

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

**Лабораторная работа No 13. Средства, применяемые при разработке
программного обеспечения в ОС типа UNIX/Linux**

Абдуллина Ляйсан Раисовна НПИбд-01-21

Содержание

1	Цель работы	4
2	Теоретическое введение	5
3	Выполнение лабораторной работы	6
3.1	1	6
3.2	2	6
3.3	3	9
3.4	4	9
3.5	5	9
3.6	6	10
3.7	7	18
4	Контрольные вопросы	20
5	Выводы	25
6	Список литературы	26

Список иллюстраций

3.1	Созданный каталог	6
3.2	Созданные файлы	7
3.3	Реализация функций калькулятора в файле calculate.c	7
3.4	Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора	8
3.5	Основной файл main.c, реализующий интерфейс пользователя к калькулятору	8
3.6	Созданные файлы	9
3.7	Созданный файл	10
3.8	Исправленный файл	11
3.9	Компиляция файлов	11
3.10	Запуск отладчика	12
3.11	Запуск программы внутри отладчика	12
3.12	Просмотр кода (постранично)	13
3.13	Просмотр строк 12-15	13
3.14	Просмотр строк не основного файла	14
3.15	Установка точки останова	14
3.16	Информация о точках останова	15
3.17	Остановка у точки	15
3.18	Значение Numeral	16
3.19	Значение Numeral	17
3.20	Убираем точки останова	17
3.21	Загрузка утилиты	18
3.22	Информация о calculate.c	18
3.23	Информация о main.c	19

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.

3 Выполнение лабораторной работы

3.1 1

В домашнем каталоге создадим подкаталог `~/work/os/lab_prog` - через команду `mkdir` (скриншот 3.1)


A screenshot of a terminal window titled "Терминал" (Terminal). The window shows a user named "lrabdullina" on a "fedora" system, working in the directory "~/work/os". The terminal output shows the following commands and results:

```
[lrabdullina@fedora ~]$ cd work/os/  
[lrabdullina@fedora os]$ mkdir lab_prog  
[lrabdullina@fedora os]$ ls  
lab06  lab_prog  
[lrabdullina@fedora os]$
```

Скриншот 3.1: Созданный каталог

3.2 2

Создадим в нём файлы: `calculate.h`, `calculate.c`, `main.c` - через команду `touch` (скриншот 3.1)

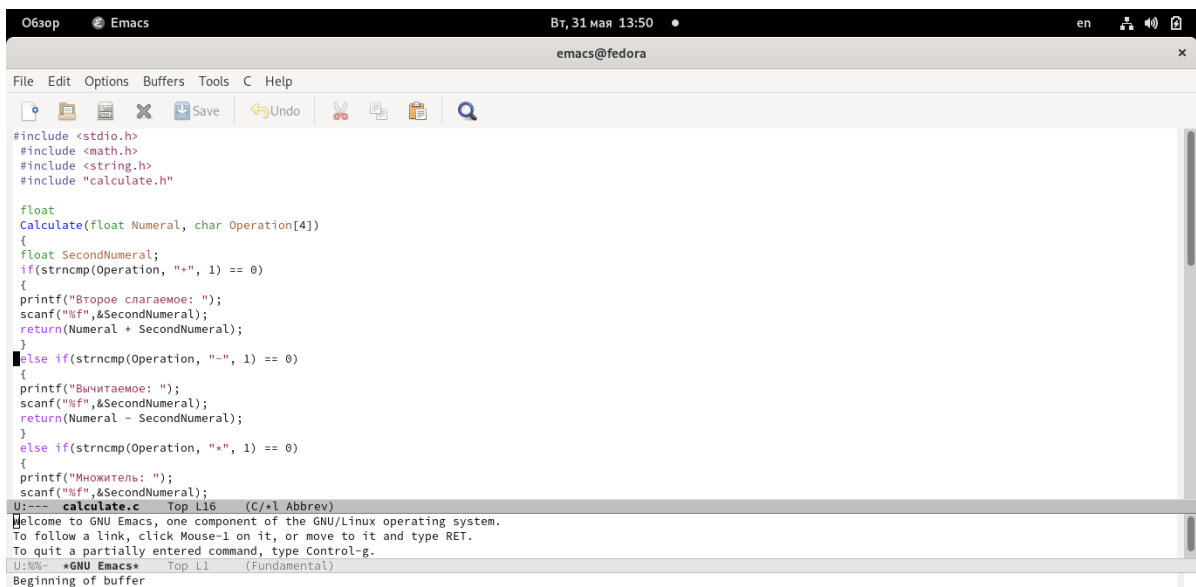


```
Обзор Терминал Вт, 31 мая 13:41 en
lrabdu1lina@fedora:~/work/os/lab_prog

[lrabdu1lina@fedora ~]$ cd work/os/
[lrabdu1lina@fedora os]$ mkdir lab_prog
[lrabdu1lina@fedora os]$ ls
lab06 lab_prog
[lrabdu1lina@fedora os]$ cd lab
bash: cd: lab: Нет такого файла или каталога
[lrabdu1lina@fedora os]$ cd lab_prog/
[lrabdu1lina@fedora lab_prog]$ touch calculate.h calculate.c main.c.
[lrabdu1lina@fedora lab_prog]$ ls
calculate.c calculate.h main.c.
[lrabdu1lina@fedora lab_prog]$
```

