

# Лабораторная работа № 13

---

Беличева Д.М

<sup>1</sup>RUDN University, Moscow, Russian Federation

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.

1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. 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.
3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`.

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):
- Запустите отладчик GDB, загрузив в него программу для отладки:  
`gdb ./calcul`
  - Для запуска программы внутри отладчика введите команду `run: run`
  - Для постраничного (по 9 строк) просмотра исходного кода используйте команду `list: 1 list`
  - Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами: `list 12,15`
  - Для просмотра определённых строк не основного файла используйте `list` с параметрами: `list calculate.c:20,29`
  - Установите точку останова в файле `calculate.c` на строке номер 21:  
`list calculate.c:20,27 break 21`

- Выведите информацию об имеющихся в проекте точка останова: `info breakpoints` – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова. 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.
  - Сравните с результатом вывода на экран после использования команды: `display Numeral`
  - Уберите точки останова
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

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.

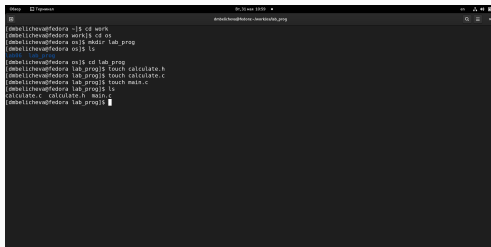
Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.



Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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.

# Выполнение лабораторной работы

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.
2. Создайте в нём файлы: calculate.h, calculate.c, main.c. (рис. 1)

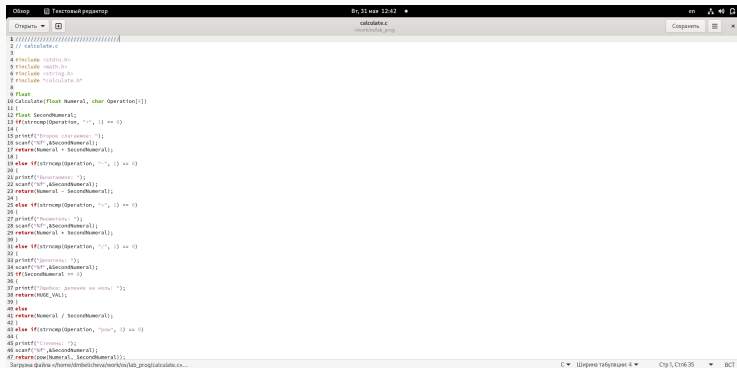


```
osbelichayv@fedora ~$ cd work
osbelichayv@fedora work$ cd os
osbelichayv@fedora os$ mkdir lab_prog
osbelichayv@fedora os$ ls
lab_prog
osbelichayv@fedora os$ cd lab_prog
osbelichayv@fedora lab_prog$ touch calculate.h
osbelichayv@fedora lab_prog$ touch calculate.c
osbelichayv@fedora lab_prog$ touch main.c
osbelichayv@fedora lab_prog$ ls
calculate.c  calculate.h  main.c
osbelichayv@fedora lab_prog$
```

Figure 1: Терминал

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. (рис. 2, 3, 4)

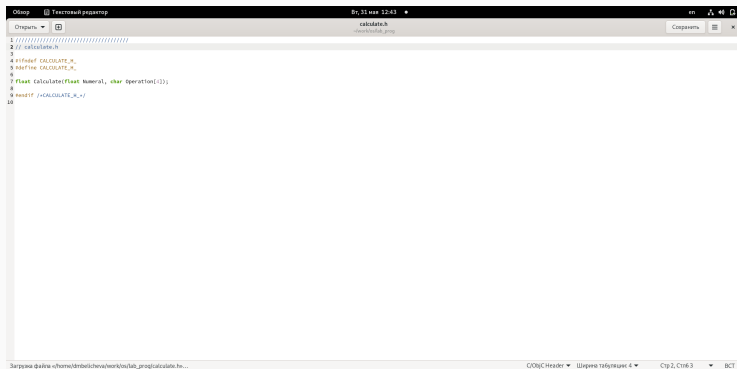
# Выполнение лабораторной работы



```
1 // calculate.c
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include <stdlib.h>
8
9 float
10 calculate(float Numeral, char Operation[])
11 {
12     float SecondNumeral;
13     if(strlen(Operation) == 1)
14     {
15         printf("Введите операцию: ");
16         scanf("%c", &SecondNumeral);
17         return Numeral + SecondNumeral;
18     }
19     else if(strlen(Operation) == 2)
20     {
21         printf("Введите операцию: ");
22         scanf("%c", &SecondNumeral);
23         return Numeral - SecondNumeral;
24     }
25     else if(strlen(Operation) == 3)
26     {
27         printf("Введите операцию: ");
28         scanf("%c", &SecondNumeral);
29         return Numeral * SecondNumeral;
30     }
31     else if(strlen(Operation) == 4)
32     {
33         printf("Введите операцию: ");
34         scanf("%c", &SecondNumeral);
35         if(SecondNumeral == 0)
36         {
37             printf("Ошибка: деление на ноль");
38             return MODE_VAL;
39         }
40         else
41             return Numeral / SecondNumeral;
42     }
43     else if(strlen(Operation) == 5)
44     {
45         printf("Введите операцию: ");
46         scanf("%c", &SecondNumeral);
47         return pow(Numeral, SecondNumeral);
48     }
49     return 0;
50 }
```

