

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №14

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

дисциплина: Операционные системы

Студентка:

Бочкарева Елена Дмитриевна

Студенческий билет номер №: 1032207514

Группа:

НПМбв-01-19

МОСКВА

2023

Оглавление

11.1. Цель работы	3
11.1.1. Запускаю операционную систему (рис.1).....	3
11.1.2. Вхожу от имени пользователя edbochkareva. Ввожу пароль (рис.2).....	3
11.3. Последовательность выполнения работы.....	4
11.3.8. Описание и комментарии по созданию кода в соответствии с заданием лабораторной работы:	4
11.5. Ответы на контрольные вопросы.....	20
11.5.1. Как получить информацию о возможностях программ gcc, make, gdb и др.?	20
11.5.2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.	21
11.5.3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.	21
11.5.4. Каково основное назначение компилятора языка C в UNIX?	22
11.5.5. Для чего предназначена утилита make?.....	23
11.5.6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.	23
11.5.7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?	24
11.5.8. Назовите и дайте основную характеристику основным командам отладчика gdb.....	24
11.5.9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.	25
11.5.10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.....	26
11.5.11. Назовите основные средства, повышающие понимание исходного кода программы.	26
11.5.12. Каковы основные задачи, решаемые программой splint.....	28
Примеры команды Split в Linux	28
1. Разделите файлы на несколько файлов	28
2. Разделите файлы на несколько файлов с определенными номерами строк.....	29
3. Разделение файлов на n файлов	29
5. Разделите и укажите длину суффикса	29
6. Разделить с помощью числового суффикса заказа	29
7. Добавьте шестнадцатеричные суффиксы для разделения файлов	29
8. Разделите файлы на несколько файлов определенного размера.....	30
9. Разбивка файлов на несколько с определенным размером файла	30
Выводы, согласованные с целью	30

11.1. 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.

11.1.1. Запускаю операционную систему (рис.1)

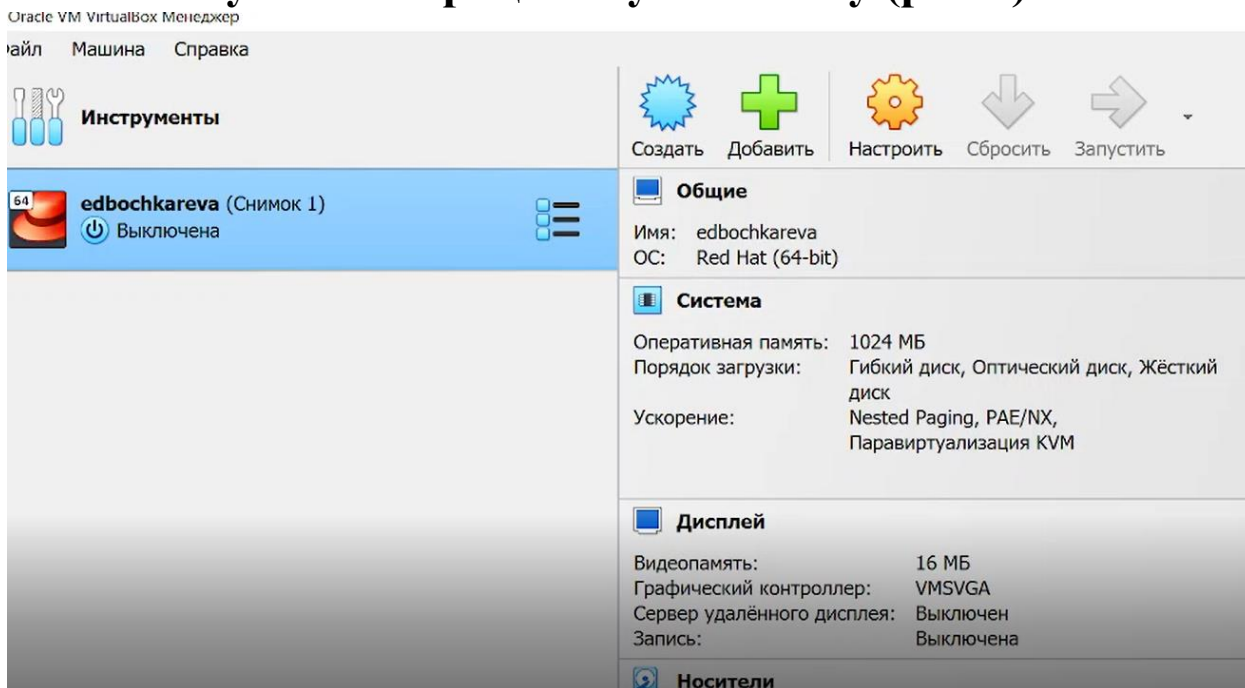
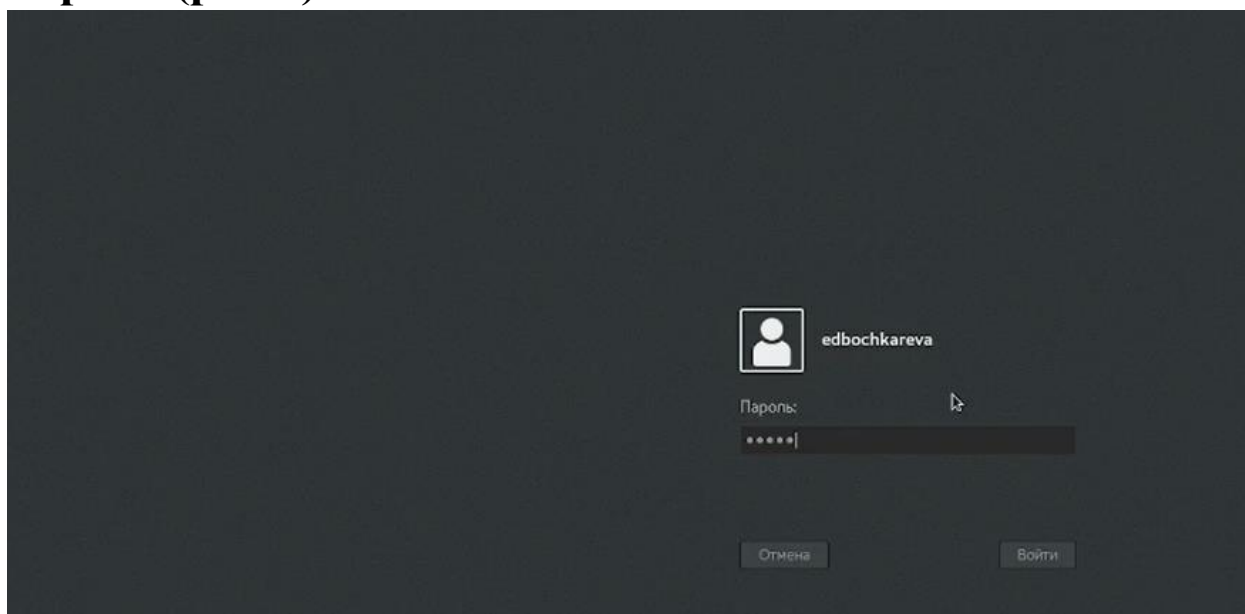


Рис.1: Рисунок 1

11.1.2. Вхожу от имени пользователя edbochkareva. Ввожу пароль (рис.2).



