

# **Отчёта по лабораторной работе №14**

**Операционный Систем**

Коне Сирики НФИБД-01-20

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выводы</b>	<b>19</b>

## List of Tables

# List of Figures

2.1	рисунок 1 . . . . .	6
2.2	рисунок 2 . . . . .	6
2.3	рисунок 3 . . . . .	7
2.4	рисунок 4 . . . . .	7
2.5	рисунок 5 . . . . .	8
2.6	рисунок 6 . . . . .	8
2.7	рисунок 7 . . . . .	9
2.8	рисунок 8 . . . . .	9
2.9	рисунок 9 . . . . .	10
2.10	рисунок 10 . . . . .	10
2.11	рисунок 11 . . . . .	10
2.12	рисунок 12 . . . . .	11
2.13	рисунок 13 . . . . .	11
2.14	рисунок 14 . . . . .	12
2.15	рисунок 15 . . . . .	12
2.16	рисунок 16 . . . . .	12
2.17	рисунок 17 . . . . .	13
2.18	рисунок 18 . . . . .	13
2.19	рисунок 19 . . . . .	13
2.20	рисунок 20 . . . . .	14
2.21	рисунок 21 . . . . .	14
2.22	рисунок 22 . . . . .	15

# 1 Цель работы

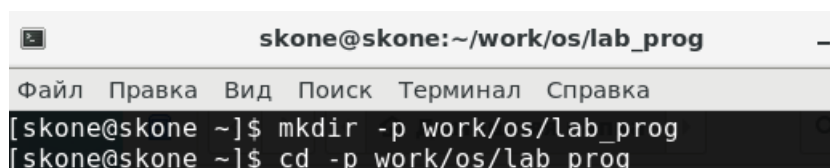
приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями # Задание

-Этапы разработки приложений -Компиляция исходного текста и построение исполняемого файла -Тестирование и отладка Анализ исходного текста программы

## 2 Выполнение лабораторной работы

Ход работы: 1. В домашнем каталоге создала подкаталог ~/work/os/lab\_prog.

(рис. 2.1)

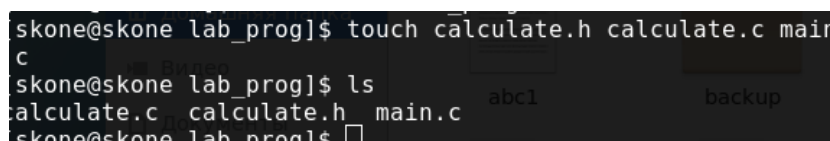


```
skone@skone:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
[skone@skone ~]$ mkdir -p work/os/lab_prog
[skone@skone ~]$ cd -p work/os/lab prog
```

Figure 2.1: рисунок 1

2. Создал в нём файлы: calculate.h, calculate.c, main.c. 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 этого программа выводит результат и останавливается. Код для программы приведён в задании к лабораторной работе.

(рис. 2.2)

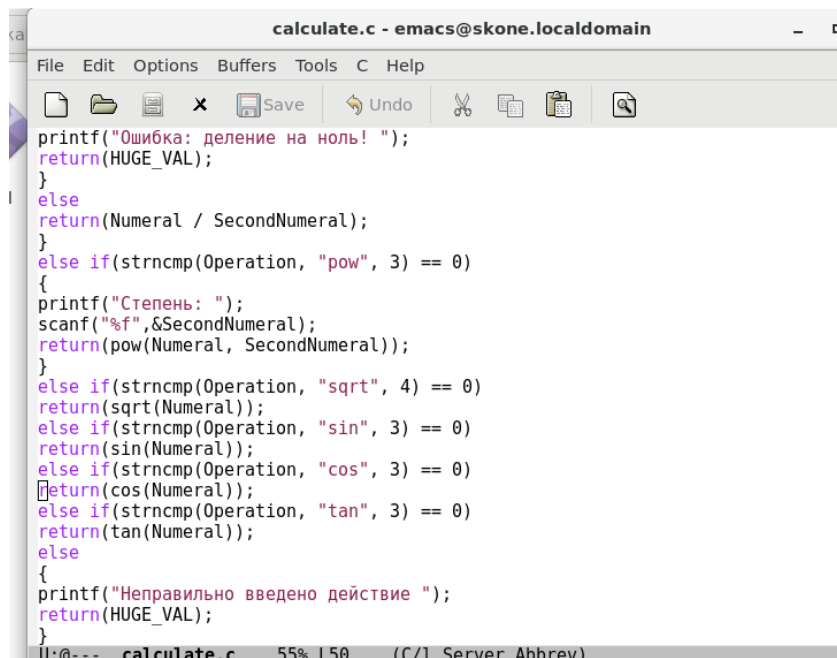


```
skone@skone lab_prog]$ touch calculate.h calculate.c main
c
skone@skone lab_prog]$ ls
calculate.c  calculate.h  main.c
skone@skone lab_prog]$
```

