

# Лабораторная работа-13

Средства, применяемые при разработке программного обеспечения в  
ОС типа UNIX/Linux

Арсоева Залина НБИбд-01-21

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	8
Вывод:	21
Ответы на контрольные вопросы:	22

## Список иллюстраций

0.1	Создаю подкаталог . . . . .	8
0.2	Создаю файлы . . . . .	9
0.3	Скрипт . . . . .	10
0.4	Пишу скрипт . . . . .	11
0.5	Скрипт . . . . .	12
0.6	Основной файл мэйн . . . . .	12
0.7	Компиляция . . . . .	12
0.8	Файлы . . . . .	13
0.9	Создала Мэйкфайл . . . . .	13
0.10	Исправила мэйкфайл . . . . .	14
0.11	Запустила программу . . . . .	15
0.12	Использую команды . . . . .	16
0.13	Использую команды . . . . .	16
0.14	Параметры Лист . . . . .	17
0.15	Информация о точках останова . . . . .	17
0.16	Запуск программы . . . . .	18
0.17	Сравнение . . . . .	18
0.18	Удаляю точки останова . . . . .	19
0.19	splint . . . . .	19
0.20	splint . . . . .	20

## Список таблиц

## 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

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 результаты и остановится. Реализация функций калькулятора в файле `calculate.h`:

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

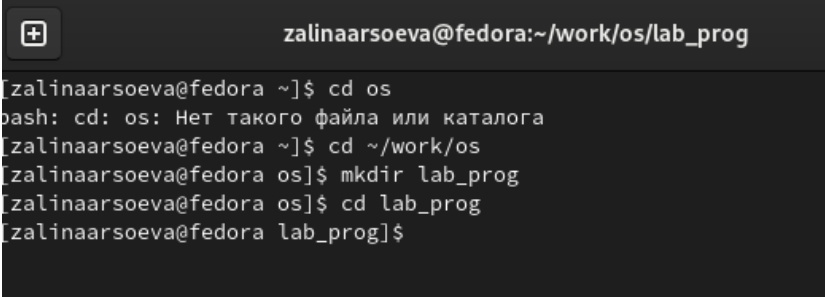
3. Выполните компиляцию программы посредством `gcc`:
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со следующим содержанием: Поясните в отчёте его содержание.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`):
  - Запустите отладчик GDB, загрузив в него программу для отладки: Кулябов Д.С. и др. Операционные системы 109 `1 gdb ./calcul`
  - Для запуска программы внутри отладчика введите команду `run`: `1 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`

спараметрами: 1 list 12,15 – Для просмотра определённых строк не основного файла используйте list с па- раметрами: 1 list calculate.c:20,29 – Установите точку останова в файле calculate.c на строке номер 21: 1 list calculate.c:20,27 2 break 21 – Выведите информацию об имеющихся в проекте точках останова: 1 info breakpoints – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: 1 run 2 5 3 - 4 backtrace – Отладчик выдаст следующую информацию: 1 #0 Calculate (Numeral=5, Operation=0x7fffffff280 "-") 2 at calculate.c:21 3 #1 0x00000000400b2b 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: 110 Лабораторная работа No 13. Средства, применяемые при разработке программного... 1 print Numeral На экран должно быть выведено число 5. – Сравните с результатом вывода на экран после использования команды: 1 display Numeral – Уберите точки останова: 1 info breakpoints 2 delete 1

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

# Выполнение лабораторной работы

1. В домашнем каталоге создала подкаталог `~/work/os/lab_prog`. (рис. [-@fig:001])



```
zalinaarsoeva@fedora:~/work/os/lab_prog
[ zalinaarsoeva@fedora ~ ]$ cd os
bash: cd: os: Нет такого файла или каталога
[ zalinaarsoeva@fedora ~ ]$ cd ~/work/os
[ zalinaarsoeva@fedora os ]$ mkdir lab_prog
[ zalinaarsoeva@fedora os ]$ cd lab_prog
[ zalinaarsoeva@fedora lab_prog ]$
```

