

Лабораторная работа №14

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

Ким Михаил Алексеевич

Содержание

1	Цель работы	3
2	Выполнение лабораторной работы.	4
3	Выводы	23
4	Термины	24

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.

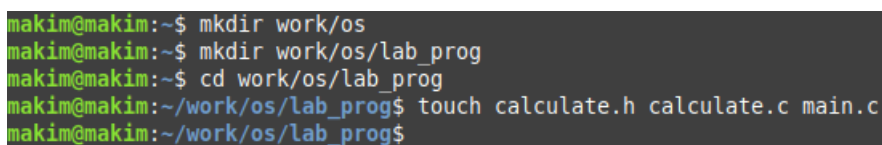
2. Выполнение лабораторной работы.

1. В домашнем каталоге создаём подкаталог `~/work/os/lab_prog`. (рис. 2.1)

```
mkdir -p ~/work/os/lab_prog  
/либо/  
mkdir work/os  
mkdir 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. (рис. 2.1)

```
cd ~/work/os/lab_prog  
touch calculate.h calculate.c main.c
```



```
makim@makim:~$ mkdir work/os  
makim@makim:~$ mkdir work/os/lab_prog  
makim@makim:~$ cd work/os/lab_prog  
makim@makim:~/work/os/lab_prog$ touch calculate.h calculate.c main.c  
makim@makim:~/work/os/lab_prog$
```

Рис. 2.1: Подготовка рабочей среды

3. Реализуем функций калькулятора в файле `calculate.c`. (рис. 2.2)

```

// calculate.c

// подключаем необходимые библиотеки и заголовчный файл
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

// объявляем функцию типа float,
// которая будет возвращать результат вычисления
// функция принимает значение float (первое число)
// и массив char (операция)
float Calculate(float Numeral, char Operation[4])
{
    // создаем переменную типа float,
    // в которой будет храниться второе число
    float SecondNumeral;

    // используем ветвления, чтобы определить операцию
    // strcmp - сравнение строк с ограничением количества сравниваемых симв
    // strcmp(const char *str1, const char *str2, size_t n)
    // str1, str2 - указатели на сравниваемые строки.
    // size_t n - количество символов для сравнения.
    // Возвращаемое значение: 0 - если первые n символов сравниваемых строк

    // в зависимости от операции, мы попадаем в одно из ветвлений и возвраща
    // если необходимо просим ввести второе число

    // по такому принципу пишется все нижеперечисленное

```

```

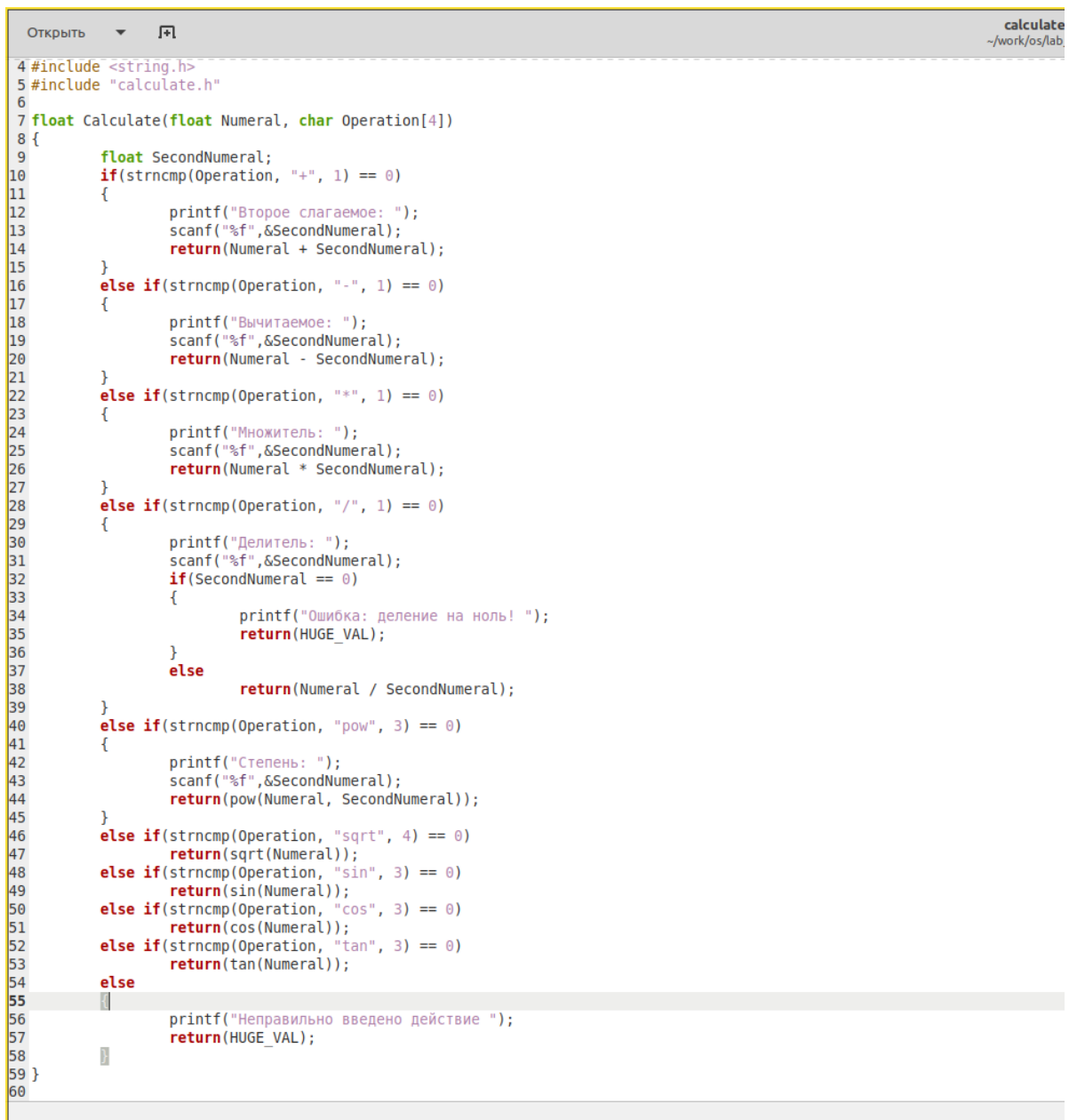
if (strncmp(Operation, "+", 1) == 0)
{
    printf("Второе слагаемое: ");
    scanf("%f", &SecondNumeral);
    return (Numeral + SecondNumeral);
}
else if (strncmp(Operation, "-", 1) == 0)
{
    printf("Вычитаемое: ");
    scanf("%f", &SecondNumeral);
    return (Numeral - SecondNumeral);
}
else if (strncmp(Operation, "*", 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));
    }
    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);
    }
}

```