Скриншот 3.2: Созданные файлы

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 выведет результат и останавливается. (скриншоты 3.3, 3.4, 3.5)



```
Обзор Emacs Вт, 31 мая 13:50 en
emacs@fedora

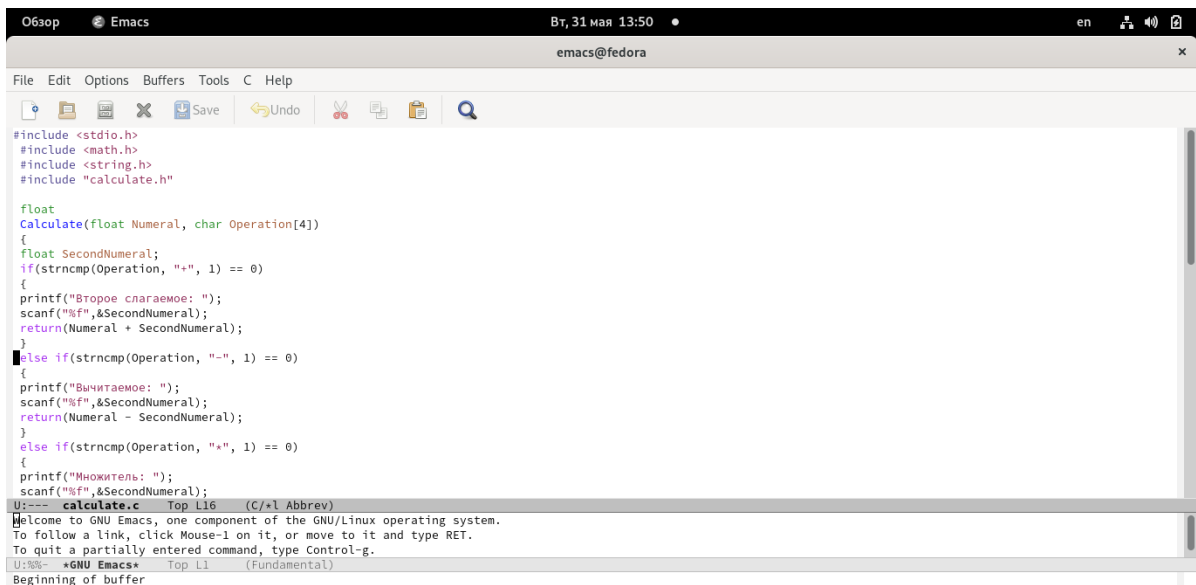
File Edit Options Buffers Tools C Help

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strcmp(Operation, "+", 1) == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-", 1) == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strcmp(Operation, "*", 1) == 0)
    {
        printf("Множитель: ");
        scanf("%f", &SecondNumeral);
    }
}

U:--- calculate.c Top L16 (C/*l Abbrev)
Welcome to GNU Emacs, one component of the GNU/Linux operating system.
To follow a link, click Mouse-1 on it, or move to it and type RET.
To quit a partially entered command, type Control-g.
U:%%- *GNU Emacs* Top L1 (Fundamental)
Beginning of buffer
```

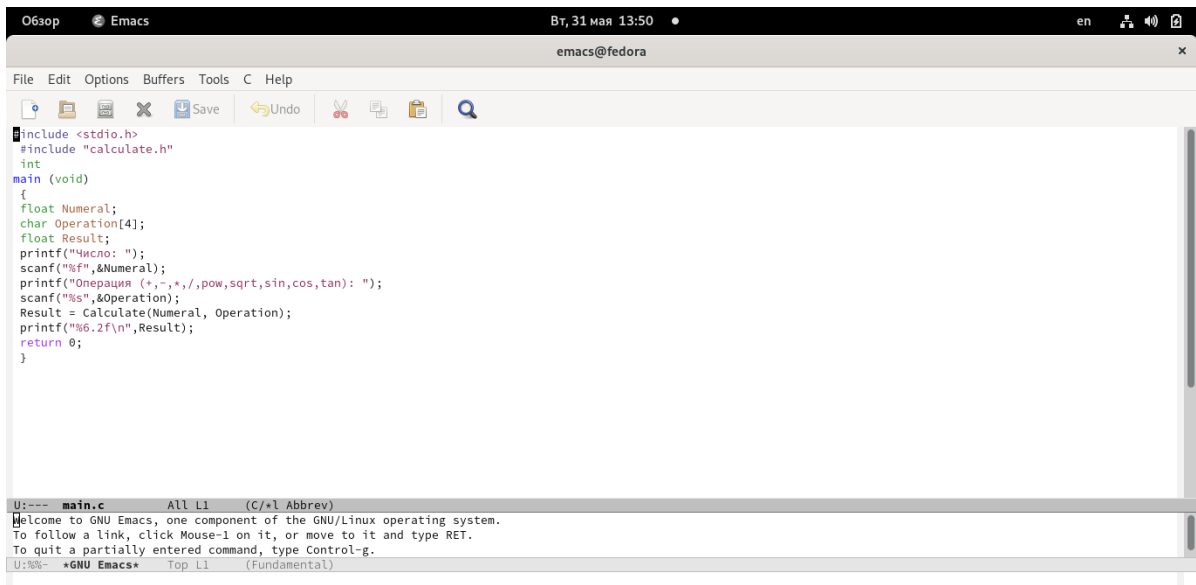
Скриншот 3.3: Реализация функций калькулятора в файле calculate.c

A screenshot of the Emacs editor window titled 'emacs@fedora'. The window shows the 'calculate.h' header file. The code includes standard headers like <stdio.h>, <math.h>, and <string.h>. It defines a 'Calculate' function that takes a float 'Numeral' and a char array 'Operation' of size 4. The function uses 'strcmp' to check the operation and performs addition, subtraction, or multiplication. The status bar at the bottom indicates 'U:--- calculate.c Top L16 (C/*l Abbrev)' and 'Beginning of buffer'.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strcmp(Operation, "+") == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-") == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strcmp(Operation, "*") == 0)
    {
        printf("Множитель: ");
        scanf("%f",&SecondNumeral);
    }
}
```

Скриншот 3.4: Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора

A screenshot of the Emacs editor window titled 'emacs@fedora'. The window shows the 'main.c' source file. The code includes 'stdio.h' and 'calculate.h'. The 'main' function prompts the user for a number and an operation, calls the 'Calculate' function, and prints the result. The status bar at the bottom indicates 'U:--- main.c All L1 (C/*l Abbrev)' and 'Beginning of buffer'.

```
#include <stdio.h>
#include "calculate.h"

