

# **Лабораторная работа 13**

**Отчет по лабораторной работе 13**

Куркина Евгения Вячеславовна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
	Список литературы	19

## Список иллюстраций

2.1	Создание подкаталога . . . . .	6
2.2	Скрипт калькулятора . . . . .	7
2.3	Текст интерфейсного файла . . . . .	7
2.4	Скрипт интерфейса . . . . .	8
2.5	Компиляция программы . . . . .	8
2.6	Исходный текст . . . . .	9
2.7	Измененный текст . . . . .	9
2.8	удаление файлов и компиляция файлов . . . . .	10
2.9	Компиляция файлов . . . . .	10
2.10	Запуск отладчика . . . . .	10
2.11	Команда run . . . . .	10
2.12	Команда list . . . . .	11
2.13	Команда просмотра с 12 по 15 строк . . . . .	11
2.14	Команда просмотра определенных строк . . . . .	11
2.15	Установка точки останова . . . . .	12
2.16	Информация о точках останова . . . . .	12
2.17	Проверка точки останова . . . . .	12
2.18	Значение переменной . . . . .	13
2.19	Сравнение результатов . . . . .	13
2.20	Удаление точки останова . . . . .	13
2.21	Анализ кода файла calcilate.c . . . . .	14
2.22	Анализ кода файла main.c . . . . .	14

## **Список таблиц**

# 1 Цель работы

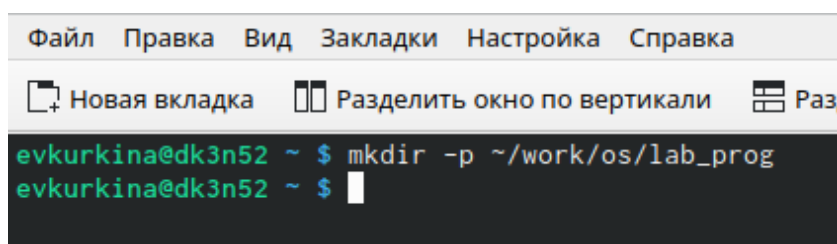
Здесь приводится формулировка цели лабораторной работы. Формулировки цели для каждой лабораторной работы приведены в методических указаниях.

Цель данной лабораторной работы — Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями. # Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

## 2 Выполнение лабораторной работы

- 1) В домашнем каталоге создаю подкаталог ~/work/os/lab\_prog (рис. 2.1).



The image shows a terminal window with a menu bar at the top containing 'Файл', 'Правка', 'Вид', 'Закладки', 'Настройка', and 'Справка'. Below the menu bar are three buttons: 'Новая вкладка', 'Разделить окно по вертикали', and 'Разделить окно по горизонтали'. The terminal text shows the user 'evkurkina@dk3n52' in the home directory '~' executing the command 'mkdir -p ~/work/os/lab\_prog'. The prompt returns to '\$' on the next line.

Рис. 2.1: Создание подкаталога

- 2) Создала файлы calculate.h, calculate.c, main.c. Написала текст примитивнейшего калькулятора, который 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. 2.2). Далее написала интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис. 2.3), а также текст основного файла main.c, реализующий интерфейс пользователя к калькулятору (рис. 2.4).

```
Приложения  Места  GNU Emacs

File Edit Options Buffers Tools C Help

// 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)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));
}

U:--- calculate.c  Top L39  (C/*l Abbrev) Ср мая 25 17:01 2.79
the '--debug-init' option to view a complete error backtrace.
```

Рис. 2.2: Скрпит калькулятора

```
Приложения  Места  GNU Emacs

emacs@dk3n52

File Edit Options Buffers Tools C Help

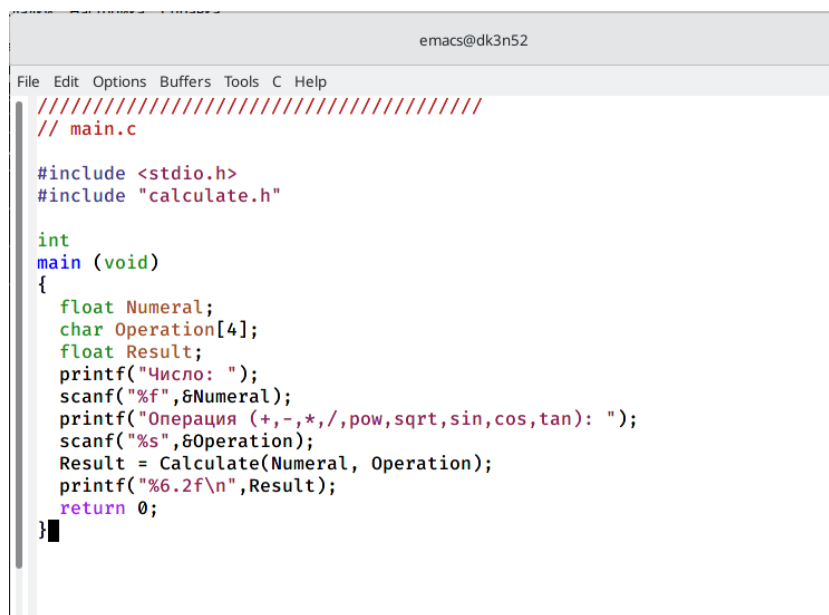
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 2.3: Текст интерфейсного файла



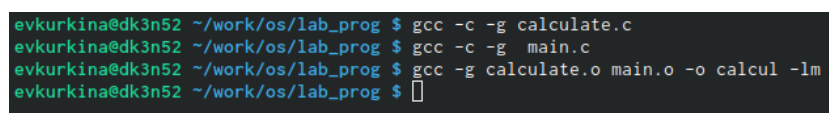
```
emacs@dk3n52
File Edit Options Buffers Tools C Help
////////////////////////////////////
// main.c