Рис. 0.1: Создаю подкаталог

2. Создала в нём файлы: `calculate.h`, `calculate.c`, `main.c`. Это примитивнейший калькулятор, способный складывать, вычитать, умножать, делить, возводить число в степень, вычислять квадратный корень, вычислять `sin`, `cos`, `tan`. При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается. (рис. [-@fig:002])



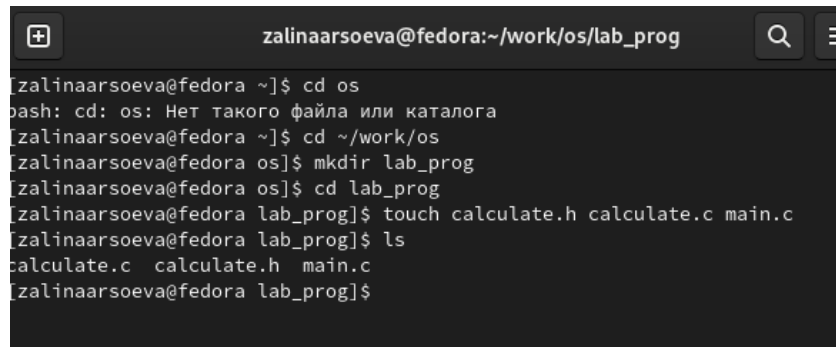
A terminal window with a dark background and light text. The title bar at the top shows a plus icon, the username and host 'zalinaarsoeva@fedora', the current directory '~/work/os/lab\_prog', a search icon, and a menu icon. The terminal content shows a series of commands and their outputs: 'cd os' results in an error 'bash: cd: os: Нет такого файла или каталога'; 'cd ~/work/os' changes the directory; 'mkdir lab\_prog' creates the directory; 'cd lab\_prog' changes to the new directory; 'touch calculate.h calculate.c main.c' creates three files; 'ls' lists the files 'calculate.c', 'calculate.h', and 'main.c'.

Рис. 0.2: Создаю файлы

Реализация функций калькулятора в файле `calculate.c`: (рис. [-@fig:003]) (рис. [-@fig:004])

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <string.h>
4  #include "calculate.h"
5
6  float
7  Calculate(float Numeral, char Operation[4])
8  {
9      float SecondNumeral;
10     if(strncmp(Operation, "+", 1) == 0)
11     {
12         printf("Второе слагаемое: ");
13         scanf("%f", &SecondNumeral);
14         return(Numeral + SecondNumeral);
15     }
16     else if(strncmp(Operation, "-", 1) == 0)
17     {
18         printf("Вычитаемое: ");
19         scanf("%f", &SecondNumeral);
20         return(Numeral - SecondNumeral);
21     }
22     else if(strncmp(Operation, "*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f", &SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strncmp(Operation, "/", 1) == 0)
29     {
30         printf("Делитель:");
31         scanf("%f", &SecondNumeral);
32         if(SecondNumeral == 0)
33         {
34             printf("Ошибка: деление на ноль");
35             return(HUGE_VAL);
36         }
37         else
38             return(Numeral / SecondNumeral);
39     }
40     else if(strncmp(Operation, "pow", 3) == 0)
```

Рис. 0.3: Скрипт

```
22     else if(strncmp(Operation, "*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f", &SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strncmp(Operation, "/", 1) == 0)
29     {
30         printf("Делитель:");
31         scanf("%f", &SecondNumeral);
32         if(SecondNumeral == 0)
33         {
34             printf("Ошибка: деление на ноль");
35             return(HUGE_VAL);
36         }
37         else
38             return(Numeral / SecondNumeral);
39     }
40     else if(strncmp(Operation, "pow", 3) == 0)
41     {
42         printf("Степень: ");
43         scanf("%f", &SecondNumeral);
44         return(pow(Numeral, SecondNumeral));
45     }
46     else if(strncmp(Operation, "sqrt", 4) == 0)
47     |     return(sqrt(Numeral));
48     else if(strncmp(Operation, "sin", 3) == 0)
49     |     return(sin(Numeral));
50     else if(strncmp(Operation, "cos", 3) == 0)
51     |     return(cos(Numeral));
52     else if(strncmp(Operation, "tan", 3) == 0)
53     |     return(tan(Numeral));
54     else
55     |
56     |     printf("Неправильно введено действие ");
57     |     return(HUGE_VAL);
58     |
59 }
```

Рис. 0.4: Пишу скрипт

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:(рис. [-@fig:005])