11.3. Последовательность выполнения работы

11.3.1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.

11.3.2. 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. Кулябов Д. С. и др.

11.3.3. Выполните компиляцию программы посредством `gcc`:
`gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm`

11.3.4. При необходимости исправьте синтаксические ошибки.

11.3.5. Создайте `Makefile` со следующим содержанием: Кулябов Д. С. и др.

11.3.6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`)

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

11.3.8. Описание и комментарии по созданию кода в соответствии с заданием лабораторной работы:

1. В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`.
2. В каталоге создадим файлы `calculate.c`, `calculate.h`, `main.c`
3. Выполним компиляцию программы с помощью `gcc`.

4. Создадим Makefile (рисунок 5). Makefile описывает правила обработки файлов и необходим для отслеживания связей между файлами. Далее следует описание созданного Makefile:
 - в начале файла указаны три переменные, значения которых используются в командах ниже.
 - первая цель - `calcul`, она зависит от файлов `calculate.o` и `main.o`.
Ниже со знака табуляции приведена команда для компиляции `calcul`.
 - в свою очередь цель `calculate.o` зависит от `calculate.c` и `calculate.h`. Ниже также указана команда для компиляции цели.
 - цель `main.o` зависит от файлов `main.c` и `calculate.h`
 - в последнюю очередь указана команда для удаления файлов, связанных с программой.
5. Выполним отладку скомпилированной программы `calcul` с помощью команды `gdb`.
6. Запустим отладчик с помощью команды `gdb ./calcul`.
7. Запустим программу для отладки при помощи команды `run`
8. Посмотрим постраничный вывод исходного кода программы `calcul`
9. Выведем на экран исходный код с 12 по 15 строку - команда `list 12,15`
10. Просмотрим код файла `calculate.c` - `list calculate.c:20,29`
11. Установим точку остановки в файле `calculate.c` на строке 21 с помощью команды `break 21`
12. Выведем информацию об имеющихся точках остановки - команда `info breakpoints`

13. Запустим программу на отладку, чтобы убедиться, что точка остановки сработала.
14. С помощью команды `backtrace` увидим какие функции вызывались от начала и до текущего места программы.
15. Выведем значение переменной `Numeral` с помощью команды `print`
16. Выведем значение переменной `Numeral` с помощью команды `display`
17. Уберем точки остановки
18. С помощью команды `splint` проанализируем код файлов `calculate.c` и `main.c` Утилита покажет предупреждающие сообщения о спорных местах в коде и варианты их решения.

В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`.

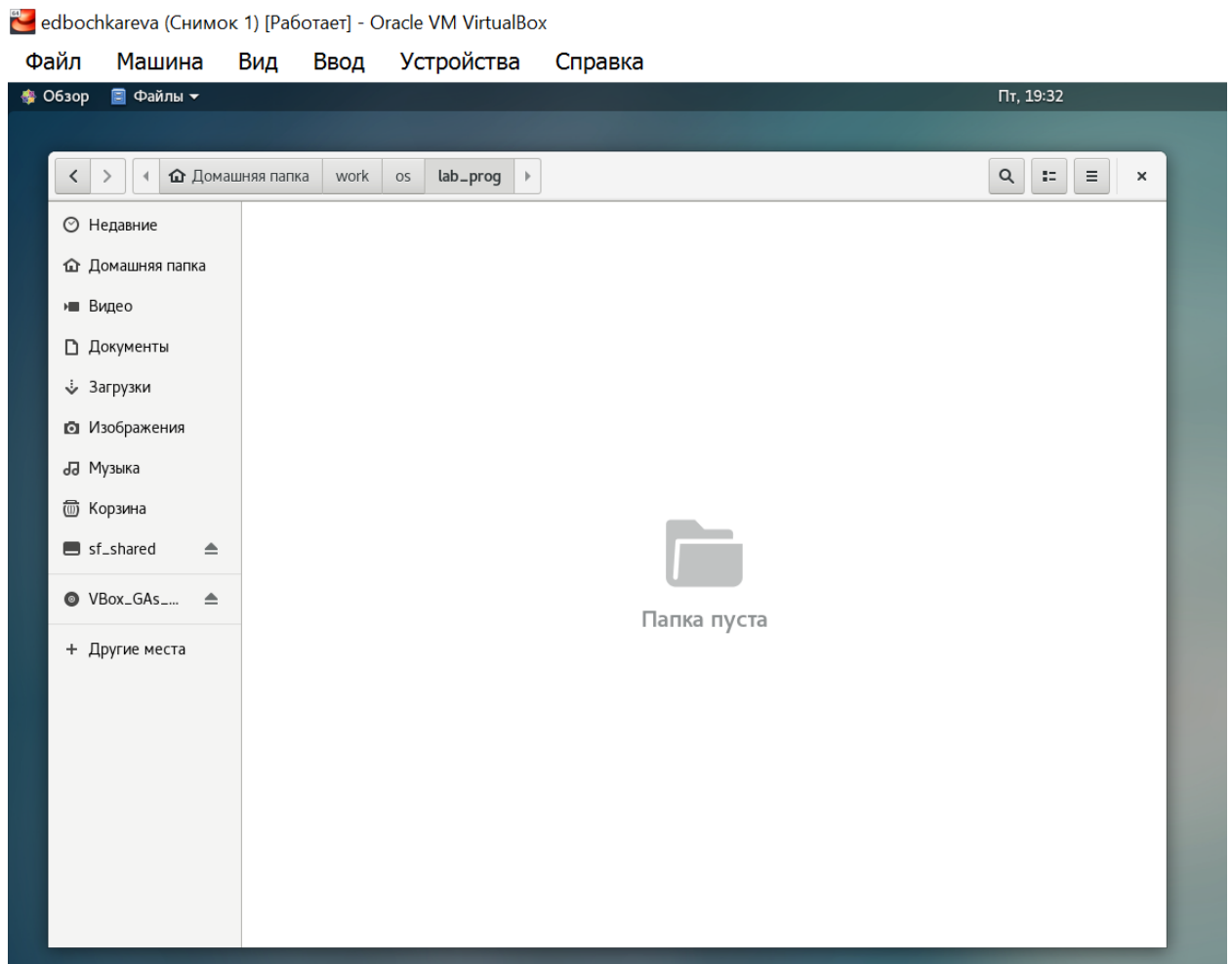


Рис.3: Рисунок 3

Запускаю emacs

```
[edbochkareva@edbochkareva lab_prog]$ emacs
```

Рис.4: Рисунок 4

Открываю редактор emacs



Рис.5: Рисунок 5

Выполняю компиляцию программы с помощью gcc

```
[root@edbochkareva edbochkareva]# cd ./work/os/lab_prog
[root@edbochkareva ] lab_prog]# gcc -c calculate.c -g
[root@edbochkareva ] lab_prog]# gcc -c main.c -g
[root@edbochkareva ] lab_prog]# gcc calculate.c main.o -o calcul -lm
[root@edbochkareva ] lab_prog]#
```

