

# **Отчёт по лабораторной работе №14**

**Дисциплина: Операционные системы**

Аветисян Давид Артурович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>20</b>
<b>5</b>	<b>Выводы</b>	<b>25</b>

## **Список таблиц**

## Список иллюстраций

3.1	Создал каталог и файлы в нём . . . . .	8
3.2	Реализация функций калькулятора в файле <code>calculate.c</code> . . . . .	9
3.3	Реализация функций калькулятора в файле <code>calculate.c</code> . . . . .	10
3.4	Интерфейсный файл <code>calculate.h</code> . . . . .	10
3.5	Основной файл <code>main.c</code> . . . . .	11
3.6	Выполнил компиляцию программы посредством <code>gcc</code> . . . . .	11
3.7	Создал <code>Makefile</code> с необходимым содержанием . . . . .	12
3.8	Далее исправил <code>Makefile</code> . . . . .	13
3.9	Используем команды <code>make</code> . . . . .	13
3.10	Запустил отладчик <code>GDB</code> . . . . .	14
3.11	Запуск программы внутри отладчика . . . . .	14
3.12	Использовал команду « <code>list</code> » . . . . .	15
3.13	Просмотр строк с 12 по 15 . . . . .	15
3.14	Просмотр определённых строк не основного файла . . . . .	16
3.15	Установил точку останова в файле <code>calculate.c</code> . . . . .	16
3.16	Вывел информацию об имеющихся в проекте точках останова . . . . .	16
3.17	Запустил программу внутри отладчика до точки останова . . . . .	17
3.18	Посмотрел, чему равно <code>Numeral</code> . . . . .	17
3.19	Сравнил с результатом вывода на экран . . . . .	17
3.20	Убрал точку останова . . . . .	17
3.21	Проанализировал код файла <code>calculate.c</code> . . . . .	18
3.22	Проанализировал код файла <code>main.c</code> . . . . .	19

# 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.

## 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`:  
`gcc -c calculate.c`  
`gcc -c main.c`  
`gcc calculate.o main.o -o calcul -lm`
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`. 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
  - Для запуска программы внутри отладчика введите команду `run`
  - Для постраничного (по 10 строк) просмотра исходного кода используйте команду `list`
  - Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами
  - Для просмотра определённых строк не основного файла используйте `list` с параметрами

- Установите точку останова в файле `calculate.c` на строке номер 21
  - Выведите информацию об имеющихся в проекте точка останова
  - Запустите программу внутри отладчика и убедитесь, что программа останавливается в момент прохождения точки останова
  - Отладчик выдаст информацию, а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места
  - Посмотрите, чему равно на этом этапе значение переменной `Numeral`. На экран должно быть выведено число 5
  - Сравните с результатом вывода на экран после использования команды
  - Уберите точки останова
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

### 3 Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог `calculate` с помощью команды «`mkdir calculate`».
2. Создал в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`, используя команды «`cd calculate`» и «`touch calculate.h calculate.c main.c`» (рис. -fig. 3.1).

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14$ mkdir calculate
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14$ ls
calculate  pres14  README.md  report14
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14$ cd calculate
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ touch calculate.h calculate.c main.c
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ ls
calculate.c  calculate.h  main.c
```

Рис. 3.1: Создал каталог и файлы в нём

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.

Открыв редактор Емас, приступил к редактированию созданных файлов.

Реализация функций калькулятора в файле `calculate.c` (рис. -fig. 3.2) (рис. -fig. 3.3).



```

Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strncmp(Operation,"+",1) == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral+SecondNumeral);
    }
    else if(strncmp(Operation,"-",1) == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral-SecondNumeral);
    }
    else if(strncmp(Operation,"*",1) == 0)
    {
        printf("Множитель: ");
        scanf("%f",&SecondNumeral);
        return(Numeral*SecondNumeral);
    }
    else if(strncmp(Operation,"/",1) == 0)
    {
        printf("Делитель: ");
        scanf("%f",&SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка: деление на ноль!");
            return(HUGE_VAL);
        }
        else
            return(Numeral/SecondNumeral);
    }
    else if(strncmp(Operation,"pow",3) == 0)
    {
        printf("Степень: ");
        scanf("%f",&SecondNumeral);
        return(pow(Numeral,SecondNumeral));
    }
}

```

Рис. 3.2: Реализация функций калькулятора в файле calculate.c