```

C calculate.c  C calculate.h x
C calculate.h
1  #ifndef CALCULATE_H_
2  #define CALCULATE_H_
3
4  float Calculate(float Numeral, char Operation[4]);
5
6  #endif

```

Рис. 0.5: Скрипт

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:(рис. [-@fig:006])

```

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

```

Рис. 0.6: Основной файл мэйн

3.Выполнила компиляцию программы посредством gcc:(рис. [-@fig:007])(рис. [-@fig:008])

```

[zalinaarsoeva@fedora lab_prog]$ gcc -c calculate.c
[zalinaarsoeva@fedora lab_prog]$ gcc -c main.c
[zalinaarsoeva@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm

```

Рис. 0.7: Компиляция

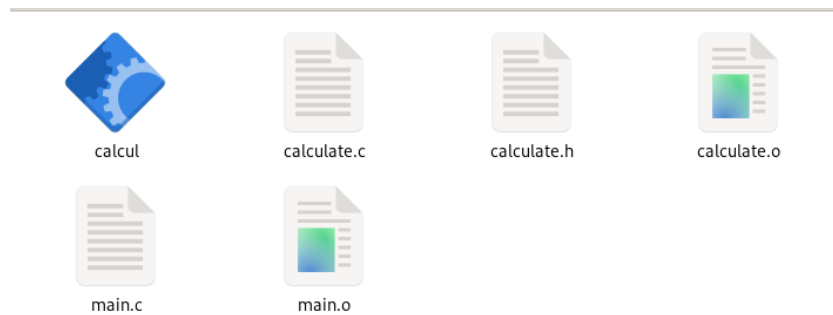


Рис. 0.8: Файлы

4. Исправила синтаксические ошибки.
5. Создала Makefile.(рис. [-@fig:009])

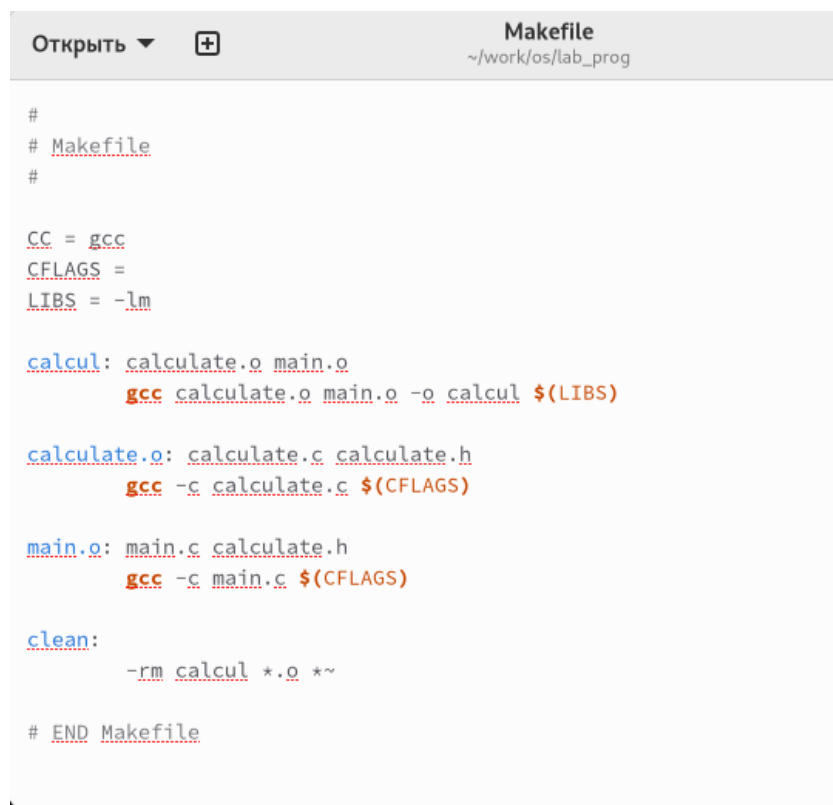
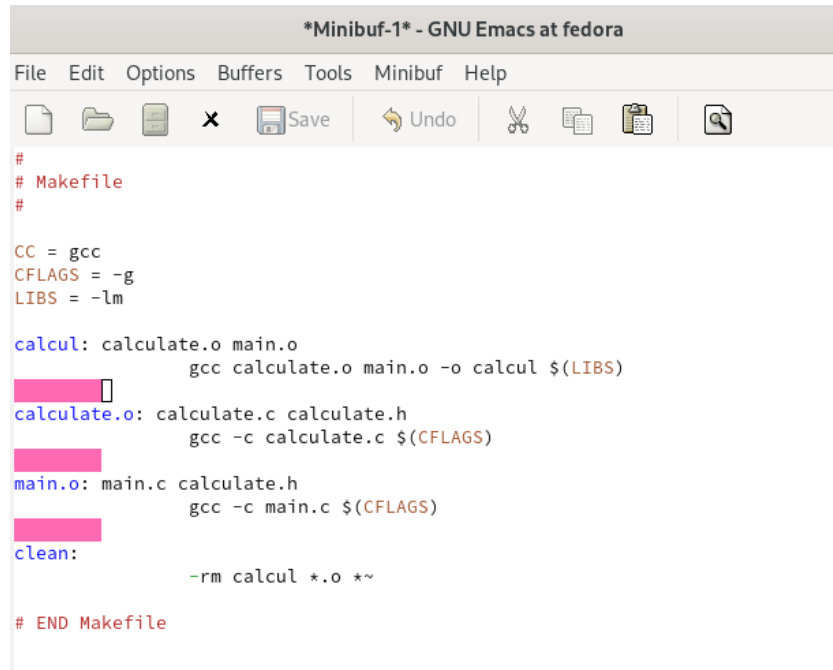


Рис. 0.9: Создала Мэйкфайл

В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик.

6. С помощью gdb выполнила отладку программы calcul (перед использованием gdb исправил Makefile): – запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` – для запуска программы внутри отладчика ввела команду `run`(рис. [-@fig:010])(рис. [-@fig:011])



