

# **Отчёт по лабораторной работе №12**

**Средства для создания приложений в ОС UNIX**

Тагиев Павел Фаикович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
4.1	Набор кода, компиляция и запуск . . . . .	7
4.2	Написание и разбор Makefile . . . . .	11
4.3	Отладка программы с использованием GDB . . . . .	12
4.4	Анализ кода с помощью линтера splint . . . . .	14
<b>5</b>	<b>Контрольные вопросы</b>	<b>16</b>
<b>6</b>	<b>Выводы</b>	<b>22</b>
	<b>Список литературы</b>	<b>23</b>

## Список иллюстраций

4.1	Создание каталогов и файлов . . . . .	7
4.2	Файл <code>calculate.c</code> . . . . .	8
4.3	Файл <code>calculate.h</code> . . . . .	9
4.4	Файл <code>main.c</code> . . . . .	9
4.5	Компиляция и запуск . . . . .	10
4.6	Написанный <code>Makefile</code> . . . . .	11
4.7	Запуск отладчика GDB . . . . .	12
4.8	Команда <code>list</code> . . . . .	13
4.9	Точки останова . . . . .	13
4.10	<code>splint</code> для <code>calculate.c</code> . . . . .	14
4.11	<code>splint</code> для <code>main.c</code> . . . . .	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 [1].

## 2 Задание

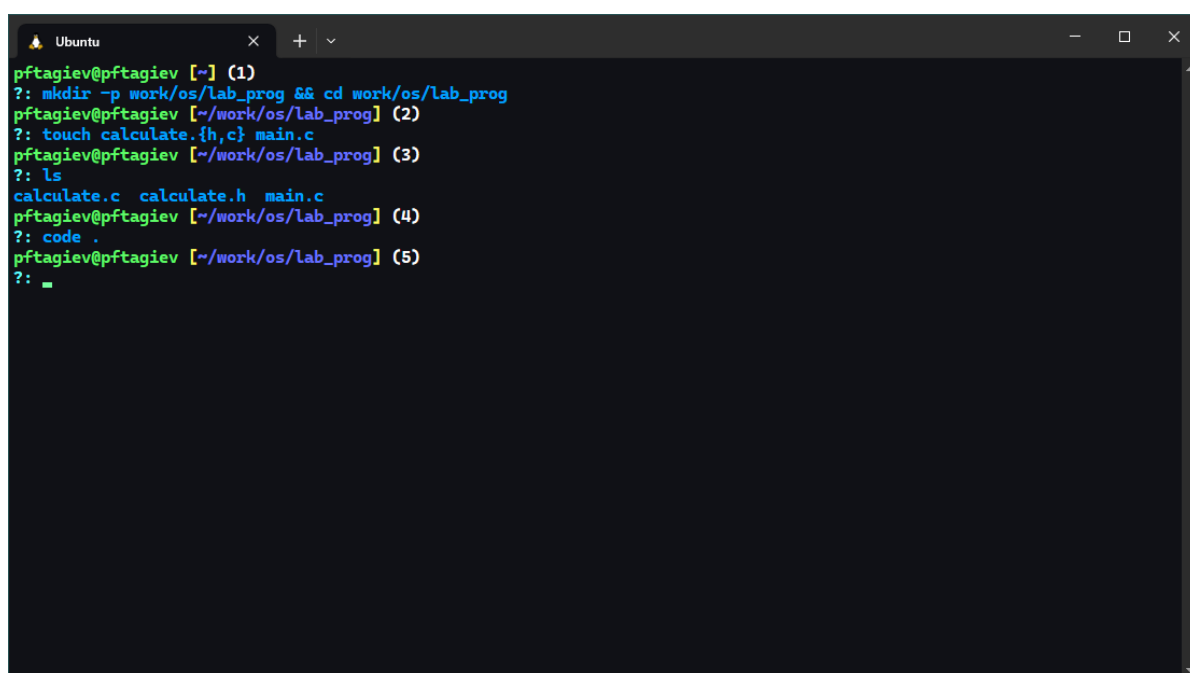
1. Провести сборку программы простейшего калькулятора
2. Произвести отладку программы используя [gdb](#)
3. Изучить предупреждения линтера `splint`

