

Лабораторная работа №13

Отчёт по лабораторной работе №13

Макарова Анастасия Михайловна

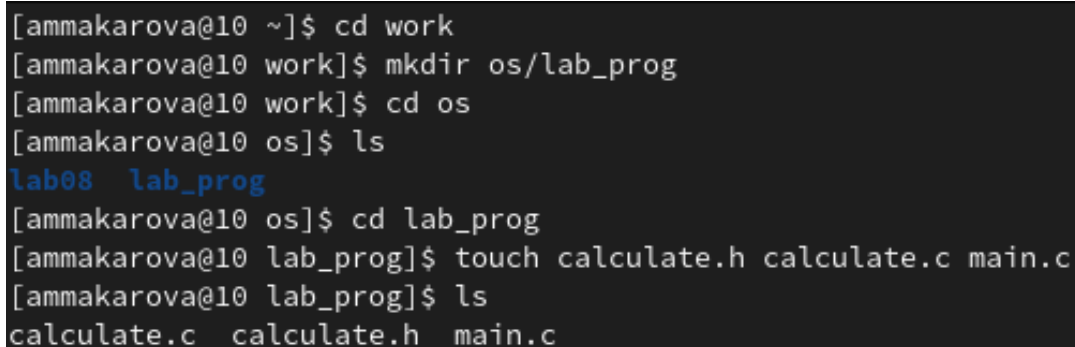
Содержание

Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования калькулятора с простейшими функциями.

Выполнение лабораторной работы

1. В домашнем каталоге создадим подкаталог `~/work/os/lab_prog` и создадим в нём файлы: `calculate.h`, `calculate.c`, `main.c` (Рис.1).



```
[ammakarova@10 ~]$ cd work
[ammakarova@10 work]$ mkdir os/lab_prog
[ammakarova@10 work]$ cd os
[ammakarova@10 os]$ ls
lab08  lab_prog
[ammakarova@10 os]$ cd lab_prog
[ammakarova@10 lab_prog]$ touch calculate.h calculate.c main.c
[ammakarova@10 lab_prog]$ ls
calculate.c  calculate.h  main.c
```

Рис.1

2. Реализуем функции калькулятора в файле `calculate.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).

```
ammakarova@10:~/work/os/lab_prog — mcedit calculate.c
calculate.c [----] 22 L: [ 1+11 12/ 63] *(208 /1602b) 0010 0x00A
// 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);
    }
}
```

Рис.2

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (Рис.3).

```
ammakarova@10:~/work/os/lab_prog — mcedit calculate.h
calculate.h [----] 50 L: [ 1+ 6 7/ 10] *(150 / 176b) 0010 0x00A
// calculate.h

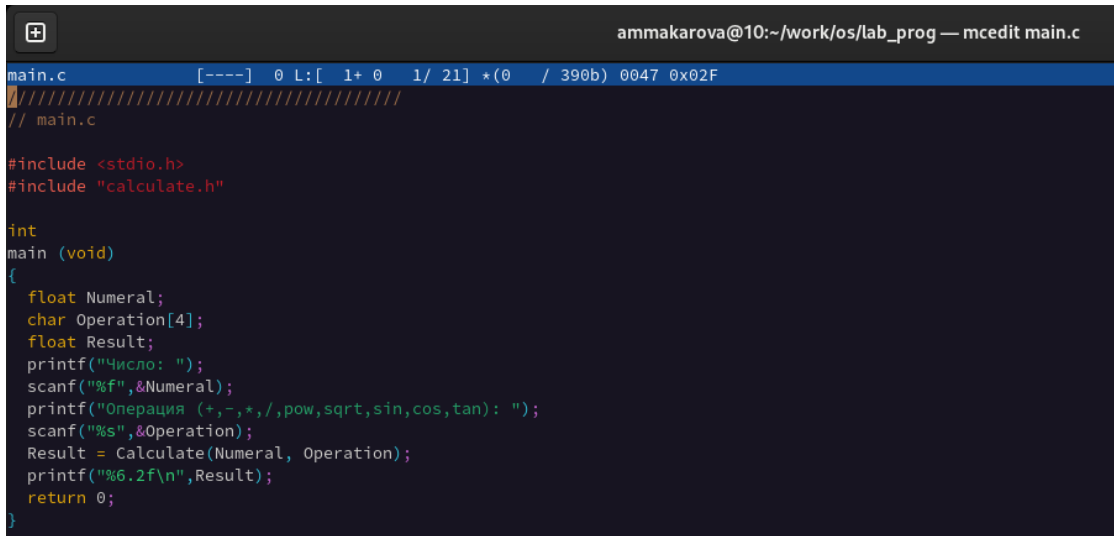
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис.3

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (Рис.4).



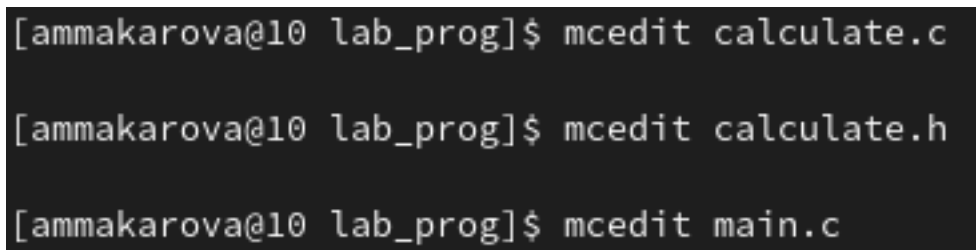
```
main.c [----] 0 L: [ 1+ 0 1/ 21] *(0 / 390b) 0047 0x02F
// main.c

