

# **Отчет по лабораторной работе №13**

**дисциплина: операционные системы**

**Шмаков Максим Павлович**

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	8
Выводы	18
Контрольные вопросы	19

# Список иллюстраций

0.1. рис. 1 . . . . .	8
0.2. рис. 2 . . . . .	8
0.3. рис. 3 . . . . .	9
0.4. рис. 4 . . . . .	10
0.5. рис. 5 . . . . .	10
0.6. рис. 6 . . . . .	10
0.7. рис. 7 . . . . .	11
0.8. рис. 8 . . . . .	11
0.9. рис. 9 . . . . .	12
0.10. рис. 10 . . . . .	12
0.11. рис. 11 . . . . .	13
0.12. рис. 12 . . . . .	13
0.13. рис. 13 . . . . .	14
0.14. рис. 14 . . . . .	14
0.15. рис. 15 . . . . .	14
0.16. рис. 16 . . . . .	15
0.17. рис. 17 . . . . .	15
0.18. рис. 18 . . . . .	16
0.19. рис. 19 . . . . .	16
0.20. рис. 20 . . . . .	16
0.21. рис. 21 . . . . .	17

## **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 результат и остановится.
3. Выполните компиляцию программы посредством `gcc`:
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со следующим содержанием:
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 9 строк) просмотра исходного код используйте команду `list`: – Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами 12, 15: – Для просмотра определённых строк не основного файла используйте `list` с параметрами `calculate.c:20,29`: – 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 остановится в момент прохождения точки останова: – Посмотрите, чему равно на этом этапе значение переменной `Numeral` с помощью `print Numeral` и сравните с `display Numeral`: – Уберите точки останова:

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

# Выполнение лабораторной работы

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

```
[mpshmakov@fedora ~]$ mkdir work1/os/lab_prog
```

Рис. 0.1.: рис. 1

2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`. (рис. [-@fig:002])

```
[mpshmakov@fedora ~]$ cd work1/os/lab_prog  
[mpshmakov@fedora lab_prog]$ touch calculate.h calculate.c main.c
```

Рис. 0.2.: рис. 2

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

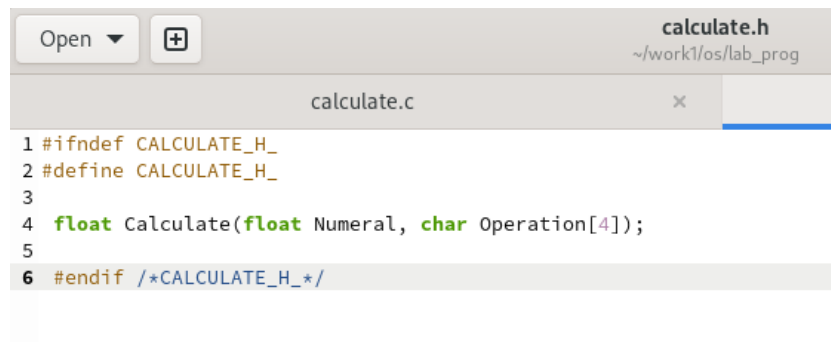




```
1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4 #include "calculate.h"
5
6 float
7 Calculate(float Numeral, char Operation[4])
8 {
9     float SecondNumeral;
10    if(strncmp(Operation, "+", 1) == 0)
11    {
12        printf("Второе слагаемое: ");
13        scanf("%f",&SecondNumeral);
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)
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
49    Saving file "/home/mpshmakov/work1/os/lab_prog/calculate.c"...
```

Рис. 0.3.: рис. 3

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора: (рис. [-@fig:004])



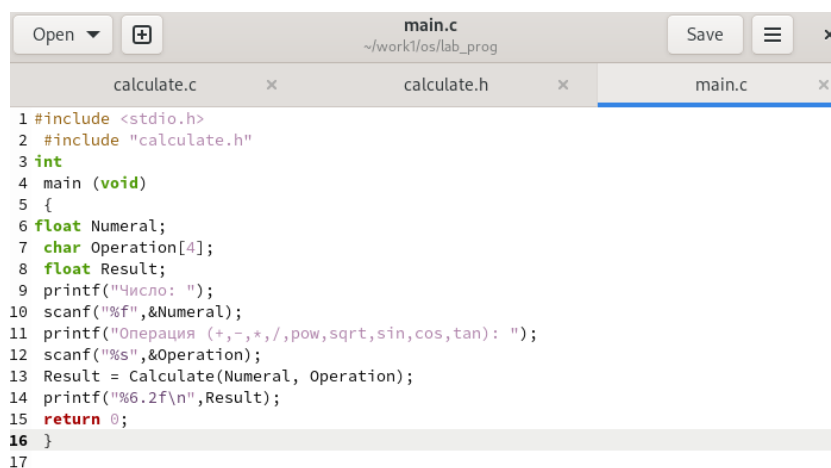
```
calculate.h
~/work1/os/lab_prog

calculate.c
x

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

Рис. 0.4.: рис. 4

Основной файл main.c, реализующий интерфейс пользователя к калькулятору: (рис. [-@fig:005])



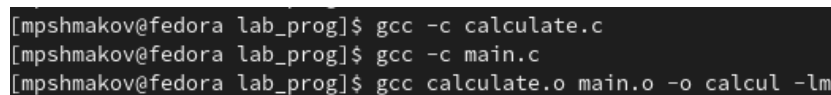
```
main.c
~/work1/os/lab_prog

