

# **Лабораторная работа №13**

**1032224521**

Атанесов Александр Николаевич

# Содержание

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Цель работы</b>                          | <b>5</b>  |
| <b>2</b> | <b>Задание</b>                              | <b>6</b>  |
| <b>3</b> | <b>Выполнение лабораторной работы</b>       | <b>7</b>  |
| <b>4</b> | <b>Выводы</b>                               | <b>17</b> |
| <b>5</b> | <b>Пояснение содержимого файла Makefile</b> | <b>18</b> |
| <b>6</b> | <b>Ответы на контрольные вопросы</b>        | <b>19</b> |
|          | <b>Список литературы</b>                    | <b>23</b> |

## Список иллюстраций

|      |  |    |
|------|--|----|
| 3.1  | Использую команду mkdir . . . . .            | 7  |
| 3.2  | Использую команду touch . . . . .            | 7  |
| 3.3  | Использую команду chmod . . . . .            | 7  |
| 3.4  | Использую команду nano . . . . .             | 8  |
| 3.5  | Использую редактор nano . . . . .            | 9  |
| 3.6  | Использую команду nano . . . . .             | 10 |
| 3.7  | Использую редактор nano . . . . .            | 10 |
| 3.8  | Использую команду nano . . . . .             | 10 |
| 3.9  | Использую редактор nano . . . . .            | 11 |
| 3.10 | Использую команду chmod . . . . .            | 11 |
| 3.11 | Использую команду nano . . . . .             | 11 |
| 3.12 | Использую редактор nano . . . . .            | 12 |
| 3.13 | Использую команду gdb./calcul, run . . . . . | 13 |
| 3.14 | Использую команду list 12,15 . . . . .       | 14 |
| 3.15 | Использую команду break . . . . .            | 15 |
| 3.16 | Использую команду splint . . . . .           | 16 |

## Список таблиц

# 1 Цель работы

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

## 2 Задание

1. Создать простейший калькулятор в UNIX;

## 3 Выполнение лабораторной работы

1. Создаю папку lab\_prog. (рис. [3.1])

```
[aatanesov@fedora os]$ mkdir lab_prog
```

Рис. 3.1: Используя команду mkdir

### 3.1

2. Создаю файлы calculate.c calculate.h main.c. (рис. [3.2])

```
[aatanesov@fedora lab_prog]$ touch calculate.h calculate.c main.c
```

Рис. 3.2: Используя команду touch

### 3.2


3. Делаем эти файлы исполняемыми. (рис. [3.3])

```
[aatanesov@fedora lab_prog]$ chmod +x calculate.h calculate.c main.c
```

Рис. 3.3: Используя команду chmod

### 3.3

4. Открываю файл `calculate.c` через `nano`. (рис. [3.4])



```
[aatanesov@fedora lab_prog]$ nano calculate.c
```

Рис. 3.4: Используя команду `nano`

### 3.4

5. Пишу код для будущего калькулятора. (рис. [3.5])



```
GNU nano 6.4 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: ");
        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));
}
```

Рис. 3.5: Используя редактор nano

## 3.5

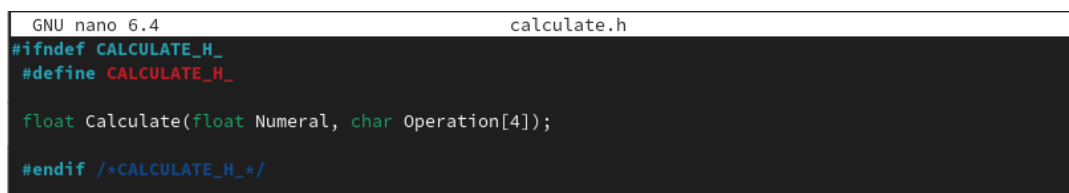
6. Открываю файл calculate.h через nano. (рис. [3.6])

```
[aatanesov@fedora lab_prog]$ nano calculate.h
```

Рис. 3.6: Используя команду nano

## 3.6

7. Пишу необходимый код для calculate.h . (рис. [3.7])



```
GNU nano 6.4 calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 3.7: Используя редактор nano

## 3.7

8. Открываю файл main.c через nano. (рис. [3.8])

```
[aatanesov@fedora lab_prog]$ nano main.c
```

Рис. 3.8: Используя команду nano

## 3.8

9. Пишу необходимый код, представленный в выполнение лабораторной работы №13 . (рис. [3.9])

```

GNU nano 6.4                                     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. 3.9: Используя редактор nano

## 3.9

10. Создаю файл Makefile. (рис. [3.10])

```
[aatanesov@fedora lab_prog]$ touch Makefile
```