```
4 #include <string.h>
5 #include "calculate.h"
6
7 float 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    else if(strncmp(Operation, "sin", 3) == 0)
49        return(sin(Numeral));
50    else if(strncmp(Operation, "cos", 3) == 0)
51        return(cos(Numeral));
52    else if(strncmp(Operation, "tan", 3) == 0)
53        return(tan(Numeral));
54    else
55    {
56        printf("Неправильно введено действие ");
57        return(HUGE_VAL);
58    }
59 }
60
```

Рис. 2.2: Исходный код calculate.c

4. Реализуем заголовочный файл calculate.h, описывающий формат вызова функции калькулятора. (рис. 2.3)

```
// calculate.h
```

```
// Директива #ifndef проверяет, определено ли имя CALCULATE_H,
```



```
// если нет, то управление передаётся директиве #define
// и определяется интерфейс класса. Если же имя CALCULATE_H_
// уже определено, управление передаётся директиве #endif.
// Таким образом, исключается возможность многократного
// определения класса CALCULATE_H_.

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif/*CALCULATE_H_*/
```

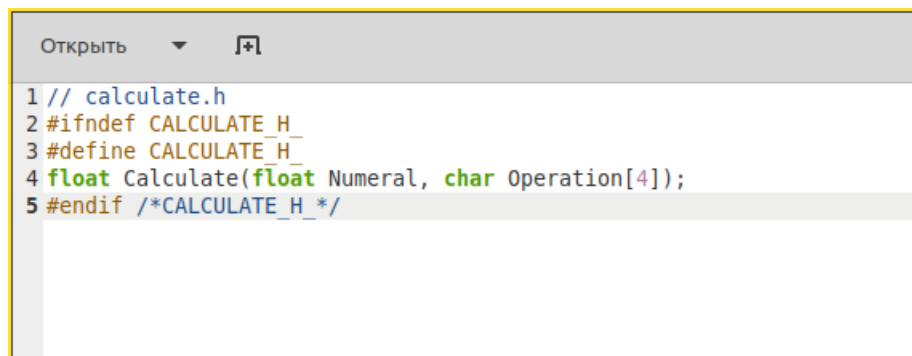


Рис. 2.3: Исходный код calculate.h

5. Пишем основной файл main.c, реализующий интерфейс пользователя к калькулятору. (рис. 2.4)

```
// main.c

// подключаем библиотеки
#include <stdio.h>
#include "calculate.h"
```

```

int main (void){
    // объявляем переменную, хранящую первое число
    float Numeral;

    // объявляем массив, где будет храниться операция
    char Operation[4];

    // объявляем переменную, где будет храниться результат
    float Result;

    // просим ввести и принимаем первое число
    printf("Число: ");
    scanf("%f",&Numeral);

    // просим ввести и принимаем операцию
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",&Operation[0]);

    // записываем в Result возвращенное значение из Calculate
    Result = Calculate(Numeral, Operation);