Рис.6: Рисунок 6

Создаю файл calculate.h

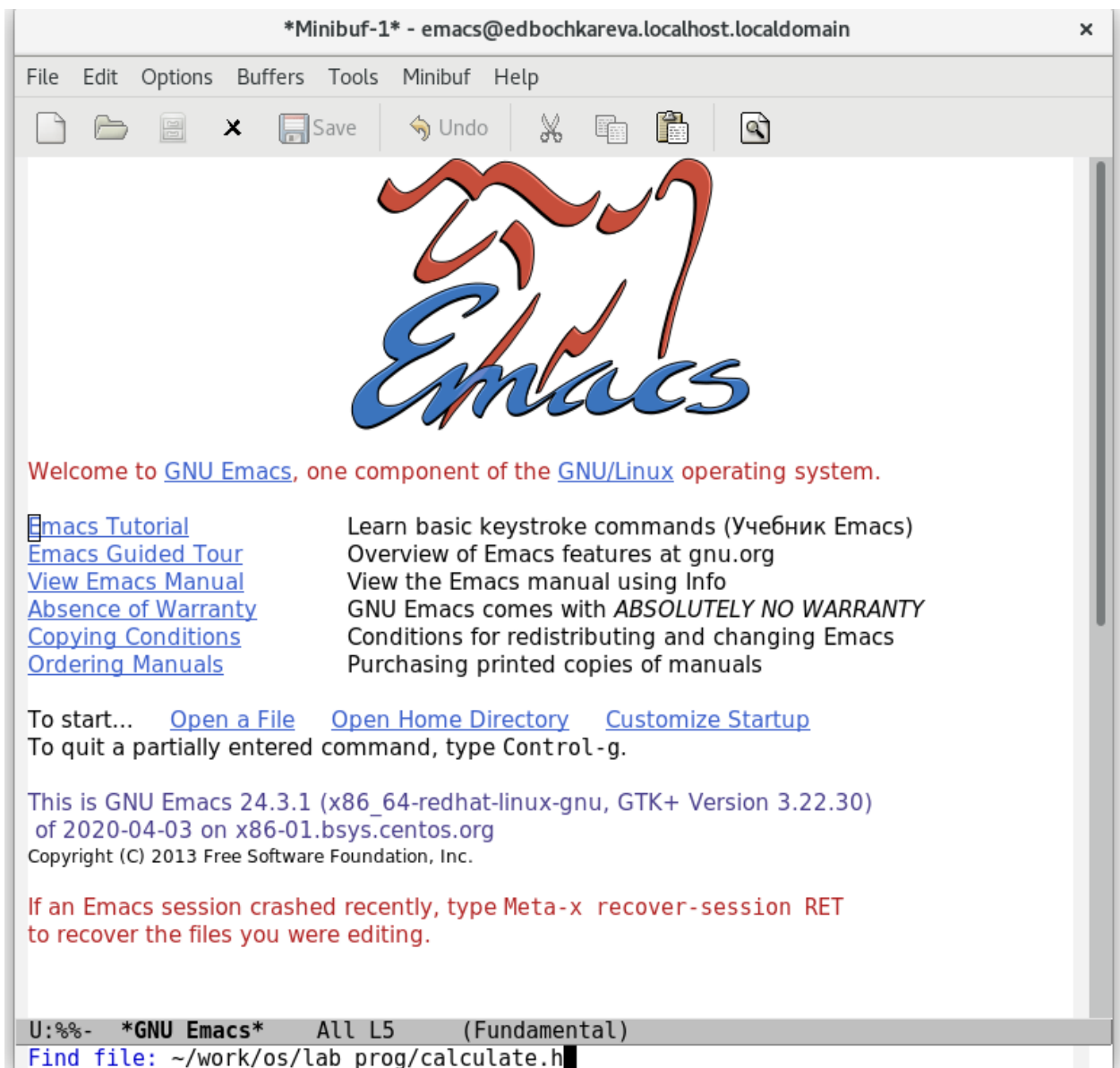
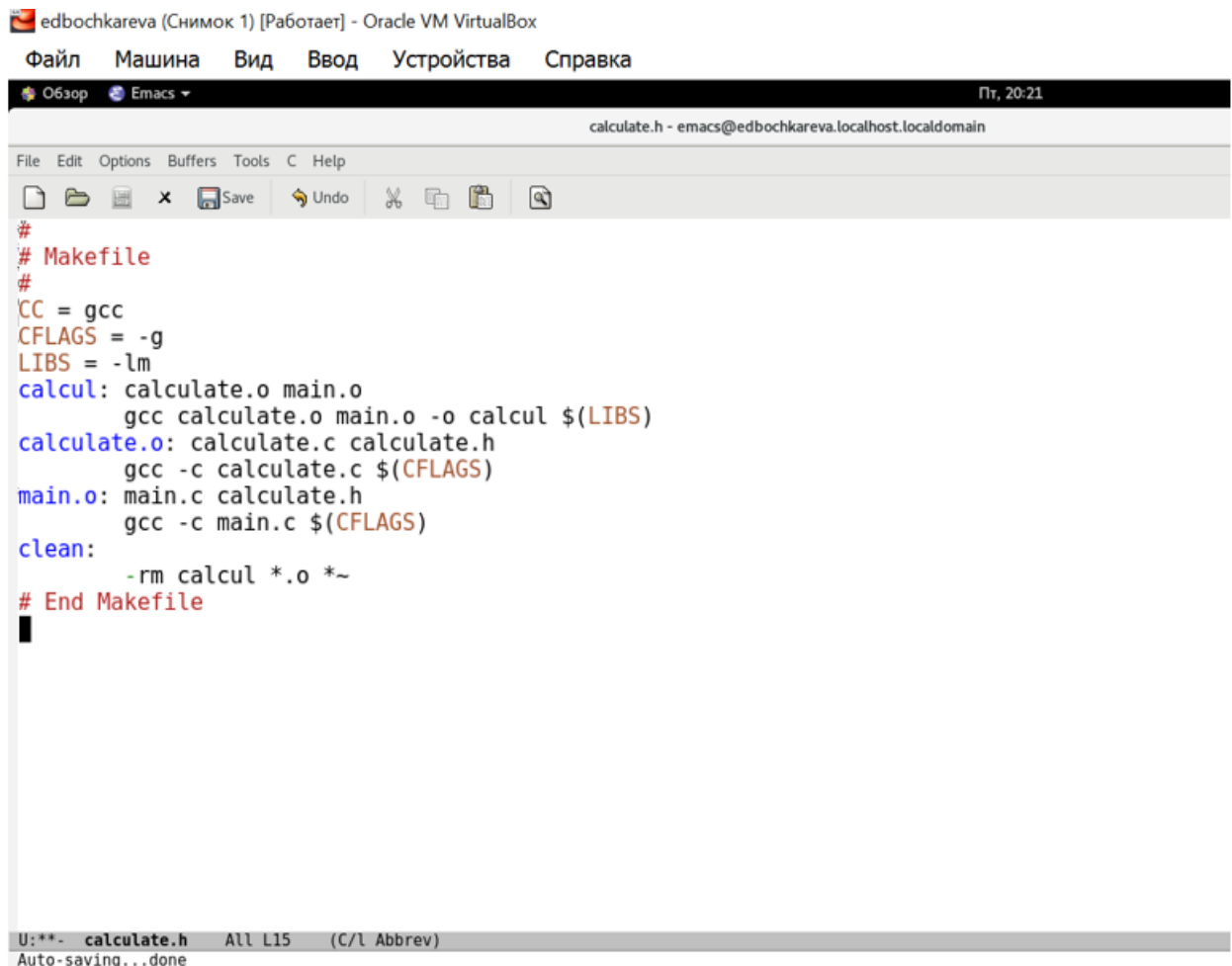