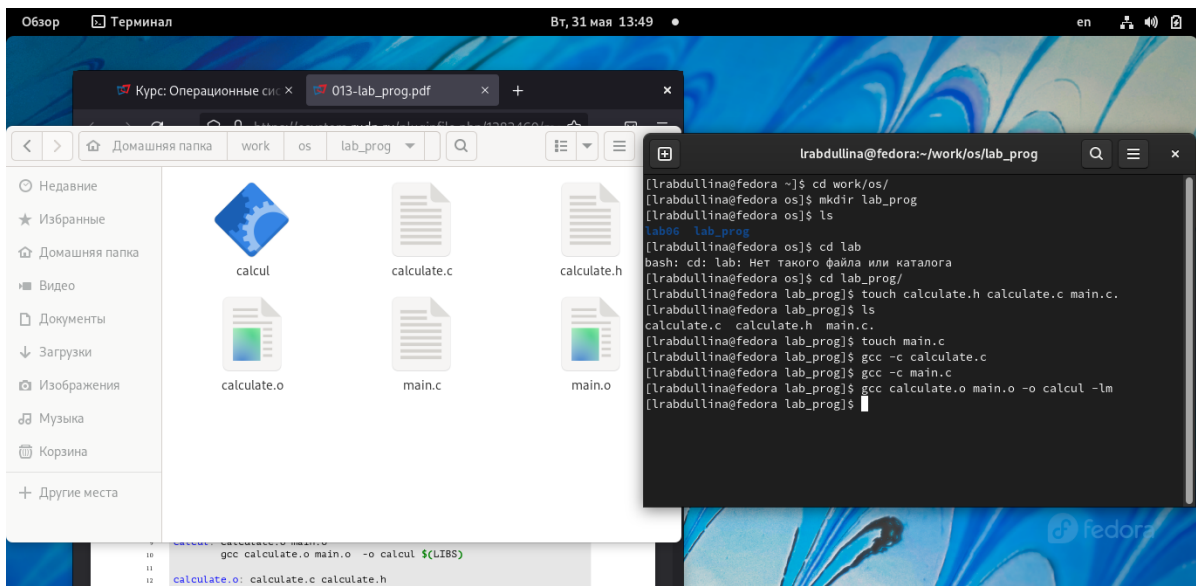
int
main(void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",&Operation);
    Result = Calculate(Numeral, Operation);
    printf("%.2f\n",Result);
    return 0;
}
```

Скриншот 3.5: Основной файл main.c,реализующий интерфейс пользователя к калькулятору

3.3 3

Выполним компиляцию программы посредством gcc, используя команды (скриншот 3.6):

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



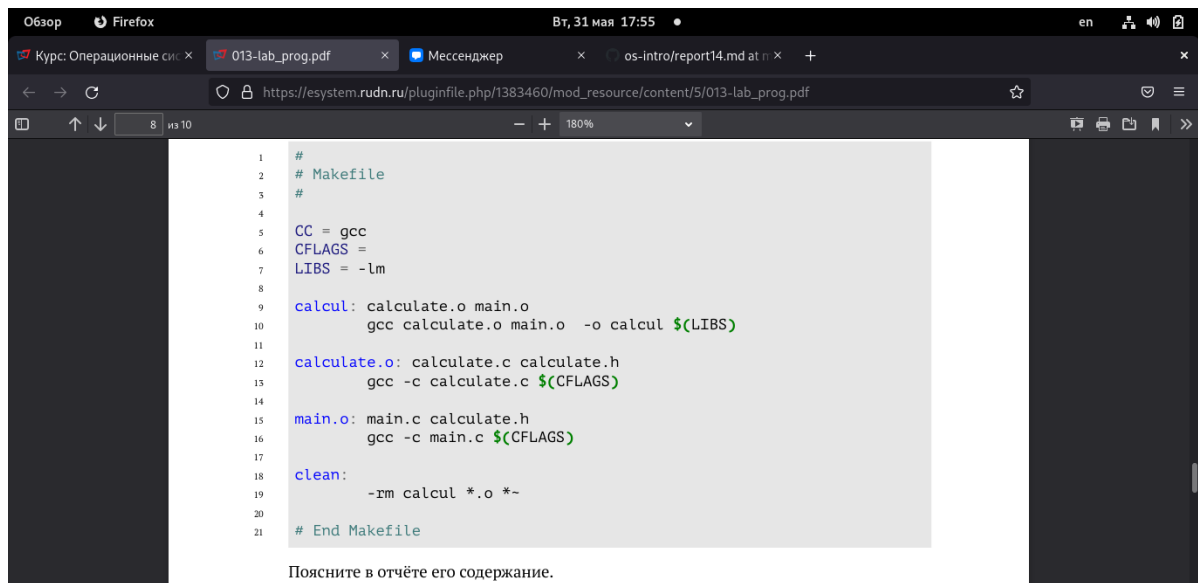
Скриншот 3.6: Созданные файлы

3.4 4

Необходимости исправлять синтаксические ошибки не было.

3.5 5

Создадим Makefile со следующим содержанием (скриншот 3.7):

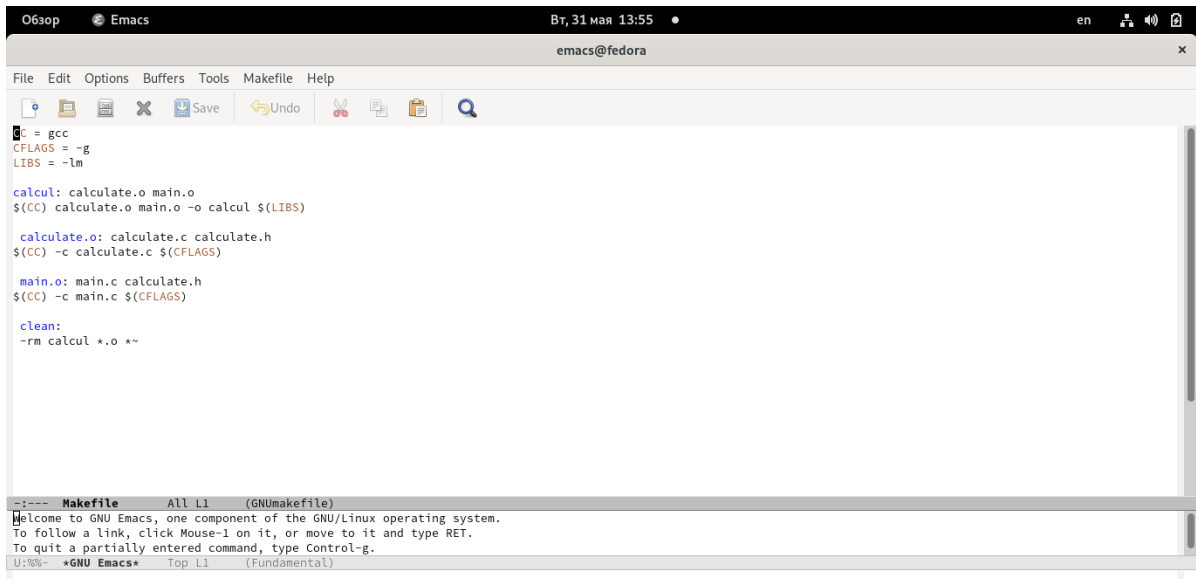


Скриншот 3.7: Созданный файл

Данный файл необходим для автоматической компиляции файлов `calculate.c` (цель `calculate.o`), `main.c` (цель `main.o`), а также их объединения в один исполняемый файл `calcul` (цель `calcul`). Цель `clean` нужна для автоматического удаления файлов. Переменная `CC` отвечает за утилиту для компиляции. Переменная `CFLAGS` отвечает за опции в данной утилите. Переменная `LIBS` отвечает за опции для объединения объектных файлов в один исполняемый файл.