#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. 2.4: Скрипт интерфейса

3) Выполнила компиляцию программы посредством gcc (рис. 2.5).

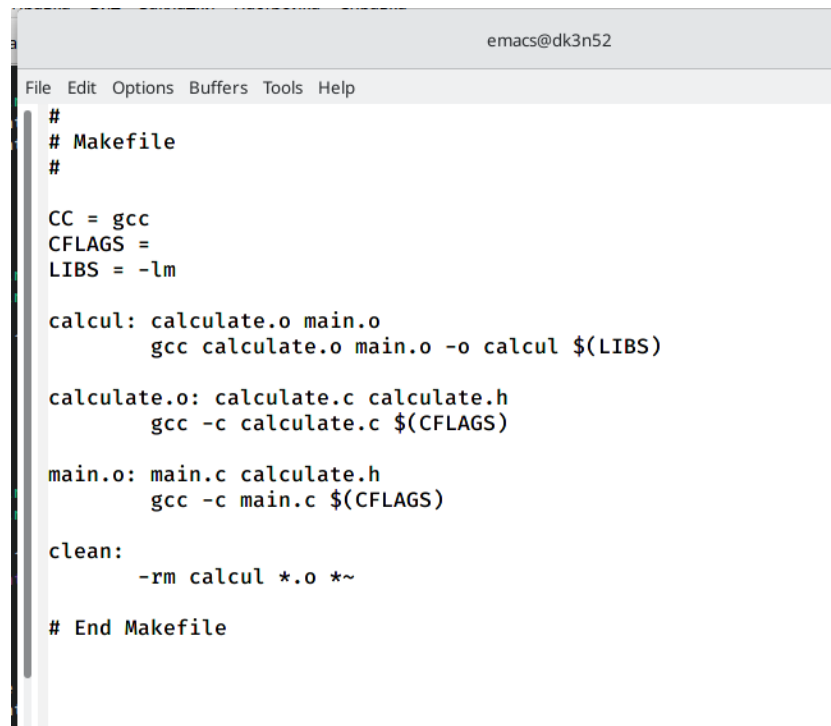


```
evkurkina@dk3n52 ~/work/os/lab_prog $ gcc -c -g calculate.c
evkurkina@dk3n52 ~/work/os/lab_prog $ gcc -c -g main.c
evkurkina@dk3n52 ~/work/os/lab_prog $ gcc -g calculate.o main.o -o calcul -lm
evkurkina@dk3n52 ~/work/os/lab_prog $
```