```

else if(strncmp(Operation,"sqrt",4) == 0)
    return(sqrt(Numeral));
else if(strncmp(Operation,"sin",3) == 0)
    return(sin(Numeral));
else if(strncmp(Operation,"cos",3) == 0)
    return(cos(Numeral));
else if(strncmp(Operation,"tan",3) == 0)
    return(tan(Numeral));
else
{
    printf("Неправильно введено действие ");
    return(HUGE_VAL);
}
}

```

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

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

```

#ifndef CALCULATE_H_
#define CALCULATE_H_

float Calculate(float Numeral, char Operation[4]);

#endif

```

Рис. 3.4: Интерфейсный файл calculate.h

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

```

#include <stdio.h>
#include "calculate.h"

int
main (void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",Operation);
    Result = Calculate(Numeral,Operation);
    printf("%6.2f\n",Result);
    return 0;
}

```

Рис. 3.5: Основной файл main.c

3. Выполнил компиляцию программы посредством gcc, используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (рис. -fig. 3.6).

```

daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ gcc -c calculate.c
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ gcc -c main.c
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ gcc calculate.o main.o -o calcul -lm

```

Рис. 3.6: Выполнил компиляцию программы посредством gcc

4. В ходе компиляции программы никаких ошибок выявлено не было.
5. Создал Makefile с необходимым содержанием (рис. -fig. 3.7). Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS

отвечает за опции для объединения объектных файлов в один исполняемый файл.

```
CC = gcc
CFLAGS =
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 *~
```

Рис. 3.7: Создал Makefile с необходимым содержанием

6. Далее исправил Makefile (рис. -fig. 3.8). В переменную CFLAGS добавил опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделал так, что утилита компиляции выбирается с помощью переменной CC.

```

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 *~

```

Рис. 3.8: Далее исправил Makefile

После этого я удалил исполняемые и объектные файлы из каталога с помощью команды «make clean». Выполнил компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (рис. -fig. 3.9).

```

daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ make clean
rm calcul *.o *~
гм: невозможно удалить '*~': Нет такого файла или каталога
make: [Makefile:15: clean] Ошибка 1 (игнорирование)
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ make calculate.o
gcc -c calculate.c -g
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ make main.o
gcc -c main.c -g
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ make calcul
gcc calculate.o main.o -o calcul -lm

```

Рис. 3.9: Используем команды make

Далее с помощью gdb выполнил отладку программы calcul. Запустил отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul» (рис. -fig. 3.10).

```

daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ 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...

```

Рис. 3.10: Запустил отладчик GDB

Для запуска программы внутри отладчика ввёл команду «run» (рис. -fig. 3.11).

```

(gdb) run
Starting program: /home/daavetisyan/work/2020-2021/os-intro/laboratory/lab14/calculate/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
      8.00
[Inferior 1 (process 2375) exited normally]

```

Рис. 3.11: Запуск программы внутри отладчика

Для постраничного (по 10 строк) просмотра исходного кода использовал команду «list» (рис. -fig. 3.12).

```

(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) list
11         scanf("%f",&Numeral);
12         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13         scanf("%s",Operation);
14         Result = Calculate(Numeral,Operation);
15         printf("%6.2f\n",Result);
16         return 0;
17     }

```

Рис. 3.12: Использовал команду «list»

Для просмотра строк с 12 по 15 основного файла использовал команду «list 12,15» (рис. -fig. 3.13).

```

(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);

```

Рис. 3.13: Просмотр строк с 12 по 15

Для просмотра определённых строк не основного файла использовал команду «list calculate.c:20,29» (рис. -fig. 3.14).

```

(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     {

```

Рис. 3.14: Просмотр определённых строк не основного файла

Установил точку останова в файле calculate.c на строке номер 18, используя команды «list calculate.c:15,22» и «break 18» (рис. -fig. 3.15).

```

(gdb) list calculate.c:15,22
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)
(gdb) break 18
Breakpoint 3 at 0x555555552dd: file calculate.c, line 18.

```

Рис. 3.15: Установил точку останова в файле calculate.c

Вывел информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (рис. -fig. 3.16).

```

(gdb) info breakpoints
Num   Type             Disp Enb Address              What
3     breakpoint       keep y  0x0000555555552dd in calculate at calculate.c:18

```

Рис. 3.16: Вывел информацию об имеющихся в проекте точках останова



Запустил программу внутри отладчика и убедился, что программа остановилась в момент прохождения точки останова. Использовал команды «run», «5», «-» и «backtrace» (рис. -fig. 3.17).