calculate.c x calculate.h x main.c x

1 #include <stdio.h>
2 #include "calculate.h"
3 int
4 main (void)
5 {
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 }
17
```

Рис. 0.5.: рис. 5

3. Выполните компиляцию программы посредством gcc: (рис. [-@fig:006])



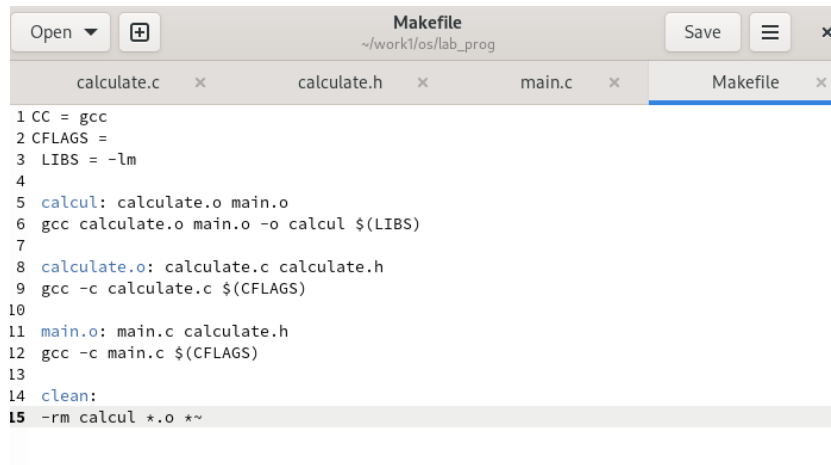
```
[mpshmakov@fedora lab_prog]$ gcc -c calculate.c
[mpshmakov@fedora lab_prog]$ gcc -c main.c
[mpshmakov@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рис. 0.6.: рис. 6

4. При необходимости исправьте синтаксические ошибки.

Ошибок нет.

5. Создайте Makefile со следующим содержанием: (рис. [-@fig:007])

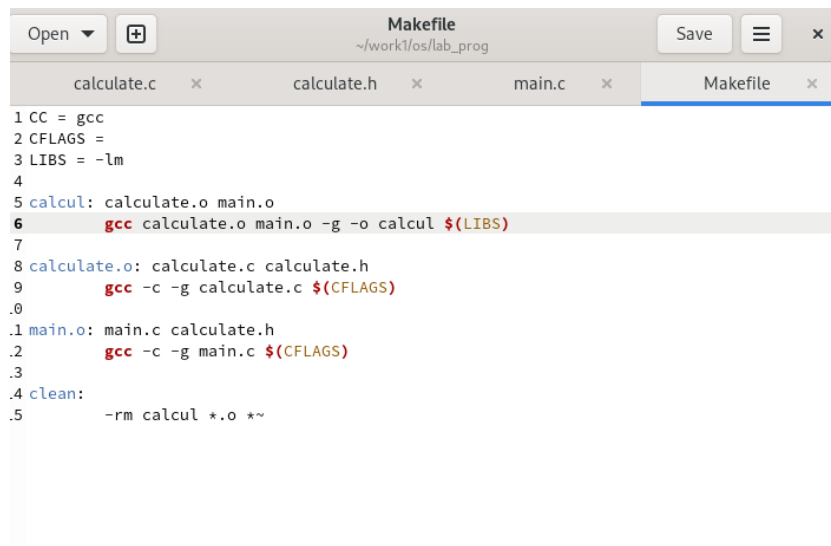


```
1 CC = gcc
2 CFLAGS =
3 LIBS = -lm
4
5 calcul: calculate.o main.o
6 gcc calculate.o main.o -o calcul $(LIBS)
7
8 calculate.o: calculate.c calculate.h
9 gcc -c calculate.c $(CFLAGS)
10
11 main.o: main.c calculate.h
12 gcc -c main.c $(CFLAGS)
13
14 clean:
15 -rm calcul *.o *~
```

Рис. 0.7.: рис. 7

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

Добавил в Makefile параметр -g, который нужен для отладки. (рис. [-@fig:008])



```
1 CC = gcc
2 CFLAGS =
3 LIBS = -lm
4
5 calcul: calculate.o main.o
6 gcc calculate.o main.o -g -o calcul $(LIBS)
7
8 calculate.o: calculate.c calculate.h
9 gcc -c -g calculate.c $(CFLAGS)
10
11 main.o: main.c calculate.h
12 gcc -c -g main.c $(CFLAGS)
13
14 clean:
15 -rm calcul *.o *~
```

Рис. 0.8.: рис. 8

– Запустите отладчик GDB, загрузив в него программу для отладки: (рис. [-@fig:009])

```
[mpshmakov@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-2.fc35
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

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

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /home/mpshmakov/work1/os/lab_prog/./calcul...
(No debugging symbols found in ./calcul)
(gdb)
```