Рис. 2.5: Компиляция программы

4) Создала Makefile переписала в него данный текст (рис. 2.6), затем изменила его до рабочего состояния рис. 2.7). Данный файл необходим для автоматической компиляции файлов calculate.c. В переменную GLASS добавила опцию -g, утилита компиляции выбирается с помощью переменной CC. Далее я удалила файлы из каталога и выполнила компиляцию файлов (рис. 2.8)(рис. 2.9)





```
#
# Makefile
#

CC = gcc
CFLAGS =
LIBS = -lm

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

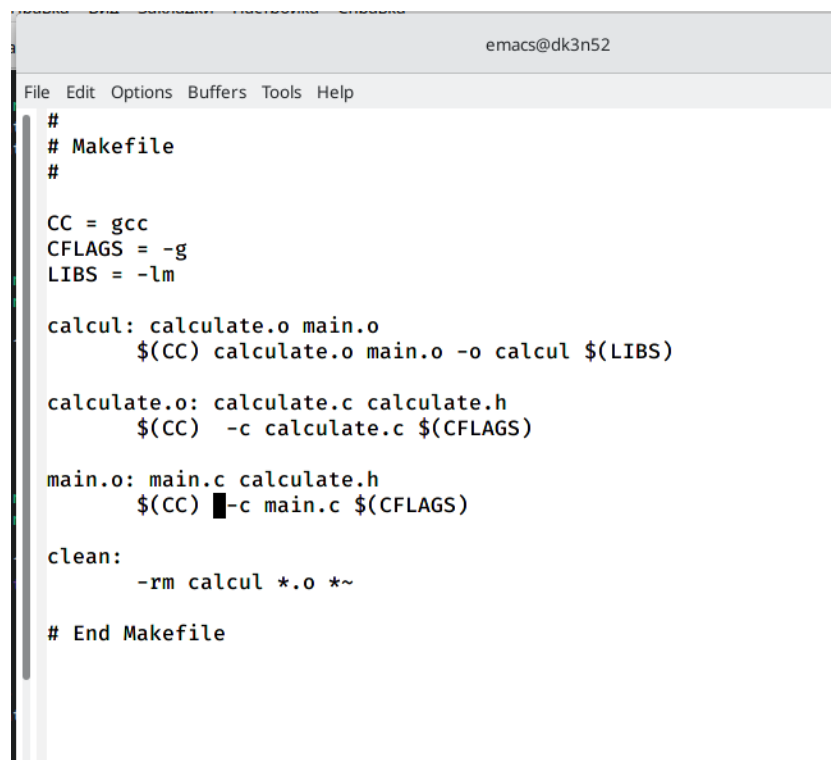
calculate.o: calculate.c calculate.h
    gcc -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
    gcc -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# End Makefile
```

Рис. 2.6: Исходный текст



```
#
# 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. 2.7: Измененный текст

```
evkurkina@dk3n52 ~/work/os/lab_prog $ make clean
rm calcul *.o *~
evkurkina@dk3n52 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
evkurkina@dk3n52 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
```

Рис. 2.8: удаление файлов и компиляция файлов

```
evkurkina@dk3n52 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
```

Рис. 2.9: Компиляция файлов

5) Запустила отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` (рис. 2.10).

```
evkurkina@dk3n52 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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: /afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/work/os/lab_prog/calcul
Число: █
```

Рис. 2.10: Запуск отладчика

6) Запустила программу внутри отладчика командой `run` (рис. 2.11).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/work/os/lab_prog/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
30.00
[Inferior 1 (process 7604) exited normally]
(gdb) █
```

Рис. 2.11: Команда run

7) Для постраничного (по 9 строк) просмотра исходного кода использовала команду `list` (рис. 2.12).

```

(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;
(gdb) list
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16     scanf("%s",Operation);
17     Result = Calculate(Numeral, Operation);
18     printf("%6.2f\n",Result);
19     return 0;
20 }
(gdb)

```

Рис. 2.12: Команда list

8) Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15 (рис. 2.13). Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29 (рис. 2.14).

```

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

```

Рис. 2.13: Команда просмотра с 12 по 15 строк

```

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

```

Рис. 2.14: Команда просмотра определенных строк

- 9) Установила точку отладки на 21 строке: `list calculate.c:20,27 break 21` (рис. 2.15).

```
(gdb) list calculate.c:20,27
20     printf("Вычитаемое: ");
21     scanf("%f",&SecondNumeral);
22     return(Numeral - SecondNumeral);
23 }
24 else if(strncmp(Operation, "*", 1) == 0)
25 {
26     printf("Множитель: ");
27     scanf("%f",&SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x5555555525f: file calculate.c, line 21.
(gdb)
```

Рис. 2.15: Установка точки останова

- 10) Вывела информацию об имеющихся точках останова: `info breakpoints` (рис. 2.16).

```
(gdb) info breakpoints
Num Type      Disp Enb Address          What
1   breakpoint keep y  0x00005555555525f in Calculate at calculate.c:21
(gdb)
```

Рис. 2.16: Информация о точках останова

- 11) Запустила программу внутри отладчика и убедилась, что программа останавливается в момент прохождения точки останова: `run 5 - backtrace`. Отладчик выдал следующую информацию: 1 #0 Calculate (Numeral=5, Operation=0x7fffffffcd280 "-") 2 at calculate.c:21 3 #1 0x0000000000400b2b in main () at main.c:17 (рис. 2.17).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffcd44 "-") at calculate.c:21
21     scanf("%f",&SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffcd44 "-") at calculate.c:21
#1 0x00005555555555a9 in main () at main.c:17
(gdb)
```

Рис. 2.17: Проверка точки останова

- 12) Просмотрела, чему равно на этом этапе значение переменной Numeral: `print Numeral` (рис. 2.18).

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

Рис. 2.18: Значение переменной

13) Сравнила с результатом вывода на экран после использования команды: `display Numeral` (рис. 2.19).

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

Рис. 2.19: Сравнение результатов

14) Убрала точку останова `info breakpoints delete` (рис. 2.20).

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint        keep y   0x000055555555525f in calculate at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Рис. 2.20: Удаление точки останова

15) С помощью утилиты `splint` проанализировала коды файлов `calculate.c` и `main.c` (рис. 2.21). (рис. 2.22).