    // выводим результат
    printf("%.2f\n",Result);

    return 0;
}

```



```
1 // main.c
2 #include <stdio.h>
3 #include "calculate.h"
4
5 int main (void)
6 {
7     float Numeral;
8     char Operation[4];
9     float Result;
10    printf("Число: ");
11    scanf("%f",&Numeral);
12    printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
13    scanf("%s",&Operation);
14    Result = Calculate(Numeral, Operation);
15    printf("%6.2f\\n",Result);
16    return 0;
17 }
18
```

Рис. 2.4: Исходный код main.c

6. Выполняем компиляцию программы посредством gcc и проверяем работу калькулятора. (рис. 2.5)

```
gcc -c calculate.c
```

```
gcc -c main.c
```

```
gcc calculate.o main.o -o calcul -lm
```

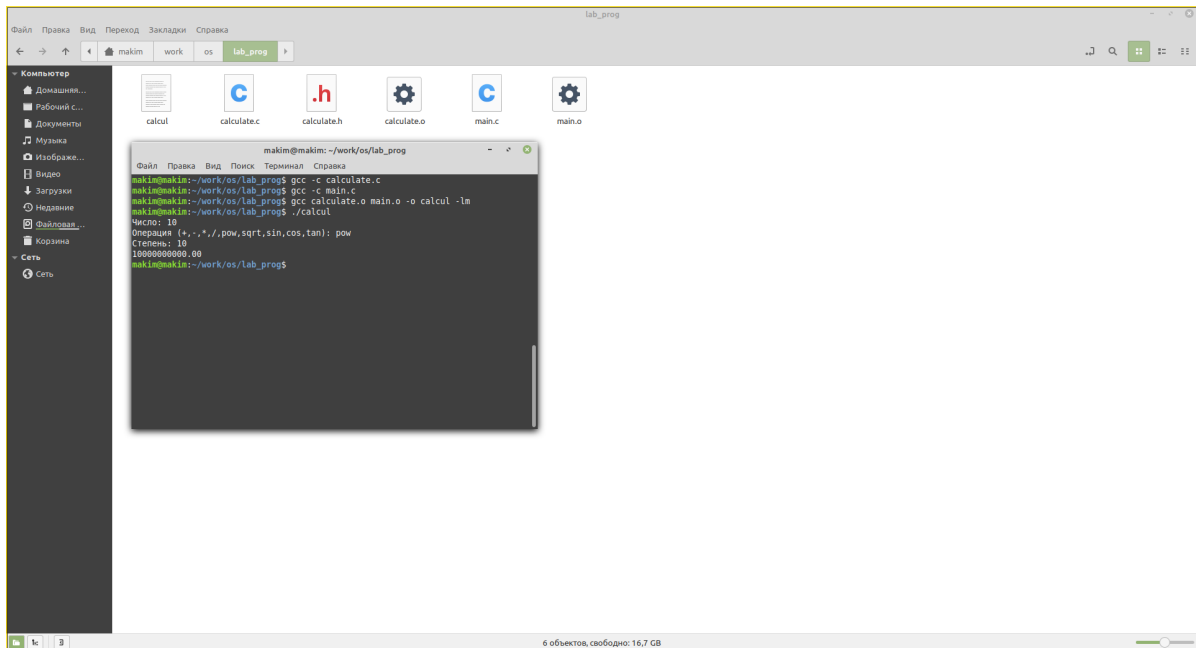


Рис. 2.5: Компиляция

7. Создаём Makefile. Для создания используем образец из работы. Попутно исправляем ошибки в образце, которые препятствуют оптимальной работе отладчика. (рис. 2.6)

```
#
```

```
# Makefile
```

```
#
```

```
CC = gcc
```

```
CFLAGS =
```

```
LIBS = -lm
```

```
// цель calcul - отвечает за создание исполняемого файла calcul
```

```
// на основе calculate.o main.o (зависит от этих файлов)
```

```
calcul: calculate.o main.o
```

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

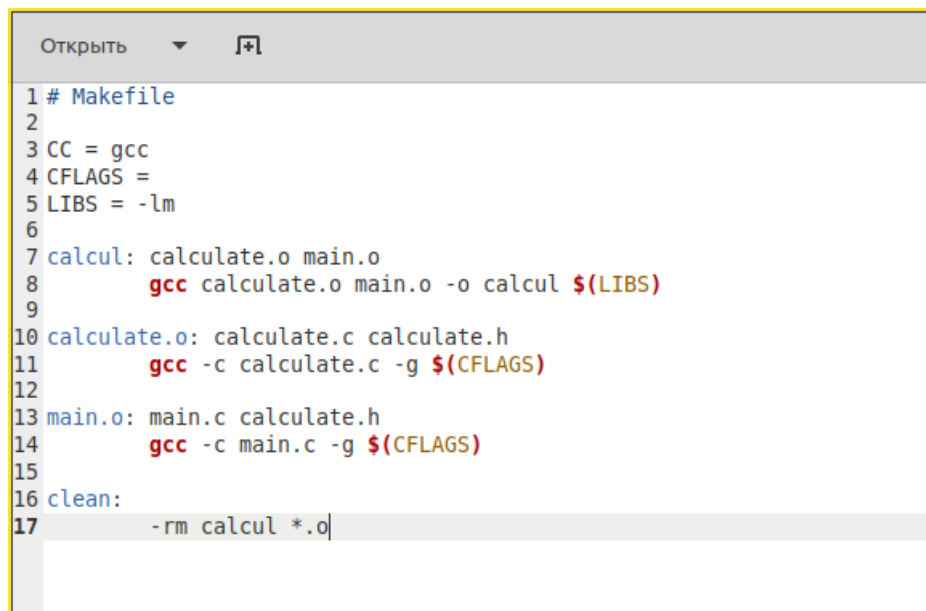
```

