

Отчёт по лабораторной работе

Грузинова Елизавета Константиновна

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

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.

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.

Тексты реализаций функций калькулятора в файле `calculate.h`, интерфейсного файла `calculate.h`, описывающий формат вызова функции-калькулятора и основного файла `main.c`, реализующий интерфейс пользователя к калькулятору приведены при выполнении лабораторной работы.

3. Выполните компиляцию программы посредством gcc:

```
1 gcc -c calculate.c 2 gcc -c main.c 3 gcc calculate.o main.o -o calcul -lm
```

4. При необходимости исправьте синтаксические ошибки.

5. Создайте Makefile со следующим содержанием (текст также приведен при выполнении лабораторной работы). Поясните в отчёте его содержание.

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

– Запустите отладчик GDB, загрузив в него программу для отладки:

```
1 gdb ./calcul
```

– Для запуска программы внутри отладчика введите команду run:

```
1 run
```

– Для постраничного (по 9 строк) просмотра исходного код используйте команду `list`:

```
1 list
```

– Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами:

```
1 list 12,15
```


– Для просмотра определённых строк не основного файла используйте `list` с параметрами:

```
1 list calculate.c:20,29
```

– Установите точку останова в файле `calculate.c` на строке номер 21:

```
1 list calculate.c:20,27
```

```
2 break 21
```

– Выведите информацию об имеющихся в проекте точка останова:

1 info breakpoints

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:

1 run

2 5

3 -

4 backtrace

– Отладчик выдаст следующую информацию:

```
1 #0 Calculate (Numeral=5, Operation=0x7fffffff280 "-") 2 at calculate.c:21 3  
#1 0x000000000400b2b in main () at main.c:17
```

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места.

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя

```
1 print Numeral
```

На экран должно быть выведено число 5.

– Сравните с результатом вывода на экран после использования команды:

1 display Numeral

– Уберите точки останова:

1 info breakpoints 2 delete 1

7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

Этапы разработки приложений.

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 создание подкаталога ~/work/os/lab_prog.
(рис. 1)

```
[ekgruzinova@fedora ~]$ mkdir lab_prog  
[ekgruzinova@fedora ~]$ cd lab_prog/
```

Figure 1: Создала каталог lab_prog и перешла в него

2. Создание в нём файлов: calculate.h, calculate.c, main.c. (рис. 2)

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

Figure 2: Создала необходимые файлы и проверила с помощью ls

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. Тексты реализаций функций калькулятора в файле `calculate.h`, интерфейсного файла `calculate.h`, описывающий формат вызова функции-калькулятора и основного файла `main.c`, реализующий интерфейс пользователя к калькулятору приведены при выполнении лабораторной работы. (рис. 3, 4, 5, 6, 7)

```
#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)
```

Figure 3: Текст файла calculate.c (1)

Выполнение лабораторной работы

```
{
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));
}
```

Figure 4: Текст файла calculate.c (2)

```
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);
}
```

Figure 5: Текст файла calculate.c (3)

```
#ifndef CALCULATE_H_  
#define CALCULATE_H_  
  
float Calculate(float Numeral, char Operation[4]);  
  
#endif /*CALCULATE_H_*/
```

Figure 6: Текст файла calculate.h (1)

```
#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;
}
```

Figure 7: Текст файла main.c

3. Выполнение компиляции программы посредством gcc (рис. 8):

```
1 gcc -c calculate.c 2 gcc -c main.c 3 gcc calculate.o main.o -o calcul -lm
```

```
[ekgruzinova@fedora lab_prog]$ gcc -c calculate.c
[ekgruzinova@fedora lab_prog]$ gcc -c main.c
[ekgruzinova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[ekgruzinova@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
```

Figure 8: Компиляция через терминал

4. При необходимости исправьте синтаксические ошибки (их не было обнаружено).
5. Создание Makefile со следующим содержанием (текст также приведен при выполнении лабораторной работы). Поясните в отчёте его содержание. (рис. 9)

Выполнение лабораторной работы

```
1 #
2 # Makefile
3 #
4
5 CC=gcc
6 CFLAGS=
7 LIBS=-lm
8
9 calcul: calculate.o main.o
10     gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13     gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16     gcc -c main.c $(CFLAGS)
17
18 clean:
19     -rm calcul *.o *~
20
21 # End Makefile
```

Figure 9: Текст в файле Makefile

6. С помощью gdb выполнение отладки программы calcul (перед использованием gdb исправьте Makefile) (рис. 10, 11):

```
#  
# Makefile  
#  
  
CC=gcc  
CFLAGS=  
LIBS=-lm  
  
calcul: calculate.o main.o  
    gcc calculate.o main.o -o calcul -g $(LIBS)  
  
calculate.o: calculate.c calculate.h  
    gcc -c calculate.c $(CFLAGS) -g  
  
main.o: main.c calculate.h  
    gcc -c main.c $(CFLAGS) -g  
  
clean:  
    -rm calcul *.o *~  
  
# End Makefile
```

