

# **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №13**

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

Анастасия Павловна Баранова, НБИбд-01-21

# Содержание

|          |                                       |           |
|----------|---------------------------------------|-----------|
| <b>1</b> | <b>Цель работы</b>                    | <b>4</b>  |
| <b>2</b> | <b>Задание</b>                        | <b>5</b>  |
| <b>3</b> | <b>Выполнение лабораторной работы</b> | <b>7</b>  |
| <b>4</b> | <b>Вывод</b>                          | <b>15</b> |
| <b>5</b> | <b>Ответы на контрольные вопросы</b>  | <b>16</b> |

## Список иллюстраций

|      |   |    |
|------|---|----|
| 3.1  | В домашнем каталоге создам подкаталог ~/work/os/lab_prog. . . .                                 | 7  |
| 3.2  | Создам в нём файлы: calculate.h, calculate.c, main.c. . . . .                                   | 7  |
| 3.3  | Реализация функций калькулятора в файле calculate.c. . . . .                                    | 8  |
| 3.4  | Интерфейсный файл calculate.h. . . . .  | 8  |
| 3.5  | Основной файл main.c. . . . .   | 9  |
| 3.6  | Выполню компиляцию программы посредством gcc. . . . .   | 9  |
| 3.7  | Создам Makefile со следующим содержанием. . . . .   | 9  |
| 3.8  | Запущу отладчик GDB. . . . .  | 10 |
| 3.9  | Перед использованием gdb исправлю Makefile. . . . .   | 10 |
| 3.10 | Для запуска программы внутри отладчика введу команду run. . .                                   | 11 |
| 3.11 | Для постраничного просмотра кода использую команду list. . .                                    | 11 |
| 3.12 | Использую list с параметрами: list calculate.c:20,29. . . . .                                   | 12 |
| 3.13 | Установлю точку останова. . . . .   | 12 |
| 3.14 | Выведу информацию об имеющихся точках останова. . . . .   | 12 |
| 3.15 | Запущу программу внутри отладчика. . . . .  | 13 |
| 3.16 | Посмотрю, чему равно на этом этапе значение переменной Numeral. 13                              |    |
| 3.17 | Сравню с результатом вывода на экран после использования ко-<br>манды: display Numeral. . . . . | 13 |
| 3.18 | Уберу точки останова. . . . .   | 14 |
| 3.19 | С помощью утилиты splint проанализирую код файла calculate.c. .                                 | 14 |
| 3.20 | С помощью утилиты splint проанализирую код файла main.c. . . .                                  | 14 |

# 1 Цель работы

Целью данной лабораторной работы является 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.

## 2 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 результат и остановится.
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: `gdb ./calcul` – Для запуска программы внутри отладчика введите команду `run: run` – Для постраничного (по 9 строк) просмотра исходного код используйте команду `list: 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: `run 5`

- `backtrace` – Отладчик выдаст следующую информацию: `#0 Calculate (Numeral=5, Operation=0x7ffffffd280 "-") at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17` а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрите, чему равно на этом этапе значение переменной `Numeral`, введя: `print Numeral` На экран должно быть выведено число 5. – Сравните с результатом вывода на экран после использования команды: `display Numeral` – Уберите точки останова: `info breakpoints delete 1`
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

### 3 Выполнение лабораторной работы

В домашнем каталоге создам подкаталог `~/work/os/lab_prog`. (рис. 3.1)

```
anastasia@Anastasia-PC:~/work/os$ mkdir lab_prog
```

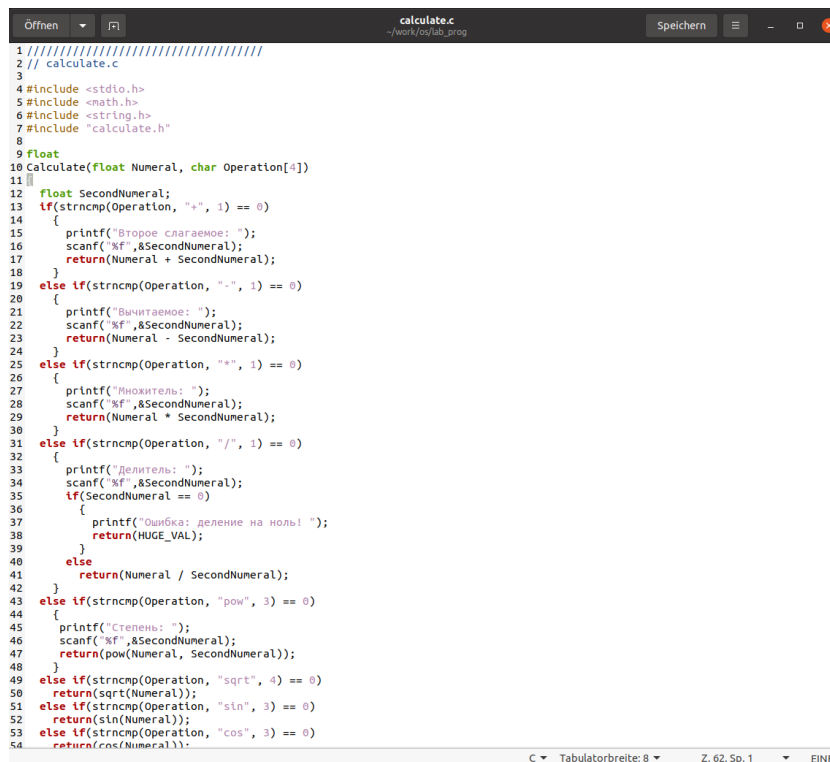
Рис. 3.1: В домашнем каталоге создам подкаталог `~/work/os/lab_prog`.