Figure 2.2: рисунок 2

Реализация функций калькулятора в файле calculate.h:

(рис. 2.3)

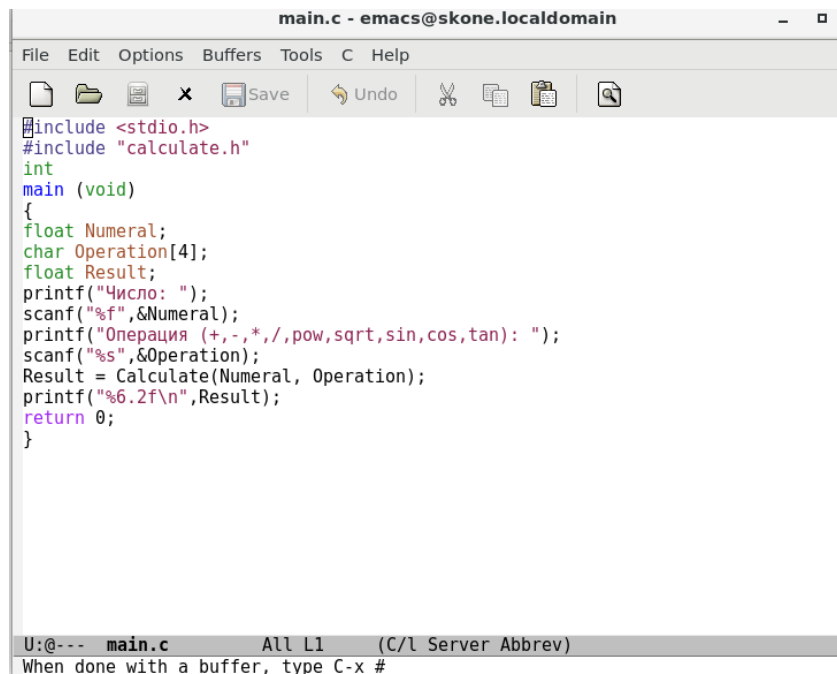


```
calculate.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons: Save, Undo, Cut, Copy, Paste, Find]

printf("Ошибка: деление на ноль! ");
return(HUGE_VAL);
}
else
return(Numeral / SecondNumeral);
}
else if(strncmp(Operation, "pow", 3) == 0)
{
printf("Степень: ");
scanf("%f",&SecondNumeral);
return(pow(Numeral, SecondNumeral));
}
else if(strncmp(Operation, "sqrt", 4) == 0)
return(sqrt(Numeral));
else if(strncmp(Operation, "sin", 3) == 0)
return(sin(Numeral));
else if(strncmp(Operation, "cos", 3) == 0)
return(cos(Numeral));
else if(strncmp(Operation, "tan", 3) == 0)
return(tan(Numeral));
else
{
printf("Неправильно введено действие ");
return(HUGE_VAL);
}
}
U:~-- calculate.c 55% L50 (C/l Server Abbrev)
```

Figure 2.3: рисунок 3

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:  
(рис. 2.4)



```
main.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons: Save, Undo, Cut, Copy, Paste, Find]

#include <stdio.h>
#include "calculate.h"
int
main (void)
{
float Numeral;
char Operation[4];
float Result;
printf("Число: ");
scanf("%f",&Numeral);
printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
scanf("%s",&Operation);
Result = Calculate(Numeral, Operation);
printf("%.2f\n",Result);
return 0;
}
U:~-- main.c All L1 (C/l Server Abbrev)
When done with a buffer, type C-x #
```

Figure 2.4: рисунок 4

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:  
(рис. 2.5)