### 3 Теоретическое введение

- *GNU Compiler Collection* (обычно используется сокращение GCC) — набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU. GCC является свободным программным обеспечением, распространяется в том числе фондом свободного программного обеспечения (FSF) на условиях GNU GPL и GNU LGPL и является ключевым компонентом GNU toolchain. Он используется как стандартный компилятор для свободных UNIX-подобных операционных систем [2].
- *GNU Debugger* (GDB) — переносимый отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования, включая Си, C++, Free Pascal, FreeBASIC, Ada, Фортран и Rust. GDB — свободное программное обеспечение, распространяемое по лицензии GPL [3].
- *Splint* (Secure Programming Lint) — представляет собой инструмент программирования для статической проверки программ на языке Си на предмет уязвимостей безопасности и ошибок кодирования [4].

## 4 Выполнение лабораторной работы

### 4.1 Набор кода, компиляция и запуск

A screenshot of a terminal window titled 'Ubuntu'. The terminal shows a series of commands and their outputs, numbered 1 through 5. Command 1: 'mkdir -p work/os/lab\_prog && cd work/os/lab\_prog'. Command 2: 'touch calculate.{h,c} main.c'. Command 3: 'ls'. Command 4: 'code .'. Command 5: ' '. The output for command 3 shows 'calculate.c calculate.h main.c'. The terminal has a dark background with light-colored text.

```
pftagiev@pftagiev [~] (1)
?: mkdir -p work/os/lab_prog && cd work/os/lab_prog
pftagiev@pftagiev [~/work/os/lab_prog] (2)
?: touch calculate.{h,c} main.c
pftagiev@pftagiev [~/work/os/lab_prog] (3)
?: ls
calculate.c calculate.h main.c
pftagiev@pftagiev [~/work/os/lab_prog] (4)
?: code .
pftagiev@pftagiev [~/work/os/lab_prog] (5)
?: _
```

Рис. 4.1: Создание каталогов и файлов

Создадим каталог `~/work/os/lab_prog` и перейдем в него, как показано на рис. 4.1 на промпте (1). Затем создадим файлы в которых будет располагаться исходный код нашего калькулятора (промпт (2)). После запустим редактор кода *VSCode* (промпт (4)).

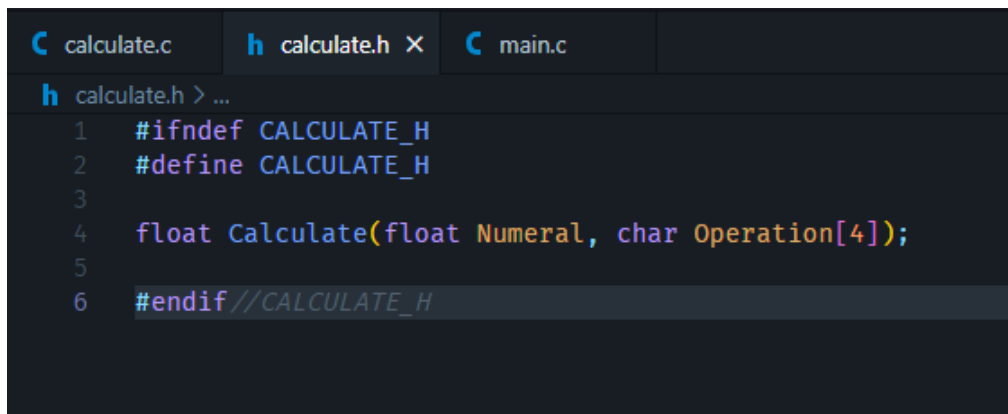
В созданных ранее файлах напомним код простейшего калькулятора, как показано на рис. 4.2, 4.3, 4.4.