Figure 10: Изменения в файле Makefile

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

Figure 11: Запуск измененного Makefile

- Запустите отладчик GDB, загрузив в него программу для отладки (рис. 12):

```
1 gdb ./calcul
```

```
[ekgruzinova@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.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Figure 12: Вызов для программы calcul отладчика GDB

– Для запуска программы внутри отладчика введите команду `run` (рис. 13):

1 `run`

```
(gdb) run
Starting program: /home/ekgruzinova/lab_prog/calcul

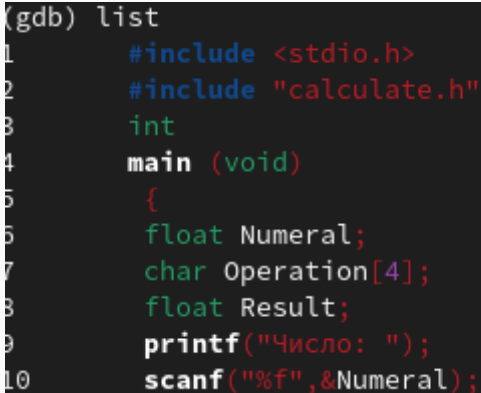
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
5.00
[Inferior 1 (process 10245) exited normally]
```

Figure 13: Запуск программы внутри отладчика

Выполнение лабораторной работы

- Для постраничного (по 9 строк) просмотра исходного код используйте команду list (рис. 14):

1 list



```
(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);
```

Figure 14: Вывод текста программы в отладчике

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами (рис. 15):

1 list 12,15

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

Figure 15: Вывод определенного текста программы в отладчике

– Для просмотра определённых строк не основного файла используйте `list` с параметрами (рис. 16):

1 `list calculate.c:20,29`

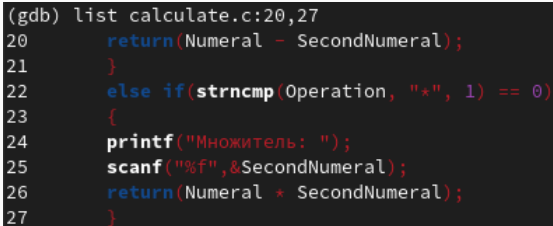
```
(gdb) list calculate.c:20,29
20     return(Numeral - SecondNumeral);
21     }
22     else if(strncmp(Operation, "*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strncmp(Operation, "/",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

Figure 16: Вывод определенного текста не основного файла в отладчике

Выполнение лабораторной работы

– Установите точку останова в файле calculate.c на строке номер 21 (рис. 17, 18):

1 list calculate.c:20,27



```
(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     }
```

Figure 17: Вывод необходимого образца текста

2 break 21

```
(gdb) break 21  
Breakpoint 1 at 0x401247: file calculate.c, line 22.
```

Figure 18: Создание точки останова

– Выведите информацию об имеющихся в проекте точка останова (рис. 19):

1 info breakpoints

```
(gdb) info breakpoints
Num    Type           Disp Enb Address            What
1      breakpoint    keep y   0x0000000000401247 in calculate
                                at calculate.c:22
```

Figure 19: Информация о точках останова

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова (рис. 20):

1 run

2 5

3 -

4 backtrace

```
(gdb) run
Starting program: /home/ekgruzinova/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 5, Calculate (Numeral=5, Operation=0x7fffffffdf54 "*") at calculate.c:22
22     else if (strcmp (Operation, "+") == 0)
```

Figure 20: Запуск программы после установления точки останова

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места (рис. 21).

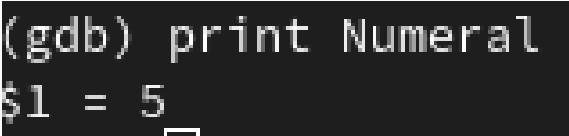
```
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf54 "/") at calculate.c:22
#1  0x00000000004014eb in main () at main.c:13
```

Figure 21: Работа команды backtrace

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя

```
1 print Numeral
```

На экран должно быть выведено число 5 (рис. 22).

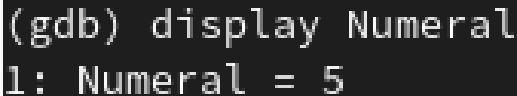


```
(gdb) print Numeral
$1 = 5
```

Figure 22: Работа команды print в отладчике

– Сравните с результатом вывода на экран после использования команды (рис. 23):

1 display Numeral



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

Figure 23: Работа команды display в отладчике

– Уберите точки останова (рис. 24):

1 info breakpoints 2 delete 1

```
(gdb) info breakpoints
Num      Type           Disp Enb Address            What
5        breakpoint     keep y   0x0000000000401247 in calculate at calculate.c:22
breakpoint already hit 1 time
(gdb) delete 5
(gdb) info breakpoints
No breakpoints or watchpoints.
```

Figure 24: Удаление точки останова

7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`. (рис. 25, 26)

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

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:31:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:5: Dangerous equality comparison involving float types:
                    SecondNumeral == 0
```

Figure 25: Анализ файла calculate.c


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

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)
main.c: (in function main)
main.c:10:2: 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:12:13: 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 --- 4 code warnings
```

Figure 26: Анализ файла main.c

Выводы

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

Спасибо за внимание!