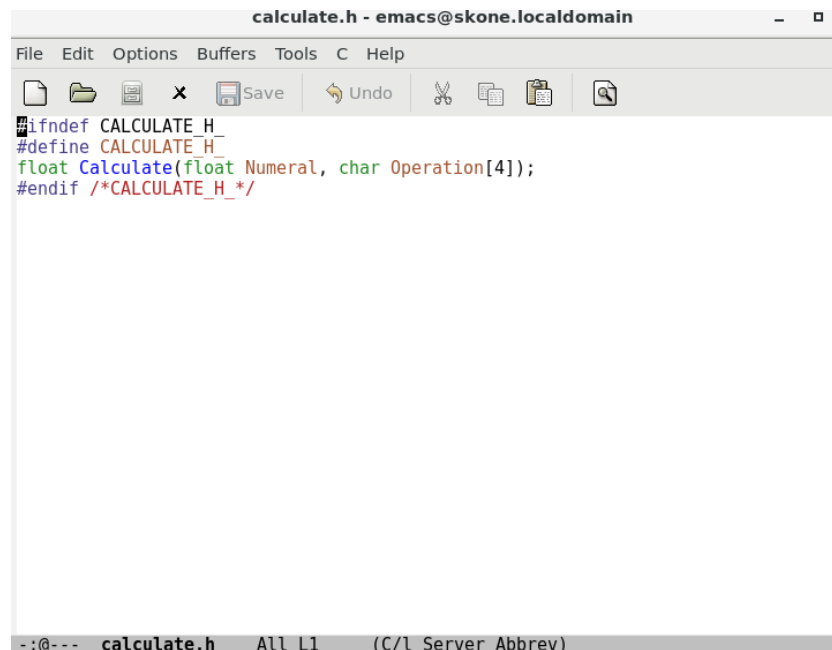


Figure 2.5: рисунок 5

3. Выполнил компиляцию программы посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm

(рис. 2.6)

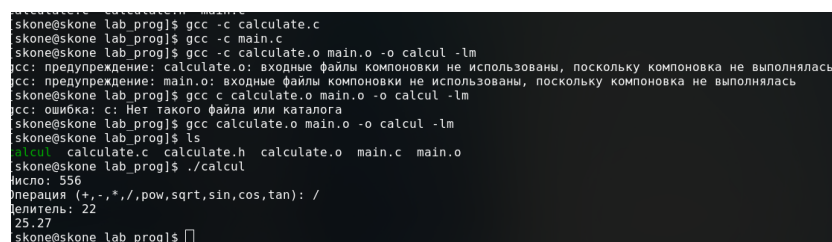


Figure 2.6: рисунок 6

4. Исправил синтаксические ошибки: Scanf("%s",&Operation) -> scanf("%s",Operation)
5. Создал Makefile со следующим содержанием: # # Makefile #

CC = gcc #задание переменных CFLAGS = LIBS = -lm



calcul: calculate.o main.o gcc calculate.o main.o -o calcul \$(LIBS) #получение исполняемого файла

calculate.o: calculate.c calculate.h gcc -c calculate.c \$(CFLAGS) #компилирование файла calculate.c

main.o: main.c calculate.h gcc -c main.c \$(CFLAGS) #компилирование файла main.c

clean: -rm calcul .o ~ #очистка каталога от файлов, появившихся в процессе компиляции # End Makefile

(рис. 2.7)

```
[skone@skone lab_prog]$ touch Makefile.sh
[skone@skone lab_prog]$ make clean
cat Makefile.sh > Makefile
```

Figure 2.7: рисунок 7

(рис. 2.8)

