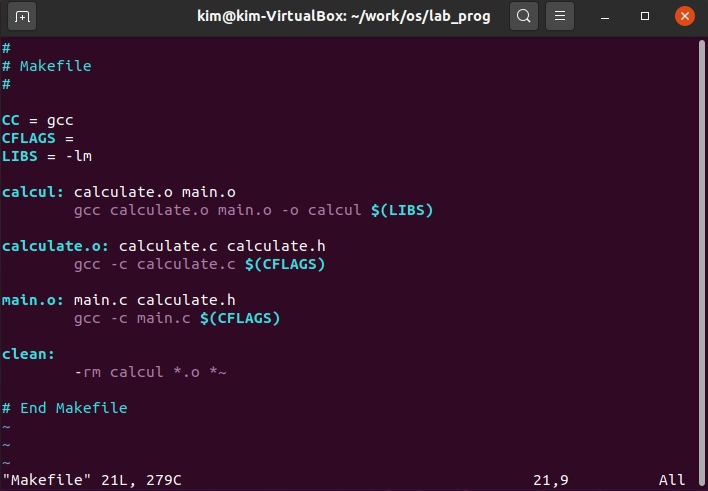
gcc -c calculate.c  
gcc -c main.c  
gcc calculate.o main.o -o calcul -lm

*Рисунок 6: gcc*



1. При необходимости исправьте синтаксические ошибки: Как мы видим в *(Рисунок 6)* ошибок нет.
2. Создайте Makefile со следующим содержанием:

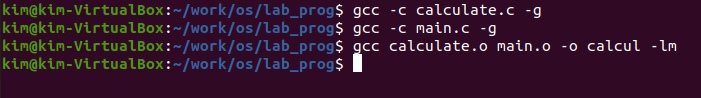
*Рисунок 7: Создайте Makefile*



1. С помощью gdb выполните отладку программы calcul(перед использованием gdb исправьте Makefile):

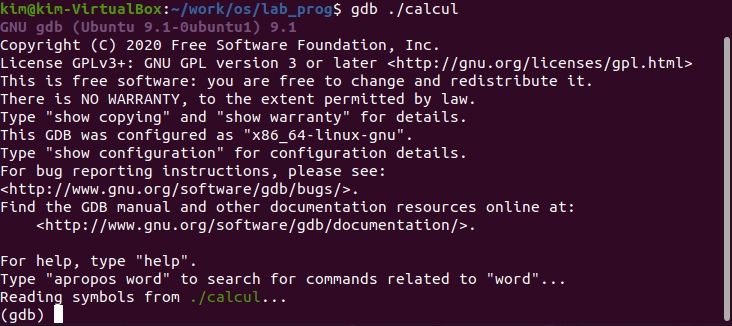
Для этого сначала мы запустим некоторую команду *(Рисунок 8)*, потому что при запуске команды gdb возникнут некоторые ошибки.

*Рисунок 8: gcc c -g*



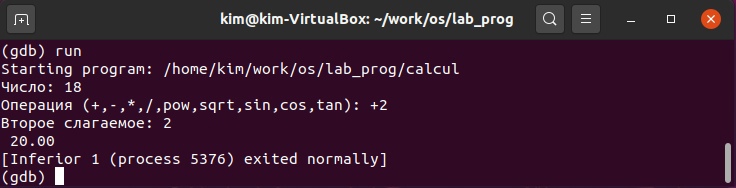
* Запустите отладчик GDB, загрузив в него программу для отладки: gdb ./calcul *(Рисунок 9)*

*Рисунок 9: Запустите отладчик GDB*



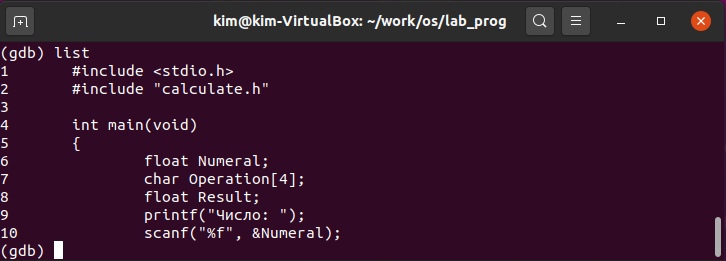
* Для запуска программы внутри отладчика введите команду run: run *(Рисунок 10)*