```
#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    gcc calculate.o main.o -o calcul $(LIBS)

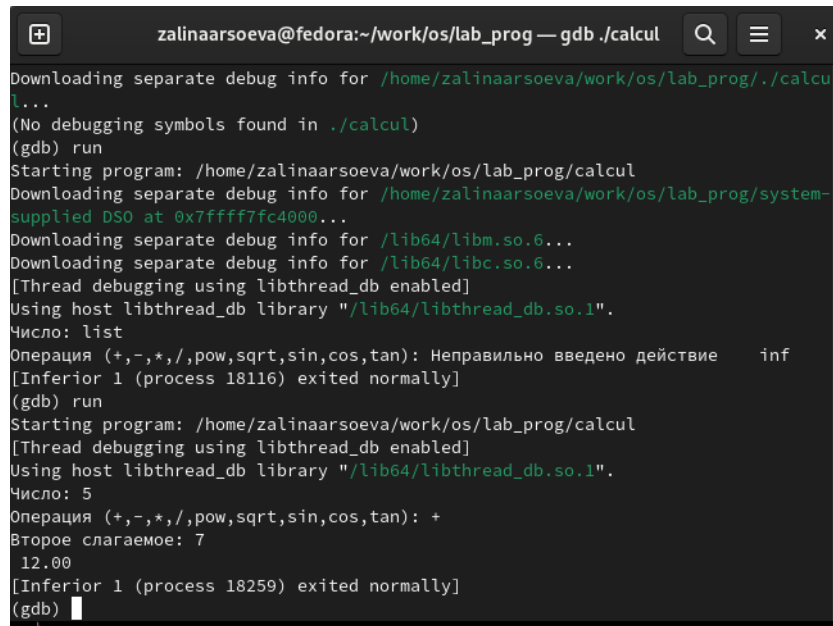
calculate.o: calculate.c calculate.h
    gcc -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
    gcc -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# END Makefile
```

Рис. 0.10: Исправила мэйкфайл