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

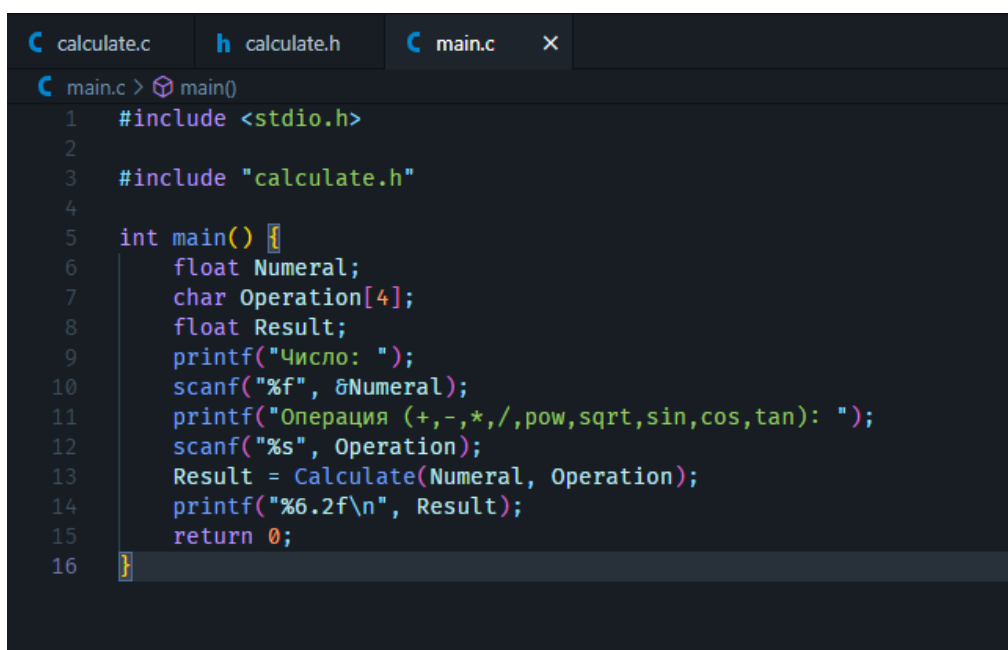
Рис. 4.2: Файл calculate.c



A screenshot of a code editor with three tabs: 'calculate.c', 'calculate.h' (active), and 'main.c'. The active tab shows the following code:

```
1  #ifndef CALCULATE_H
2  #define CALCULATE_H
3
4  float Calculate(float Numeral, char Operation[4]);
5
6  #endif //CALCULATE_H
```

Рис. 4.3: Файл calculate.h

A screenshot of a code editor with three tabs: 'calculate.c', 'calculate.h', and 'main.c' (active). The active tab shows the following code:

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

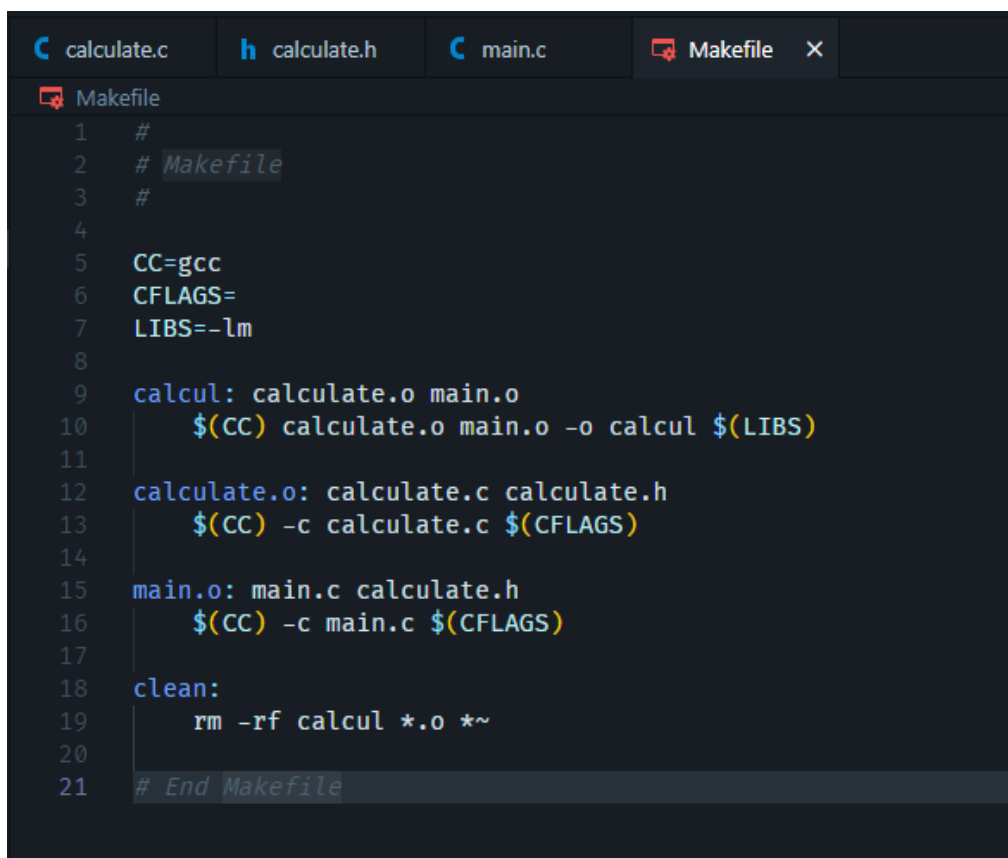
Рис. 4.4: Файл main.c

Компиляцию набранной программы можно увидеть на рис. 4.5 в промптах (1)-(3). Проверку собранной программы можно увидеть в промптах (4)-(6), все на том же рис. 4.5.

```
Ubuntu
pftagiev@pftagiev [~/work/os/lab_prog] (1)
?: gcc -c calculate.c
pftagiev@pftagiev [~/work/os/lab_prog] (2)
?: gcc -c main.c
pftagiev@pftagiev [~/work/os/lab_prog] (3)
?: gcc calculate.o main.o -o calcul -lm
pftagiev@pftagiev [~/work/os/lab_prog] (4)
?: ./calcul
Число: 100
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 2
10000.00
pftagiev@pftagiev [~/work/os/lab_prog] (5)
?: ./calcul
Число: 123
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /
Делитель: 0
Ошибка: деление на ноль!  inf
pftagiev@pftagiev [~/work/os/lab_prog] (6)
?: ./calcul
Число: 1
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): cos
0.54
pftagiev@pftagiev [~/work/os/lab_prog] (7)
?: =
```