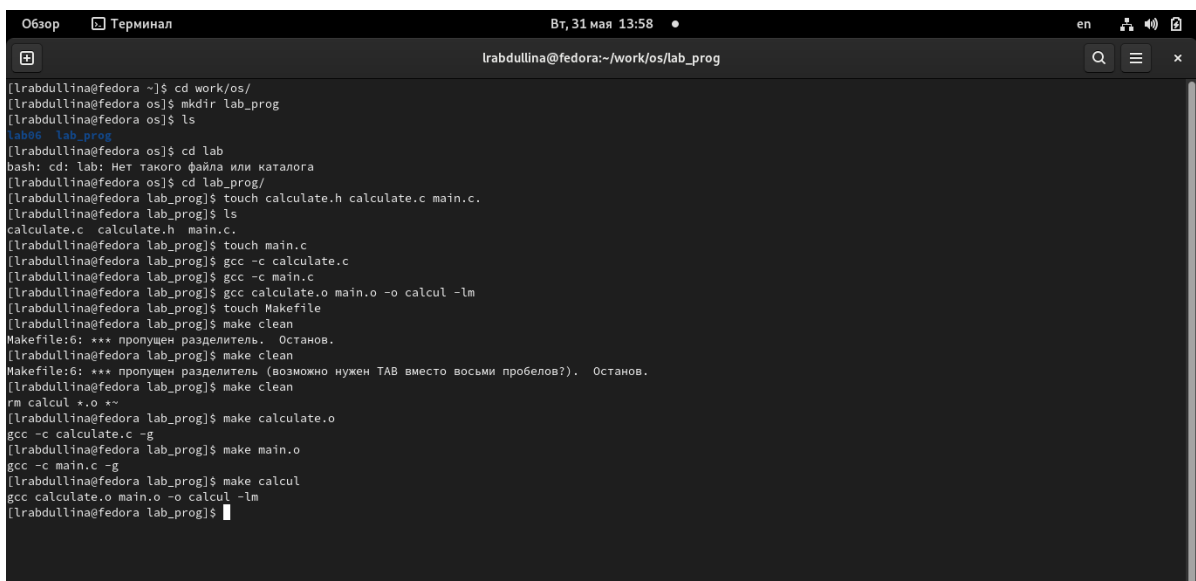
3.6 6

С помощью `gdb` выполним отладку программы `calcul`, но перед использованием `gdb` исправим `Makefile`. В переменную `CFLAGS` добавим опцию `-g`, необходимую для компиляции объектных файлов и их использования в программе отладчика `GDB`. Сделаем так, что утилита компиляции выбирается с помощью переменной `CC`. (скриншот 3.8)



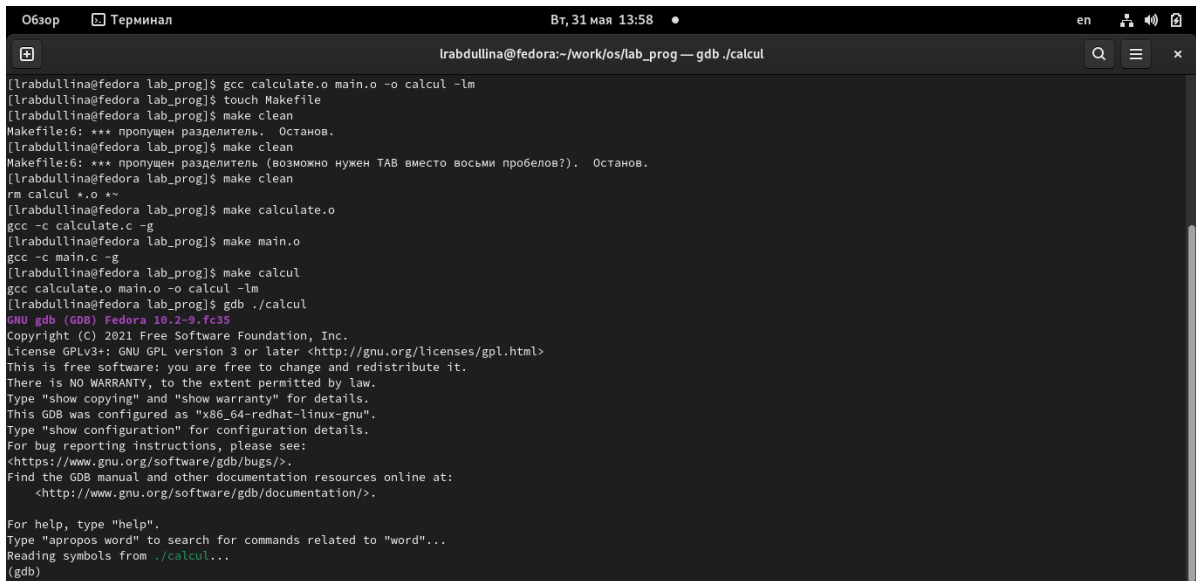
Скриншот 3.8: Исправленный файл

После этого удалим исполняемые и объектные файлы из каталога с помощью команды «make clean». Выполним компиляцию файлов, используя команды «make calculate.o», «make main.o», «male calcul» (скриншот 3.9)



Скриншот 3.9: Компиляция файлов

- Запустим отладчик GDB, загрузив в него программу для отладки “gdb ./calcul” (скриншот 3.10)



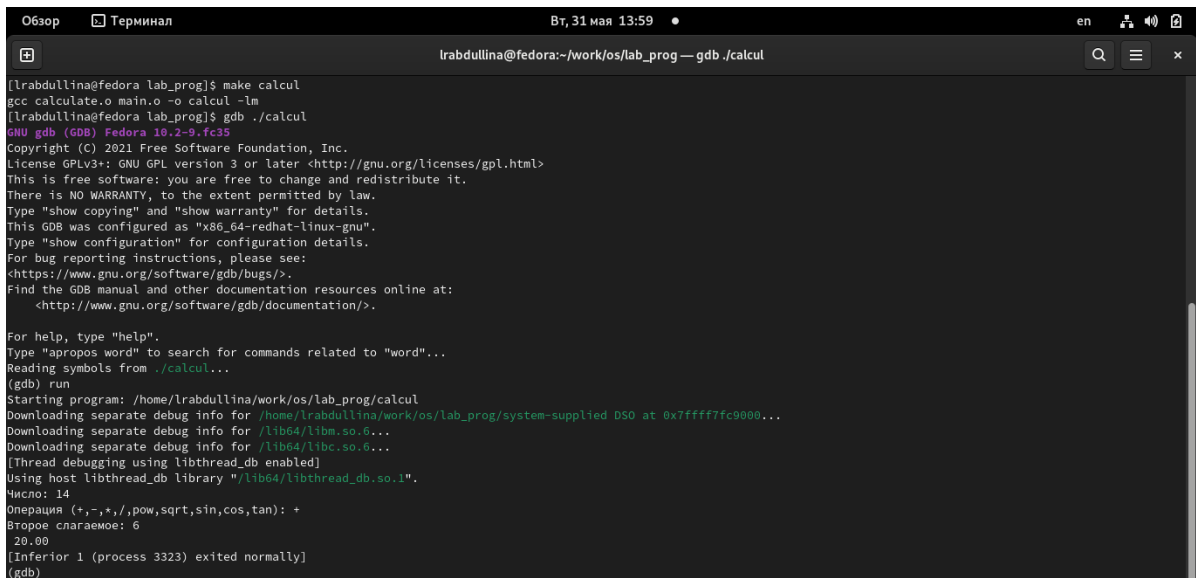
```
Обзор Терминал Вт, 31 мая 13:58 en
lrabduullina@fedora:~/work/os/lab_prog — gdb ./calcul

