

Отчет по лабораторной работе №14

Дисциплина: Операционные системы

Тихонова Екатерина Андреевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы:	19
4	Выводы	23

Список иллюстраций

2.1	Создаем подкаталог	6
2.2	Создаем файлы	6
2.3	Редактируем файлы	7
2.4	Редактируем файлы	7
2.5	Файл	8
2.6	Файл	9
2.7	Компилируем	9
2.8	Проверяем работу	10
2.9	Исправляем файл	11
2.10	Выполняем компиляцию	11
2.11	Выполняем компиляцию	11
2.12	Выполняем отладку	12
2.13	Ввела команду	12
2.14	Используем команду	13
2.15	Используем команду	13
2.16	Используем команду	14
2.17	Используем команду	14
2.18	Используем команду	15
2.19	Используем команду	15
2.20	Используем команду	15
2.21	Сравниваем	16
2.22	Убрала точки	16
2.23	Убрала точки	17
2.24	Используем команду	18
2.25	Используем команду	18

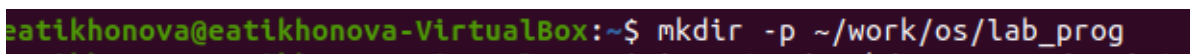
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

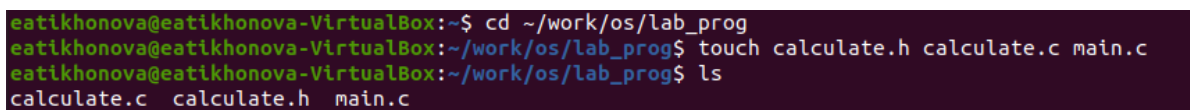
1. В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` помощью команды «`mkdir -p ~/work/os/lab_prog`»



```
eatikhonova@eatikhonova-VirtualBox:~$ mkdir -p ~/work/os/lab_prog
```

Рис. 2.1: Создаем подкаталог

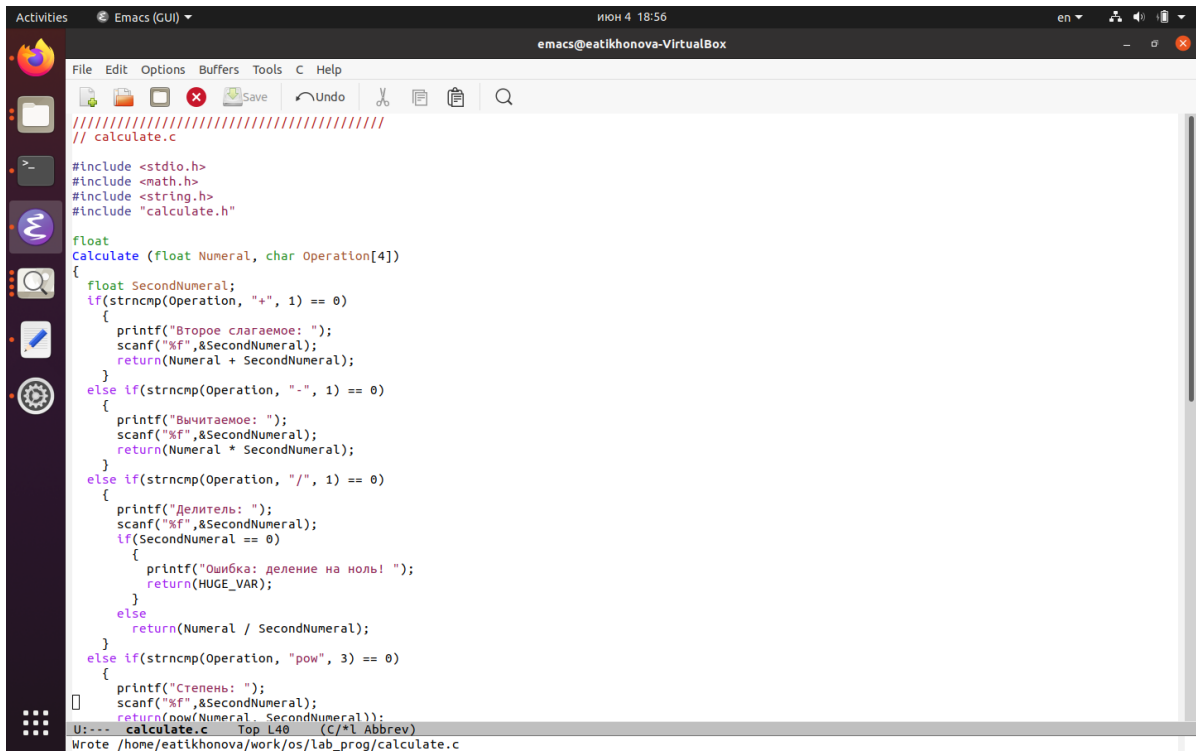
Создала в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`, используя команды «`cd ~/work/os/lab_prog`» и «`touch calculate.h calculate.c main.c`»



```
eatikhonova@eatikhonova-VirtualBox:~$ cd ~/work/os/lab_prog
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ ls
calculate.c calculate.h main.c
```