Рис. 4.5: Компиляция и запуск

## 4.2 Написание и разбор Makefile



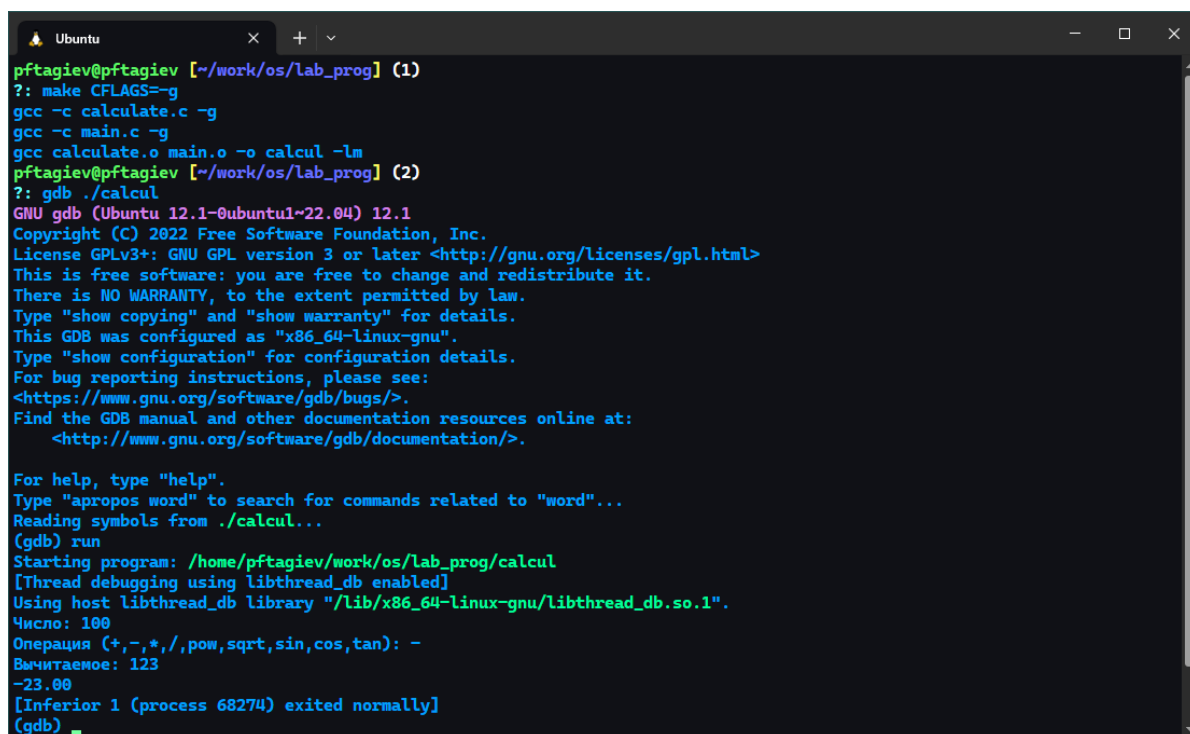
```
1 #
2 # Makefile
3 #
4
5 CC=gcc
6 CFLAGS=
7 LIBS=-lm
8
9 calcul: calculate.o main.o
10     $(CC) calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13     $(CC) -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16     $(CC) -c main.c $(CFLAGS)
17
18 clean:
19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

Рис. 4.6: Написанный Makefile

На рис. 4.6 можно увидеть написанный Makefile. Давайте разберем его содержимое. В строках 5 - 7 объявляются переменные с помощью которых можно влиять на процесс сборки программы. Например, изменить компилятор (переменная `CC`) или включить дополнительные опции (переменные `CFLAGS` и `LIBS`).

Далее объявлены таргеты сборки. `calculate.o` и `main.o`, создают объектные файлы, а таргет `calcul` компанует созданные объектные файлы в исполняемый с именем `calcul`. Последний таргет `clean` очищает директорию от созданных объектных и временных файлов.

## 4.3 Отладка программы с использованием GDB



```
pftagiev@pftagiev [~/work/os/lab_prog] (1)
?: make CFLAGS=-g
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
pftagiev@pftagiev [~/work/os/lab_prog] (2)
?: gdb ./calcul
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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/pftagiev/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Число: 100
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): -
Вычитаемое: 123
-23.00
[Inferior 1 (process 68274) exited normally]
(gdb) _
```

Рис. 4.7: Запуск отладчика GDB

Соберем написанную программу с флагом `-g`, чтобы компилятор добавил отладочную информацию. Для этого напишем `make CFLAGS=-g`, как показано на рис. 4.7 в промпте (1). Затем запустим отладчик *GDB* передав ему собранную программу. Все на том же рис. 4.7 можно увидеть использование команды `run`.

Посмотрим исходный код программы используя команду `list` в разных вариациях (рис. 4.8).

Используем точки останова (рис. 4.9, ориентируйтесь по приглашению оболочки: `(gdb)`). Выведем кусочек программы командой `list`, затем установим *breakpoint* на строке 14 (строка 14 соответствует строке 21 в задании). Выведем информацию о имеющихся точках останова командой `info breakpoints`. Затем запустим программу командой `run`. Используя команду `backtrace`, убедимся что программа остановилась в момент прохождения точки останова.