Figure 2: Текст программы

# Выполнение лабораторной работы



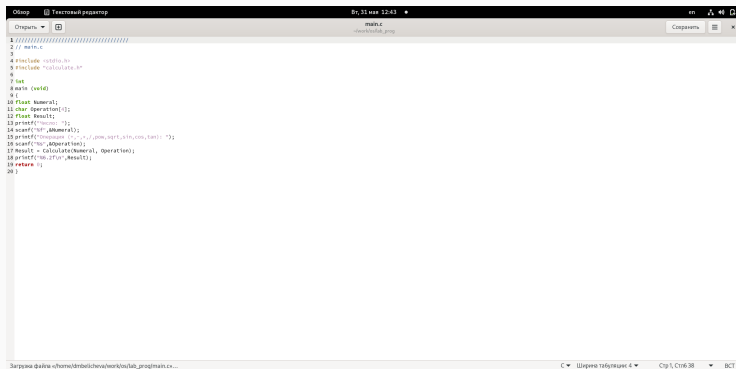
The screenshot shows a web browser window with a code editor. The browser's address bar shows the file path: `calculate.h` and `~/work/lab_prog`. The code editor contains the following C code:

```
1 // calculate.h
2
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6
7 float calculate(float numeral, char operation[1]);
8
9 #endif //CALCULATE_H_
10
```

The status bar at the bottom of the editor shows the file path: `Зернова g:\file\c\home\drnbolsheva\work\on\lab_prog\calculate.h...`, the file type: `C/C++ Header`, the window title: `Шерина таблица 4`, the page number: `Стр.2, Стр.63`, and the language: `ВСТ`.

Figure 3: Текст программы

# Выполнение лабораторной работы



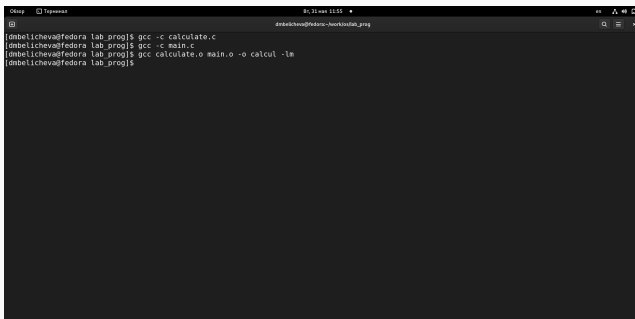
The image shows a screenshot of a text editor window titled "Текстовый редактор". The editor displays the source code for a C program named "main.c". The code is as follows:

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

The editor interface includes a top bar with the file name "main.c" and the path "~work/lab\_prog". The status bar at the bottom shows the file path "Surpina @linux ~/home/andrulicheva/work/os/lab\_prog/main.c...", the number of lines "Стр 1, Стр 38", and the encoding "ВСТ".

