

# Отчет

Лабораторная работа №13

Павлова Варвара Юрьевна НПМбд-02-21

# Содержание

Цель работы	5
Задание	6
Теоретическое введение	8
Выполнение лабораторной работы	9
Выводы	19
Список литературы	20

# Список иллюстраций

0.1	создание подкаталога . . . . .	9
0.2	создание файлов . . . . .	9
0.3	calculate.c . . . . .	10
0.4	calculate.c . . . . .	10
0.5	calculate.h . . . . .	11
0.6	main.c . . . . .	11
0.7	компиляция программы . . . . .	11
0.8	создание файла . . . . .	12
0.9	создание Makefile . . . . .	12
0.10	изменение файла . . . . .	13
0.11	makefile . . . . .	13
0.12	запуск отладчика . . . . .	14
0.13	запуск программы . . . . .	14
0.14	постраничный просмотр . . . . .	15
0.15	просмотр строк 12-15 . . . . .	15
0.16	просмотр строк не основного файла . . . . .	15
0.17	установка точки останова . . . . .	16
0.18	вывод информации о точках остановок . . . . .	16
0.19	запуск внутри отладчика . . . . .	16
0.20	сравнение значений Numeral . . . . .	17
0.21	удаление точек остановок . . . . .	17
0.22	calculate.c . . . . .	17
0.23	main.c . . . . .	18

## Список таблиц

## 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.

# Задание

1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.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.
3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`): Запустите отладчик GDB, загрузив в него программу для отладки.
  - Для запуска программы внутри отладчика введите команду `run`.
  - Постранично (по 9 строк) просмотрите исходный код.
  - Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами.
  - Для просмотра определённых строк не основного файла используйте `list` с параметрами.
  - Установите точку остановки в файле `calculate.c` на строке номер 21.
  - Выведите информацию об имеющихся в проекте точках остановок.
  - Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки остановки.

- Посмотрите, чему равно на этом этапе значение переменной `Numeral`.
  - Сравните с результатом вывода на экран.
  - Уберите точки остановок.
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

# 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.



# Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог `~/work/os/lab_prog`. (рис. [-@fig:001])

```
[vypavlova@fedora ~]$ mkdir ~/work/os/lab_prog  
[vypavlova@fedora ~]$
```

Рис. 0.1: создание подкаталога

2. Создаю в нём файлы: `calculate.h`, `calculate.c`, `main.c`. (рис. [-@fig:002]) Пишу код для работы программы в созданных файлах. (рис. [-@fig:003]) (рис. [-@fig:004]) (рис. [-@fig:005]) (рис. [-@fig:006])

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

Рис. 0.2: создание файлов

```

////////////////////////////////////
// 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. 0.3: calculate.c

```

    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));
    else if(strncmp(Operation, "tan", 3) == 0)
        return(tan(Numeral));
    else
    {
        printf("Неправильно введено действие");
        return(HUGE_VAL);
    }
}

```

U:--- calculate.c Bot L34 (C/\*l Abbrev)

Рис. 0.4: calculate.c

```

////////////////////////////////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

```

Рис. 0.5: calculate.h

```

////////////////////////////////////
// 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. 0.6: main.c

3. Выполняю компиляцию программы посредством gcc.(рис. [-@fig:007]) Ошибок не нашлось.

```

[vyurvlova@fedora lab_prog]$ gcc -c calculate.c
[vyurvlova@fedora lab_prog]$ gcc -c main.c
[vyurvlova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm

```

Рис. 0.7: компиляция программы

4. Создаю Makefile. (рис. [-@fig:008]) Пишу в файле текст, указанный в лабораторной работе. (рис. [-@fig:009]) Меняю текст, чтобы я могла работать с отладчиком. (рис. [-@fig:010]) Запускаю Makefile. (рис. [-@fig:011])

```
[vypavlova@fedora lab_prog]$ touch Makefile
[vypavlova@fedora lab_prog]$ emacs &
[1] 4848
```

Рис. 0.8: создание файла

```
#
# 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. 0.9: создание Makefile

```

#
# 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. 0.10: изменение файла

```

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

```

Рис. 0.11: makefile

6. С помощью gdb выполняю отладку программы calcul: Запускаю отладчик GDB, загрузив в него программу для отладки. (рис. [-@fig:012])
  - Для запуска программы внутри отладчика ввожу команду run. (рис. [-@fig:013])
  - Постранично (по 9 строк) просматриваю исходный код. (рис. [-@fig:014])
  - Для просмотра строк с 12 по 15 основного файла использую list с параметрами 12,15. (рис. [-@fig:015])

- Для просмотра определённых строк не основного файла использую list с параметрами имя\_файла: 20,29. (рис. [-@fig:016])
- Устанавливаю точку остановки в файле calculate.c на строке номер 21. (рис. [-@fig:017])
- Вывожу информацию об имеющихся в проекте точках остановок. (рис. [-@fig:018])
- Запускаю программу внутри отладчика и убеждаюсь, что программа останавливается в момент прохождения точки остановки. (рис. [-@fig:019])
- Посматриваю, чему равно на этом этапе значение переменной Numeral и сравниваю с результатом вывода на экран. (рис. [-@fig:020])
- Убираю точки остановок. (рис. [-@fig:021])

```
[vypavlova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 10.2-9.fc35
Copyright (C) 2021 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:
```

Рис. 0.12: запуск отладчика

```
(gdb) run
Starting program: /home/vypavlova/work/os/lab_prog/calcul
Downloading separate debug info for /home/vypavlova/work/os/lab_p
lied DSO at 0x7ffff7fc9000...
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: 2
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое4
6.00
[Inferior 1 (process 5457) exited normally]
(gdb) □
```

Рис. 0.13: запуск программы

```

[Inferior 1 (process 3437) exited normally]
(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) █

```

Рис. 0.14: постраничный просмотр

```

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

```

Рис. 0.15: просмотр строк 12-15

```

(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);
(gdb) █

```

Рис. 0.16: просмотр строк не основного файла

```
(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.
```

Рис. 0.17: установка точки останова

```
(gdb) info breakpoints
Num      Type             Disp Enb Address              What
1        breakpoint      keep y   0x000000000040120f in calculate
                                                at calculate.c:21
```

Рис. 0.18: вывод информации о точках остановок

```
(gdb) run
Starting program: /home/vypavlova/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=0x7fffffffdf34 "-")
at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf34 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:17
```

Рис. 0.19: запуск внутри отладчика



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

Рис. 0.20: сравнение значений Numeral

```
(gdb) info breakpoints
Num    Type             Disp Enb Address                  What
1      breakpoint      keep y   0x000000000040120f in calculate
                                           at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
```

Рис. 0.21: удаление точек остановок

7. С помощью утилиты splint анализирую коды файлов calculate.c и main.c.(рис. [-@fig:022])(рис. [-@fig:023])

```
[vypavlova@fedora 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
```

Рис. 0.22: calculate.c

```
[vypavlova@fedora 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:16:15: 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
```

Рис. 0.23: main.c

## Выводы

Выполняя данную лабораторную работу 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.

## Список литературы