Создам в нём файлы: `calculate.h`, `calculate.c`, `main.c`. (рис. 3.2) 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.

```
anastasia@Anastasia-PC:~/work/os$ touch calculate.h
anastasia@Anastasia-PC:~/work/os$ touch calculate.c
anastasia@Anastasia-PC:~/work/os$ touch main.c
anastasia@Anastasia-PC:~/work/os$ ls
calculate.c calculate.h lab06 lab_prog main.c
anastasia@Anastasia-PC:~/work/os$
```

Рис. 3.2: Создам в нём файлы: `calculate.h`, `calculate.c`, `main.c`.

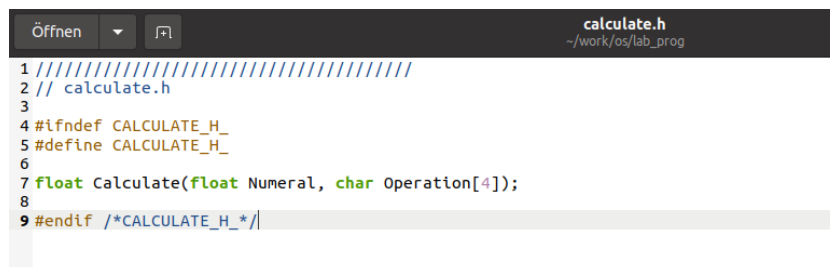
Реализация функций калькулятора в файле `calculate.c` (рис. 3.3):



```
1 //////////////////////////////////////////////////
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include "calculate.h"
8
9 float
10 Calculate(float Numeral, char Operation[4])
11 {
12     float SecondNumeral;
13     if(strncmp(Operation, "+", 1) == 0)
14     {
15         printf("Второе слагаемое: ");
16         scanf("%f", &SecondNumeral);
17         return(Numeral + SecondNumeral);
18     }
19     else if(strncmp(Operation, "-", 1) == 0)
20     {
21         printf("Вычитаемое: ");
22         scanf("%f", &SecondNumeral);
23         return(Numeral - SecondNumeral);
24     }
25     else if(strncmp(Operation, "*", 1) == 0)
26     {
27         printf("Умножитель: ");
28         scanf("%f", &SecondNumeral);
29         return(Numeral * SecondNumeral);
30     }
31     else if(strncmp(Operation, "/", 1) == 0)
32     {
33         printf("Делитель: ");
34         scanf("%f", &SecondNumeral);
35         if(SecondNumeral == 0)
36         {
37             printf("Ошибка: деление на ноль! ");
38             return(HUGE_VAL);
39         }
40         else
41             return(Numeral / SecondNumeral);
42     }
43     else if(strncmp(Operation, "pow", 3) == 0)
44     {
45         printf("Степень: ");
46         scanf("%f", &SecondNumeral);
47         return(pow(Numeral, SecondNumeral));
48     }
49     else if(strncmp(Operation, "sqrt", 4) == 0)
50         return(sqrt(Numeral));
51     else if(strncmp(Operation, "sin", 3) == 0)
52         return(sin(Numeral));
53     else if(strncmp(Operation, "cos", 3) == 0)
54         return(cos(Numeral));
55 }
```

Рис. 3.3: Реализация функций калькулятора в файле calculate.c.

Интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора (рис. 3.4):



```
1 //////////////////////////////////////////////////
2 // calculate.h
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6
7 float Calculate(float Numeral, char Operation[4]);
8
9 #endif /*CALCULATE_H_*/
```

Рис. 3.4: Интерфейсный файл calculate.h.