Figure 4: Текст программы

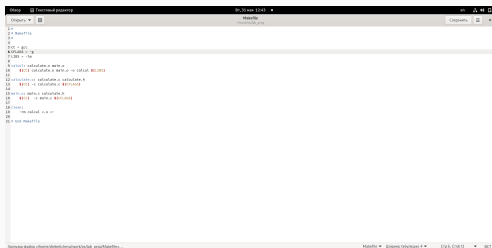
## 3. Выполните компиляцию программы посредством gcc. (рис. 5)



```
dnbelicheva@fedora lab_prog$ gcc -c calculate.c
dnbelicheva@fedora lab_prog$ gcc -c main.c
dnbelicheva@fedora lab_prog$ gcc calculate.o main.o -o calcul -lm
dnbelicheva@fedora lab_prog$
```

Figure 5: Компиляция программ

4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile. (рис. 6)



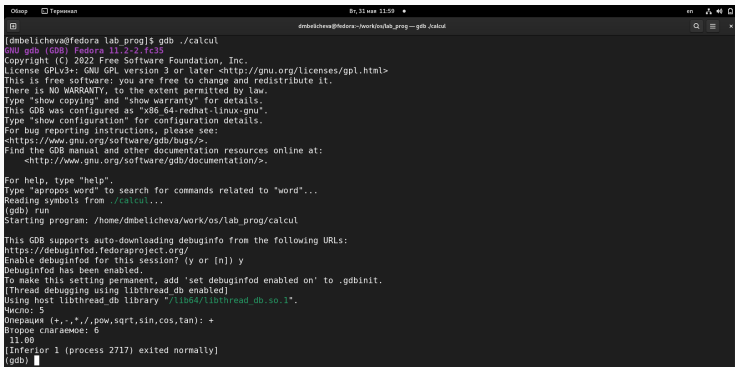
```
1 #  
2 #  
3 #  
4 #  
5 #  
6 #  
7 #  
8 #  
9 #  
10 #  
11 #  
12 #  
13 #  
14 #  
15 #  
16 #  
17 #  
18 #  
19 #  
20 #  
21 #  
22 #  
23 #  
24 #  
25 #  
26 #  
27 #  
28 #  
29 #  
30 #  
31 #  
32 #  
33 #  
34 #  
35 #  
36 #  
37 #  
38 #  
39 #  
40 #  
41 #  
42 #  
43 #  
44 #  
45 #  
46 #  
47 #  
48 #  
49 #  
50 #  
51 #  
52 #  
53 #  
54 #  
55 #  
56 #  
57 #  
58 #  
59 #  
60 #  
61 #  
62 #  
63 #  
64 #  
65 #  
66 #  
67 #  
68 #  
69 #  
70 #  
71 #  
72 #  
73 #  
74 #  
75 #  
76 #  
77 #  
78 #  
79 #  
80 #  
81 #  
82 #  
83 #  
84 #  
85 #  
86 #  
87 #  
88 #  
89 #  
90 #  
91 #  
92 #  
93 #  
94 #  
95 #  
96 #  
97 #  
98 #  
99 #  
100 #
```

Figure 6: Текст Makefile



6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):
  - Запустите отладчик GDB, загрузив в него программу для отладки:  
`gdb ./calcul`
  - Для запуска программы внутри отладчика введите команду `run`:  
`run` (рис. 7)

# Выполнение лабораторной работы



```
Olseap  Terepanen
Br, 31 мая 11:59
dmbelicheva@fedora: /work/os/lab_prog -- gdb ./calcul

[dmbelicheva@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-2.fc35
Copyright (C) 2022 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

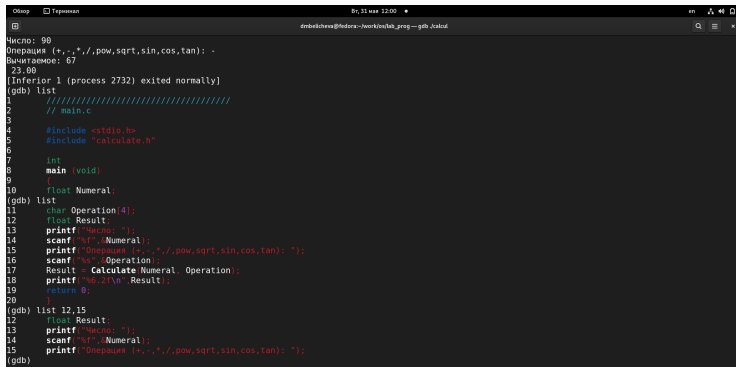
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /home/dmbelicheva/work/os/lab_prog/calcul

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
(Thread debugging using libthread_db enabled)
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 6
11.00
[Inferior 1 (process 2717) exited normally]
(gdb)
```