Рис. 2.2: Создаем 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. Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле `calculate.c`

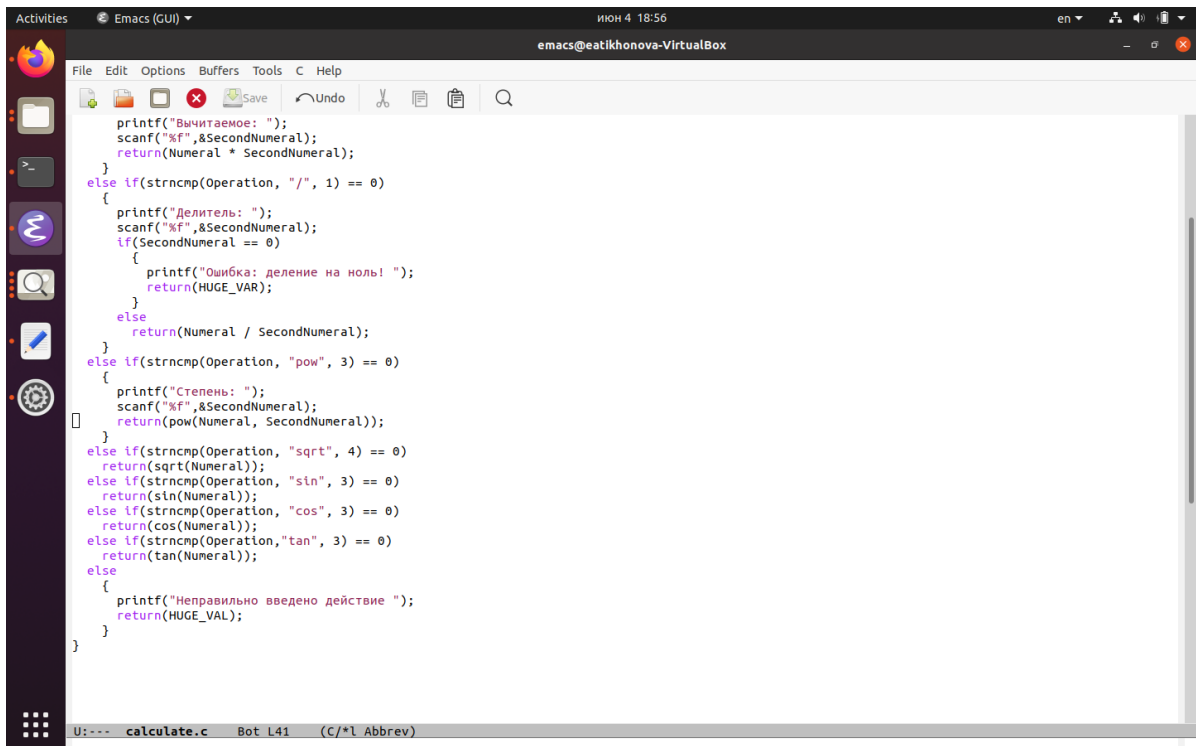


```
////// calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate (float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strcmp(Operation, "+") == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-") == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strcmp(Operation, "/") == 0)
    {
        printf("Делитель: ");
        scanf("%f",&SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка: деление на ноль! ");
            return(HUGE_VAL);
        }
        else
            return(Numeral / SecondNumeral);
    }
    else if(strcmp(Operation, "pow", 3) == 0)
    {
        printf("Степень: ");
        scanf("%f",&SecondNumeral);
        return(pow(Numeral, SecondNumeral));
    }
}

U:--- calculate.c Top L40 (C/*l Abbrev)
Wrote /home/eatikhonova/work/os/Lab_prog/calculate.c
```

Рис. 2.3: Редактируем файлы



```
printf("Вычитаемое: ");
scanf("%f",&SecondNumeral);
return(Numeral * SecondNumeral);
}
else if(strcmp(Operation, "/") == 0)
{
    printf("Делитель: ");
    scanf("%f",&SecondNumeral);
    if(SecondNumeral == 0)
    {
        printf("Ошибка: деление на ноль! ");
        return(HUGE_VAL);
    }
    else
        return(Numeral / SecondNumeral);
}
else if(strcmp(Operation, "pow", 3) == 0)
{
    printf("Степень: ");
    scanf("%f",&SecondNumeral);
    return(pow(Numeral, SecondNumeral));
}
else if(strcmp(Operation, "sqrt", 4) == 0)
    return(sqrt(Numeral));
else if(strcmp(Operation, "sin", 3) == 0)
    return(sin(Numeral));
else if(strcmp(Operation, "cos", 3) == 0)
    return(cos(Numeral));
else if(strcmp(Operation, "tan", 3) == 0)
    return(tan(Numeral));
else
{
    printf("Неправильно введено действие ");
    return(HUGE_VAL);
}
}

U:--- calculate.c Bot L41 (C/*l Abbrev)
```

