

# **Отчёт к лабораторной работе №14**

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

Кекишева Анастасия Дмитриевна

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Вывод	15
5	Библиография	16

## Список таблиц

## Список иллюстраций

3.1	Файл calculate.c . . . . .	7
3.2	Файл calculate.h . . . . .	8
3.3	Файл main.c . . . . .	8
3.4	Компиляция программ . . . . .	9
3.5	Makefile . . . . .	9
3.6	Компиляция Makefile . . . . .	9
3.7	Отладку программы calcul . . . . .	10
3.8	Просмотр кода . . . . .	11
3.9	Установка и удаление точки останова . . . . .	12
3.10	Код файла calculate.c . . . . .	13
3.11	Код файла main.c . . . . .	13

# 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.

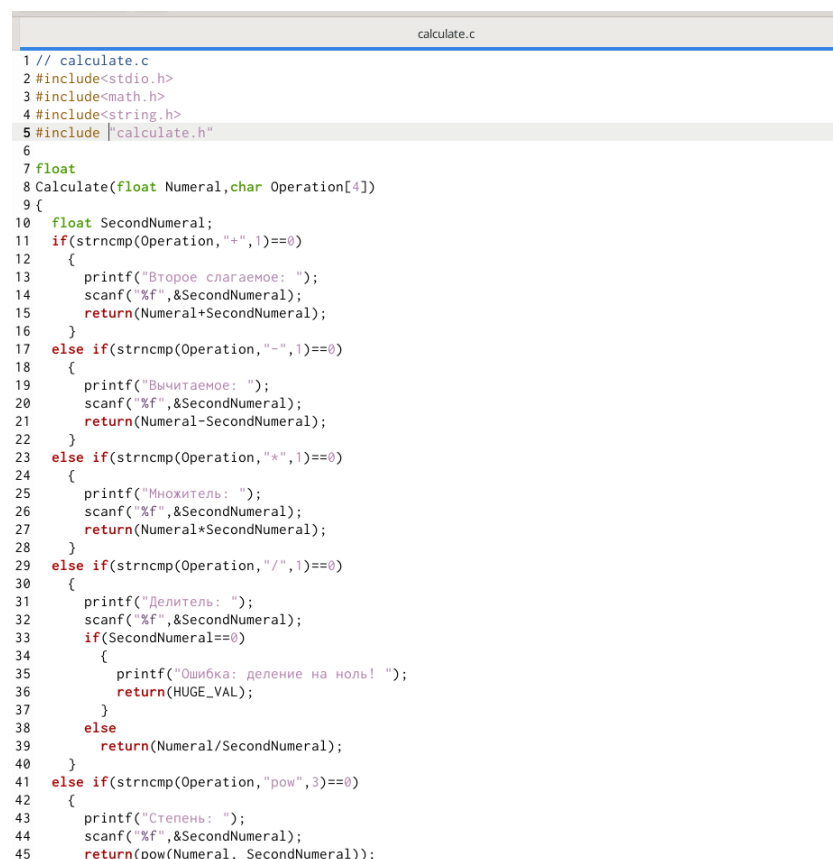
## 2 Задание

**Выполнить следующие пункты:**

1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`.
3. Выполните компиляцию программы посредством `gcc`:  
`> gcc -c calculate.c > gcc -c main.c > gcc calculate.o main.o -o calcul -lm`
4. Создайте `Makefile` со следующим содержанием.
5. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`)
6. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

### 3 Выполнение лабораторной работы

Перед выполнением лабораторной работы я хорошо ознакомилась с теоритическим материалом для её выполнения Ссылка 1



```
1 // calculate.c
2 #include<stdio.h>
3 #include<math.h>
4 #include<string.h>
5 #include "calculate.h"
6
7 float
8 Calculate(float Numeral, char Operation[4])
9 {
10     float SecondNumeral;
11     if(strncmp(Operation, "+", 1) == 0)
12     {
13         printf("Второе слагаемое: ");
14         scanf("%f", &SecondNumeral);
15         return(Numeral + SecondNumeral);
16     }
17     else if(strncmp(Operation, "-", 1) == 0)
18     {
19         printf("Вычитаемое: ");
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)
30     {
31         printf("Делитель: ");
32         scanf("%f", &SecondNumeral);
33         if(SecondNumeral == 0)
34         {
35             printf("Ошибка: деление на ноль! ");
36             return(HUGE_VAL);
37         }
38         else
39             return(Numeral / SecondNumeral);
40     }
41     else if(strncmp(Operation, "pow", 3) == 0)
42     {
43         printf("Степень: ");
44         scanf("%f", &SecondNumeral);
45         return(pow(Numeral, SecondNumeral));
```