Figure 7: GDB

- Для постраничного (по 9 строк) просмотра исходного код используйте команду `list`:
- Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами: `list 12,15` (рис. 8)

# Выполнение лабораторной работы

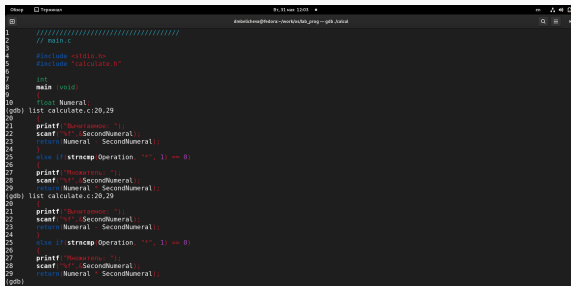


```
Число: 99
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Выводимое: 67
23.00
[Inferior 1 (process 2732) exited normally]
(gdb) list
1  //////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) 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("%6.2f\n",Result);
19     return 0;
20 }
(gdb) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb)
```

Figure 8: GDB

# Выполнение лабораторной работы

- Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29 (рис. 9)

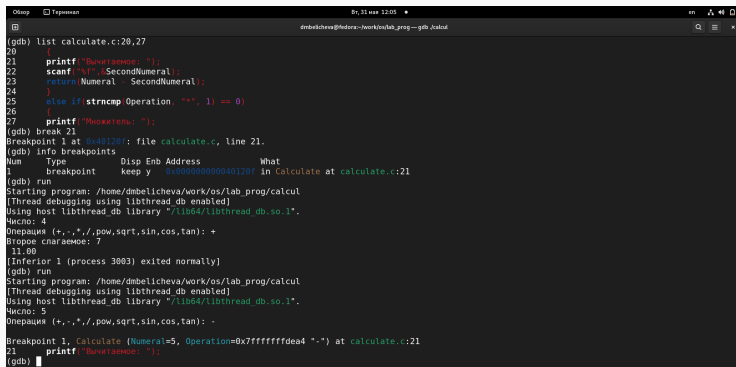


```
1 //////////////////////////////////////////////////
2 // main.c
3
4 #include <stdio.h>
5 #include "calculate.h"
6
7 int
8 main (void)
9 {
10     float Numeral;
11     (gdb) list calculate.c:20,29
12
13     printf("Buvarenko: ");
14     scanf("%f",&SecondNumeral);
15     return Numeral * SecondNumeral;
16 }
17
18 else if (strcmp Operation, "+") == 0)
19 {
20     printf("Buvarenko: ");
21     scanf("%f",&SecondNumeral);
22     return Numeral + SecondNumeral;
23 }
24
25 (gdb) list calculate.c:20,29
26
27     printf("Buvarenko: ");
28     scanf("%f",&SecondNumeral);
29     return Numeral * SecondNumeral;
30 }
31
32 else if (strcmp Operation, "+") == 0)
33 {
34     printf("Buvarenko: ");
35     scanf("%f",&SecondNumeral);
36     return Numeral + SecondNumeral;
37 }
38
39 (gdb)
```

Figure 9: GDB

- Установите точку останова в файле `calculate.c` на строке номер 21:  
`list calculate.c:20,27 break 21`
- Выведите информацию об имеющихся в проекте точке останова:  
`info breakpoints` – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова.
- 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. (рис. 10)

# Выполнение лабораторной работы



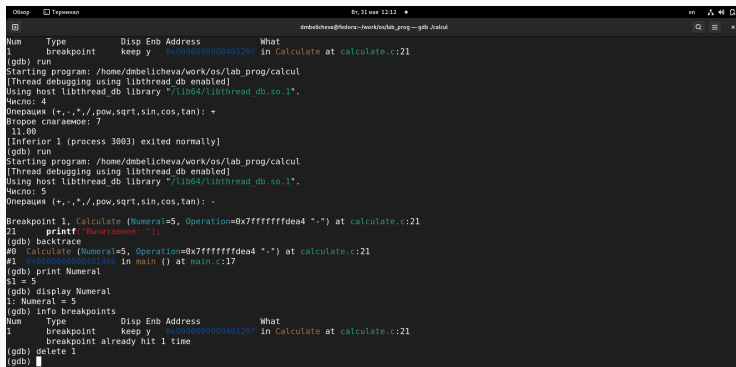
```
(gdb) list calculate.c:20,27
20      {
21          printf("Вычитаемое: ");
22          scanf("%f", &SecondNumeral);
23          return (Numeral - SecondNumeral);
24      }
25      else if (strcmp(Operation, "**") == 0)
26      {
27          printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x000000000040120f in Calculate at calculate.c:21
(gdb) run
Starting program: /home/dmbelicheva/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 4
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 7
11.00
[Inferior 1 (process 3003) exited normally]
(gdb) run
Starting program: /home/dmbelicheva/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdea4 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb)
```