Рис. 2.4: Редактируем файлы

Интерфейсный файл `calculate.h`, описывающий формат вызова функции калькулятора

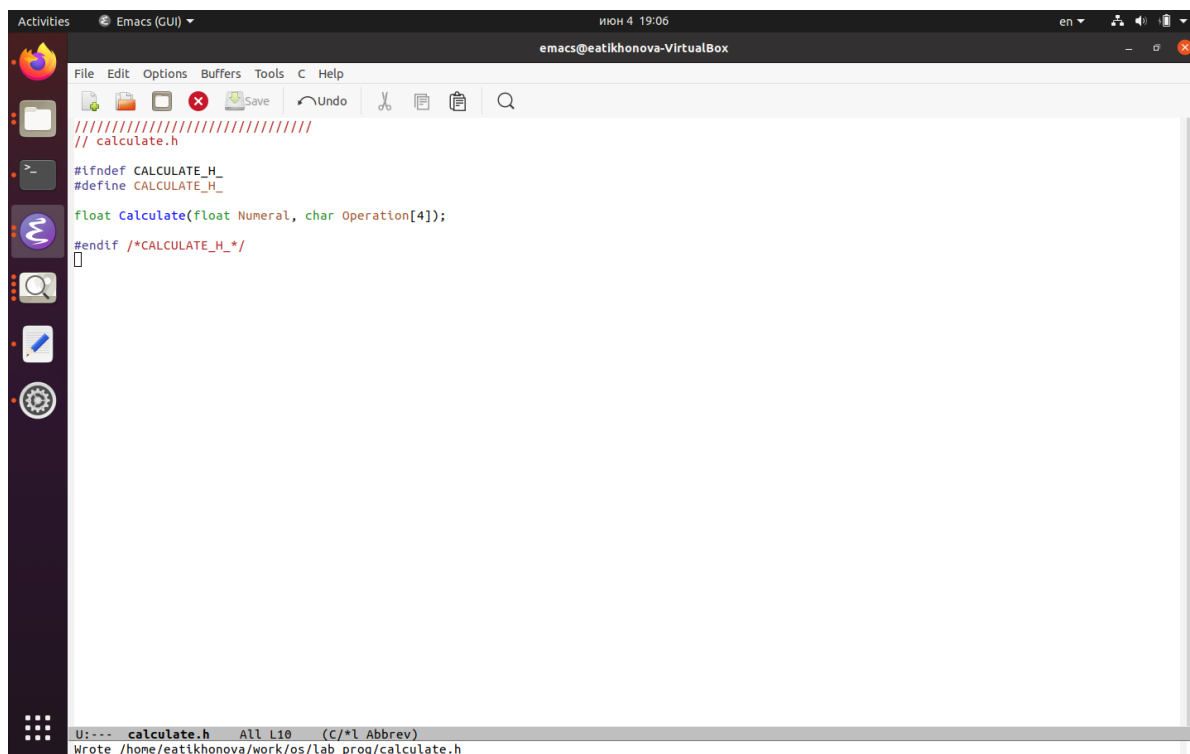


Рис. 2.5: Файл

Основной файл `main.c`, реализующий интерфейс пользователя к калькулятору

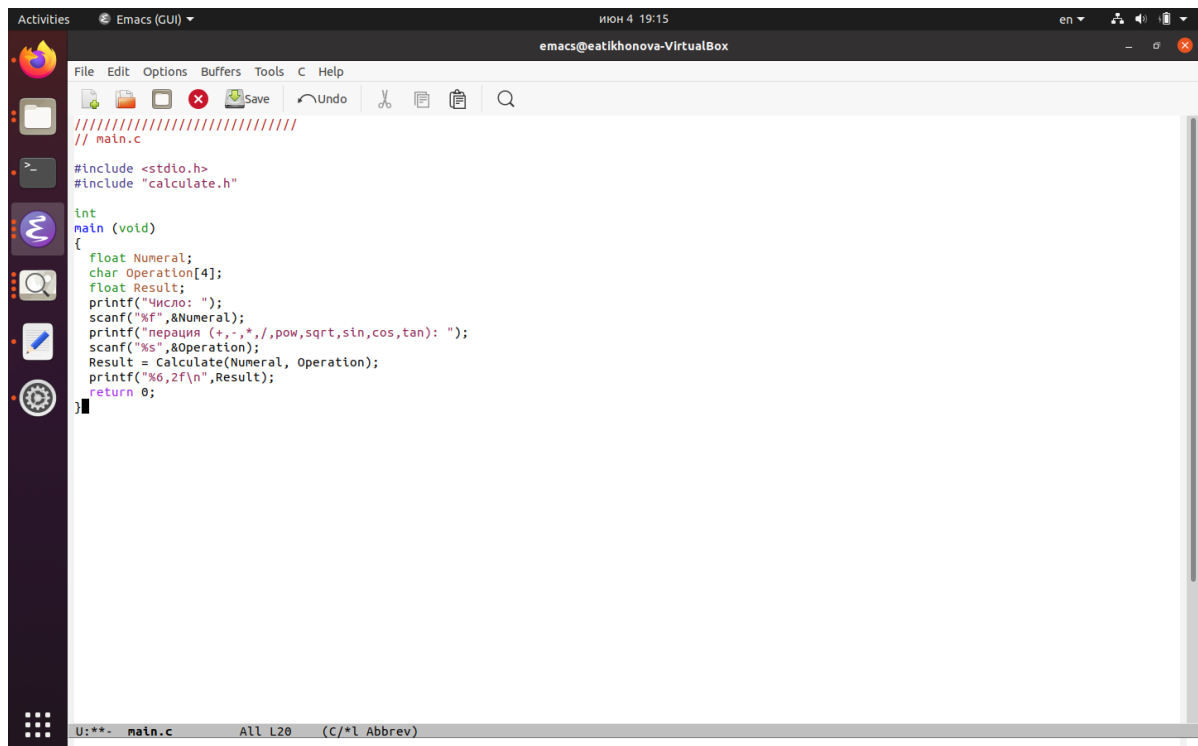


Рис. 2.6: Файл

3. Выполнила компиляцию программы посредством gcc(версия компилятора: 8.3.0-19), используя команды «gcc-calculate.c», «gcc-cmain.c» и «gcc calculate.o main.o-o calcul-lm»

```
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ gcc -c calculate.c
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ gcc -c main.c
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ gcc calculate.o main.o calcul -lm
```

Рис. 2.7: Компилируем

4. В ходе компиляции программы никаких ошибок выявлено не было.
5. Создала Makefile с необходимым содержанием.