```
(gdb) run
Starting program: /home/daavetisyan/work/2020-2021/os-intro/laboratory/lab14/calculate/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 3, Calculate (Numeral=5, Operation=0x7fffffffdf04 "-") at calculate.c:18
18      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf04 "-") at calculate.c:18
#1 0x00005555555555bd in main () at main.c:14
```

Рис. 3.17: Запустил программу внутри отладчика до точки останова

Посмотрел, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (рис. -fig. 3.18).

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

Рис. 3.18: Посмотрел, чему равно Numeral

Сравнил с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (рис. -fig. 3.19).

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

Рис. 3.19: Сравнил с результатом вывода на экран

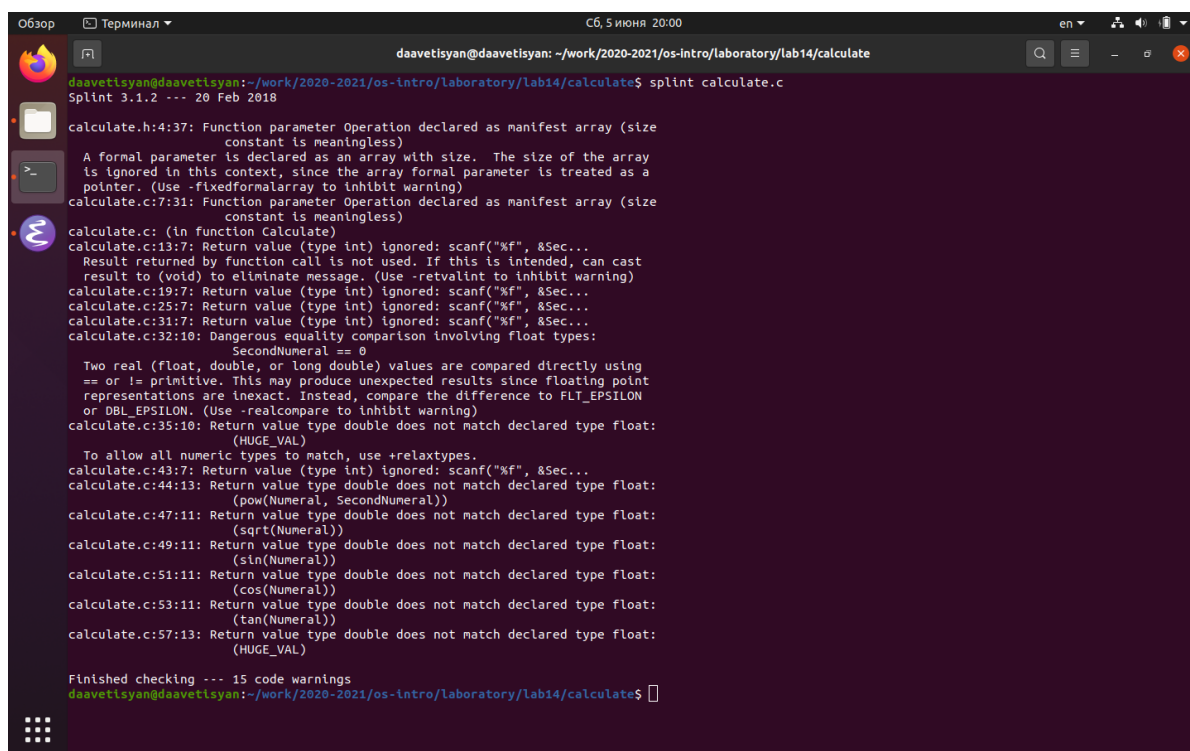
Убрал точку останова с помощью команд «info breakpoints» и «delete 3» (рис. -fig. 3.20).