*Рисунок 10: запуска программы внутри отладчика*



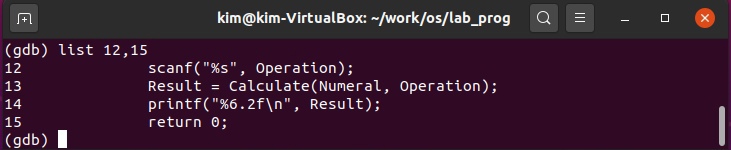
* Для постраничного (по 9 строк) просмотра исходного код используйте команду list: list *(Рисунок 11)*

*Рисунок 11: постраничного (по 9 строк)*



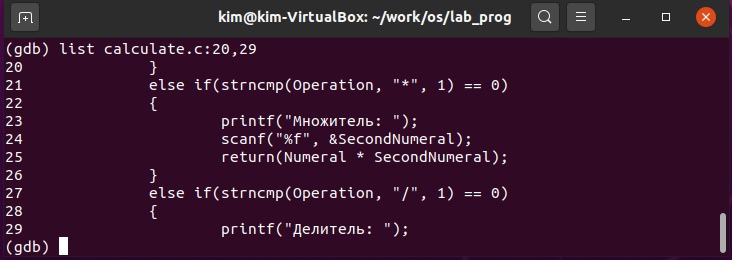
* Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15 *(Рисунок 12)*

*Рисунок 12: просмотра строк с 12 по 15*



* Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29 *(Рисунок 13)*

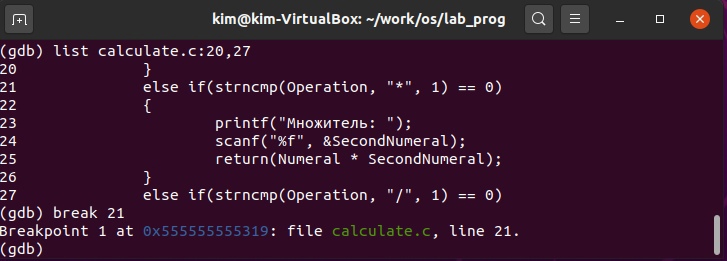
*Рисунок 13: просмотра определённых строк*



* Установите точку останова в файле calculate.c на строке номер 21: *(Рисунок 14)*

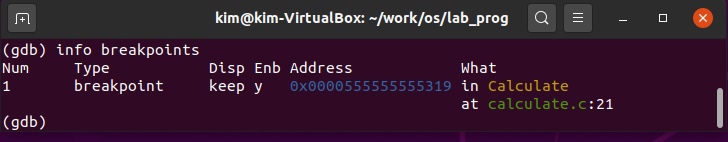
list calculate.c:20,27  
break 21

*Рисунок 14: Установите точку останова*



* Выведите информацию об имеющихся в проекте точка останова: info breakpoints *(Рисунок 15)*

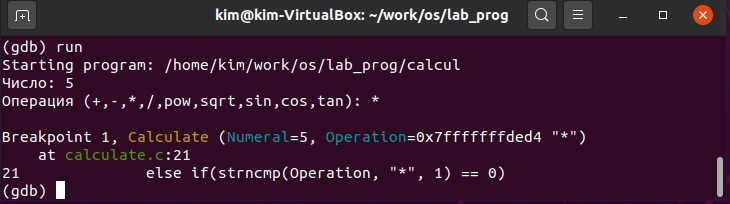
*Рисунок 15: Выведите информацию об имеющихся*



* Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:

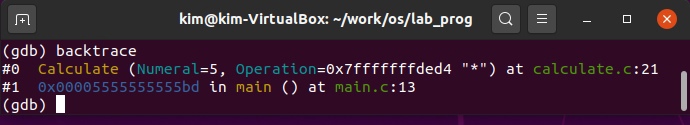
run  
5  
\*  
backtrace

*Рисунок 16: Запустите программу*



Как мы видим , здесь я использовал символ*, потому что мои точки останова здесь в строке 21-символ \* , поэтому нам нужно поставить \*, иначе он не будет показывать сообщения, как в* (Рисунок 16)\*