Рис.7: Рисунок 7

Создаю Makefile. Makefile описывает правила обработки файлов и необходим для отслеживания связей между файлами.

Описание созданного Makefile:

- в начале файла указаны три переменные, значения которых используются в командах ниже.
- первая цель - `calcul`, она зависит от файлов `calculate.o` и `main.o`.
- ниже со знака табуляции приведена команда для компиляции `calcul`.
- в свою очередь цель `calculate.o` зависит от `calculate.c` и `calculate.h`.
- ниже также указана команда для компиляции цели.
- цель `main.o` зависит от файлов `main.c` и `calculate.h`
- в последнюю очередь указана команда для удаления файлов, связанных с программой.



```
#
# Makefile
#
CC = gcc
CFLAGS = -g
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 *~
# End Makefile
```

Рис.8: Рисунок 8

Создаю файл `calculate.c`

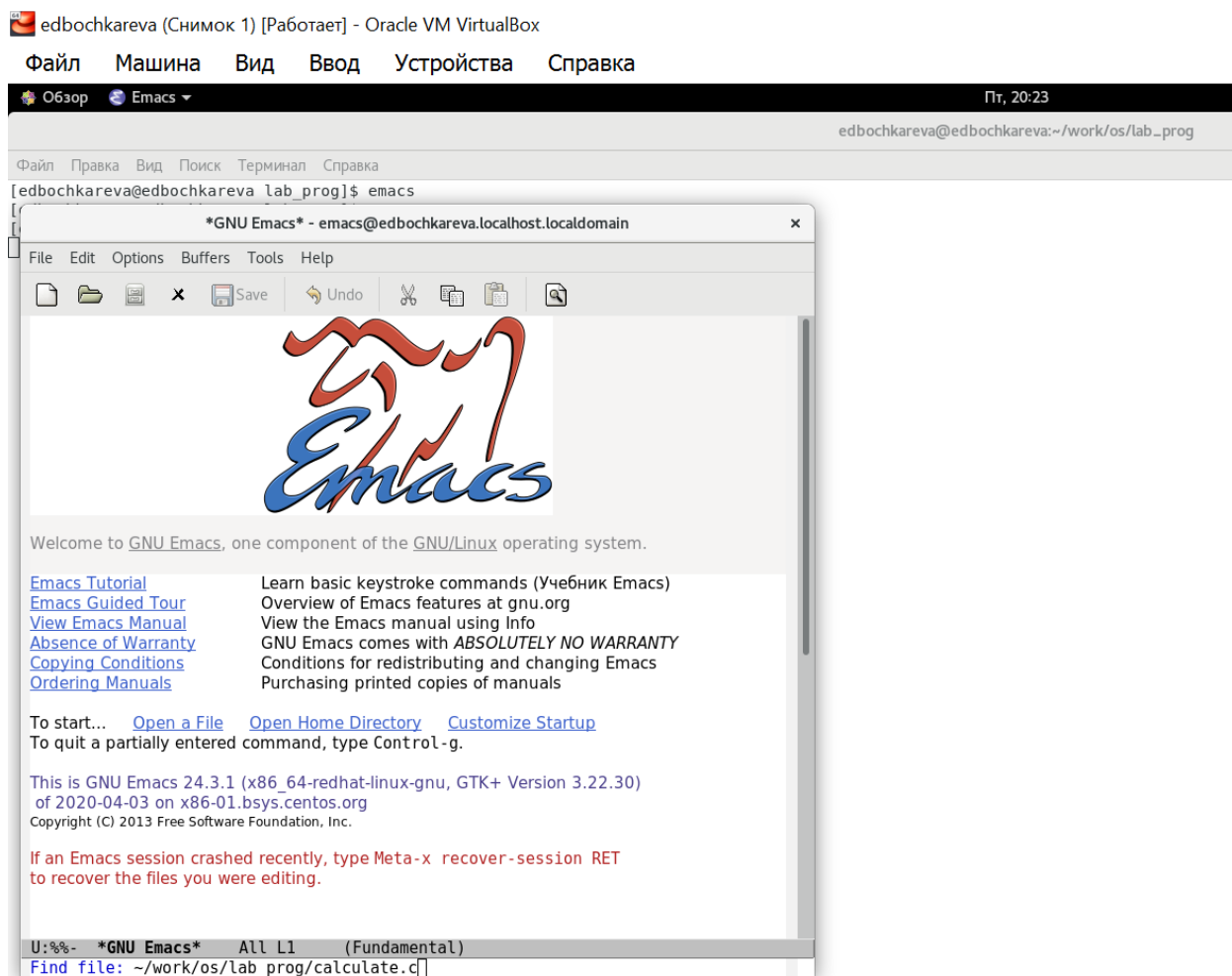
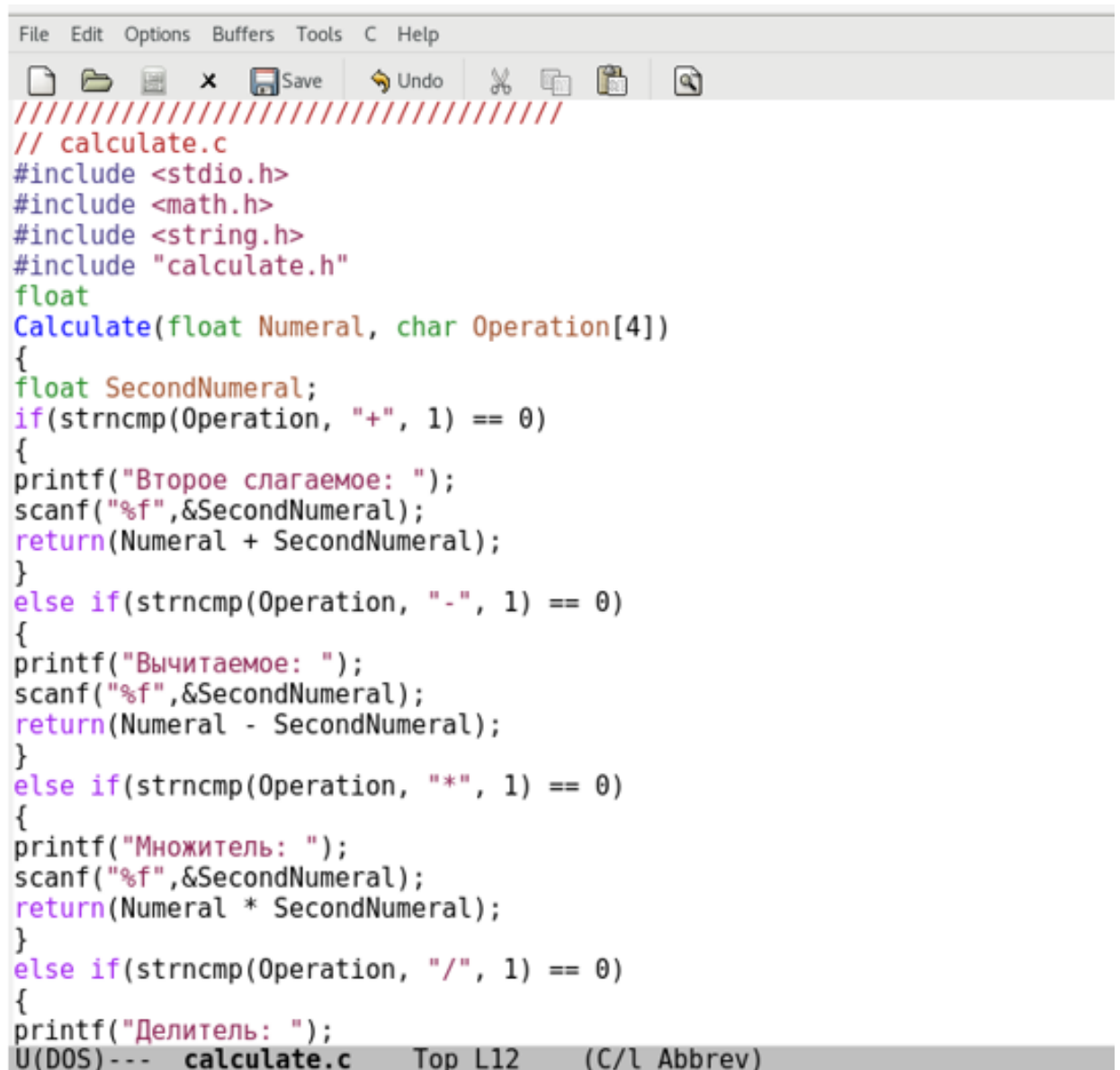