// calculate.o - отвечает за создание объектного файла calculate.o
// на основе calculate.c
// не забываем добавить опцию -g, чтобы отладочная информация содержалась в
calculate.o: calculate.c calculate.h
    gcc -c calculate.c -g $(CFLAGS)

// main.o - отвечает за создание объектного файла main.o
// на основе main.c
// не забываем добавить опцию -g, чтобы отладочная информация содержалась в
main.o: main.c calculate.h
    gcc -c main.c -g $(CFLAGS)

// clean отвечает за удаление объектных файлов и самого исполняемого файла
clean:
    -rm calcul *.o

```



```

Открыть ▼ [F2]
1 # Makefile
2
3 CC = gcc
4 CFLAGS =
5 LIBS = -lm
6
7 calcul: calculate.o main.o
8     gcc calculate.o main.o -o calcul $(LIBS)
9
10 calculate.o: calculate.c calculate.h
11     gcc -c calculate.c -g $(CFLAGS)
12
13 main.o: main.c calculate.h
14     gcc -c main.c -g $(CFLAGS)
15
16 clean:
17     -rm calcul *.o

```

Рис. 2.6: Makefile

8. Используем Makefile. (рис. 2.7-2.9)

Компиляция по-отдельности:

```
make calculate.o
```

```
make main.o
```

```
make calcul
```

Полная компиляция сразу:

```
make
```

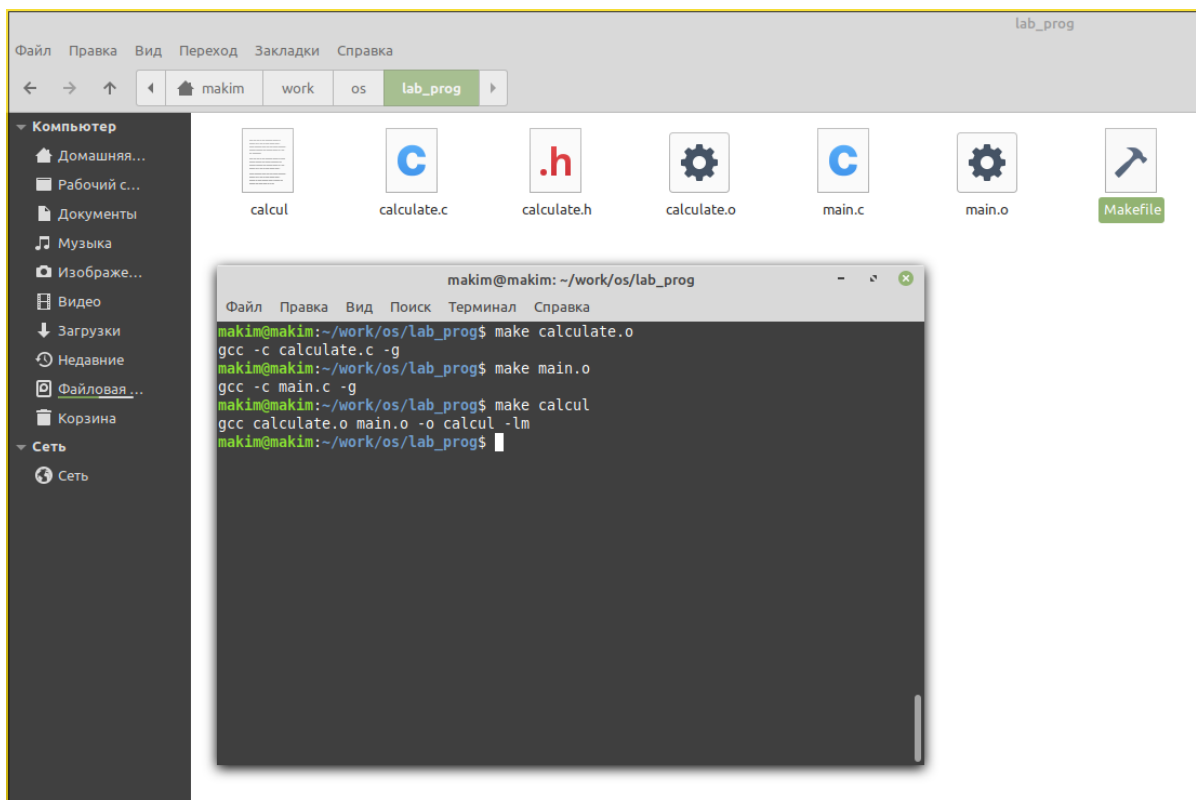


Рис. 2.7: Использование Makefile

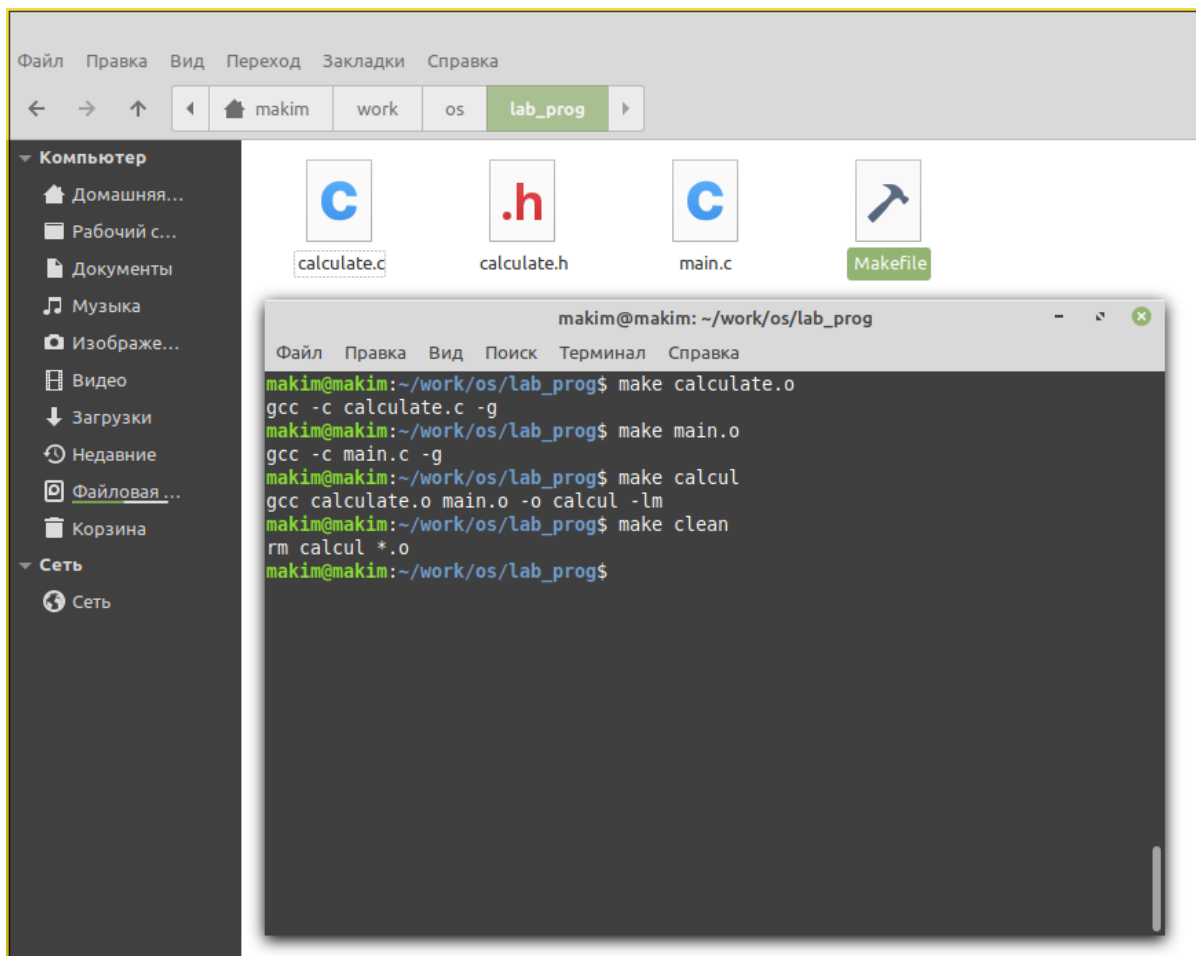


Рис. 2.8: Использование Makefile

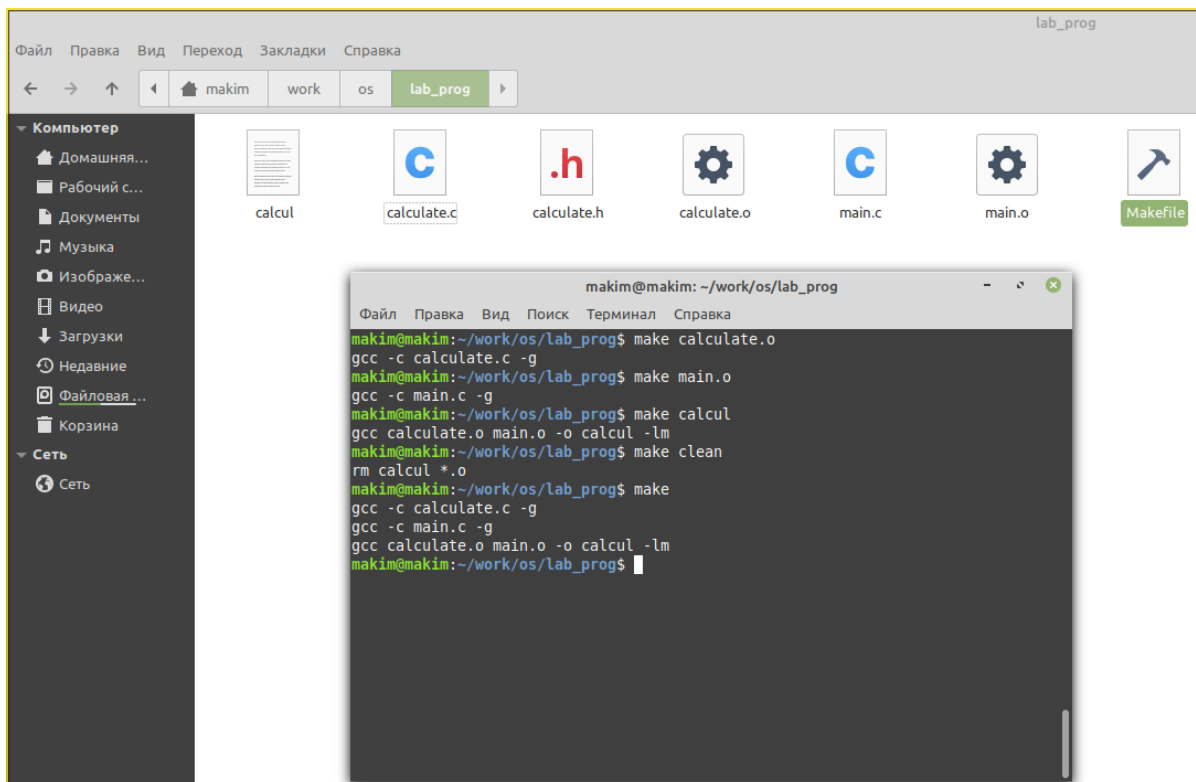


Рис. 2.9: Использование Makefile

9. Запускаем отладчик GDB, загрузив в него программу для отладки. (рис. 2.10)