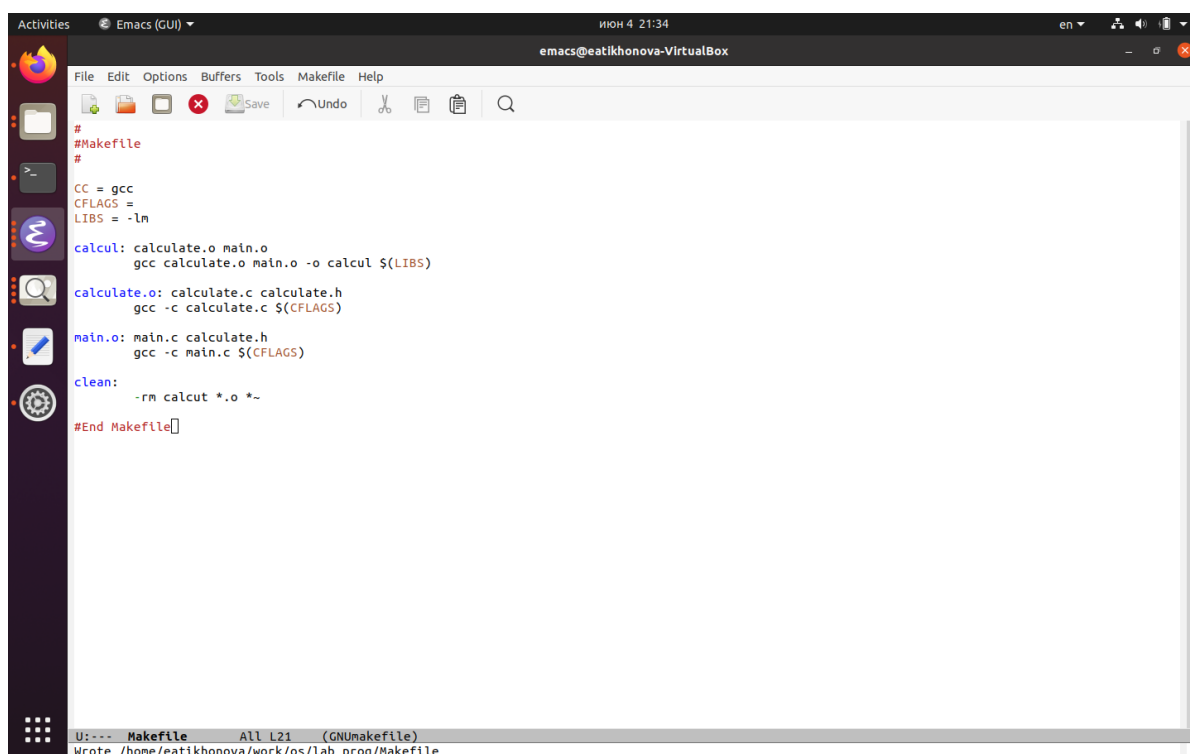


Рис. 2.8: Проверяем работу

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 исправил Makefile

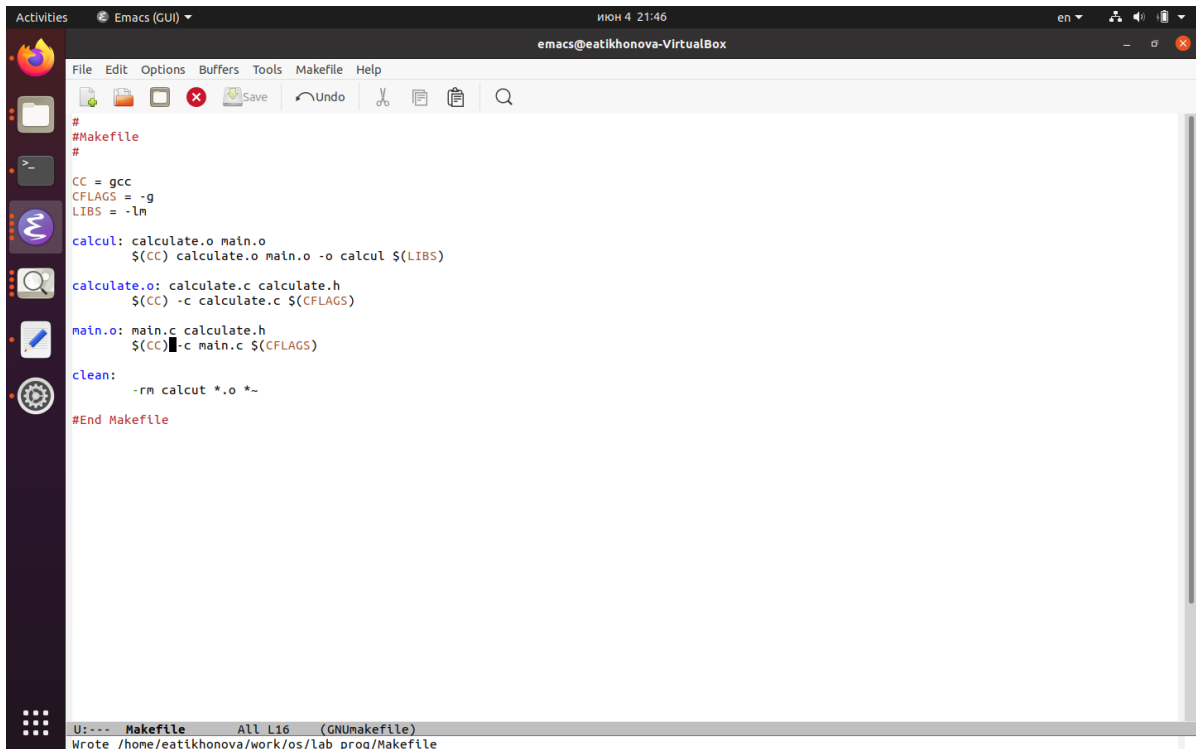


Рис. 2.9: Исправляем файл

Временную 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 «makeclear». Выполнила компиляцию файлов, используя команды «makecalculate.o», «makemain.o», «malecalcul»

```

eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ make clean
rm calcul *.o *~

```

Рис. 2.10: Выполняем компиляцию

```

eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ make calculate.o
gcc -c calculate.c -g
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ make main.o
gcc -c main.c -g
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ make calcul
gcc calculate.o main.o -o calcul -lm

```

Рис. 2.11: Выполняем компиляцию

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul»

```
e@tikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
```

Рис. 2.12: Выполняем отладку

Для запуска программы внутри отладчика ввела команду «run»

```
(gdb) run
Starting program: /home/kaleontjeva/work/os/lab_prog/calcul
Число: 6
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): *
Множитель: 5
30.00
[Inferior 1 (process 4551) exited normally]
(gdb) □
```

Рис. 2.13: Ввела команду

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list»

```

(gdb) list
1  //////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) list
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
16     scanf("%s",&Operation);
17     Result = Calculate(Numeral, Operation);
18     printf("%.2f\n",Result);
19     return 0;
20 }
(gdb)

```

Рис. 2.14: Используем команду

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12, 15»

```

(gdb) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
(gdb)