#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("%.2f\n",Result);
    return 0;
}
```

Рис.4

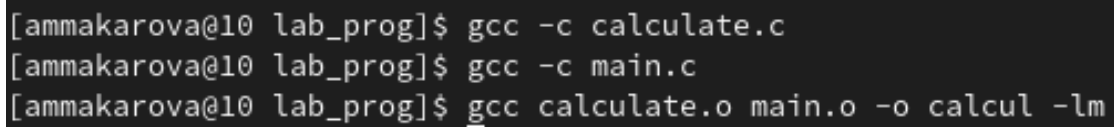
При редактировании программ используем редактор mcedit (Рис.5).



```
[ammakarova@10 lab_prog]$ mcedit calculate.c
[ammakarova@10 lab_prog]$ mcedit calculate.h
[ammakarova@10 lab_prog]$ mcedit main.c
```

Рис.5

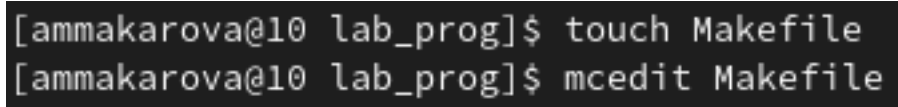
3. Выполним компиляцию программы посредством gcc (Рис.6).



```
[ammakarova@10 lab_prog]$ gcc -c calculate.c
[ammakarova@10 lab_prog]$ gcc -c main.c
[ammakarova@10 lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рис.6

4. Создадим Makefile и откроем его в редакторе mcedit (Рис.7).