Рис. 3.10: Используя команду chmod

## 3.10

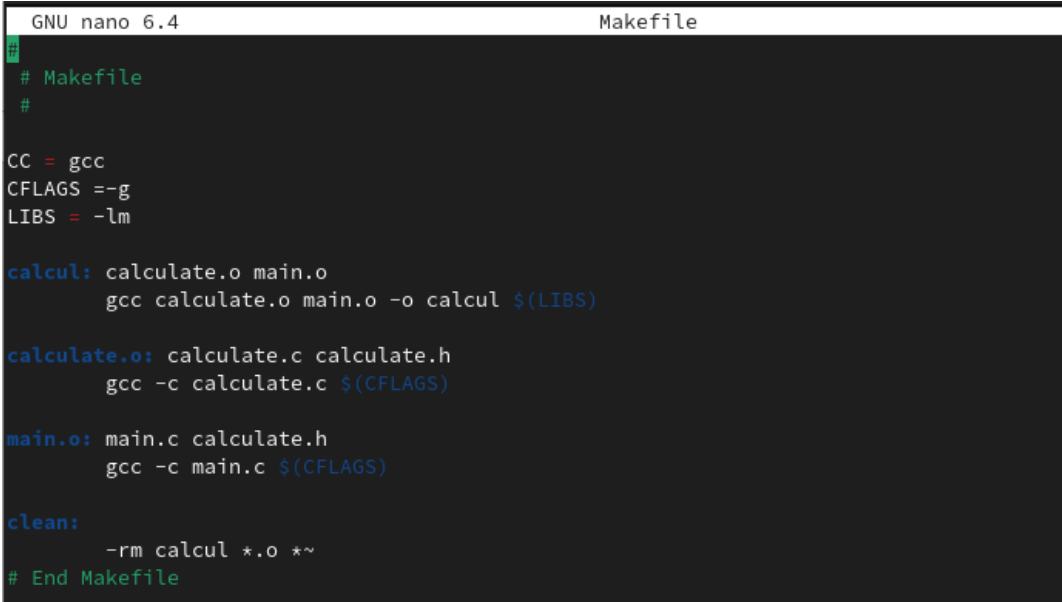
11. Открываю файл Makefile через nano. (рис. [3.11])

```
[aatanesov@fedora lab_prog]$ nano Makefile
```

Рис. 3.11: Используя команду nano

## 3.11

12. Пишу необходимый код. (рис. [3.12])

A screenshot of a terminal window showing the GNU nano 6.4 text editor. The editor is editing a file named 'Makefile'. The content of the Makefile is as follows:

```
# 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. 3.12: Использую редактор nano

## 3.12

13. Запускаю файл калькулятор. (рис. [3.13])

```
[aatanesov@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 13.1-3.fc37
Copyright (C) 2023 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) run
Starting program: /home/aatanesov/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Выводимое: 2
  3.00
[Inferior 1 (process 5460) exited normally]
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.36-9.fc37.x86_64
```

Рис. 3.13: Используя команду gdb./calcul, run

## 3.13

14. Вывожу 10 строк кода . (рис. [??])

```
(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);
```

(image/13.p

width=90%}

## 3.14

15. Вывожу с 12 по 15 строку кода нашей программы. (рис. [3.14])

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

Рис. 3.14: Используя команду list 12,15

## 3.15

16. Вывожу с 20 по 29 строку кода программы. (рис. [??])

```
(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      {
```

(image/18.png){#fig:

width=90%}

## 3.16

17. Вывожу с 20 по 27 и ставлю точку остановки . (рис. [3.15])

```

(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) nfo breakpoints
Undefined command: "nfo". Try "help".
(gdb) info breakpoints
Num    Type             Disp Enb Address              What
1      breakpoint      keep y   0x0000000000401247 in Calculate at calculate.c:22
(gdb)

```

Рис. 3.15: Используя команду break

## 3.17

18. Вывожу автоматический анализ кода программ main.c и calculate.c . (рис. [3.16])

```

[aatanesov@fedora lab_prog]$ splint calculate.c main.c
Splint 3.1.2 --- 23 Jul 2022