Рис.9: Рисунок 9

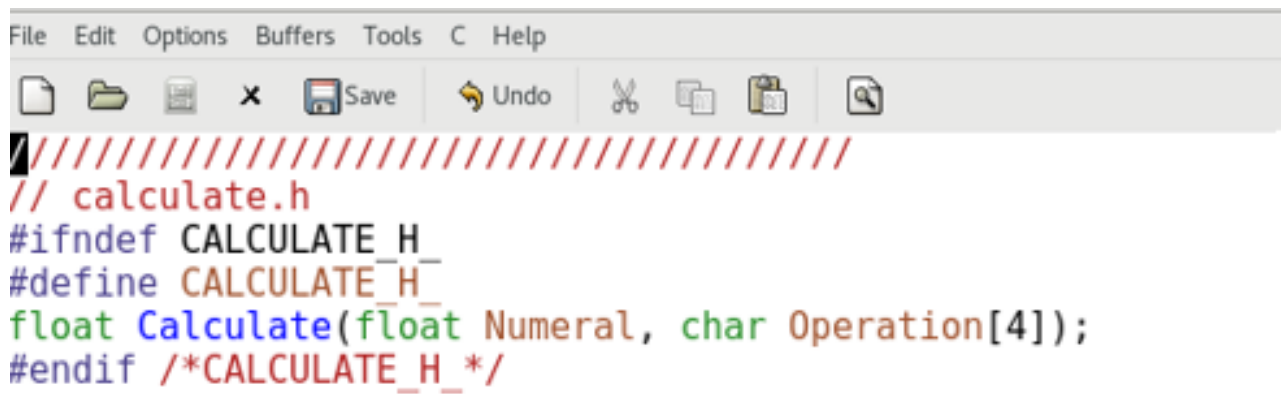
Ввожу //calculate.c



```
File Edit Options Buffers Tools C Help
Save Undo
////////////////////////////////////
// calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"
float
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: ");
}
U(DOS) --- calculate.c Top L12 (C/l Abbrev)
```