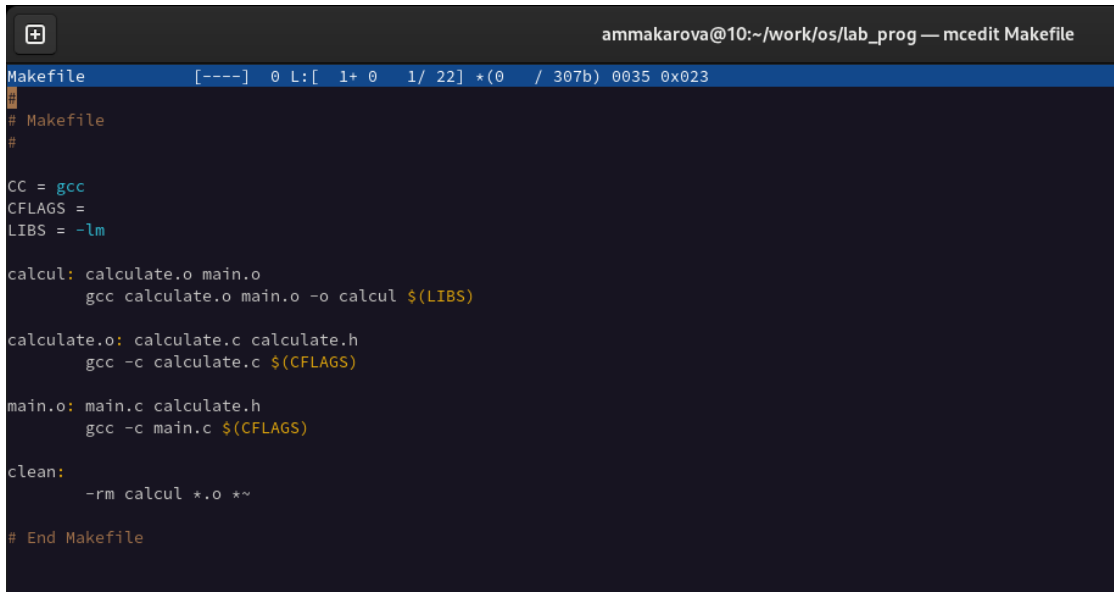


```
[ammakarova@10 lab_prog]$ touch Makefile
[ammakarova@10 lab_prog]$ mcedit Makefile
```

Рис.7

Этот файл необходим для того, чтобы автоматически компилировать файлы calculate.c, main.c, а также объединять их в один исполняемый файл 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 (Рис.8).



```
Makefile [-----] 0 L:[ 1+ 0 1/ 22] *(0 / 307b) 0035 0x023
# Makefile
#
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 *~

# End Makefile
```

Рис.8

Перед использованием gdb исправим Makefile. В переменную CFLAGS добавим опцию -g, которая необходима для компиляции объектных файлов и их использования в отладчике GDB. Утилита компиляции будет выбираться с помощью переменной CC (Рис.9).

```

Makefile      [-M--]  0 L:[  1+21  22/ 22] *(316 /
#
# Makefile
#

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 *~

# End Makefile

```

Рис.9

5. Удалим исполняемые и объектные файлы из каталога с помощью команды `make clean`. Выполним компиляцию файлов с помощью команд `make calculate.o`, `make main.o`, `make calcul` (Рис.10).

```

[ammakarova@10 lab_prog]$ make clean
rm calcul *.o *~
[ammakarova@10 lab_prog]$ make calculate.o
gcc -c calculate.c -g
[ammakarova@10 lab_prog]$ make main.o
gcc -c main.c -g
[ammakarova@10 lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm

```

Рис.10

6. С помощью `gdb` выполним отладку программы `calcul`.

- Запустим отладчик GDB, загрузив в него программу для отладки (Рис.11):

```
[ammakarova@10 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...
(gdb) █
```

Рис.11

- Для запуска программы внутри отладчика введем команду run (Рис.12):

```
(gdb) run
Starting program: /home/ammakarova/work/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.
Downloading separate debug info for /home/ammakarova/work/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: 0
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sin
0.00
[Inferior 1 (process 10797) exited normally]
```

Рис.12

- Для постраничного (по 9 строк) просмотра исходного код используем команду list (Рис.13):

```
(gdb) list
1      //////////////////////////////////////////
2      // main.c
3
4      #include <stdio.h>
5      #include "calculate.h"
6
7      int
8      main (void)
9      {
10         float Numeral;
```

Рис.13

- Для просмотра строк с 12 по 15 основного файла используем list с параметрами (Рис.14):

```
(gdb) list 12,15
12         float Result;
13         printf("Число: ");
14         scanf("%f",&Numeral);
15         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
```

Рис.14

- Для просмотра определённых строк не основного файла используем list с параметрами (Рис.15):

```
(gdb) list calculate.c:20,29
20         {
21             printf("Вычитаемое: ");
22             scanf("%f",&SecondNumeral);
23             return(Numeral - SecondNumeral);
24         }
25         else if(strncmp(Operation, "*", 1) == 0)
26         {
27             printf("Множитель: ");
28             scanf("%f",&SecondNumeral);
29             return(Numeral * SecondNumeral);
```