[lrabduullina@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[lrabduullina@fedora lab_prog]$ touch Makefile
[lrabduullina@fedora lab_prog]$ make clean
Makefile:6: *** пронуцен разделитель. Останов.
[lrabduullina@fedora lab_prog]$ make clean
Makefile:6: *** пронуцен разделитель (возможно нужен TAB вместо восьми пробелов?). Останов.
[lrabduullina@fedora lab_prog]$ make clean
rm calcul *.o *~
[lrabduullina@fedora lab_prog]$ make calculate.o
gcc -c calculate.c -g
[lrabduullina@fedora lab_prog]$ make main.o
gcc -c main.c -g
[lrabduullina@fedora lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
[lrabduullina@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:
<https://www.gnu.org/software/gdb/bugs/>.
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)
```

Скриншот 3.10: Запуск отладчика

- Для запуска программы внутри отладчика введем команду “run” и произведем арифметическую операцию (скриншот 3.11)



```
Обзор Терминал Вт, 31 мая 13:59 en
lrabduullina@fedora:~/work/os/lab_prog — gdb ./calcul

[lrabduullina@fedora lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
[lrabduullina@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:
<https://www.gnu.org/software/gdb/bugs/>.
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: /home/lrabduullina/work/os/lab_prog/calcul
Downloading separate debug info for /home/lrabduullina/work/os/lab_prog/system-supplied DSO at 0x7ffff7fc9808...
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: 14
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
20.00
[Inferior 1 (process 3323) exited normally]
(gdb)
```

Скриншот 3.11: Запуск программы внутри отладчика

- Для постраничного (по 9 строк) просмотра исходного код используем команду “list”(скриншот 3.12)

```
Обзор Терминал Вт, 31 мая 13:59 en
lrabdullina@fedora:~/work/os/lab_prog — gdb ./calcul

For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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: /home/lrabdullina/work/os/lab_prog/calcul
Downloading separate debug info for /home/lrabdullina/work/os/lab_prog/system-supplied 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: 14
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
20.00
[Inferior 1 (process 3323) 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)
```

Скриншот 3.12: Просмотр кода (постранично)

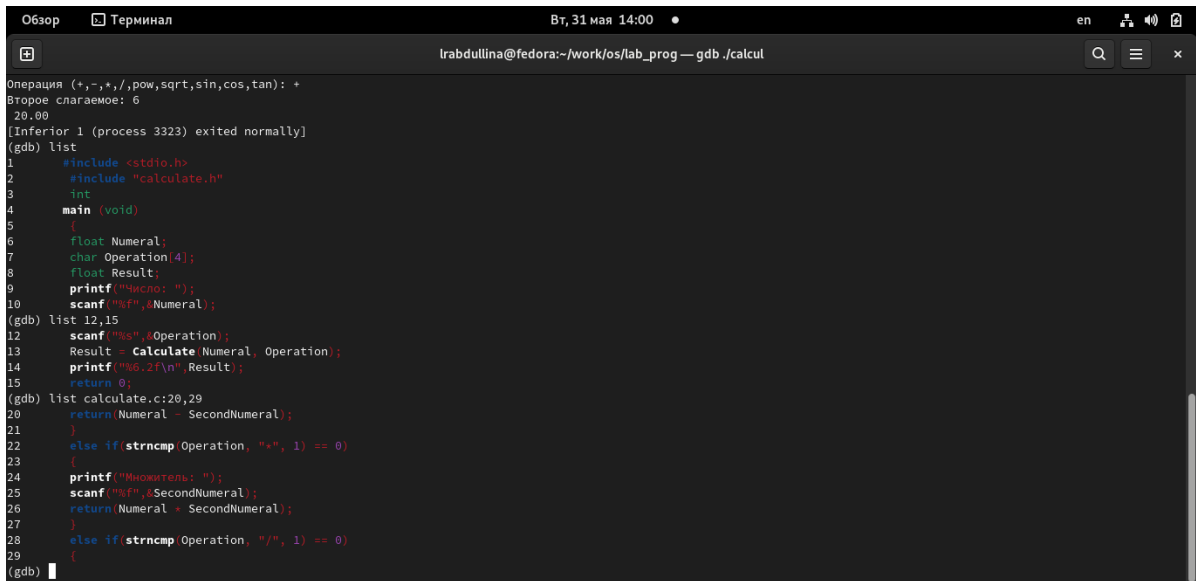
- Для просмотра строк с 12 по 15 основного файла используем “list 12,15” (скриншот 3.13)

```
Обзор Терминал Вт, 31 мая 14:00 en
lrabdullina@fedora:~/work/os/lab_prog — gdb ./calcul