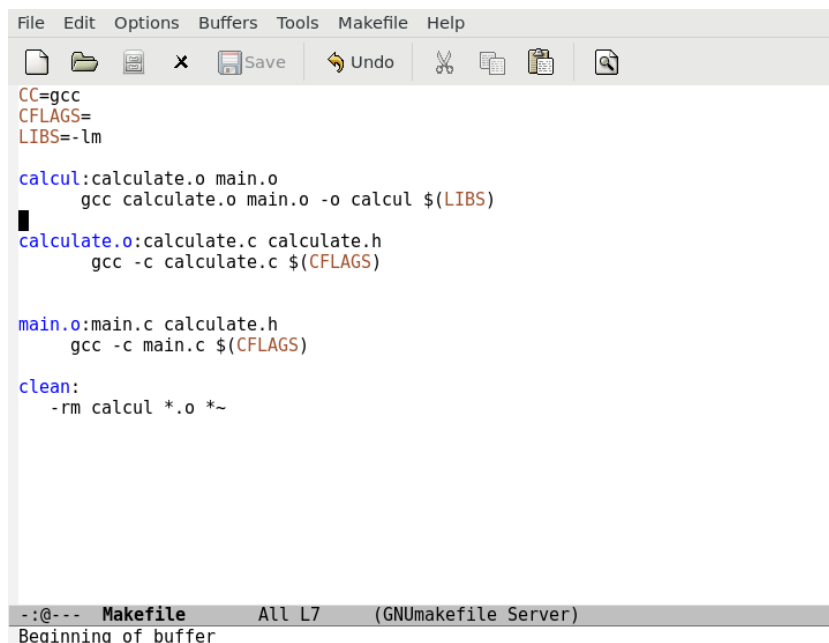
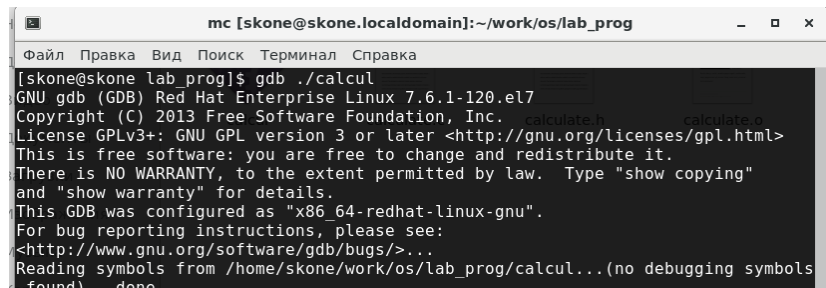


Figure 2.8: рисунок 8

6. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправила Makefile: CFLAGS=-g):

(рис. 2.9)

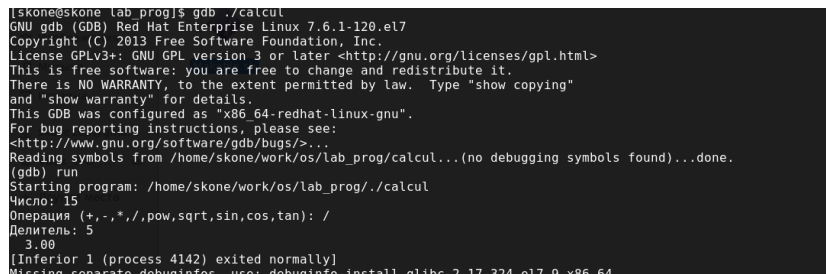


```
mc [skone@skone.localdomain]:~/work/os/lab_prog
Файл Правка Вид Поиск Терминал Справка
[skone@skone lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/skone/work/os/lab_prog/calcul...(no debugging symbols
found)...done
```

Figure 2.9: рисунок 9

– Запустил отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul`

(рис. 2.10)

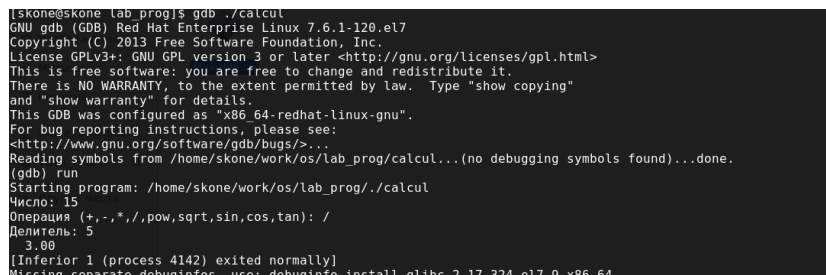


```
[skone@skone lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/skone/work/os/lab_prog/calcul...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/skone/work/os/lab_prog/./calcul
Число: 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /
Делитель: 5
3.00
[Inferior 1 (process 4142) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7.x86_64
```

Figure 2.10: рисунок 10

– Для запуска программы внутри отладчика ввёл команду `run: run`

(рис. 2.11)



```
[skone@skone lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/skone/work/os/lab_prog/calcul...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/skone/work/os/lab_prog/./calcul
Число: 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /
Делитель: 5
3.00
[Inferior 1 (process 4142) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7.x86_64
```

Figure 2.11: рисунок 11

– Для постраничного (по 9 строк) просмотра исходного код использовал команду `list: list`