Рис. 3.1: Файл calculate.c

```

calculate.h
1 ///////////////////////////////////////////////////
2 //calculate.h
3 #ifndef CALCULATE_H_
4 #define CALCULATE_H_
5
6 float Calculate(float Numeral, char Operation[4]);
7
8 #endif /*CALCULATE_H_*/

```

Рис. 3.2: Файл calculate.h

```

main.c
1 //main.c
2 #include <stdio.h>
3 #include "calculate.h"
4 int main(void)
5 {
6     float Numeral;
7     char *Operation;
8     float Result;
9     printf("Число: ");
10    scanf("%f", &Numeral);
11    printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
12    scanf("%s", Operation);
13    Result = Calculate(Numeral, Operation);
14    printf("%.2f\n", Result);
15    return 0;
16 }

```

Рис. 3.3: Файл main.c

1. В домашнем каталоге создала подкаталог ~/work/os/lab\_prog.
2. Создала в нём файлы: calculate.h (рис. 3.1), calculate.c (рис. 3.2), main.c (рис. 3.3). Это примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается.



```

adkekisheva@dk8n78 ~ $ cd work
adkekisheva@dk8n78 ~/work $ cd os
adkekisheva@dk8n78 ~/work/os $ cd lab_prog/
adkekisheva@dk8n78 ~/work/os/lab_prog $ gcc -c calculate.c
adkekisheva@dk8n78 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:12:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
   12 |     scanf("%s", &operation);
       |           ^~
       |           |
       |           | char (*)[4]
       |           |
       |           char *
adkekisheva@dk8n78 ~/work/os/lab_prog $ gcc -c main.c
adkekisheva@dk8n78 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm

```

Рис. 3.4: Компиляция программ

3. Выполнила компиляцию программы посредством gcc (рис. 3.4):

gcc -c calculate.c

gcc -c main.c

gcc calculate.o main.o -o calcul -lm.

Сначала у меня вылезла ошибка при компиляции main.c, чтобы её исправить, я сделал переменную Operation указателем на тип char. И в scanf передавала не ссылку, а значение этой переменной.

```

1 #
2 #Makefile
3 #
4 CC = gcc
5 CFLAGS = -g
6 LIBS = -lm
7
8 calcul: calculate.o main.o
9     gcc calculate.o main.o -o calcul $(LIBS)
10 calculate.o: calculate.c calculate.h
11     gcc -c calculate.c $(CFLAGS)
12 main.o: main.c calculate.h
13     gcc -c main.c $(CFLAGS)
14 clean:
15     -rm calcul *.o *~
16 # End Makefile

```

Рис. 3.5: Makefile

```

adkekisheva@dk8n78 ~/work/os/lab_prog $ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm

```

Рис. 3.6: Компиляция Makefile

4. Создала Makefile (рис. 3.5). После знака = перед CFLAGS поставила опцию -g, чтобы отладочная информация содержалась в результирующем бинарном файле. Его содержание следующее: сначала задаётся список целей, в качестве цели в Makefile может выступать имя файла или название какого-то действия разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами начинаются с табуляции. Зависимость задаёт исходные параметры (условия), которые необходимо выполнить для достижения цели. Далее, скомпилировала программы командой make (рис. 3.6).

```
adkekisheva@dk8n78 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.1 vanilla) 10.1
Copyright (C) 2020 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/a/d/adkekisheva/work/os/lab_prog/calcul
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
10.00
[Inferior 1 (process 5601) exited normally]
```

Рис. 3.7: Отладку программы calcul

6. С помощью gdb выполнила отладку программы calcul:

- Запустила отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` (рис. 3.7);
- Для запуска программы внутри отладчика ввела команду `run` (рис. 3.7);

```

(gdb) list
1      //main.c
2      #include <stdio.h>
3      #include "calculate.h"
4      int main(void)
5      {
6          float Numeral;
7          char *Operation;
8          float Result;
9          printf("Число: ");
10         scanf("%f",&Numeral);
(gdb) list 12,15
12         scanf("%s",Operation);
13         Result=Calculate(Numeral, Operation);
14         printf("%.2f\n",Result);
15         return 0;
(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)

```

Рис. 3.8: Просмотр кода

- Для постраничного (по 9 строк) просмотра исходного код воспользовалась командой list (рис. 3.8);
- Для просмотра строк с 12 по 15 основного файла воспользовалась командой list с параметрами: list 12,15;
- Для просмотра определённых строк не основного файла использовала list с параметрами: list calculate.c:20,29;

```

(gdb) break 21
Breakpoint 1 at 0x55555540097e: file calculate.c, line 21.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000055555540097e in Calculate at calculate.c:21
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/d/adkekisheva/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 4

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffff010 "-") at calculate.c:21
21 return(Numeral-SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffff010 "-") at calculate.c:21
#1 0x0000555555400c31 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000055555540097e in Calculate at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.

```

Рис. 3.9: Установка и удаление точки останова

- Установила точку останова в файле calculate.c на строке номер 21 командой `break 21` (рис. 3.9);
- Вывела информацию об имеющихся в проекте точка останова `info breakpoints`;
- Запустила программу внутри отладчика и убедилась, что программа останавливается в момент прохождения точки останова;
- Посмотрела, чему равно на этом этапе значение переменной `Numeral`, введя: `print Numeral`. На экран вывелось число 5.
- Сравнила с результатом вывода на экран после использования команды: `display Numeral`;
- Удалила точку останова `delete 1`;

```

adkekisheva@dk8n78 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:36: 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:8:30: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:14:7: 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:20:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:26:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:10: 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:36:10: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:44:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:45:13: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:48:11: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:50:11: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:52:11: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:54:11: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:58:13: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings

```

Рис. 3.10: Код файла calculate.c

```

Finished checking --- 4 code warnings
adkekisheva@dk8n78 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:36: 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: Unallocated storage Operation passed as out parameter to scanf:
        Operation
    An rvalue is used that may not be initialized to a value on some execution
    path. (Use -usedef to inhibit warning)
main.c:12:3: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 4 code warnings
adkekisheva@dk8n78 ~/work/os/lab_prog $

```

Рис. 3.11: Код файла main.c

7. С помощью утилиты splint проанализировала коды файлов calculate.c (рис. 3.10) и main.c (рис. 3.11). Для файла main.c: первое предупреждение говорит о том, что параметр Operation воспринимается как массив. Далее, это

это главная функция, эта функция возвращает целочисленный тип. Размер массива в этом контексте игнорируется, так как формальный параметр массива обрабатывается как указатель. Используется значение `value`, которое не может быть инициализировано в значение на некотором пути выполнения.

Для `calculate.c`: результат, возвращаемый вызовом функции, не используется. Если это предназначено, можно привести результат к `(void)`, чтобы исключить сообщение. Опасное сравнение равенства с использованием типов с плавающей точкой: второе число `== 0`. Два реальных значения (`float`, `double` или `long double`) сравниваются непосредственно с помощью `!=` примитивно. Это может привести к неожиданным результатам, поскольку представления с плавающей запятой неточны.

## 4 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.

## 5 Библиография

1. Ссылка 1

### **Контрольные вопросы:**

1. Информацию об этих программах можно получить с помощью функций `info` и `man`.
2. 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. 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 -p1 выводит префиксы в форме которая подходит для команды patch -p1.

4. Основное назначение компилятора make языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.
6. В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. 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. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.

7. Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных

переменных, а также внутренних регистров микроконтроллера и напряжений

на выводах этой микросхемы.

8. `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

9.

- Выполнила компиляцию программы;
- Увидела ошибки в программе;

- Открыла редактор и исправила программу;
  - Загрузила программу в отладчик gdb: run — отладчик выполнил программу, ввела требуемые значения.
  - Использовала другие команды отладчика и проверила работу программы;
10. Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
11. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным.
- Система

разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

- cscope - исследование функций, содержащихся в программе;
- splint — критическая проверка программ, написанных на языке Си.

12.

- Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
- Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
- Общая оценка мобильности пользовательской программы.