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /home/lrabdullina/work/os/lab_prog/calcul
Downloading separate debug info for /home/lrabdullina/work/os/lab_prog/system-supplied 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: 14
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
20.00
[Inferior 1 (process 3323) 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) list 12,15
12         scanf("%s",&Operation);
13         Result = Calculate(Numeral, Operation);
14         printf("%6.2f\n",Result);
15         return 0;
(gdb)
```

Скриншот 3.13: Просмотр строк 12-15

- Для просмотра определённых строк не основного файла используем “list calculate.c:20,29” (скриншот 3.14)

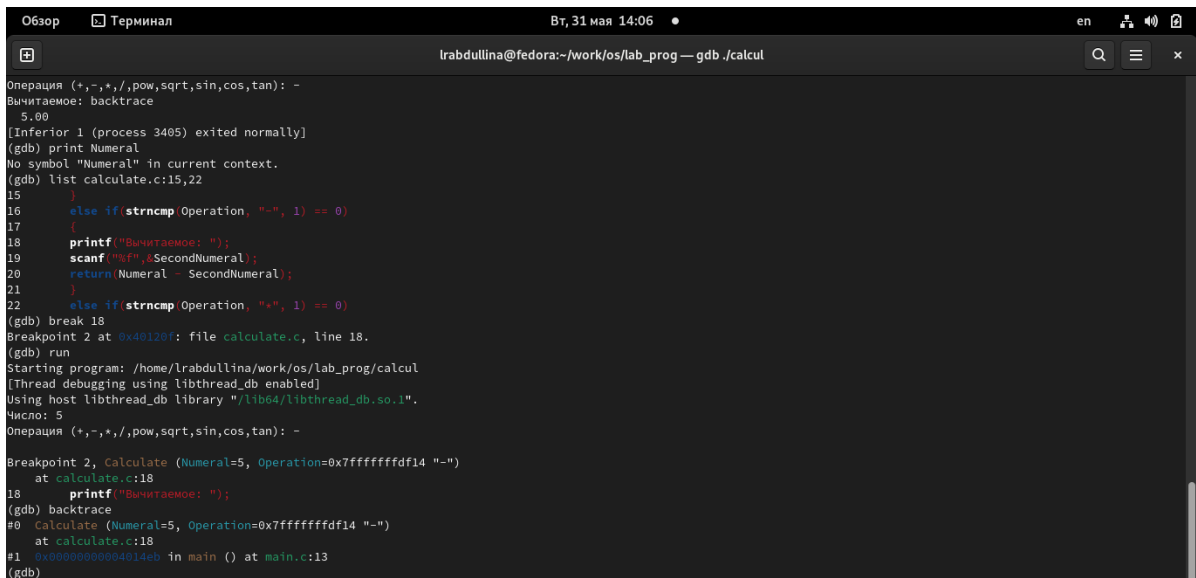


```
Обзор Терминал Вт, 31 мая 14:00 en
lrabdullina@fedora:~/work/os/lab_prog — gdb ./calcul

Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
20.00
[Inferior 1 (process 3323) 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) list 12,15
12         scanf("%s",&Operation);
13         Result = Calculate(Numeral, Operation);
14         printf("%0.2f\n",Result);
15         return 0;
(gdb) list calculate.c:20,29
20         return Numeral - SecondNumeral;
21     }
22     else if(strcmp(Operation, "+") == 0)
23     {
24         printf("Нумератор: ");
25         scanf("%f",&SecondNumeral);
26         return Numeral + SecondNumeral;
27     }
28     else if(strcmp(Operation, "/") == 0)
29     {
```

Скриншот 3.14: Просмотр строк не основного файла

- Установим точку останова в файле calculate.c на строке номер 18 с командами “list calculate.c:20,27”, “break 18” (скриншот 3.15)



```
Обзор Терминал Вт, 31 мая 14:06 en
lrabdullina@fedora:~/work/os/lab_prog — gdb ./calcul

Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Выводимое: backtrace
5.00
[Inferior 1 (process 3405) exited normally]
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) list calculate.c:15,22
15     }
16     else if(strcmp(Operation, "-") == 0)
17     {
18         printf("Выводимое: ");
19         scanf("%f",&SecondNumeral);
20         return Numeral - SecondNumeral;
21     }
22     else if(strcmp(Operation, "+") == 0)
(gdb) break 18
Breakpoint 2 at 0x40120f: file calculate.c, line 18.
(gdb) run
Starting program: /home/lrabdullina/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=0x7fffffffdf14 "-")
at calculate.c:18
18     printf("Выводимое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf14 "-")
at calculate.c:18
#1 0x00000000004014eb in main () at main.c:13
(gdb)
```

Скриншот 3.15: Установка точки останова

- Выведем информацию об имеющихся в проекте точках останова “info breakpoints” (скриншот 3.16)

```
Обзор Терминал Вт, 31 мая 14:01 en
lrabduullina@fedora:~/work/os/lab_prog — gdb ./calcul

(gdb) list 12,15
12     scanf("%s",&Operation);
13     Result = Calculate(Numeral, Operation);
14     printf("%g.2f\n",Result);
15     return 0;
(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 {
(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 }
(gdb) break 21
Breakpoint 1 at 0x401247: file calculate.c, line 22.
(gdb) info breakpoints
Num Type      Disp Enb Address      What
1   breakpoint keep y  0x0000000000401247 in Calculate
                                at calculate.c:22
(gdb)
```

Скриншот 3.16: Информация о точках останова

- Запустите программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова. (скриншот 3.17)

```
Обзор Терминал Вт, 31 мая 14:06 en
lrabduullina@fedora:~/work/os/lab_prog — gdb ./calcul

Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вводимое: backtrace
5.00
[Inferior 1 (process 3405) exited normally]
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) list calculate.c:15,22
15 }
16 else if(strncmp(Operation, "-", 1) == 0)
17 {
18     printf("Вводимое: ");
19     scanf("%f",&SecondNumeral);
20     return(Numeral - SecondNumeral);
21 }
22 else if(strncmp(Operation, "+", 1) == 0)
(gdb) break 18
Breakpoint 2 at 0x40120f: file calculate.c, line 18.
(gdb) run
Starting program: /home/lrabduullina/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): -