```
(gdb) list
1      #include <stdio.h>
2
3      #include "calculate.h"
4
5      int main() {
6          float Numeral;
7          char Operation[4];
8          float Result;
9          printf("Число: ");
10         scanf("%f", &Numeral);
(gdb) list 12,15
12         scanf("%s", Operation);
13         Result = Calculate(Numeral, Operation);
14         printf("%6.2f\n", Result);
15         return 0;
(gdb) list calculate.c:20,29
20         return Numeral * SecondNumeral;
21     } else if (strcmp(Operation, "/" , 1) == 0) {
22         printf("Делитель: ");
23         scanf("%f", &SecondNumeral);
24         if (SecondNumeral == 0) {
25             printf("Ошибка: деление на ноль!");
26             return HUGE_VAL;
27         }
28         return Numeral / SecondNumeral;
29     } else if (strcmp(Operation, "pow", 3) == 0) {
(gdb) _
```

Рис. 4.8: Команда list

```
(gdb) list calculate.c:14,18
14         printf("Вычитаемое: ");
15         scanf("%f", &SecondNumeral);
16         return Numeral - SecondNumeral;
17     } else if (strcmp(Operation, "*", 1) == 0) {
18         printf("Множитель: ");
(gdb) break 14
Breakpoint 1 at 0x12df: file calculate.c, line 14.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x00000000000012df in Calculate at calculate.c:14
(gdb) run
Starting program: /home/pftagiev/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf74 "-") at calculate.c:14
14         printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf74 "-") at calculate.c:14
#1 0x0000555555555641 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x0000555555555641 in Calculate at calculate.c:14
breakpoint already hit 1 time
(gdb) delete 1
(gdb) _
```

Рис. 4.9: Точки останова

Теперь выведем значение переменной `Numeral` на экран, используя команду

print, как видно значение переменной равно 5. Сравним вывод команды print с выводом команды display.

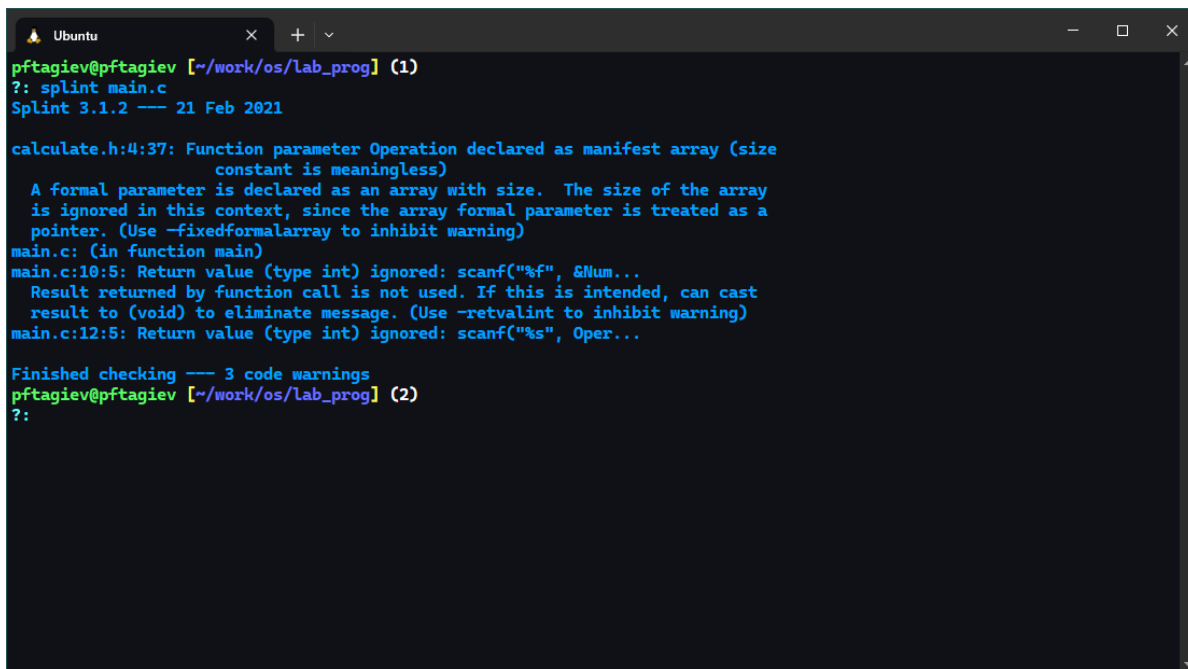
В заключение, удалим созданную точку останова и выйдем из отладчика комбинацией клавиш **CTRL + d**.

## 4.4 Анализ кода с помощью линтера splint

Рис. 4.10: splint для calculate.c

Разберем вывод утилиты splint для файла calculate.c (рис. 4.10). Сначала, выводится информация о том, что длина массива указанная в сигнатуре функции Calculate не имеет никакого смысла и игнорируется. Далее несколько раз выводится информация о том что мы игнорируем возвращаемое значение

функции `scanf`. Затем идут предупреждения о неявном преобразовании типа `double` в тип `float`. В сумме 15 предупреждений.