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. 3.5):





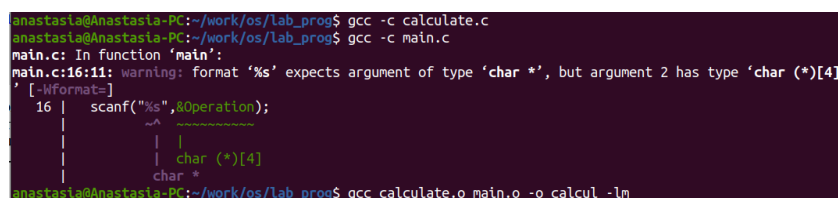
```

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[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 }

```

Рис. 3.5: Основной файл main.c.

Выполню компиляцию программы посредством gcc (рис. 3.6):



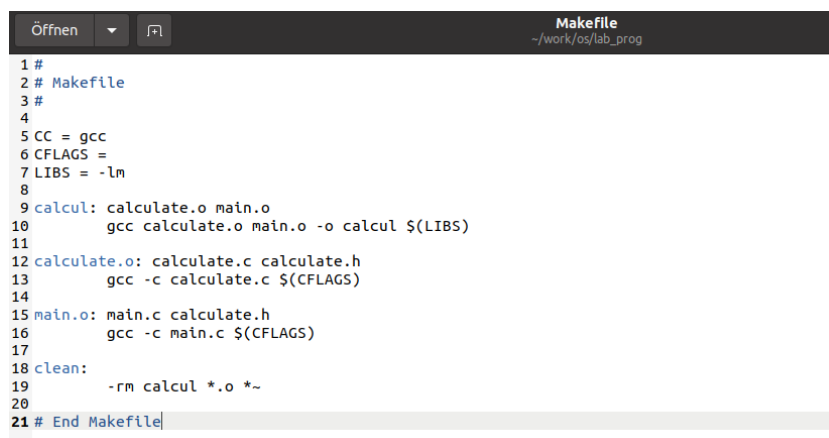
```

anastasia@Anastasia-PC:~/work/os/lab_prog$ gcc -c calculate.c
anastasia@Anastasia-PC:~/work/os/lab_prog$ gcc -c main.c
main.c: In function 'main':
main.c:16:11: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[4]' [-Wformat=]
   16 |     scanf("%s",&Operation);
      |           ^
      |           |
      |           | char (*)[4]
      |           |
      |           char *
anastasia@Anastasia-PC:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm

```

Рис. 3.6: Выполню компиляцию программы посредством gcc.

Создам Makefile со следующим содержанием (рис. 3.7):



```

1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS =
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10     gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13     gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16     gcc -c main.c $(CFLAGS)
17
18 clean:
19     -rm calcul *.o *~
20
21 # End Makefile

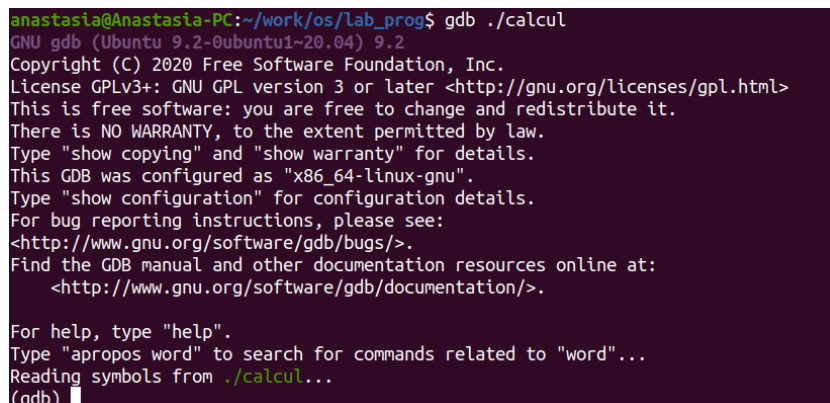
```

Рис. 3.7: Создам Makefile со следующим содержанием.

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

С помощью gdb выполню отладку программы calcul (перед использованием gdb исправлю Makefile):

- Запущу отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` (рис. 3.8, рис. 3.9)



```
anastasia@Anastasia-PC:~/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/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb)
```

Рис. 3.8: Запущу отладчик GDB.



```
Makefile - GNU Emacs at Anastasia-PC
File Edit Options Buffers Tools Makefile Help

#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm

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

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

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

clean:
-rm calcul *.o *~

# End Makefile
```

Рис. 3.9: Перед использованием gdb исправлю Makefile.