Рис.10: Рисунок 10

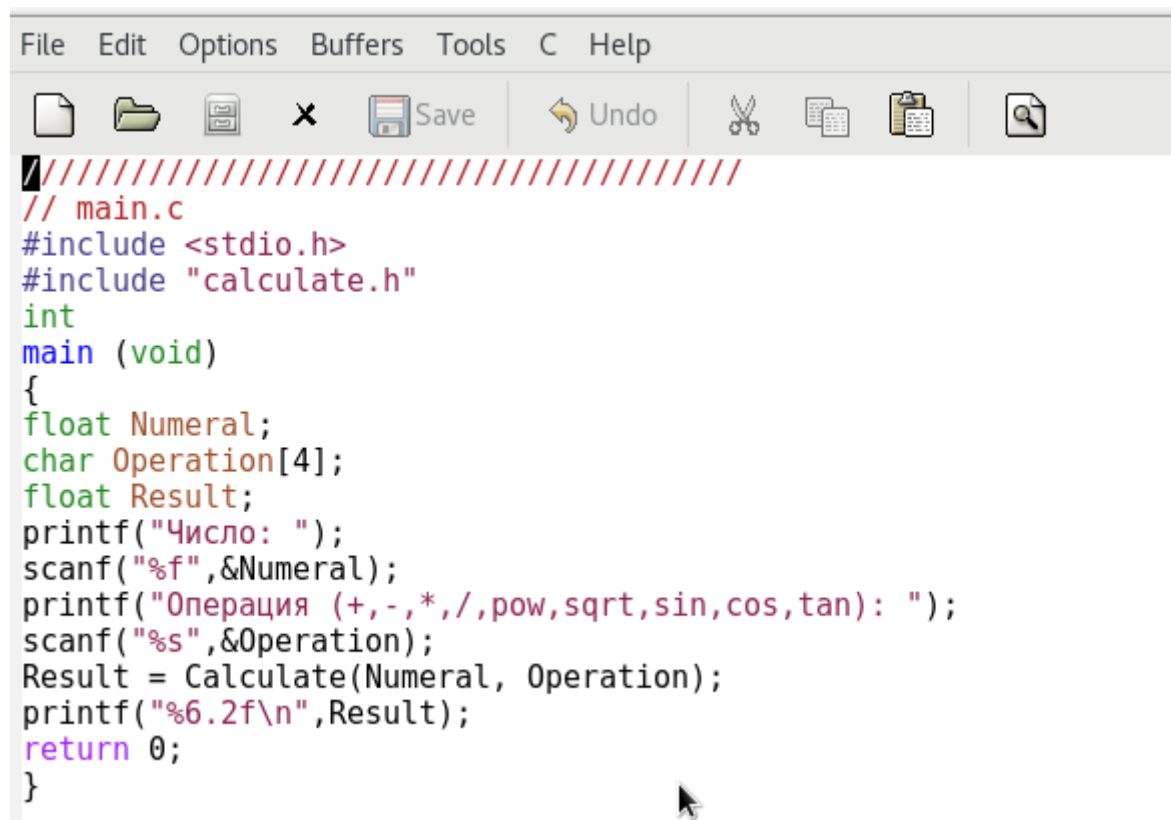
Ввожу //calculate.h



```
File Edit Options Buffers Tools C Help
Save Undo
////////////////////////////////////
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
```

Рис.11: Рисунок 11

Ввожу //main.c



```
File Edit Options Buffers Tools C Help
Save Undo
////////////////////////////////////
// main.c
#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.12: Рисунок 12

Набираю команды cd и gcc:

```
[root@edbochkareva edbochkareva]# cd ./work/os/lab_prog
[root@edbochkareva lab_prog]# gcc -c calculate.c -g
[root@edbochkareva lab_prog]# gcc -c main.c -g
[root@edbochkareva lab_prog]# gcc calculate.c main.o -o calcul -lm
[root@edbochkareva lab_prog]# ls./
```

Рис.12: Рисунок 12

Ввожу ls./

```
[root@edbochkareva lab_prog]# ls ./
calcul      calculate.c~ calculate.o  main.o
calculate.c calculate.h  main.c      Makefile
[root@edbochkareva lab_prog]# gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/edbochkareva/work/os/lab_prog/./calcul...done.
(nmh)
```

Рис.13: Рисунок 13

Выполнение кода по заданию

```

This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/edböchkareva/work/os/lab_prog/./calcul...done.
(gdb) run
Starting program: /home/edböchkareva/work/os/lab_prog/./calcul
Число: 2
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
4.00
[Inferior 1 (process 25296) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
(gdb) list
1      ////////////////////////////////////////// main.c
2      #include <stdio.h>
3      #include "calculate.h"
4      int
5      main (void)
6      {
7      float Numeral;
8      char Operation[4];
9      float Result;
10     printf("Число: ");
(gdb) █

```

Рис.14: Рисунок 14

Выполнение кода по заданию

```

Starting program: /home/edböchkareva/work/os/lab_prog/./calcul
Число: 2
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
4.00
[Inferior 1 (process 25296) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
(gdb) list
1      ////////////////////////////////////////// main.c
2      #include <stdio.h>
3      #include "calculate.h"
4      int
5      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)

```

Рис.16: Рисунок 16

Выполнение кода по заданию

```
5      int
6      main (void)
7      {
8          float Numeral;
9          char Operation[4];
10         float Result;
11         printf("Число: ");
(gdb) list 12,15
12         scanf("%f",&Numeral);
13         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
14         scanf("%s",&Operation);
15         Result = Calculate(Numeral, Operation);
(gdb) list calculate.c:20,29
20         scanf("%f",&SecondNumeral);
21         return(Numeral - SecondNumeral);
22     }
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
28     }
29     else if(strncmp(Operation, "/", 1) == 0)
(gdb)
```

Рис.17: Рисунок 17

```

15     Result = Calculate(Numeral, Operation);
(gdb) list calculate.c:20,29
20     scanf("%f",&SecondNumeral);
21     return(Numeral - SecondNumeral);
22 }
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
28     }
29     else if(strncmp(Operation, "/", 1) == 0)
(gdb) list calculate.c:20,27
20     scanf("%f",&SecondNumeral);
21     return(Numeral - SecondNumeral);
22 }
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
(qdb)

```

Рис.18: Рисунок 18


```

22     }
23     else if(strncmp(0peration, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
28     }
29     else if(strncmp(0peration, "/", 1) == 0)
(gdb) list calculate.c:20,27
20     scanf("%f",&SecondNumeral);
21     return(Numeral - SecondNumeral);
22     }
23     else if(strncmp(0peration, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint      keep y   0x00000000004007fd      in Calculate
                                                at calculate.c:21
(gdb)

```

Рис.19: Рисунок 19

Выполнение кода по заданию

```

20     scanf("%f",&SecondNumeral);
21     return(Numeral - SecondNumeral);
22     }
23     else if(strncmp(0peration, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint      keep y   0x00000000004007fd      in Calculate
                                                at calculate.c:21
(gdb) run
Starting program: /home/edbochkareva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 0

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf20 "-")
at calculate.c:21
21     return(Numeral - SecondNumeral);
(gdb) █

```

Рис.20: Рисунок 20

Выполнение кода по заданию

```
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint       keep y   0x00000000004007fd in Calculate
                                           at calculate.c:21

(gdb) run
Starting program: /home/edbochkareva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 0

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf20 "-")
  at calculate.c:21
21     return(Numeral - SecondNumeral);
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf20 "-") at calculate.c:21
#1  0x0000000000400a7d in main () at main.c:15
(gdb) █
```

Рис.21: Рисунок 21

Выполнение кода по заданию

```

25     printf("Множитель: ");
26     scanf("%f",&SecondNumeral);
27     return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x00000000004007fd in Calculate
                                                at calculate.c:21