```

evkurkina@dk3n52 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 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:9:31: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:15:5: 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:21:6: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:6: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:6: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:9: 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:37:12: Return value type double does not match declared type float:
    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:45:6: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:46:12: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:49:9: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:51:9: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:53:9: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:55:9: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:59:12: Return value type double does not match declared type float:
    (HUGE_VAL)

Finished checking --- 15 code warnings
evkurkina@dk3n52 ~/work/os/lab_prog $

```

Рис. 2.21: Анализ кода файла calculate.c

```

evkurkina@dk3n52 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 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:16:3: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
evkurkina@dk3n52 ~/work/os/lab_prog $

```

Рис. 2.22: Анализ кода файла main.c

## 16) Контрольные вопросы:

1). Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help(-h) для каждой команды.

2). Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

планирование, включающее сбор и анализ требований к функционалу и другим характеристикам;  
проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, создание архитектуры;  
непосредственная разработка приложения: кодирование – по сути создание исходного кода;  
документирование. Для создания исходного текста программы разработчик может воспользоваться компилятором.

3). Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) `.c` воспринимаются как программы на языке C, файлы с расширением `.cpp` – как файлы на языке C++, а файлы с расширением `.o` считаются объектными. Например, в команде `gcc -c main.c`: `gcc` по расширению (суффиксу) `.c` распознает тип файла для компиляции и формирует объектный модуль – файл с расширением `.o`. Если требуется получить исполняемый файл с определённым именем (например, `hello`), то требуется воспользоваться опцией `-o` в качестве параметра задать имя создаваемого файла: `gcc -o hello main.c`. 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 C калькулятора с простейшими функциями `main.c`.

4). Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

5). Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой `make`. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

6). Для работы с утилитой `make` необходимо в корне рабочего каталога с Вашим проектом создать файл с названием `makefile` или `Makefile`, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае `Makefile` имеет следующий синтаксис: `цели : зависимости <команда 1> ...`. Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и

список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды – собственно действия, которые необходимо выполнить для достижения цели. Общий синтаксис Makefile имеет вид: `target1 [target2...]:[:] [dependment1...][  
[  
(tab)commands] [#commentary][  
(tab)commands] [#commentary]`. Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться). Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (`\`). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса Makefile: `## Makefile for abcd.c  
CC = gcc  
CFLAGS =  
# Compile abcd.c normally  
abcd: abcd.c`

`(CFLAGS) abcd.c  
clean:-rm abcd.o ~# End Makefile for abcd.c`. В этом примере в начале файла заданы три переменные: `CC` и `CFLAGS`. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем `clean` производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

7). Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией `-g` компилятора `gcc`: `gcc -c file.c -g`. После этого для начала работы с `gdb` необходимо в командной стро-



ке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: `gdbfile.o`

#### 8). Основные команды отладчика gdb:

`backtrace` – вывод на экран пути к текущей точке останова (по сути вывод – названий функций, в которых произошла остановка);  
`break` – установить точку останова (в качестве параметра может быть указан номер строки в исходном коде);  
`clear` – удалить все точки останова в функции;  
`continue` – продолжить выполнение программы;  
`delete` – удалить точку останова;  
`display` – добавить выражение в список выражений, значения которых отображаются при выполнении программы;  
`finish` – выполнить программу до момента выхода из функции;  
`info breakpoints` – вывести на экран список используемых точек останова;  
`info watchpoints` – вывести на экран список используемых контрольных выражений;  
`list` – вывести на экран исходный код (в ходе выполнения данной лабораторной работы);  
`next` – выполнить программу пошагово, но без выполнения вызываемых в программе функций;  
`print` – вывести значение указываемого в качестве параметра выражения;  
`run` – запуск программы на выполнение;  
`set` – установить новое значение переменной;  
`step` – пошаговое выполнение программы;  
`watch` – установить контрольное выражение, при изменении значения которого программа будет остановлена.  
d. Более подробную информацию по работе с gdb можно получить с помощью команд `gdb` и `mangdb`.

#### 9). Схема отладки программы показана в 6 пункте лабораторной работы.

10). При первом запуске компилятор не выдал никаких ошибок, но в коде программы `main.c` допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке `scanf("%s", &Operation);` нужно убрать знак `&`, потому что имя массива символов уже является указателем на первый элемент этого массива.

11). Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: `gdb` – исследование

ние функций, содержащихся в программе, **lint** – критическая проверка программ, написанных на языке Си.

12). Утилита **splint** анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора Санализатор **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. # Выводы

Во время выполнения данной лабораторной работы, я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений ОС UNIX/LINUX на примере создания на языке программирования C калькулятора с простейшими функциями.

## **Список литературы**