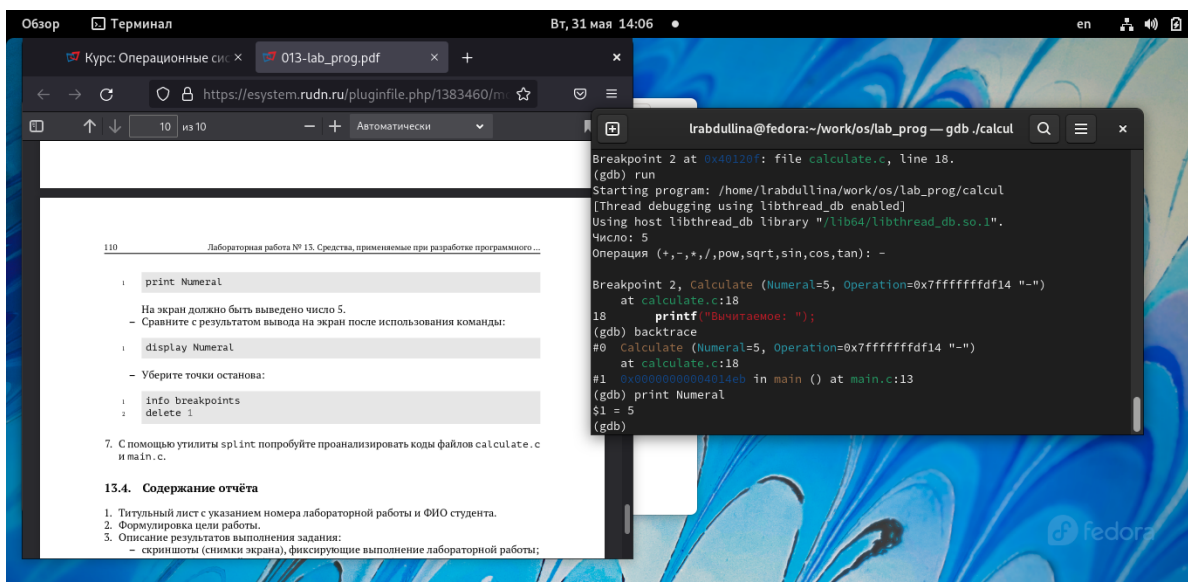
Breakpoint 2, Calculate (Numeral=5, Operation=0x7fffffffdf14 "-")
at calculate.c:18
18     printf("Вводимое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf14 "-")
at calculate.c:18
#1 0x00000000004014eb in main () at main.c:13
(gdb)
```

Скриншот 3.17: Остановка у точки

- Отладчик выдал следующую информацию: #0 Calculate (Numeral=5, Operation=0x7fffffffdf14 "-") at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17

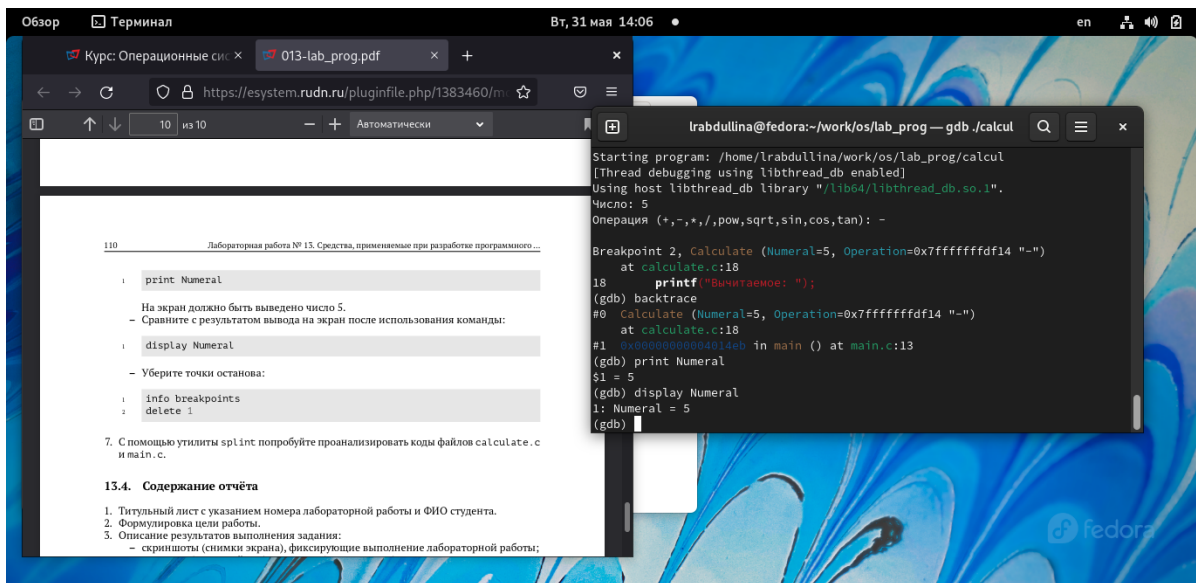
а команда “backtrace” показала весь стек вызываемых функций от начала программы до текущего места.

- Посмотрим, чему равно на этом этапе значение переменной Numeral, введя “print Numeral”. На экране выведено число 5. (скриншот 3.18)



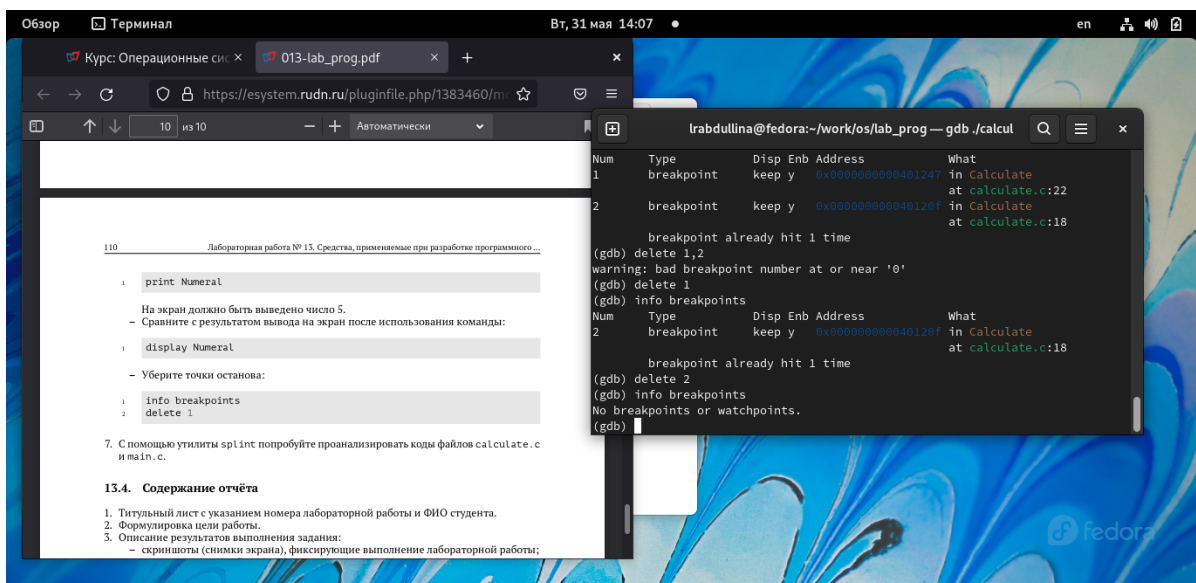
Скриншот 3.18: Значение Numeral

- Сравним с результатом вывода на экран после использования команды “display Numeral”. Значения совпадают (скриншот 3.19)



Скриншот 3.19: Значение Numeral

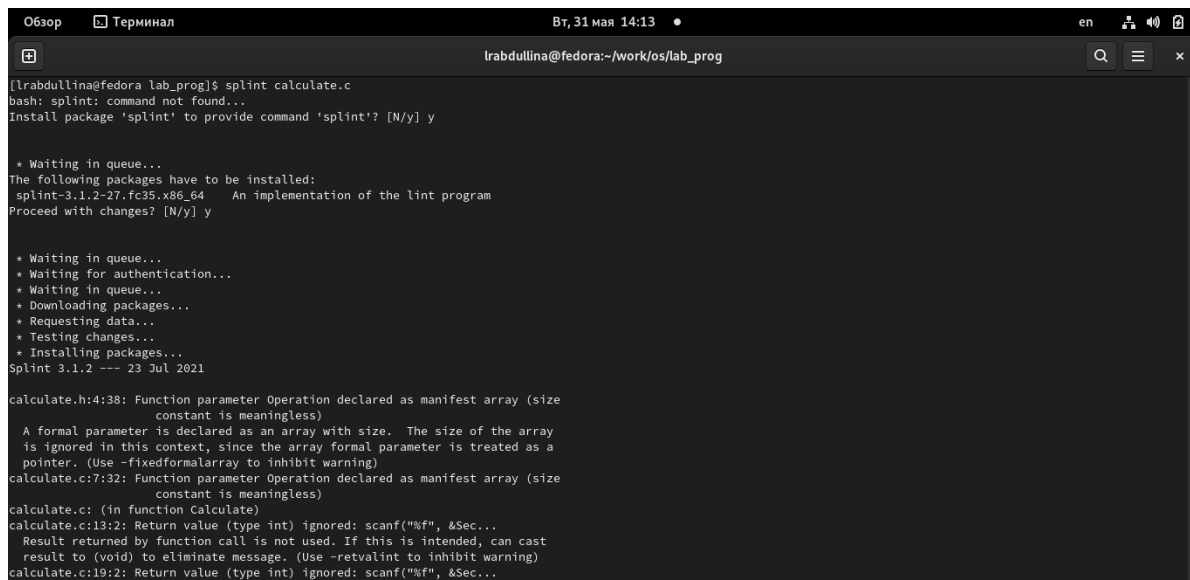
- Уберем точки останова “info breakpoints”, “delete 1” (скриншот 3.20)



Скриншот 3.20: Убираем точки останова

3.7 7

С помощью утилиты `splint` попробуем проанализировать коды файлов `calculate.c` и `main.c`. (скриншот 3.21,3.22, 3.23)



