

# **Отчёт по лабораторной работе №13**

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

Хусаинова Динара Айратовна

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Ход работы	8
4	Контрольные вопросы	14
5	Вывод	19

## Список иллюстраций

3.1	Создание файлов и каталога . . . . .	8
3.2	calculate.h . . . . .	8
3.3	calculate.c . . . . .	9
3.4	main.c . . . . .	9
3.5	Компиляция . . . . .	10
3.6	Makefile . . . . .	10
3.7	Запуск калькулятора . . . . .	10
3.8	Извлекаем квадратный корень и просматриваем строки файла . .	11
3.9	Просматриваем строки с 12 по 15 . . . . .	11
3.10	Просматриваем строки с 20 по 27 . . . . .	11
3.11	Ставим точки останова и наблюдаем, как останавливается програм- ма при попытке вычислить выражение . . . . .	12
3.12	Просматриваем значение переменной и удаляем точки останова .	12
3.13	Файл calculate.c . . . . .	13
3.14	Файл main.c . . . . .	13

## List of Tables

# 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 Теоретическое введение

Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

- планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
- проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
- непосредственная разработка приложения:
- кодирование — по сути создание исходного текста программы (возможно в нескольких вариантах);
- анализ разработанного кода;
- сборка, компиляция и разработка исполняемого модуля;
- тестирование и отладка, сохранение произведённых изменений;
- документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др.

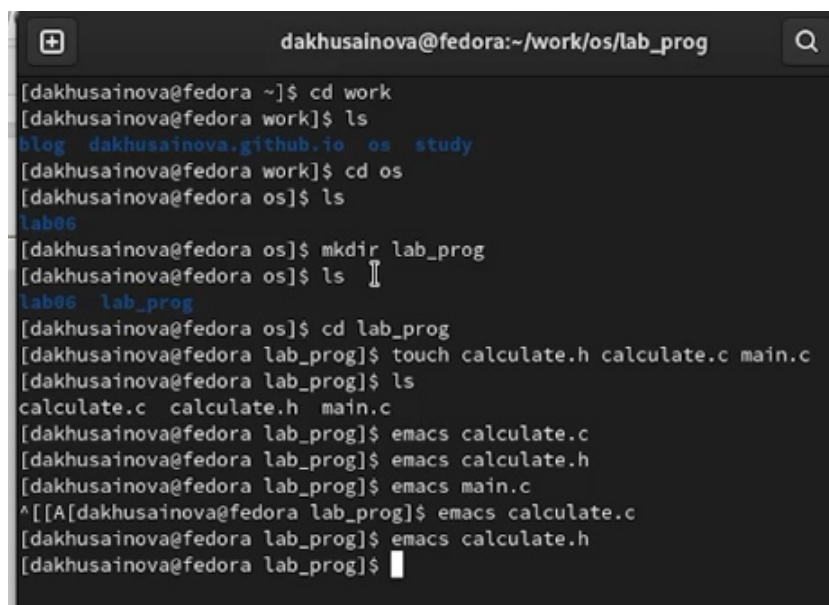
После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcs, которая интерпрети-

рует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C — как файлы на языке C++, а файлы с расширением .o считаются объектными.