Рис. 0.9.: рис. 9

– Для запуска программы внутри отладчика введите команду run: (рис. [-@fig:010])

```
(gdb) run
Starting program: /home/mpshmakov/work1/os/lab_prog/calcul
Downloading separate debug info for /home/mpshmakov/work1/os/lab_prog/system-supplied DSO at 0x7ffff7fc5000...
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: 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /
Делитель: 3
5.00
[Inferior 1 (process 43756) exited normally]
(gdb)
```

Рис. 0.10.: рис. 10

– Для постраничного (по 9 строк) просмотра исходного код используйте команду list: (рис. [-@fig:011])

```

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

```

Рис. 0.11.: рис. 11

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами 12, 15: (рис. [-@fig:012])

```

(gdb) list 12,15
12     scanf("%s",&Operation);
13     Result = Calculate(Numeral, Operation);
14     printf("%6.2f\n",Result);
15     return 0;
(gdb)

```

Рис. 0.12.: рис. 12

– Для просмотра определённых строк не основного файла используйте list с параметрами calculate.c:20,29: (рис. [-@fig:013])

```
(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. 0.13.: рис. 13

– Установите точку останова в файле calculate.c на строке номер 21: (рис. [-@fig:014])

```
(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.
(gdb)
```

Рис. 0.14.: рис. 14

– Выведите информацию об имеющихся в проекте точка останова: (рис. [-@fig:015])

```
(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1     breakpoint keep y  0x0000000000401247 in Calculate at calculate.c:22
(gdb)
```

Рис. 0.15.: рис. 15

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: (рис. [-@fig:016])

```
(gdb) run
Starting program: /home/mpshmakov/work1/os/lab_prog/calcul

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

Breakpoint 1, Calculate (Numeral=1, Operation=0x7fffffffdf34 "*") at calculate.c:22
22     else if(strncmp(Operation, "*", 1) == 0)
(gdb) backtrace
#0 Calculate (Numeral=1, Operation=0x7fffffffdf34 "*") at calculate.c:22
#1 0x00000000004014eb in main () at main.c:13
(gdb)
```

Рис. 0.16.: рис. 16

– Посмотрите, чему равно на этом этапе значение переменной Numeral с помощью print Numeral и сравните с display Numeral:

Результат вывода 2ух команд отличается, но они обе показывают значение переменной. (рис. [-@fig:017]) (рис. [-@fig:018])

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

Рис. 0.17.: рис. 17

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

Рис. 0.18.: рис. 18

– Уберите точки останова: (рис. [-@fig:019])

```
(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1      breakpoint keep y   0x0000000000401247 in calculate at calculate.c:22
breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Рис. 0.19.: рис. 19

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c. (рис. [-@fig:020]) (рис. [-@fig:021])

```
[mpshmakov@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:4:38: 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:10:2: 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:12:13: 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:12:10: Corresponding format code
main.c:12:2: Return value (type int) ignored: scanf("%s", &Ope...
Finished checking --- 4 code warnings
```

Рис. 0.20.: рис. 20



```

[mpshmakov@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:4:38: 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:13:2: 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:19:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:5: 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:35:8: Return value type double does not match declared type float:
    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:43:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:8: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:47:8: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:49:8: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:51:8: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:53:8: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:57:8: Return value type double does not match declared type float:
    (HUGE_VAL)

Finished checking --- 15 code warnings

```

Рис. 0.21.: рис. 21

## Выводы

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

# Контрольные вопросы

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

Чтобы получить информацию о возможностях этих программ нужно прописать в консоль `man gcc`, `man make` итд...

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

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. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

Суффикс это составная часть имени файла. Система сборки каких-либо программ (например язык java) требует, чтобы имена файлов исходного кода заканчивались на . java. 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.

#### 4. Каково основное назначение компилятора языка C в UNIX?

Компилятор — программа, переводящая написанный на языке программирования текст в набор машинных кодов.

#### 5. Для чего предназначена утилита make?

Для компиляции файлов calculate.c, calculate.h и main.c.

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

Общий синтаксис Makefile имеет вид: 1 target1 [target2...]:[:] [dependment1...] 2 [(tab)commands] [#commentary] 3 [(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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Основным свойством присущим всем программам отладки является возможность ставить точки останова в программе. Для того, чтобы поставить точку останова в отладчике gdb нужно прописать break и сточку кода, где нужно остановить исполнение программы.

8. Назовите и дайте основную характеристику основным командам 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 по шагово, 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.

– Запустите отладчик GDB, загрузив в него программу для отладки:

`gdb ./calcul`

– Для запуска программы внутри отладчика введите команду run:

`run`

– Для постраничного (по 9 строк) просмотра исходного кода используйте команду list:

`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  
0x000000000400b2b 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
```

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

Выводит в терминал место, где была допущена ошибка.

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

- 1) Знание языка программирования исходного кода.
- 2) Программа splint может помочь понять ошибки в коде.

12. Каковы основные задачи, решаемые программой splint?

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.