```
gdb ./calcul
```

10. Для запуска программы внутри отладчика вводим команду run. (рис. 2.10)

```
run
```

11. Для постраничного (по 9 строк) просмотра исходного код используем команду list. (рис. 2.10)

```
list
```

12. Для просмотра строк с 12 по 15 основного файла используем list с параметрами. (рис. 2.10)


```
list 12,15
```

13. Для просмотра определённых строк не основного файла используем list с параметрами. (рис. 2.10)

```
list calculate.c:20,29
```

```

makim@makim:~/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) run
Starting program: /home/makim/work/os/lab_prog/calcul
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 12
24.00
[Inferior 1 (process 4525) exited normally]
(gdb) list
1      // main.c
2      #include <stdio.h>
3      #include "calculate.h"
4
5      int main (void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Число: ");
(gdb) list 12,15
12         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13         scanf("%s",Operation);
14         Result = Calculate(Numeral, Operation);
15         printf("%6.2f\n",Result);
(gdb) list calculate.c:20,29
20         return(Numeral - SecondNumeral);
21     }
22     else if(strncmp(Operation, "*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strncmp(Operation, "/", 1) == 0)
29     {
(gdb)

```

Рис. 2.10: Отладчик GDB

14. Устанавливаем точку остановки в файле calculate.c на строке номер 22, чтобы программа остановилась, после ввода операции вычитания. (рис.

2.11)

```
list calculate.c:15,25  
break 18
```

15. Выводим информацию об имеющихся в проекте точка останова. (рис. 2.11)

```
info breakpoints
```

16. Запускаем программу внутри отладчика и убеждаемся, что программа остановилась в момент прохождения точки останова. (рис. 2.11)

```
run  
5  
-
```

17. Используем команду `backtrace`, которая показывает весь стек вызываемых функций от начала программы до текущего места. (рис. 2.11)

```
backtrace
```

18. Просматриваем, чему равно на этом этапе значение переменной `Numeral`. (рис. 2.11)

```
print Numeral
```

19. Сравниваем с результатом вывода на экран после использования команды `display`. (рис. 2.11)

```
display Numeral
```

20. Убираем точки останова. (рис. 2.11)

info breakpoints

delete 2