## 3 Ход работы

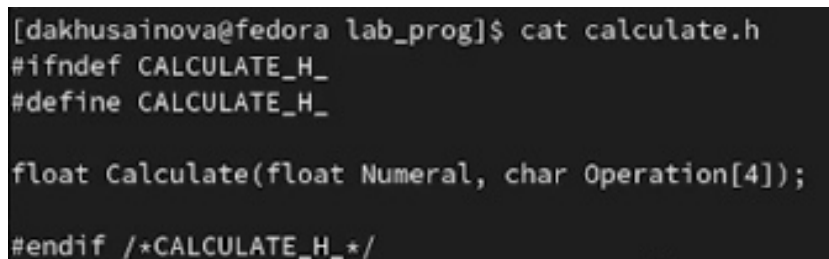
1. В домашнем каталоге создаем подкаталог `~/work/os/lab_prog`. А после создаем в нём файлы: `calculate.h`, `calculate.c`, `main.c`. (рис. 3.1).



```
dakhusainova@fedora:~/work/os/lab_prog
[dakhusainova@fedora ~]$ cd work
[dakhusainova@fedora work]$ ls
blog dakhusainova.github.io os study
[dakhusainova@fedora work]$ cd os
[dakhusainova@fedora os]$ ls
lab06
[dakhusainova@fedora os]$ mkdir lab_prog
[dakhusainova@fedora os]$ ls
lab06 lab_prog
[dakhusainova@fedora os]$ cd lab_prog
[dakhusainova@fedora lab_prog]$ touch calculate.h calculate.c main.c
[dakhusainova@fedora lab_prog]$ ls
calculate.c calculate.h main.c
[dakhusainova@fedora lab_prog]$ emacs calculate.c
[dakhusainova@fedora lab_prog]$ emacs calculate.h
[dakhusainova@fedora lab_prog]$ emacs main.c
^[[A[dakhusainova@fedora lab_prog]$ emacs calculate.c
[dakhusainova@fedora lab_prog]$ emacs calculate.h
[dakhusainova@fedora lab_prog]$
```

Рис. 3.1: Создание файлов и каталога

2. Реализация функций калькулятора в файле `calculate.h` (рис. 3.2).



```
[dakhusainova@fedora lab_prog]$ cat calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 3.2: `calculate.h`



3. Реализация функций калькулятора в файле calculate.c( рис. 3.3).

```
[dakhusainova@fedora lab_prog]$ cat 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);}
```

Рис. 3.3: calculate.c

4. Реализация функций калькулятора в файле main.c( рис. 3.4).

```
[dakhusainova@fedora lab_prog]$ cat 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;
}
[dakhusainova@fedora lab_prog]$
```

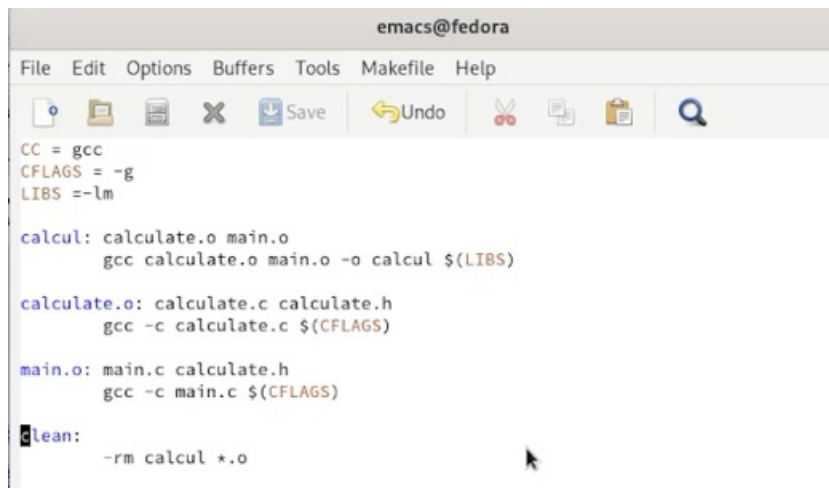
Рис. 3.4: main.c

5. Выполняем компиляцию программы посредством gcc( рис. 3.5).

```
[dakhusainova@fedora lab_prog]$ gcc -c -g calculate.c
[dakhusainova@fedora lab_prog]$ gcc -c -g main.c
[dakhusainova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рис. 3.5: Компиляция

6. Создаем Makefile со следующим содержанием( рис. 3.6).



```
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
```

Рис. 3.6: Makefile

7. Запускаем наш калькулятор и проверяем его работу. Просматриваем строки файлов, ставим точки останова, запускаем программу внутри отладчика и убеждаемся, что программа остановится в момент прохождения точки останова( рис. 3.7,3.8,3.9,3.10,3.11,3.12).

```
[dakhusainova@fedora 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.h
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/>
```

Рис. 3.7: Запуск калькулятора

```

Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sqrt
2.00
[Inferior 1 (process 6227) exited normally]
(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);
(gdb)