```
(gdb) info breakpoints
Num   Type             Disp Enb Address                      What
3     breakpoint        keep y 0x000055555555552dd in Calculate at calculate.c:18
      breakpoint already hit 1 time
(gdb) delete 3
```

Рис. 3.20: Убрал точку останова

7. С помощью утилиты splint проанализировал коды файлов calculate.c и main.c. Воспользовался командами «splint calculate.c» и «splint main.c» (рис. -fig. 3.21) (рис. -fig. 3.22).

С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохраняются. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потере данных.

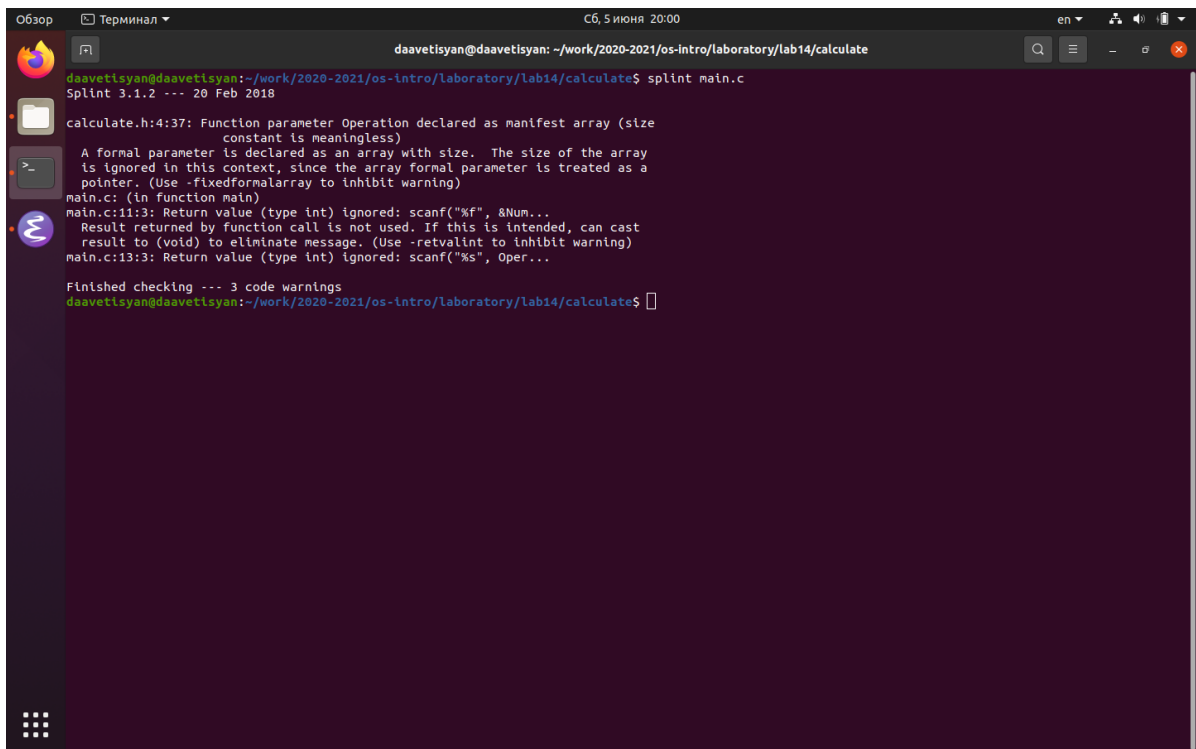


```
daavetisyan@daavetisyan: ~/work/2020-2021/os-intro/laboratory/lab14/calculate
Splint 3.1.2 --- 20 Feb 2018

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:13:7: 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:19:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:10: 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:35:10: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:43:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:13: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:47:11: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:49:11: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:51:11: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:53:11: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:57:13: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings
daavetisyan@daavetisyan: ~/work/2020-2021/os-intro/laboratory/lab14/calculate
```

Рис. 3.21: Проанализировал код файла calculate.c



```
Обзор Терминал C6, 5 июня 20:00 en
daavetisyan@daavetisyan: ~/work/2020-2021/os-intro/laboratory/lab14/calculate
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

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:11:3: 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:13:3: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab14/calculate$
```

Рис. 3.22: Проанализировал код файла main.c

## 4 Контрольные вопросы

1. Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.
2. 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.

3. Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. 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. Например, в команде «gcc -c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль – файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».
4. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
5. Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
6. Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса.

В самом простом случае Makefile имеет следующий синтаксис:

... : ...

<команда 1>

...

Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции.

В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды – собственно действия, которые необходимо выполнить для достижения цели.

Общий синтаксис Makefile имеет вид:

```
target1 [target2...]:[:] [dependment1...]
```

```
[(tab)commands] [#commentary]
```

```
[(tab)commands] [#commentary]
```

Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

7. Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger).

Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc:

```
gcc -c file.c -g
```

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл:

```
gdb file.o
```

## 8. Основные команды отладчика gdb:

- `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

Для выхода из gdb можно воспользоваться командой `quit` (или её сокращённым вариантом `q`) или комбинацией клавиш `Ctrl-d`. Более подробную информацию по работе с gdb можно получить с помощью команд `gdb -h` и `man gdb`.

9. Схема отладки программы показана в 6 пункте лабораторной работы.
10. При первом запуске компилятор не выдал никаких ошибок, но в коде программы `main.c` допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке `scanf("%s", &Operation);` нужно убрать знак `&`, потому что имя массива символов уже является указателем на первый элемент этого массива.
11. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:
- `cscope` – исследование функций, содержащихся в программе,
  - `lint` – критическая проверка программ, написанных на языке Си.
12. Утилита `splint` анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор `splint` генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.



## 5 Выводы

В ходе выполнения данной лабораторной работы я приобрёл простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.