*Рисунок 17: gdb backtrace*



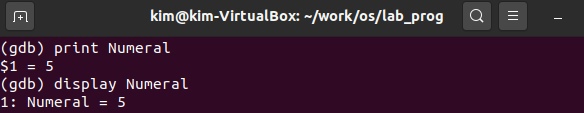
* Отладчик выдаст следующую информацию:

#0 Calculate (Numeral=5, Operation=0x7fffffffd280 "-")  
at calculate.c:21  
#1 0x0000000000400b2b in main () at main.c:17

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места.

* – Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран должно быть выведено число 5 и Сравните с результатом вывода на экран после использования команды: display Numeral *(Рисунок 18)*

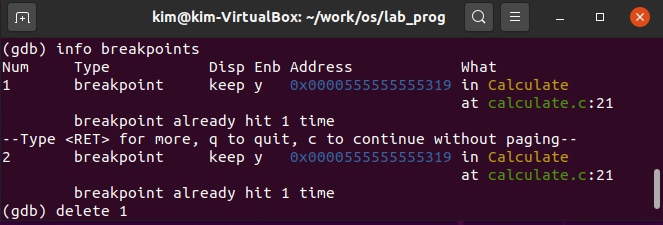
*Рисунок 18: Посмотрите Numeral*



* Уберите точки останова: *(Рисунок 19)*

info breakpoints  
delete 1

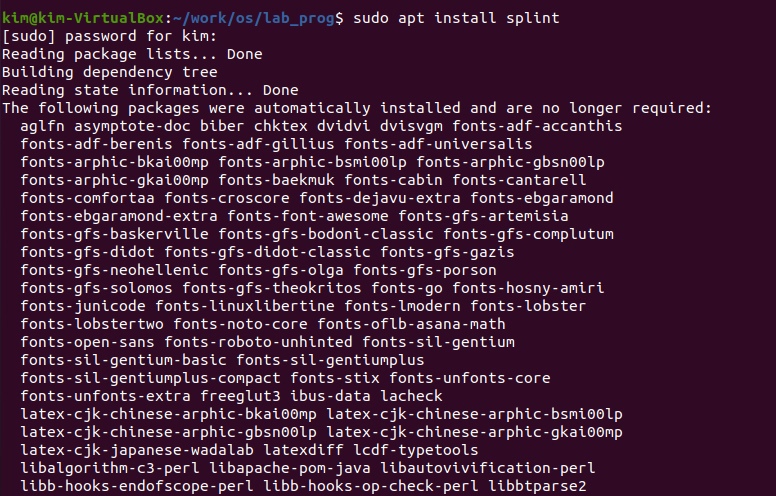
*Рисунок 19: Уберите точки останова*



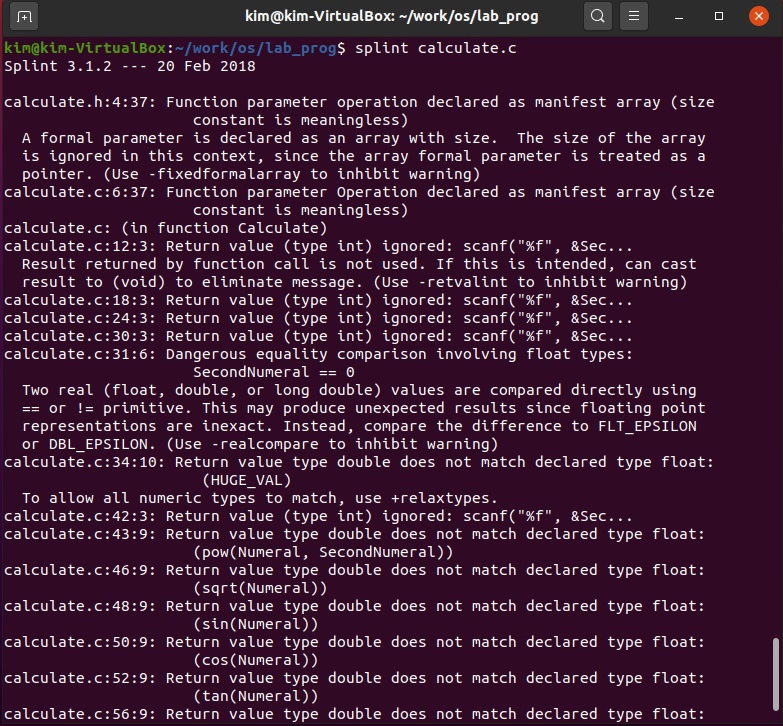
1. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