(рис. 2.12)

```

(gdb) list
1      //main.c
2      #include<stdio.h>
3      #include"calculate.h"
4
5      int main(void)
6      {
7          float Numeral,Result;
8          char Operation[4];
9          printf("Число: ");
10         scanf("%f",&Numeral);
(gdb) list
11         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan,): ");
12         scanf("%s",Operation);
13         Result=Calculate(Numeral,Operation);
14         printf("%6.2f\n",Result);
15         return 0;
16     }

```

Figure 2.12: рисунок 12

– Для просмотра строк с 12 по 15 основного файла использовал list с параметрами: list 12,15

(рис. 2.13)

```

(gdb) list 12,15
12         scanf("%s",Operation);
13         Result=Calculate(Numeral,Operation);
14         printf("%6.2f\n",Result);
15         return 0;

```

Figure 2.13: рисунок 13

– Для просмотра определённых строк не основного файла использовал list с параметрами: list calculate.c:20,29

(рис. 2.14)

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

Figure 2.14: рисунок 14

- Установил точку останова в файле calculate.c на строке номер 21: `break 21` (рис. 2.15)

```
(gdb) break 21
Breakpoint 1 at 0x804870d: file calculate.c, line 21.
```

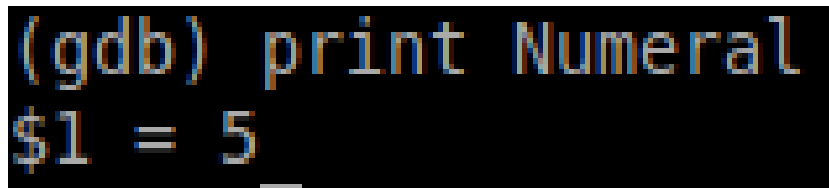
Figure 2.15: рисунок 15

- Вывёл информацию об имеющихся в проекте точка останова: `info breakpoints` (рис. 2.16)

```
Breakpoint 1 at 0x804870d: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x0804870d in Calculate at calculate.c:21
```

Figure 2.16: рисунок 16

- Запустил программу внутри отладчика и убедилась, что программа останавливается в момент прохождения точки останова: `run 5 * backtrace`
- Отладчик выдал следующую информацию: `#0 Calculate (Numeral=5, Operation=0xbffff2d8 "**") at calculate.c:21` (рис. 2.17)

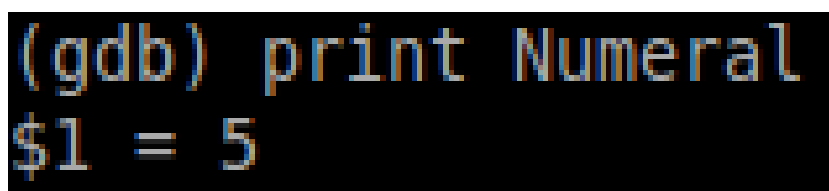


```
(gdb) print Numeral
$1 = 5
```

Figure 2.17: рисунок 17

#1 0x08048956 in main () at main.c:13 а команда backtrace показала весь стек вызываемых функций от начала программы до текущего места. – Посмотрела, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран было выведено число 5.

(рис. 2.18)

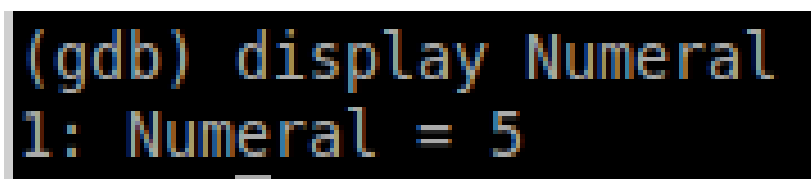


```
(gdb) print Numeral
$1 = 5
```

Figure 2.18: рисунок 18

– Сравнила с результатом вывода на экран после использования команды: display Numeral

(рис. 2.19)