(gdb) run
Starting program: /home/admin/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 0

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf20 "-")
  at calculate.c:21
21     return(Numeral - SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf20 "-") at calculate.c:21
#1 0x0000000000400a7d in main () at main.c:15
(gdb) print Numeral
$1 = 5
(gdb)

```

Рис.22: Рисунок 22

Выполнение кода по заданию

```

27     return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x00000000004007fd in Calculate
                                                at calculate.c:21

(gdb) run
Starting program: /home/edbochkareva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 0

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf20 "-")
  at calculate.c:21
21     return(Numeral - SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf20 "-") at calculate.c:21
#1 0x0000000000400a7d in main () at main.c:15
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) █

```

Рис.23: Рисунок 23

Выполнение кода по заданию

```
Starting program: /home/edböchkareva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 0

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf20 "-")
    at calculate.c:21
21      return(Numeral - SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf20 "-") at calculate.c:21
#1 0x0000000000400a7d in main () at main.c:15
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint       keep y  0x00000000004007fd in Calculate
                                     at calculate.c:21

        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

Рис.24: Рисунок 24

11.5. Ответы на контрольные вопросы

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

ОТВЕТ: Информацию можно получить с помощью команды `man <имя команды>`, информацию о которой хотим получить>.

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.

Изначально названный GNU C Compiler, поддерживал только язык Си. Позднее GCC был расширен для компиляции исходных кодов на таких языках программирования, как C++, Objective-C, Java, Фортран, Ada, Go, GAS и D.

Команда make позволяет задействовать одноименную утилиту, предназначенную для компиляции программного обеспечения из исходных кодов.

GNU Debugger — 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.

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

ОТВЕТ:

-сбор и анализ требований;

- проектирование прототипа;
- разработка приложения;
- тестирование и отладка;

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

ОТВЕТ: Суффикс (или по-другому расширение файла) нужен компилятору для определения типа файла и дальнейшей компиляции. Например, суффикс .c означает, что программа написана на языке C.

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

Компиляторы C и компилятор C++ одинаково относятся к суффиксам, но каждый раз давать файлам заголовков имена с .h (расширение) настолько общепринято, что надоедает. Есть

недостаток строгих правил, это к примеру несколько стилей именования файлов реализации в языке C++, например стандартные суффиксы .C, .crr, .crr, .c++, и .cc. Иногда встречаются файлы заголовков C++ с суффиксом .hrr.

Главное — это соблюдать единообразие при выборе суффикса.

11.5.4. Каково основное назначение компилятора языка C в UNIX?

ОТВЕТ: Компилятор — программа, переводящая написанный на языке программирования текст в набор машинных кодов. Назначение компилятора состоит в компиляции исходного кода и создании исполняемого файла.

Входная информация для компилятора есть: 1) на фазе трансляции: исходный код программы, являющийся описанием алгоритма или программы на предметно-ориентированном языке программирования; 2) на фазе компоновки: сгенерированные на фазе трансляции файлы объектных кодов модулей программы, а также файлы объектных кодов статических библиотек и данные об используемых динамических библиотеках.

На выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код[5], байт-код).

Компилировать — проводить сборку машинной программы, включая: 1) трансляцию с предметно-ориентированного языка на машинно-ориентированный язык[3]; 2) компоновка исполняемой машинно-ориентированной программы из сгенерированных на фазе трансляции объектных модулей — модулей, содержащих части кода программы на машинно-ориентированного кода программы.

Довольно часто компиляторы с языков высокого уровня выполняют лишь трансляцию исходного кода, компоновку же поручая внешнему компоновщику, — компоновщику, представляющему самостоятельную программу, вызываемую компилятором как внешняя подпрограмма.

и вот таким элементарным Makefile:

```
.PHONY: all
all: foo
.PHONY: clean
clean:
    $(RM) foo
```

Здесь видно, что файл `foo.c` является частью проектной модели, т. к. будет участвовать в процессе компиляции (посредством встроенного правила `$(CC) foo.c -o foo`), а файл `bar.c` — не является.

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

ОТВЕТ: Программа должна иметь доступ к отладочной информации в бинарных файлах. Для этого при компиляции необходимо указывать опцию `-g`.

11.5.8. Назовите и дайте основную характеристику основным командам отладчика `gdb`.

ОТВЕТ:

- `backtrace` – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от `main()`; иными словами, выводит весь стек функций;
- `break` – устанавливает точку останова; параметром может быть номер строки или название функции;
- `clear` – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
- `continue` – продолжает выполнение программы от текущей точки до конца;
- `delete` – удаляет точку останова или контрольное выражение;

- `display` – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
 - `finish` – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
 - `info breakpoints` – выводит список всех имеющихся точек останова;
 - `info watchpoints` – выводит список всех имеющихся контрольных выражений;
 - `list` – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
 - `next` – пошаговое выполнение программы, но, в отличие от команды `step`, не выполняет пошагово вызываемые функции;
 - `print` – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
 - `run` – запускает программу на выполнение;
 - `set` – устанавливает новое значение переменной
 - `step` – пошаговое выполнение программы;
- `watch` – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится.

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

ОТВЕТ: Увидеть ошибку при компиляции - Попытаться исправить ошибку в исходных файлах - Снова попытаться скомпилировать программу. Повторять до положительного результата.

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

ОТВЕТ: Компилятор указывает строку, где содержится ошибка, и сообщение об ошибке. Компиляция файла не выполняется до устранения всех критичных ошибок.

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

ОТВЕТ:

Исходный код либо используется для получения объектного кода, либо сразу выполняется интерпретатором. Другое важное назначение исходного кода — описание программы. По тексту программы можно восстановить логику её поведения. Для облегчения понимания исходного кода используются комментарии.

Комментирование, говорящие названия переменных, разбиение кода на отдельные функции, применение форматирования (табуляция в циклах и условных операторах, пустые строки между блоками).

Как комментарии оформляют в коде. Для выделения комментариев в коде используют разные символы: // — однострочные в языках C, C++, PHP, C#, Java и JavaScript; # — однострочный в Python, PHP; /* */ — многострочные в C, C++, PHP, C#, Java, JavaScript.

Говорящие названия переменных.

Использовать короткие имена только для переменных «местного значения». Называть переменные именами, не несущими смысловой нагрузки, например `a`, `e`, `p`, `mg` — можно только в том случае, если

они используются в небольшом фрагменте кода и их применение очевидно. Вообще же, название переменной должно быть понятным. Для имен переменных должны выполняться следующие правила: Имя каждой переменной должно быть уникальным; дубликаты имен не допускаются. Имена переменных могут иметь длиной до 64 байт (символов), первый символ в имени переменной должен быть буквой либо одним из символов @, #, или \$.