```
zalinaarsoeva@fedora:~/work/os/lab_prog — gdb ./calcul
Downloading separate debug info for /home/zalinaarsoeva/work/os/lab_prog/./calcul...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/zalinaarsoeva/work/os/lab_prog/calcul
Downloading separate debug info for /home/zalinaarsoeva/work/os/lab_prog/system-supplied DSO at 0x7ffff7fc4000...
Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: list
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): Неправильно введено действие inf
[Inferior 1 (process 18116) exited normally]
(gdb) run
Starting program: /home/zalinaarsoeva/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): +
Второе слагаемое: 7
12.00
[Inferior 1 (process 18259) exited normally]
(gdb)
```

Рис. 0.11: Запустила программу

- для постраничного (по 9 строк) просмотра исходного код использовал команду `list`
- для просмотра строк с 12 по 15 основного файла использовала `list` с параметрами: `list 12,15`(рис. [-@fig:012])(рис. [-@fig:013])

```

(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3
4      int
5      main (void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Число: ");
(gdb)

```

Рис. 0.12: Используя команды

```

(gdb) list 12,15
12     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13     scanf("%s",Operation);
14     Result = Calculate(Numeral,Operation);
15     printf("%6.2f\n",Result);
(gdb) list calculate.c:20,29
20     return(Numeral-SecondNumeral);
21 }
22 else if(strncmp(Operation,"+",1) == 0)
23 {
24     printf("Множитель: ");
25     scanf("%f",&SecondNumeral);
26     return(Numeral*SecondNumeral);
27 }
28 else if(strncmp(Operation,"/",1) == 0)
29 {
(gdb)

```

Рис. 0.13: Используя команды

– для просмотра определённых строк не основного файла использовал list с параметрами: list calculate.c:20,29 – установила точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 20 – вывела информацию об имеющихся в проекте точка останова: info breakpoints(рис. [-@fig:014])(рис. [-@fig:015])



```

(gdb) list calculate.c:20,29
20         return(Numeral-SecondNumeral);
21     }
22     else if(strncmp(Operation,"*",1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral*SecondNumeral);
27     }
28     else if(strncmp(Operation,"/",1) == 0)
29     {
(gdb) list calculate.c:20,27
20         return(Numeral-SecondNumeral);
21     }
22     else if(strncmp(Operation,"*",1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral*SecondNumeral);
27     }
(gdb) break 21
Breakpoint 1 at 0x401247: file calculate.c, line 22.

```

Рис. 0.14: Параметры Лист

```

(gdb) break 21
Note: breakpoint 1 also set at pc 0x401247.
Breakpoint 2 at 0x401247: file calculate.c, line 22.
(gdb) break 20
Breakpoint 3 at 0x401234: file calculate.c, line 20.
(gdb) break 21
Note: breakpoints 1 and 2 also set at pc 0x401247.
Breakpoint 4 at 0x401247: file calculate.c, line 22.
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) run
Starting program: /home/abrovkin/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): -
Вычисляемое: backtrace

Breakpoint 3, Calculate (Numeral=5, Operation=0x7fffffffdee4 "-") at calculate.c:20
20     return(Numeral-SecondNumeral);
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Undefined info command: "brskpoints". Try "help info".
(gdb) info breakpoints
Num  Type      Disp Enb Address          What
1     breakpoint keep y   0x0000000000401247 in Calculate at calculate.c:22
2     breakpoint keep y   0x0000000000401247 in Calculate at calculate.c:22
3     breakpoint keep y   0x0000000000401234 in Calculate at calculate.c:20
4     breakpoint already hit 1 time
      breakpoint keep y   0x0000000000401247 in Calculate at calculate.c:22
(gdb) delete 1
(gdb) delete 2
(gdb) delete 3

```

Рис. 0.15: Информация о точках останова

– запустила программу внутри отладчика и убедился, что программа остановится

в момент прохождения точки останова – отладчик выдал следующую информацию, 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` – сравнила с результатом вывода на экран после использования команды: `display Numeral` – убрала точки останова: `info breakpoints delete 1`(рис. [-@fig:016])(рис. [-@fig:017])(рис. [-@fig:018])