Figure 10: GDB

- Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: `print Numeral` На экран должно быть выведено число 5.
- Сравните с результатом вывода на экран после использования команды: `display Numeral`
- Уберите точки останова (рис. 11)



# Выполнение лабораторной работы



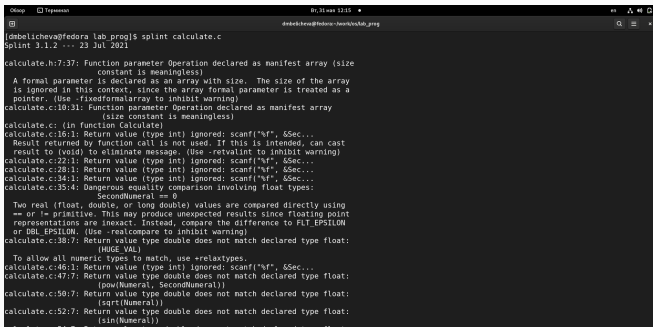
```
Olsson  Терминал
Br, 31 мая 12:12
dmbelicheva@fedora:~/work/os/lab_prog--gdb./calcul

Num  Type      Disp Enb Address      What
1    breakpoint keep y  0x00000000040120f  in Calculate at calculate.c:21
(gdb) run
Starting program: /home/dmbelicheva/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 4
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 7
11:00
[Inferior 1 (process 3003) exited normally]
(gdb) run
Starting program: /home/dmbelicheva/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdea4 "-") at calculate.c:21
21      printf("Выводимое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdea4 "-") at calculate.c:21
#1 0x0000000004014eb in main () at main.c:17
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num  Type      Disp Enb Address      What
1    breakpoint keep y  0x00000000040120f  in Calculate at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Figure 11: GDB

## 7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c. (рис. 12, 13)

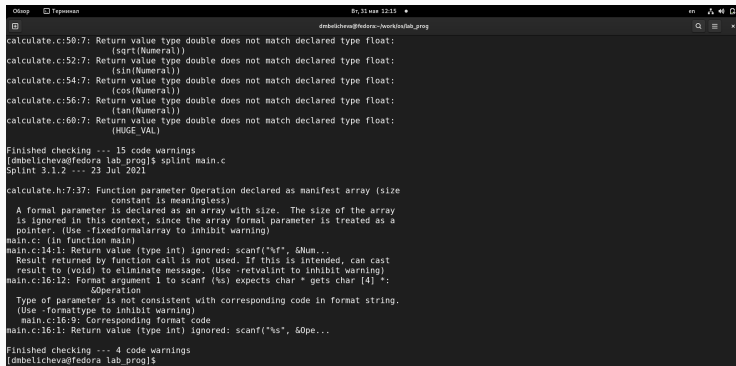


```
dbelicheva@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

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:10:31: Function parameter Operation declared as manifest array
(size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:1: 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:22:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:4: 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:38:7: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:46:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:7: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:50:7: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
```

Figure 12: Splint

# Выполнение лабораторной работы



```
calculate.c:50:7: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:56:7: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:60:7: Return value type double does not match declared type float:
    (HUGE_VAL)

Finished checking --- 15 code warnings
[dmbelicheva@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

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)
main.c: (in function main)
main.c:14:1: 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:16:12: 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:16:9: Corresponding format code
main.c:16:1: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[dmbelicheva@fedora lab_prog]$
```

Figure 13: Splint

В процессе выполнения 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.

1. Лабораторная работа № 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX Linux [Электронный ресурс]. URL:<https://esystem.rudn.ru/>.

Спасибо за внимание!