```

Рис. 2.15: Используем команду

Для просмотра определённых строк неосновного файла использовала команду «listcalculate.c: 20, 29»

```
(gdb) list calculate.c:20,29
20      {
21          printf("Вычитаемое: ");
22          scanf("%f",&SecondNumeral);
23          return(Numeral - SecondNumeral);
24      }
25      else if(strncmp(Operation, "*", 1) == 0)
26      {
27          printf("Множитель: ");
28          scanf("%f",&SecondNumeral);
29          return(Numeral * SecondNumeral);
(gdb) █
```

Рис. 2.16: Используем команду

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c: 20, 27» и «break 21»

```
(gdb) list calculate.c:20,27
20      {
21          printf("Вычитаемое: ");
22          scanf("%f",&SecondNumeral);
23          return(Numeral - SecondNumeral);
24      }
25      else if(strncmp(Operation, "*", 1) == 0)
26      {
27          printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x55555555226: file calculate.c, line 21.
(gdb) █
```

Рис. 2.17: Используем команду

Вывела информацию об имеющихся в проекте точках останова с помощью команды «infobreakpoints»


```
(gdb) info breakpoints
Num      Type           Disp Enb Address          What
1        breakpoint     keep y   0x000055555555226 in Calculate at calculate.c:21
(gdb) █
```

Рис. 2.18: Используем команду

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «-» и «backtrace»

```
(gdb) run
Starting program: /home/kaleontjeva/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffef4 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffef4 "-") at calculate.c:21
#1 0x0000555555554dd in main () at main.c:17
(gdb) █
```

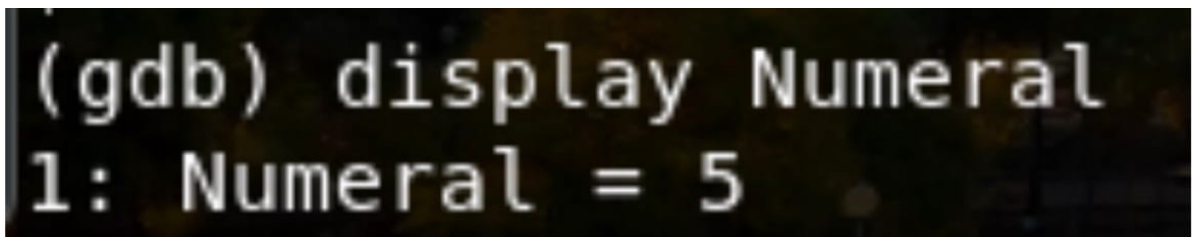
Рис. 2.19: Используем команду

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral»

```
(gdb) print Numeral
$1 = 5
```

Рис. 2.20: Используем команду

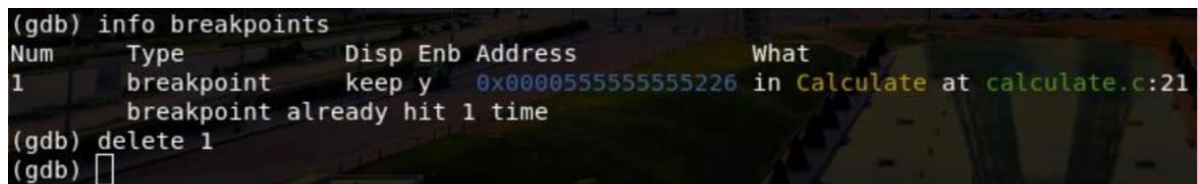
Сравнила с результатом вывода на экран после использования команды «displayNumeral». Значения совпадают



```
(gdb) display Numeral
1: Numeral = 5
```