```
(gdb) run
Starting program: /home/abrovkin/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): -
Вычитаемое: backtrace

Breakpoint 3, Calculate (Numeral=5, Operation=0x7fffffffdee4 "-") at calculate.c:20
20      return Numeral*SecondNumeral;
(gdb)
```

Рис. 0.16: Запуск программы

```
(gdb) list calculate.c:20,27
20      return Numeral*SecondNumeral;
21  }
22  else if (strcmp Operation,"*" != 0)
23  {
24      printf("Множитель: ");
25      scanf("%f",&SecondNumeral);
26      return Numeral*SecondNumeral;
27  }
(gdb) break 21
Note: breakpoint 1 also set at pc 0x401247.
Breakpoint 2 at 0x401247: file calculate.c, line 22.
(gdb) break 20
Breakpoint 3 at 0x401234: file calculate.c, line 20.
(gdb) break 21
Note: breakpoints 1 and 2 also set at pc 0x401247.
Breakpoint 4 at 0x401247: file calculate.c, line 22.
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) run
Starting program: /home/abrovkin/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): -
Вычитаемое: backtrace

Breakpoint 3, Calculate (Numeral=5, Operation=0x7fffffffdee4 "-") at calculate.c:20
20      return Numeral*SecondNumeral;
(gdb) print Numeral
$1 = 5
(gdb)
```

Рис. 0.17: Сравнение

```

(gdb) display Numeral
1: Numeral = 5
(gdb) info braskpoints
Undefined info command: "braskpoints". Try "help info".
(gdb) info breakpoints
Num  Type      Disp Enb Address          What
1    breakpoint keep y  0x0000000000401247 in Calculate at calculate.c:22
2    breakpoint keep y  0x0000000000401247 in Calculate at calculate.c:22
3    breakpoint keep y  0x0000000000401234 in Calculate at calculate.c:20
    breakpoint already hit 1 time
4    breakpoint keep y  0x0000000000401247 in Calculate at calculate.c:22
(gdb) delete 1
(gdb) delete 2
(gdb) delete 3
(gdb) delete 4
(gdb)

```

Рис. 0.18: Удаляю точки останова

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.(рис. [-@fig:019])(рис. [-@fig:020])

```

[zalinaarsoeva@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 22 Jan 2022

calculate.h:4:37: 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:7:31: Function parameter Operation declared as manifest array (size
constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:13:9: 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:19:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:21:17: Parse Error. (For help on parse errors, see splint -help
parseerrors.)
*** Cannot continue.
[zalinaarsoeva@fedora lab_prog]$

```

Рис. 0.19: splint

```
[zalinaarsoeva@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 22 Jan 2022

calculate.h:4:37: 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:11:5: 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:13:17: 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:13:13: Corresponding format code
main.c:13:5: Return value (type int) ignored: scanf("%s", &Ope...

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

Рис. 0.20: splint

## 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.

## Ответы на контрольные вопросы:

1. Информацию об этих программах можно получить с помощью функций `info` и `man`.
2. Unix поддерживает следующие основные этапы разработки приложений:
  - создание исходного кода программы; - 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.
3. Использование суффикса “.c” 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.o` 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`.

4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа `make` освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом `make-файле`, который по умолчанию имеет имя `makefile` или `Makefile`.
6. В общем случае `make-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. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС 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. Приведённый выше `make-файл` для программы `abcd.c` включает

два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем `abcd`. Второй способ позволяет включать в исполняемый модуль `testabcd` возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.

7. 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.
8. `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. 1) Выполнила компиляцию программы 2) Увидела ошибки в программе 3) Открыла редактор и исправила программу 4) Загрузила программу в отладчик `gdb` 5) `run` — отладчик выполнил программу, ввела требуемые значения. 6) Использовала другие команды отладчика и проверила работу программы
10. Отладчику не понравился формат `%s` для `&Operation`, т.к `%s` — символьный формат, а значит необходим только `Operation`.
11. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: — `cscope` - исследование функций, содержащихся в программе; — `splint` — критическая проверка программ, написанных на языке Си.
12. 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.