```

Рис. 3.8: Извлекаем квадратный корень и просматриваем строки файла

```

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

```

Рис. 3.9: Просматриваем строки с 12 по 15

```

(gdb) list calculate.c:20,27
20         scanf("%f",&SecondNumeral);
21         return(Numeral * SecondNumeral);}
22     else if(strncmp(Operation, "/", 1) == 0){
23         printf("Делитель: ");
24         scanf("%f",&SecondNumeral);
25         if(SecondNumeral == 0){
26             printf("Ошибка: деление на ноль! ");
27             return(HUGE_VAL);}

```

Рис. 3.10: Просматриваем строки с 20 по 27

```

21      return Numel_val;
(gdb) break 22
Breakpoint 2 at 0x401295: file calculate.c, line 22.
(gdb) run
Starting program: /home/dakhusainova/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /

Breakpoint 2, Calculate (Numeral=6, Operation=0x7fffffffdee4 "/") at
22      else if(strncmp(Operation, "/", 1) == 0){
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/dakhusainova/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 2, Calculate (Numeral=5, Operation=0x7fffffffdee4 "/") at
22      else if(strncmp(Operation, "/", 1) == 0){
(gdb)

```

Рис. 3.11: Ставим точки останова и наблюдаем, как останавливается программа при попытке вычислить выражение

```

(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint        keep y   0x0000000000401286 in Calculate at calculate.c:21
2     breakpoint        keep y   0x0000000000401295 in Calculate at calculate.c:22
      breakpoint already hit 1 time
(gdb) delete 1
(gdb) delete 2
Undefined command: "detele". Try "help".
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
2     breakpoint        keep y   0x0000000000401295 in Calculate at calculate.c:22
      breakpoint already hit 1 time
(gdb) delete 2
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)

```

Рис. 3.12: Просматриваем значение переменной и удаляем точки останова

8. С помощью утилиты splint проанализируем коды файлов calculate.c и main.c (рис. 3.13, 3.14).

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

calculate.h:4:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:7:31: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:5: 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:16:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:20:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:8: 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:27:13: Return value type double does not match declared type float:
```

Рис. 3.13: Файл calculate.c

```
[dakhusainova@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:4:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
main.c: (in function main)
main.c:10:3: 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:12:14: 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:12:11: Corresponding format code
main.c:12:3: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[dakhusainova@fedora lab_prog]$
```

Рис. 3.14: Файл main.c

## 4 Контрольные вопросы

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

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

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

Процесс разработки программного обеспечения обычно разделяется на следующие этапы: • планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения; • проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования; • непосредственная разработка приложения: • кодирование – по сути создание исходного текста программы (возможно в нескольких вариантах); • анализ разработанного кода; • сборка, компиляция и разработка исполняемого модуля; • тестирование и отладка, сохранение произведённых изменений; • документирование.

Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: `vi`, `vim`, `mceditor`, `emacs`, `geany` и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

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

Для имени входного файла суффикс определяет какая компиляция требуется.

Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C – 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 и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».

#### 4. Каково основное назначение компилятора языка C в UNIX?

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

#### 5. Для чего предназначена утилита make?

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. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

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: ... : ... <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.

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

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 строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdb file.o

8. Назовите и дайте основную характеристику основным командам отладчика gdb.

Основные команды отладчика gdb:

backtrace – вывод на экран пути к текущей точке останова (по сути вывод – названий всех функций)

break – установить точку останова (в качестве параметра может быть указан номер строки или название функции)



`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 достижении точки останова 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 качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк)  
`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

Для выхода из `gdb` можно воспользоваться командой `quit` (или её сокращённым вариантом `q`) или комбинацией клавиш `Ctrl-d`. Более подробную информацию по работе с `gdb` можно получить с помощью команд `gdb -h` и `man gdb`.

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

Схема отладки программы показана в 6 пункте лабораторной работы.

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

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

Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: • `cscope` – исследование функций, содержащихся в программе, • `lint` – критическая проверка программ, написанных на языке Си.

**12.** Каковы основные задачи, решаемые программой `splint`?

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 Си 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.

## 5 Вывод

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