- Для запуска программы внутри отладчика введу команду `run: run` (рис. 3.10)

```
(gdb) run
Starting program: /home/anastasia/work/os/lab_prog/calcul
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
9.00
[Inferior 1 (process 65651) exited normally]
(gdb) █
```

Рис. 3.10: Для запуска программы внутри отладчика введу команду run.

- Для постраничного (по 9 строк) просмотра исходного кода использую команду list: list (рис. 3.11)

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

Рис. 3.11: Для постраничного просмотра кода использую команду list.

- Для просмотра строк с 12 по 15 основного файла использую list с параметрами: list 12,15 (рис. 3.11)
- Для просмотра определённых строк не основного файла использую list с параметрами: list calculate.c:20,29 (рис. 3.12)

```
(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. 3.12: Используя list с параметрами: list calculate.c:20,29.

- Установлю точку останова в файле calculate.c на строке номер 21 (рис. 3.13):  
list calculate.c:20,27 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
Haltepunkt 1 at 0x5555555552dd: file calculate.c, line 21.
(gdb) █
```

Рис. 3.13: Установлю точку останова.

- Выведу информацию об имеющихся в проекте точках останова (рис. 3.14):  
info breakpoints

```
(gdb) info breakpoints
Num   Type             Disp Enb Address          What
1     breakpoint       keep y  0x000055555552dd in calculate at calculate.c:21
(gdb) █
```

Рис. 3.14: Выведу информацию об имеющихся точках останова.

- Запущу программу внутри отладчика и проверю, что программа остановится в момент прохождения точки останова (рис. 3.15): run 5  
– backtrace

```
(gdb) run
Starting program: /home/anastasia/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffddc4 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffddc4 "-") at calculate.c:21
#1 0x00005555555555bd in main () at main.c:17
(gdb)
```

Рис. 3.15: Запущу программу внутри отладчика.

- Отладчик выдал следующую информацию (рис. 3.15): #0 Calculate (Numeral=5, Operation=0x7fffffffddc4 "-") at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17 а команда backtrace показала весь стек вызываемых функций от начала программы до текущего места.
- Посмотрю, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран было выведено число 5 (рис. 3.16).

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

Рис. 3.16: Посмотрю, чему равно на этом этапе значение переменной Numeral.

- Сравню с результатом вывода на экран после использования команды: display Numeral (рис. 3.17).

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

Рис. 3.17: Сравню с результатом вывода на экран после использования команды: display Numeral.

- Уберу точки останова (рис. 3.18): info breakpoints delete 1

```
(gdb) info breakpoints
Num    Type         Disp Enb Address            What
1      breakpoint keep y   0x00005555555552dd in Calculate at calculate.c:21
(gdb) delete 1
(gdb)
```

Рис. 3.18: Уберу точки останова.

С помощью утилиты splint проанализирую коды файлов calculate.c и main.c (рис. 3.19, рис. 3.20).

```
anastasia@Anastasia-PC:~/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:10:31: 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:34:7: Return value (type int) ignored: scanf("%f", &sec...
calculate.c:35:10: 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:38:17: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:46:6: Return value (type int) ignored: scanf("%f", &sec...
calculate.c:47:12: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:50:11: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:52:11: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:54:11: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:56:11: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:60:13: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings
anastasia@Anastasia-PC:~/work/os/lab_prog$
```

Рис. 3.19: С помощью утилиты splint проанализирую код файла calculate.c.

```
anastasia@Anastasia-PC:~/work/os/lab_prog$ splint main.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)
main.c: (in function main)
main.c:14:3: 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:16:14: 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:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &ope...

Finished checking --- 4 code warnings
anastasia@Anastasia-PC:~/work/os/lab_prog$
```

Рис. 3.20: С помощью утилиты splint проанализирую код файла main.c.

## 4 Вывод

В ходе данной 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.

## 5 Ответы на контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

Ответ: Информацию об этих программах можно получить с помощью функций `info` и `man`.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. Ответ: 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. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Ответ: 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. Каково основное назначение компилятора языка C в UNIX? Ответ: Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. Для чего предназначена утилита `make`? Ответ: 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. Приведите пример структуры `Makefile`. Дайте характеристику основным элементам этого файла. Ответ: 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 коммента-

рия, 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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Ответ: 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. Назовите и дайте основную характеристику основным командам отладчика gdb. Ответ: `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. Ответ: 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. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Ответ: Отладчику не понравился формат `%s` для `&Operation`, т.к. `%s` — символьный формат, а значит необходим только `Operation`.

11. Назовите основные средства, повышающие понимание исходного кода программы. Ответ: 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. Каковы основные задачи, решаемые программой splint? Ответ: 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.