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:7:32: 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:31:1: 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)
main.c: (in function main)
main.c:10:2: Return value (type int) ignored: scanf("%f", &Num...
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 --- 18 code warnings

```

Рис. 3.16: Используя команду splint



## 4 Выводы

- Я научился создавать простые приложения и открывать их через терминал .

## 5 Пояснение содержимого файла Makefile

- Данный Makefile описывает процесс сборки программы “calcul” из исходных файлов “calculate.c” и “main.c”, а также заголовочного файла “calculate.h”.

Переменные CC, CFLAGS и LIBS содержат информацию о компиляторе, флагах компиляции и необходимых для линковки библиотеках соответственно.

Цель “calcul” (строка 9) зависит от объектных файлов “calculate.o” и “main.o”, и собирается командой “gcc calculate.o main.o -o calcul \$(LIBS)”.

Цели “calculate.o” и “main.o” (строки 12-16) компилируют соответствующие исходные файлы в объектные файлы.

Цель “clean” (строка 18) удаляет собранные объектные файлы и исполняемый файл.

Комментарии на каждой строке поясняют назначение каждой переменной или команды.

## 6 Ответы на контрольные вопросы

- 1. Для получения информации о возможностях программ gcc, make, gdb и др. можно обратиться к их официальной документации, доступной в сети Интернет, а также использовать команду man в терминале UNIX.
- 2. Основными этапами разработки приложений в UNIX являются: проектирование, написание исходного кода, компиляция, отладка, тестирование, установка и настройка приложения.
- 3. Суффикс в контексте языка программирования - это часть названия файла, указывающая на его тип и формат. Например, файл со суффиксом ".c" обозначает исходный код на языке C, а файл со суффиксом ".o" - скомпилированный объектный файл.
- 4. Основное назначение компилятора языка C в UNIX - это компиляция исходного кода на этом языке в машинный код, который может быть выполнен на компьютере.
- 5. Утилита make предназначена для автоматизации процесса сборки приложения из исходного кода и скомпилированных объектных файлов.
- 6. Пример структуры Makefile:

```
TARGET = my_program
```

```
CC = gcc
```

```
CFLAGS = -Wall -O2
```

```
$(TARGET): main.o functions.o
```

```
$(CC) -o $(TARGET) main.o functions.o
```

```
main.o: main.c functions.h
```

```
$(CC) $(CFLAGS) -c main.c
```

```
functions.o: functions.c functions.h
```

```
$(CC) $(CFLAGS) -c functions.c
```

- Основные элементы Makefile:
- TARGET - имя целевого файла приложения;
- CC - имя компилятора;
- CFLAGS - опции компилятора;
- \$(TARGET), \$(CC), \$(CFLAGS) - переменные, значения которых используются в правилах Makefile;
- \$(TARGET): main.o functions.o - правило сборки, которое указывает, что для создания целевого файла \$(TARGET) необходимы объектные файлы main.o и functions.o;
- main.o, functions.o - зависимости, т.е. файлы, которые необходимы для создания других файлов;
- main.c, functions.c, functions.h - исходные файлы приложения;
- \$(CC) -o \$(TARGET) main.o functions.o - команда сборки, которая говорит компилятору собрать файл \$(TARGET) из - - объектных файлов main.o и functions.o;
- \$(CC) \$(CFLAGS) -c main.c - команда компиляции исходного файла main.c в объектный файл main.o с опциями \$(CFLAGS).

- 7. Основное свойство, присущее всем программам отладки - это возможность управления выполнением программы, т.е. - пошаговой отладки. Для использования этого свойства необходимо установить точки останова в программе и запустить - ее в режиме отладки.
- 8. Основные команды отладчика gdb:
  - run - запустить программу;
  - break - установить точку останова в коде программы;
  - next - выполнить следующую строку кода и остановиться;
  - step - выполнить следующую строку кода и зайти в подпрограмму, если она вызывается в этой строке;
  - print - вывести значение переменной;
  - watch - установить точку останова на изменение значения переменной.
- 9. Схема отладки программы:
  - 1. Компилирование программы с опцией -g, которая добавляет отладочную информацию к скомпилированному файлу.
  - 2. Запуск gdb с указанием имени скомпилированного файла.
  - 3. Установка точек останова в нужных местах с помощью команды break.
  - 4. Запуск программы с помощью команды run.
  - 5. Пошаговое выполнение программы с помощью команд next или step.
  - 6. Использование команды print для вывода значений переменных или выражений.
  - 7. Остановка выполнения программы с помощью команды break, если необходимо.

- 8. Выход из gdb с помощью команды quit.
- 10. Компилятор при первом запуске анализирует синтаксическую корректность исходного кода программы и выдает сообщения об ошибках, если они есть. Эти сообщения содержат информацию о месте ошибки и ее характере. Размер и сложность программы также могут влиять на время компиляции.
- 11. Основные средства, повышающие понимание исходного кода программы - это комментарии в коде, описания функций и переменных, документация к проекту, а также использование среды разработки, которая облегчает чтение и изменение кода.
- 12. Основные задачи, решаемые программой splint - это статический анализ исходного кода на языке C с целью выявления потенциальных ошибок, утечки памяти и других проблем, связанных с безопасностью, надежностью и качеством кода.

## **Список литературы**