Рис.15

- Установим точку останова в файле calculate.c на строке номер 21 и выведем информацию об имеющихся в проекте точка останова (Рис.16):

```
(gdb) list calculate.c:20,27
20      {
21          printf("Вычитаемое: ");
22          scanf("%f",&SecondNumeral);
23          return(Numeral - SecondNumeral);
24      }
25      else if(strncmp(Operation, "+", 1) == 0)
26      {
27          printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type          Disp Enb Address              What
1        breakpoint    keep y   0x000000000040120f in Calculate at calculate.c:21
```

Рис.16

- Запустим программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова. Команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места (Рис.17):

```
(gdb) run
Starting program: /home/ammakarova/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:17
```

Рис.17

- Посмотрим, чему равно на этом этапе значение переменной Numeral и сравним с результатом вывода на экран после использования команды display Numeral (Рис.18):

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

Рис.18

- Уберем точки останова (Рис.19):


```
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint     keep y   0x00000000000040120f in Calculate at calculate.c:21
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
```

Рис.19

7. С помощью утилиты splint проанализируем коды файлов calculate.c и main.c. (Рис.20, 21).

```
[ammakarova@10 lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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:10:31: Function parameter Operation declared as manifest array
                    (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:7: 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:22:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:10: 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:38:17: Return value type double does not match declared type float:
                    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:46:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:13: Return value type double does not match declared type float:
                    (pow(Numeral, SecondNumeral))
calculate.c:50:11: Return value type double does not match declared type float:
```

Рис.20

```

[ammakarova@10 lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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:14:3: 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:16:14: 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:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &ope...

Finished checking --- 4 code warnings
[ammakarova@10 lab_prog]$

```

Рис.21

Вывод

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

Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.? Нужно воспользоваться командой man или опцией -help для каждой команды.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. Unix поддерживает следующие основные этапы разработки приложений:
 - создание исходного кода программы;
 - представляется в виде файла;
 - сохранение различных вариантов исходного текста;
 - анализ исходного текста; Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над

проектом программы нужно, чтобы они не делали изменений кода в одно время.

- компиляция исходного текста и построение исполняемого модуля;
 - тестирование и отладка;
 - проверка кода на наличие ошибок
 - сохранение всех изменений, выполняемых при тестировании и отладке.
3. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Использование суффикса ".c" для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл `abcd.c` должен компилироваться, а по суффиксу .o, что файл `abcd.o` является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы `abcd.c` и построения исполняемого модуля `abcd` имеет вид: `gcc -o abcd abcd.c`.

Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция `-prefix` может быть использована для установки такого префикса. Плюс к этому команда `bzr diff -pl` выводит префиксы в форме которая подходит для команды `patch -pl`.

4. Каково основное назначение компилятора языка C в UNIX? Предупреждения компилятора это очень полезный механизм, позволивший программистам избежать многих незаметных на первый взгляд ошибок, которые могли проявиться только после запуска уже скомпилированной программы (возможно, много лет спустя после первого запуска). Его, безусловно, нужно использовать, и относиться к предупреждениям следует максимально внимательно. Многие авторитетные книги по C++, написанные в жанре сборников советов (Скотта Мейерса, Герба Саттера, Андрея Александреску, Стивена Дьюхерста), содержат совет компилировать программу на максимально строгом уровне предупреждений и добиваться того, чтобы компилятору не к чему было придраться.

Компилятор GCC позволяет включать несколько десятков видов предупреждений, для каждого из которых имеется свой ключ

компиляции. Для удобства основные из них сгруппированы и включаются двумя ключами: -Wall и -Wextra

5. Для чего предназначена утилита make? Утилита, автоматизирующая процесс преобразования файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки.
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат:

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

```
[(tab)commands]
```

```
[#commentary]
```

```
[(tab)commands]
```

```
[#commentary],
```

где # — специфицирует начало комментария, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; ; — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний.

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
8. Назовите и дайте основную характеристику основным командам отладчика gdb.
 - 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;
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
 10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.
 11. Назовите основные средства, повышающие понимание исходного кода программы.
 12. Каковы основные задачи, решаемые программой `splint`? Эта утилита 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.