```
(gdb) list calculate.c:20,29
20         return(Numeral - SecondNumeral);
21     }
22     else if(strncmp(Operation, "*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strncmp(Operation, "/", 1) == 0)
29     {
(gdb) list calculate.c:15,25
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);
(gdb) break 18
Breakpoint 1 at 0x555555552dd: file calculate.c, line 18.
(gdb) info breakpoints
Num  Type      Disp Enb Address          What
1     breakpoint keep y   0x0000555555552dd in Calculate at calculate.c:18
(gdb) run
Starting program: /home/makim/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:18
18         printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:18
#1 0x0000555555555bd in main () at main.c:14
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num  Type      Disp Enb Address          What
1     breakpoint keep y   0x0000555555552dd in Calculate at calculate.c:18
      breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

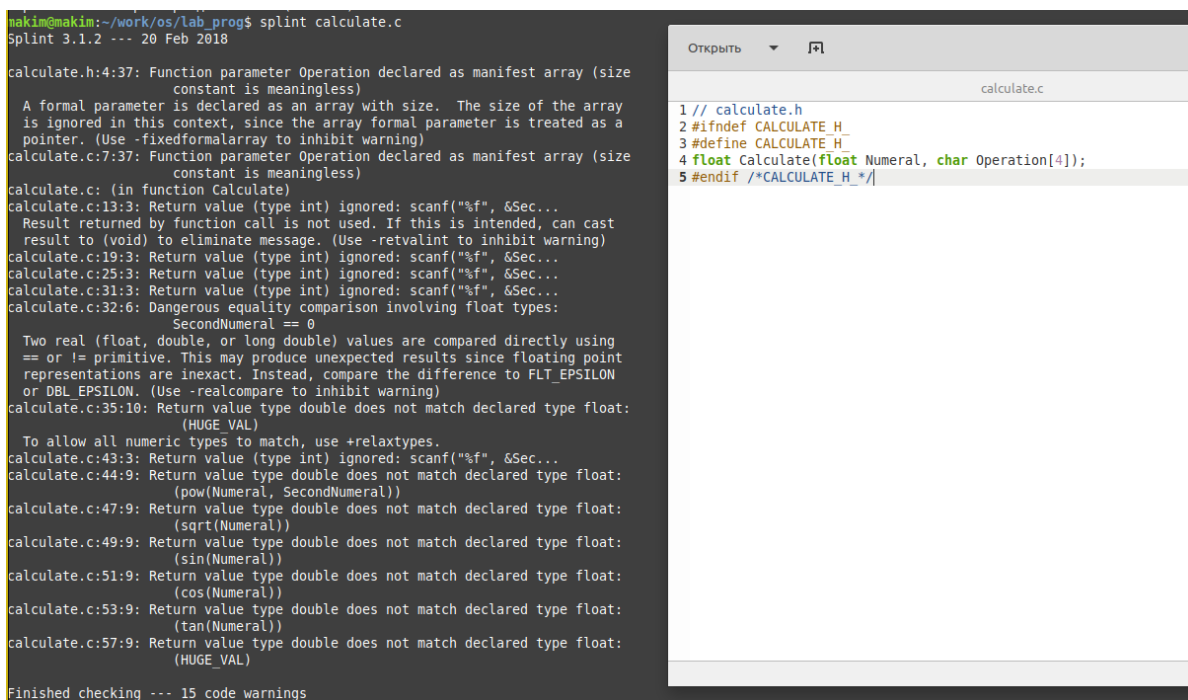
Рис. 2.11: Отладчик GDB

21. С помощью утилиты splint пробуем проанализировать коды файлов calculate.c и main.c. Найдены незначительные ошибки, не влияющие на работоспособность кода. Например, утилита предлагает посылать не весь массив Operation в функцию Calculate, а только указатель

на него. Это не является ошибкой, но оптимизирует код. Также splint предупреждает, что опасно однозначно сравнивать числа с плавающей точкой и целочисленные числа, опять же хорошее замечание, но на работу сильно не влияет. В общем, splint не нашел серьёзных ошибок, что, в целом, логично, поскольку код работает. (рис. 2.12 - 2.14)

splint calculate.c

splint main.c



```
makim@makim:~/work/os/lab_prog$ splint calculate.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)
calculate.c:7:37: Function parameter Operation declared as manifest array (size
constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:13:3: 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:19:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:6: 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:43:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:9: Return value type double does not match declared type float:
  (pow(Numeral, SecondNumeral))
calculate.c:47:9: Return value type double does not match declared type float:
  (sqrt(Numeral))
calculate.c:49:9: Return value type double does not match declared type float:
  (sin(Numeral))
calculate.c:51:9: Return value type double does not match declared type float:
  (cos(Numeral))
calculate.c:53:9: Return value type double does not match declared type float:
  (tan(Numeral))
calculate.c:57:9: Return value type double does not match declared type float:
  (HUGE_VAL)
Finished checking --- 15 code warnings
```