Следует придерживаться нескольких несложных правил при выборе имен переменных:

- Имя переменной может содержать только латинские буквы, числа и символ нижнего подчеркивания;
- Имя переменной не должно содержать пробелов;
- Имя переменной не должно начинаться с цифры;
- Регистр важен: var и Var это разные переменные.

Разбиение кода на отдельные функции, применение форматирования.

ReSharper помогает поддерживать единый стиль кода по всей кодовой базе. Настройки стиля по умолчанию основаны на общепринятых стандартах и передовых практиках. Однако, можно настроить индивидуальный стиль кода, как для постоянной работы с ReSharper, так и для отдельного решения и [поделиться настройками с командой](#). ReSharper учитывает предпочтения в стиле кода, когда предлагает варианты [автодополнения](#), [генерирует новые члены](#), применяет [шаблоны кода](#) и производит [рефакторинг](#). Нарушения стиля кода отслеживаются при помощи [инспекций](#) и мгновенно устраняются благодаря [быстрым исправлениям](#) и механизму Code Cleanup.

Откройте код. Выделите код. Выберите Редактировать > Код > Применить исходное форматирование к выделенному. Или выберите Применить исходное форматирование к выбранному в разделе Общая панель инструментов > Форматировать исходный код.

11.5.12. Каковы основные задачи, решаемые программой `split`

ОТВЕТ: Программа анализирует код, выдает ошибки и комментарии о нем.

Команда `split` в Linux позволяет разбивать файлы на несколько файлов. Есть несколько способов настроить параметры для вашего приложения. Мы покажем вам несколько примеров команды `split`, которые помогут вам понять ее использование. Можно использовать команды `ll` и `wc` для выделения изменений файла.

Примеры команды `Split` в Linux

Синтаксис команды `Split`:

```
split [options] filename [prefix]
```

1. Разделите файлы на несколько файлов

По умолчанию команда `split` создает новые файлы для каждых 1000 строк. Если префикс не указан, он будет использовать 'x'. Следующие буквы перечисляют файлы, поэтому сначала идет хаа, затем хаб и так далее.

Давайте разделим пример файла журнала:

```
split AndreyExLogFile.log
```

Если вы используете команду `ls`, вы можете увидеть несколько новых файлов в вашем каталоге.

```
andreyex@destroyer:~/Documents$ ls
```

```
AndreyExLogFile.log  xab  xad  xaf  xah  xaj  xal  xan  xap  xar
```

```
хаа                  хас  хae  хag  хai  хак  хам  хао  хаq
```

Вы можете использовать `wc` для быстрой проверки количества строк после разделения.

Помните, что ранее мы видели, что наш исходный файл содержал 17 170 строк. Таким образом, мы можем видеть, что наша программа создала как и ожидалось, 18 новых файлов. 17 из них заполнены 1000 строками в каждой, а последняя имеет оставшиеся 170 строк.

Другой способ продемонстрировать, что происходит, — запустить команду с параметром `verbose`. Если вы не знакомы с `verbose`, вы пропускаете! Он предоставляет более подробные отзывы о том, что делает ваша система, и он доступен для использования со многими командами.

```
split AndreyExLogFile.log --verbose
```

2. Разделите файлы на несколько файлов с определенными номерами строк

Мы понимаем, что вам может не понравиться, что файлы разбиты на файлы по 1000 строк. Вы можете изменить это поведение с помощью опции `-l`.

Теперь вы можете указать, сколько строк вы хотите в каждом из новых файлов.

```
split AndreyExLogFile.log -l 500
```

3. Разделение файлов на n файлов

Опция `-n` делает разделение на указанное число частей или кусков. Вы можете назначить, сколько файлов вы хотите, добавив целочисленное значение после `-n`.

```
split AndreyExLogFile.log -n 15
```

5. Разделите и укажите длину суффикса

Разделение имеет длину суффикса по умолчанию 2 [aa, ab и т. д.]. Это изменится автоматически при увеличении количества файлов, но если вы хотите изменить его вручную, это тоже возможно. Допустим, вы хотите, чтобы наши файлы были названы как-то вроде `AndreyExSeparatedLogFiles.log_aaaab`.

Как ты можешь это сделать? Опция `-a` позволяет нам указать длину суффикса.

```
split AndreyExLogFile.log AndreyExSeparatedLogFiles.log_ -a 5
```

6. Разделить с помощью числового суффикса заказа

До этого момента вы видели, что ваши файлы разделены с использованием различных буквенных комбинаций. Лично нам гораздо проще различать файлы по номерам.

Давайте сохраним длину суффикса из предыдущего примера, но изменим алфавитную организацию на числовую с помощью опции `-d`.

7. Добавьте шестнадцатеричные суффиксы для разделения файлов

Другой вариант создания суффикса — использовать встроенный шестнадцатеричный суффикс, который чередует упорядоченные буквы и цифры.

В этом примере мы объединяем несколько вещей, которые мы вам уже показали. Мы разделим файл, используя свой собственный префикс и выбрали подчеркивание для удобства чтения.

Мы использовали опцию `-x` для создания шестнадцатеричного суффикса. Затем разделили наш файл на 50 кусков и дали суффиксу длину 6.

```
split AndreyExLogFile.log _ -x -n50 -a6
```

8. Разделите файлы на несколько файлов определенного размера

Также возможно использовать размер файла, чтобы разбить файлы в разбивке. Возможно, вам нужно отправить большой файл по сети с ограниченным размером как можно более эффективно. Вы можете указать точный размер для ваших требований.

Синтаксис может стать немного сложнее, так как мы продолжаем добавлять опции. Итак, мы объясним, как работает команда, прежде чем показывать пример.

Если вы хотите создать файлы определенного размера, используйте опцию `-b`. Затем вы можете написать `n K [B]`, `n M [B]`, `n G [B]`, где `n` — это значение размера вашего файла, а `K [1024]` — это `-kibi`, `M` — `-mebi`, `G` — `-gibi` и далее. `KB [1000]` — килограмм, `МБ` — мега и т. д.

```
split AndreyExLogFile.log AndreyExSeparatedLogFiles.log_ -d -b 128KB
```

9. Разбивка файлов на несколько с определенным размером файла

Если вы хотите разбить файлы примерно на один и тот же размер, но сохранить структуру строк, это может быть лучшим выбором для вас. С помощью `-C`, вы можете указать максимальный размер. Затем программа автоматически разбивает файлы на основе полных строк.

```
split AndreyExLogFile.log AndreyExNewLogFiles.log_ -d -C 1MB
```

Выводы, согласованные с целью

В данной работе были изучены основные принципы анализа, тестирования и отладки программ в UNIX-подобных операционных системах на примере калькулятора, реализованного на языке C.