Рис. 2.21: Сравниваем

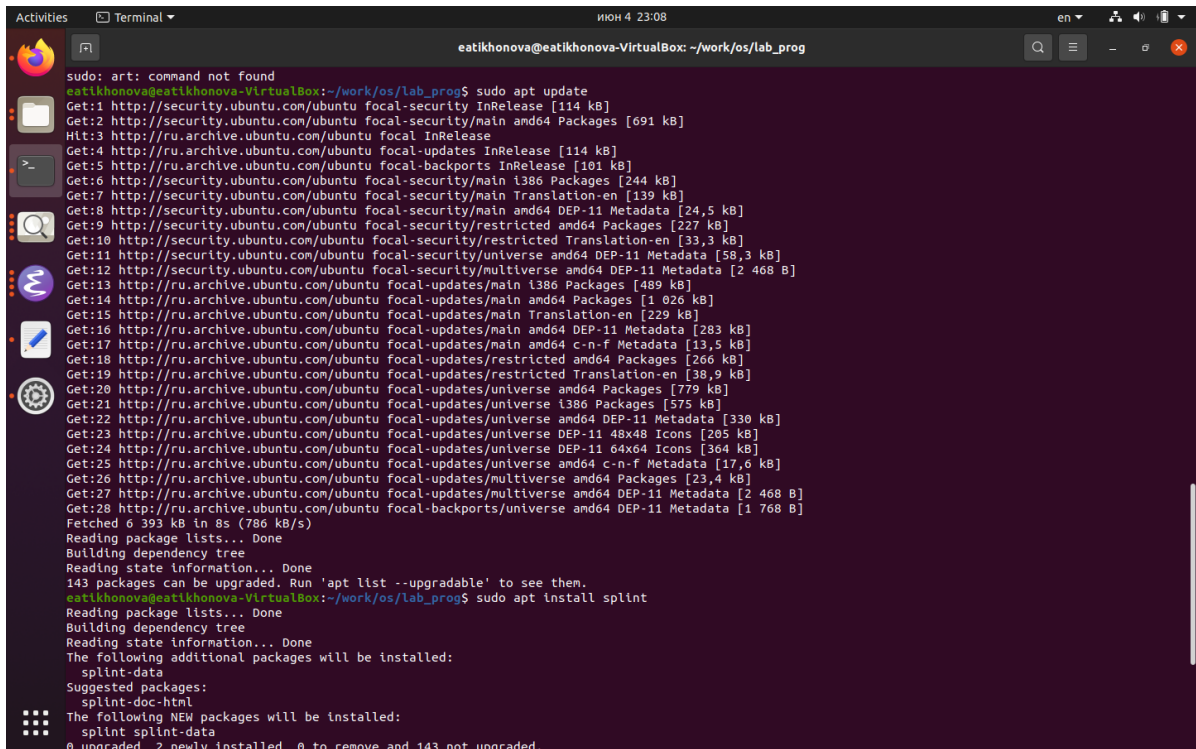
Убрала точки останова с помощью команд «infobreakpoints» и «delete1»



```
(gdb) info breakpoints
Num      Type             Disp Enb Address            What
1        breakpoint       keep y   0x000055555555226 in Calculate at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb) 
```

Рис. 2.22: Убрала точки

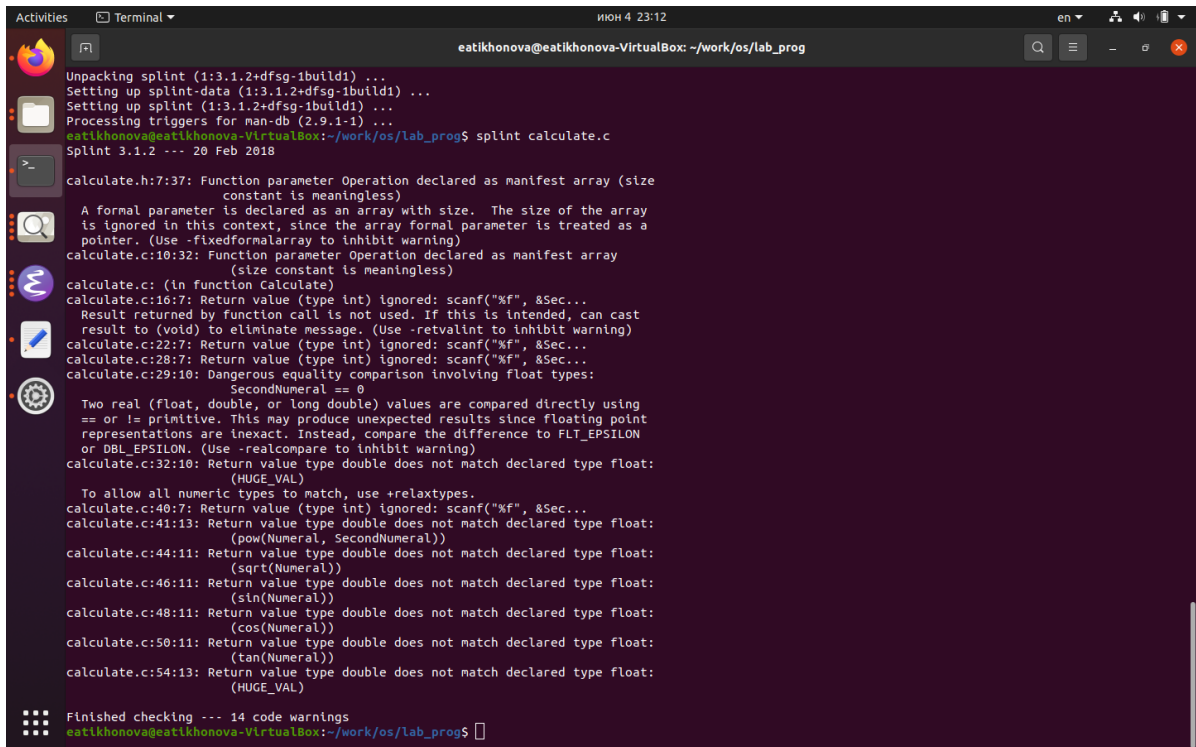
7. С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Предварительно я установила данную утилиту с помощью команд «sudoaptupdate» и «sudoaptinstallsplint»



```
Activities Terminal
eatikhonova@eatikhonova-VirtualBox: ~/work/os/lab_prog$ sudo apt update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [691 kB]
Hit:3 http://ru.archive.ubuntu.com/ubuntu focal InRelease
Get:4 http://ru.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://ru.archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main i386 Packages [244 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [139 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [24,5 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [227 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-en [33,3 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [58,3 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [2 468 B]
Get:13 http://ru.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [489 kB]
Get:14 http://ru.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1 026 kB]
Get:15 http://ru.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [229 kB]
Get:16 http://ru.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [283 kB]
Get:17 http://ru.archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [13,5 kB]
Get:18 http://ru.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [266 kB]
Get:19 http://ru.archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [38,9 kB]
Get:20 http://ru.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [779 kB]
Get:21 http://ru.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages [575 kB]
Get:22 http://ru.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [330 kB]
Get:23 http://ru.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 48x48 Icons [205 kB]
Get:24 http://ru.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 64x64 Icons [364 kB]
Get:25 http://ru.archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [17,6 kB]
Get:26 http://ru.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [23,4 kB]
Get:27 http://ru.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 DEP-11 Metadata [2 468 B]
Get:28 http://ru.archive.ubuntu.com/ubuntu focal-backports/universe amd64 DEP-11 Metadata [1 768 B]
Fetched 6 393 kB in 8s (786 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
143 packages can be upgraded. Run 'apt list --upgradable' to see them.
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ sudo apt install splint
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  splint-data
Suggested packages:
  splint-doc.html
The following NEW packages will be installed:
  splint splint-data
0 upgraded, 2 newly installed, 0 to remove and 143 not upgraded.
```