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

Figure 2.19: рисунок 19

– Убрал точки останова: info breakpoints delete 1

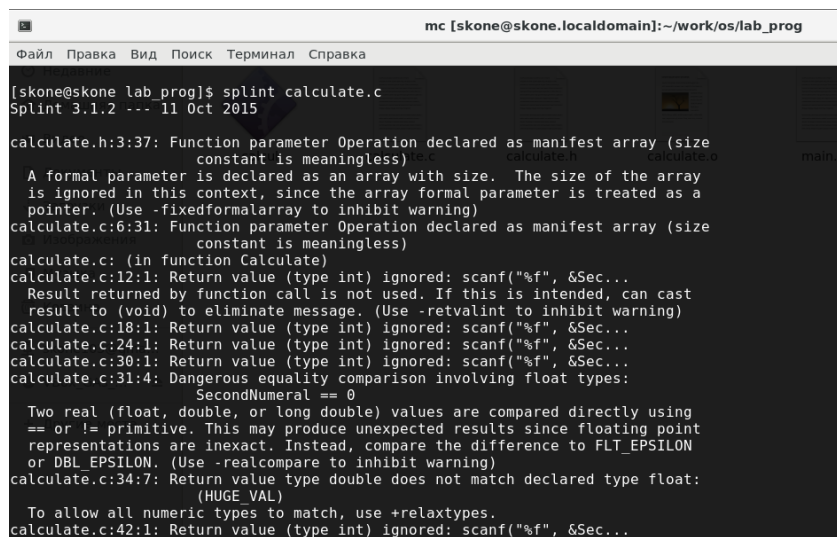
(рис. 2.20)

```
(gdb) info breakpoints
Num    Type    Disp Enb Address      What
1      breakpoint keep y   0x0804870d in Calculate at calculate.c:21
      breakpoint already hit 1 time
(gdb) delete 1
(gdb) continue
Continuing.
Множитель: 6
30.00
[Inferior 1 (process 8492) exited normally]
(gdb)
```

Figure 2.20: рисунок 20

7. С помощью утилиты splint попробовала проанализировать коды файлов calculate.c и main.c.

(рис. 2.21)



```
mc [skone@skone.localdomain]: ~/work/os/lab_prog
Файл Правка Вид Поиск Терминал Справка
[skone@skone lab_prog]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:3:37: 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:6:31: Function parameter Operation declared as manifest array (size
constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:1: 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:18:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:4: 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:34:7: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:42:1: Return value (type int) ignored: scanf("%f", &Sec...
```

Figure 2.21: рисунок 21

Например, split отмечает отсутствие необходимости указывать размер массива, который является аргументом функции. В некоторых математических функциях возвращается double, а не float.

(рис. 2.22)

```

Finished checking --- 13 code warnings
[skone@skone lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:3:37: 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)
main.c: (in function main)
main.c:10:1: Return value (type int) ignored: scanf("%f", &Num...
    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)
main.c:12:12: Format argument 1 to scanf (%s) expects char * gets char [4] *:
    &Operation
    Type of parameter is not consistent with corresponding code in format string.
    (Use -formattype to inhibit warning)
    main.c:12:9: Corresponding format code
main.c:12:1: Return value (type int) ignored: scanf("%s", &ope...

Finished checking --- 4 code warnings
[skone@skone lab_prog]$

```

Figure 2.22: рисунок 22

### Контрольные вопросы:

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?  
Прочитать man-файл.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. 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.
3. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Суффикс – это расширение файла. Позволяет определить тип файла, т.е., что с ним можно делать. Например, файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C — как файлы на языке C++, а файлы с расширением .o считаются объектными.

4. Каково основное назначение компилятора языка C в UNIX? Компиляция файлов с исходным кодом в объектные модули и получение исполняемых файлов.
5. Для чего предназначена утилита make? Make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. Общий синтаксис Makefile имеет вид:  
target1 [target2...]:[:] [dependment1...] [(tab)commands] [#commentary]  
[(tab)commands] [#commentary] 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. После # пишутся комментарии, они не обрабатываются.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Программы отладки позволяют найти и устранить ошибки в программе. Чтобы использовать их, необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле.
8. Назовите и дайте основную характеристику основным командам отладчика gdb. Основные команды: 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

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы. Запуск отладчика Запуск программы в отладчике Просмотр исходного кода основного и не основного файлов Установка точек останова и просмотр информации о них Запуск программы с установленными точками останова. Просмотр стека вызываемых функций при достижении точки останова Просмотр значения переменной в момент достижения точки останова Удаление точек останова
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Компилятор выводит найденные им ошибки с комментариями. Это такие ошибки, которые могут повлиять на работу программы.
11. Назовите основные средства, повышающие понимание исходного кода программы. Само по себе грамотное написание кода (с переносами и отступами в нужных местах) уже повышает его понимание. Также этой цели служат комментарии.

12. Каковы основные задачи, решаемые программой splint? Эта утилита 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

## 3 Выводы

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