Сначала мы установим splint с помощью sudo apt install splint *(Рисунок 20)*

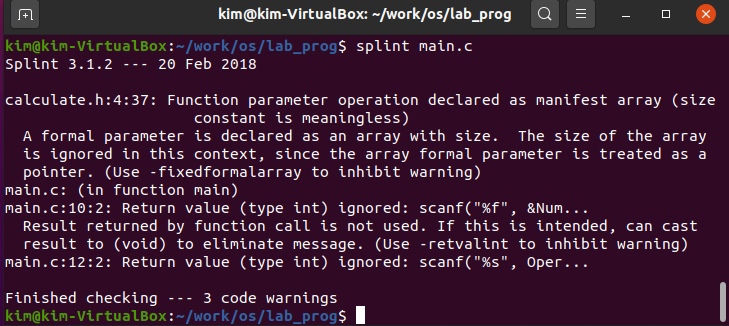
*Рисунок 20: установим splint*



*Рисунок 21: проанализировать файлов calculate.c*



*Рисунок 22: проанализировать файлов main.c*



### Контрольные вопросы:

1. C помощью функций info и man.
   * создание исходного кода программы, которая представляется в виде файла
   * сохранение различных вариантов исходного текста;
   * анализ исходного текста; необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.
   * компиляция исходного текста и построение исполняемого модуля;
   * тестирование и отладка;
   * проверка кода на наличие ошибок
   * сохранение всех изменений, выполняемых при тестировании и отладке.
2. Использование суффикса “.с” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.о является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция – prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.
3. Компиляция всей программы в целом и получении исполняемого модуля.
4. Мake-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые.
5. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами OC UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат:

* target1 [ target2...]: [:] [dependment1...]
* [(tab)commands]
* [#commentary]
* [(tab)commands]
* [#commentary],
* где # — специфицирует начало комментария, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; : — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний.

1. Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.
   * backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций,начиная от main(); иными словами, выводит весь стек функций;
   * break – устанавливает точку останова; параметром может быть номер строки или название функции;
   * clear – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
   * continue – продолжает выполнение программы от текущей точки до конца;
   * delete – удаляет точку останова или контрольное выражение;
   * display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
   * finish– выполняет программу до выхода из текущей функции; отображает возвращаемое значение,если такое имеется;
   * info breakpoints – выводит список всех имеющихся точек останова;
   * info watchpoints – выводит список всех имеющихся контрольных выражений;
   * list – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
   * next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;
   * print – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
   * run – запускает программу на выполнение;
   * set – устанавливает новое значение переменной
   * step – пошаговое выполнение программы;
   * watch – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;
   1. Выполнили компиляцию программы
   2. Увидели ошибки в программе
   3. Открыли редактор и исправили программу
   4. Загрузили программу в отладчик gdb
   5. run — отладчик выполнил программу, мы ввели требуемые значения.
   6. программа завершена, gdb не видит ошибок.
2. Не возникло
   * cscope - исследование функций, содержащихся в программе;
   * splint — критическая проверка программ, написанных на языке Си.
   1. Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
   2. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
   3. Общая оценка мобильности пользовательской программы.

### Вывод:

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.

### Библиография:

[1]:[Описание к Лабораторная №14](https://esystem.rudn.ru/pluginfile.php/1142386/mod_resource/content/2/011-lab_prog.pdf)

[2]:[Командой splint](https://www.linuxtechi.com/split-command-examples-for-linux-unix/)