Рис. 2.23: Убрала точки

Далее воспользовалась командами «splintcalculate.c» и «splintmain.c». С 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 сохраняются. 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 потере данных.



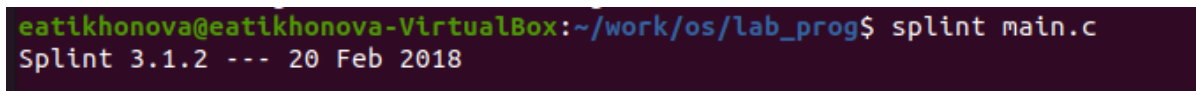
```
Activities Terminal
июн 4 23:12 en
eatikhonova@eatikhonova-VirtualBox: ~/work/os/lab_prog

Unpacking splint (1:3.1.2+dfsg-1build1) ...
Setting up splint-data (1:3.1.2+dfsg-1build1) ...
Setting up splint (1:3.1.2+dfsg-1build1) ...
Processing triggers for man-db (2.9.1-1) ...
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:7:37: Function parameter Operation declared as manifest array (size
constant is meaningless)
A formal parameter is declared as an array with size. The size of the array
is ignored in this context, since the array formal parameter is treated as a
pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:10:32: Function parameter Operation declared as manifest array
(size constant is meaningless)
calculate.c: (In function calculate)
calculate.c:16:7: Return value (type int) ignored: scanf("%f", &Sec...
Result returned by function call is not used. If this is intended, can cast
result to (void) to eliminate message. (Use -retvalint to inhibit warning)
calculate.c:22:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:29:10: Dangerous equality comparison involving float types:
SecondNumeral == 0
Two real (float, double, or long double) values are compared directly using
== or != primitive. This may produce unexpected results since floating point
representations are inexact. Instead, compare the difference to FLT_EPSILON
or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:32:10: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:40:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:41:13: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:44:11: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:46:11: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:48:11: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:50:11: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:54:13: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 14 code warnings
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$
```

Рис. 2.24: Используем команду



```
eatikhonova@eatikhonova-VirtualBox:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018
```

Рис. 2.25: Используем команду

3 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: 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.

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 C, файлы с расширением .cpp или .C – как файлы на языке C++, а файлы с расширением .o считаются объектными. Например, в команде «gcc -o main.c»: gcc по расширению (суффиксу) .c распознаёт 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 -oi в качестве параметра задать имя создаваемого файла: «gcc-ohellomain.c».4)Основное назначение компилятора языка Си в UNIXзаключается 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 Makefileимеет 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\$(CC) -o abcd \$(CFLAGS) abcd.cclean:-rm abcd .o ~# EndMakefileforabcd.cВ этом примере в начале файла заданы три переменные: CСи CFLAGS. Затем указаны цели, их за-

висимости 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-hи mangdb. 9) Схема 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: **cscore** – 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 Санализатор 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. 4

4 Выводы

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 C калькулятора с простейшими функциями.