```
Ubuntu
pftagiev@pftagiev [~/work/os/lab_prog] (1)
?: splint main.c
Splint 3.1.2 --- 21 Feb 2021

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:10:5: 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:12:5: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
pftagiev@pftagiev [~/work/os/lab_prog] (2)
?:
```

Рис. 4.11: splint для main.c

Прделаем такой же разбор, но уже для файла `main.c`. На рис. 4.11, мы видим уже знакомые нам предупреждения о игнорировании языком Си длины массива в сигнатуре функции `Calculate` и о игнорировании нами возвращаемого значения функции `scanf`. Всего 3 предупреждения.

## 5 Контрольные вопросы

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

Можно использовать команду `man имя_программы` или посетить сайт проекта *GNU* [5].

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

- 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.

В предоставленной теории [1], суффикс эквивалентен расширению файла. Если же мы говорим о языке Си, в нем суффиксом называется то, что дописывается в конце литерала, например: `3.14f`. В этом примере `f` суффикс, который говорит о том, что литерал будет иметь тип `float`.



4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора языка Си в UNIX состоит в том, чтобы преобразовывать исходный код, написанный на языке программирования Си, в исполняемый файл, который может быть запущен операционной системой UNIX.

5. Для чего предназначена утилита make? Утилита make является мощным инструментом автоматизации сборки программного обеспечения, который широко используется в UNIX-системах. Ее основное назначение — управление процессом компиляции и сборки программ, обеспечивая эффективное и удобное обновление исполняемых файлов при внесении изменений в исходный код.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

Пример структуры Makefile и его характеристику можно увидеть на лист. 5.1.

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Основное свойство, присущее всем программам отладки, — это возможность отслеживать выполнение программы, шаг за шагом, и анализировать ее состояние в любой момент времени. Это свойство называется “отладкой” или “debug mode”. Отладка позволяет разработчикам выявлять и исправлять ошибки в программе, а также понимать логику ее работы. Чтобы использовать это свойство, необходимо выполнить следующие шаги:

1. Включить отладочную информацию при компиляции программы:  
Это делается с помощью флагов компилятора, например, `-g` в компиляторах *GCC*. Флаг `-g` указывает компилятору включить отладочную информацию в объектные файлы и исполняемый файл. Отладочная информация включает в себя данные о символах (таких

как имена переменных и функций), строках кода и расположении переменных в памяти.

2. Использовать отладчик: Отладчик — это программа, которая позволяет взаимодействовать с выполняющейся целью и контролировать ее выполнение. Примерами отладчиков являются *GDB* (GNU Debugger) для C/C++ программ и *pdb* для Python-программ. Отладчик позволяет устанавливать точки останова, просматривать значения переменных, выполнять код пошагово и изучать стеки вызовов функций.
  3. Компилировать программу с отключенными оптимизациями: Некоторые оптимизации компилятора могут усложнить процесс отладки.
8. Назовите и дайте основную характеристику основным командам отладчика *gdb*.
- Основные команды отладчика *GDB* (GNU Debugger) включают в себя следующее:
- **break**: устанавливает точку останова в указанной строке кода или функции. Когда программа выполняется и достигает точки останова, она приостанавливает свое выполнение, позволяя вам проанализировать ее состояние.
  - **run**: запускает программу под контролем отладчика. Программа выполняется до первой точки останова или до завершения.
  - **continue**: продолжает выполнение программы после остановки в точке останова.
  - **print**: выводит значение выражения или переменной. Это позволяет проверять текущие значения переменных во время выполнения программы.
  - **backtrace**: отображает стек вызовов функций, показывая последовательность функций, которые были вызваны для достижения текущей точки выполнения. Это помогает понять поток управления в программе.

- `step`: выполняет программу пошагово, переходя к следующей строке кода. Если следующая строка содержит вызов функции, отладчик заходит внутрь этой функции.
- `next`: выполняет программу пошагово, но в отличие от `step`, он переходит к следующей строке кода, не заходя внутрь вызываемых функций.
- `finish`: продолжает выполнение программы до выхода из текущей функции.
- `info breakpoints`: информация о имеющихся точках останова.

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

1. Собрать программу с ключем `-g`
2. Загрузить программу в отладчик *GDB*.
3. Расставить точки останова.
4. Запустить загруженную программу командой `run`.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

К сожалению, я переписал программу калькулятора без них, так как думал что это опечатки, и у меня не возникло никаких ошибок компиляции. Просмотрев код программы из [1], я вижу одну грубую ошибку (файл `main.c` строка 16): `scanf("%s",&Operation);`, здесь не нужно брать адрес переменной `Operation`, т.к. мы передадим функции `scanf` `char**`, а она ожидает `char*`.

11. Назовите основные средства, повышающие понимание исходного кода программы.

- инструменты статического анализа, линтеры (такие как `splint`)
- современные IDE предлагают различные функции, облегчающие понимание кода, такие как подсветка синтаксиса и автодополнение

- отладчики (такие как *GDB*)

12. Каковы основные задачи, решаемые программой splint? Программа Splint предназначена для решения следующих основных задач:

1. Статический анализ кода: Splint выполняет статический анализ кода на языке C, выявляя потенциальные ошибки, проблемы безопасности и нарушения стандартов кодирования. Он проверяет код на соответствие определенным правилам и стандартам, таким как правила из руководства по стилю кодирования MISRA C.
2. Выявление ошибок времени компиляции: Splint анализирует код на наличие синтаксических и семантических ошибок, которые могут привести к ошибкам во время компиляции. Он проверяет типы переменных, соответствие аргументов функций и соблюдение правил объявления переменных.
3. Проверка безопасности: Splint специализируется на выявлении потенциальных проблем безопасности в коде, таких как переполнение буфера, использование неинициализированных переменных, ошибки управления памятью и другие распространенные уязвимости. Он помогает разработчикам писать более безопасный и защищенный от атак код.
4. Подсказки по улучшению кода: Splint предоставляет подсказки и рекомендации по улучшению качества кода.

---

### Листинг 5.1 Пример Makefile

---