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

Рис. 2.12: Предупреждения splint (calculate.c)

```

makeim@makeim: ~/work/os/lab_prog
Необходимо скачать 740 КБ архивов
После данной операции объем занятого дискового пространства возрастет на 2 883 КБ.
Хотите продолжить? [Y/n] y
Yon:1 http://mirror.docker.ru/ubuntu focal/universe amd64 splint-data all 1:3.1.2-1
Yon:2 http://mirror.docker.ru/ubuntu focal/universe amd64 splint amd64 1:3.1.2+dfsg-1
Получено 740 КБ за 0с (4 150 КБ/с)
Выбор ранее не выбранного пакета splint-data.
(Имена базы данных -- на данный момент установлено 370947 файлов и каталогов.)
Подготовка к распаковке ./splint-data 1:3.1.2+dfsg-1build1 all.deb --
Распаковывается splint-data (1:3.1.2+dfsg-1build1) --
Выбор ранее не выбранного пакета splint.
Подготовка к распаковке ./splint 1:3.1.2+dfsg-1build1 amd64.deb --
Распаковывается splint (1:3.1.2+dfsg-1build1) --
Настраивается пакет splint-data (1:3.1.2+dfsg-1build1) --
Настраивается пакет splint (1:3.1.2+dfsg-1build1) --
Оборачивается триггером для man-db (2:9.1-1) --
makeim@makeim:~/work/os/lab_prog$ splint calculate.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)
calculate.c:7:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function calculate)
calculate.c:13:13: 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:19:13: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:13: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:13: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:16: 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:35:18: Return value type double does not match declared type float:
        (HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:43:13: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:19: Return value type double does not match declared type float:
        (pow(Numeral, -SecondNumeral))
calculate.c:47:19: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:49:19: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:51:19: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:53:19: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:57:19: Return value type double does not match declared type float:
        (HUGE_VAL)
Finished checking --- 15 code warnings

7 float Calculate(float Numeral, char Operation[])
8 {
9     float SecondNumeral;
10    if(strlen(Operation, "+") == 0)
11    {
12        printf("Введите слагаемое: ");
13        scanf("%f", &SecondNumeral);
14        return(Numeral + SecondNumeral);
15    }
16    else if(strlen(Operation, "-") == 0)
17    {
18        printf("Введите вычитаемое: ");
19        scanf("%f", &SecondNumeral);
20        return(Numeral - SecondNumeral);
21    }
22    else if(strlen(Operation, "*") == 0)
23    {
24        printf("Введите множитель: ");
25        scanf("%f", &SecondNumeral);
26        return(Numeral * SecondNumeral);
27    }
28    else if(strlen(Operation, "/") == 0)
29    {
30        printf("Введите 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(strlen(Operation, "pow", 3) == 0)
41    {
42        printf("Введите степень: ");
43        scanf("%f", &SecondNumeral);
44        return(pow(Numeral, SecondNumeral));
45    }
46    else if(strlen(Operation, "sqrt", 4) == 0)
47    {
48        return(sqrt(Numeral));
49    }
50    else if(strlen(Operation, "sin", 3) == 0)
51    {
52        return(sin(Numeral));
53    }
54    else if(strlen(Operation, "cos", 3) == 0)
55    {
56        return(cos(Numeral));
57    }
58    else if(strlen(Operation, "tan", 3) == 0)
59    {
60        return(tan(Numeral));
61    }
62    printf("Некорректно введено действие ");
63    return(HUGE_VAL);
64 }

```

Рис. 2.13: Предупреждения splint (main.c)

```

makeim@makeim:~/work/os/lab_prog$ 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:11:12: 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:13:12: Return value (type int) ignored: scanf("%s", Oper...
Finished checking --- 3 code warnings
makeim@makeim:~/work/os/lab_prog$

2 #include <stdio.h>
3 #include "calculate.h"
4
5 int main (void)
6 {
7     float Numeral;
8     char Operation[4];
9     float Result;
10    printf("Число: ");
11    scanf("%f", &Numeral);
12    printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
13    scanf("%s", Operation);
14    Result = Calculate(Numeral, Operation);
15    printf("%6.2f\n", Result);
16    return 0;
17 }

```

Рис. 2.14: Предупреждения splint

3. Выводы

Мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux. Закрепили знания, полученные в прошлых работах. Создали на языке программирования С калькулятор с простейшими функциями и разобрали на нем основные навыки отладки.

4. Термины

- GCC (GNU Compiler Collection) - это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.).
- GDB (GNU Debugger) - отладчик для поиска и устранения ошибок в программе. Входит в комплект программ GNU для ОС типа UNIX.
- Утилита make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
- Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.
- Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.
- POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
- Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным.

- Каталог, он же директория, (от английского Directory) – это объект в ФС (файловой системе), необходимый для того, чтобы упростить работу с файлами.
- Домашний каталог - каталог, предназначенный для хранения собственных данных пользователя Linux. Как правило, является текущим непосредственно после регистрации пользователя в системе.
- Команда - записанный по специальным правилам текст (возможно с аргументами), представляющий собой указание на выполнение какой-либо функций (или действий) в операционной системе.