```
# Определение переменных
CC = gcc
CFLAGS = -Wall -O2

# Определение цели по умолчанию
all: программа

# Правило для сборки исполняемого файла
программа: программа.o функция.o
    $(CC) $(CFLAGS) -o программа программа.o функция.o

# Правило для компиляции исходного файла в объектный файл
.c.o:
    $(CC) $(CFLAGS) -c $< -o $@

# Правило для удаления объектных файлов и исполняемого файла
clean:
    rm -f программа.o функция.o программа
```

---

## **6 Выводы**

В этой лабораторной работе мы приобрели базовые навыки разработки, тестирования и отладки приложения в UNIX-подобных операционных системах.

## Список литературы

1. Кулябов. Операционные системы. Москва: РУДН, 2016. 118 с.
2. GNU Compiler Collection [Электронный ресурс]. 2024. URL: [https://ru.wikipedia.org/wiki/GNU\\_Compiler\\_Collection](https://ru.wikipedia.org/wiki/GNU_Compiler_Collection).
3. GNU Debugger [Электронный ресурс]. 2024. URL: [https://en.wikipedia.org/wiki/GNU\\_Debugger](https://en.wikipedia.org/wiki/GNU_Debugger).
4. Splint (programming tool) [Электронный ресурс]. 2024. URL: [https://en.wikipedia.org/wiki/Splint\\_\(programming\\_tool\)](https://en.wikipedia.org/wiki/Splint_(programming_tool)).
5. GNU Manuals Online [Электронный ресурс]. 2022. URL: <https://www.gnu.org/manual/manual.html>.