```
Обзор Терминал Вт, 31 мая 14:13 en
lrabdullina@fedora:~/work/os/lab_prog

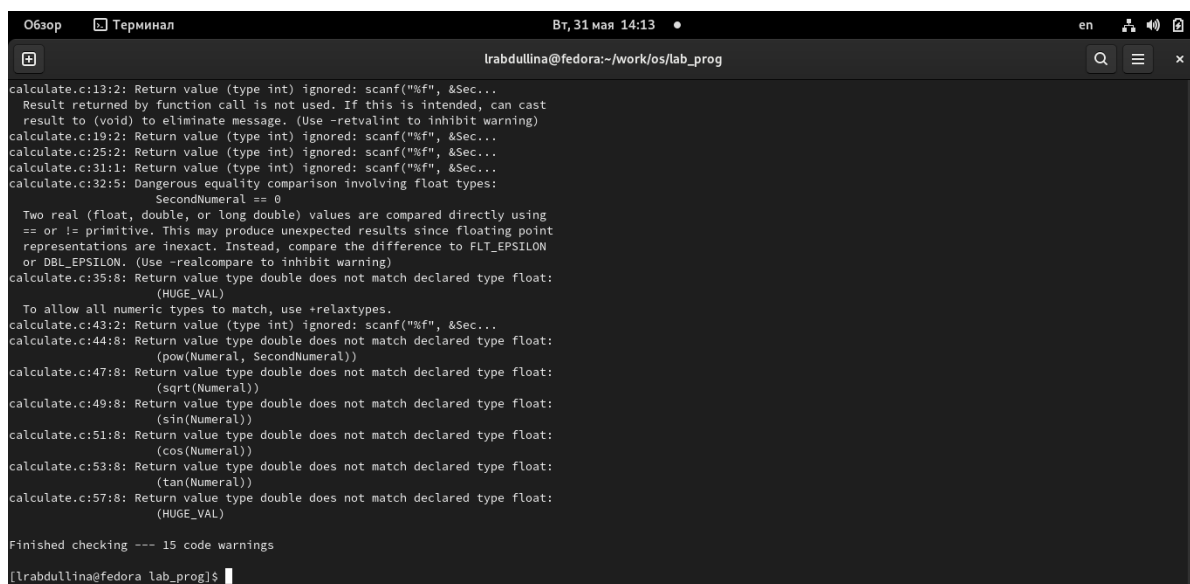
[lrabdullina@fedora lab_prog]$ splint calculate.c
bash: splint: command not found...
Install package 'splint' to provide command 'splint'? [N/y] y

* Waiting in queue...
The following packages have to be installed:
splint-3.1.2-27.fc35.x86_64 An implementation of the lint program
Proceed with changes? [N/y] y

* Waiting in queue...
* Waiting for authentication...
* Waiting in queue...
* Downloading packages...
* Requesting data...
* Testing changes...
* Installing packages...
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...
```

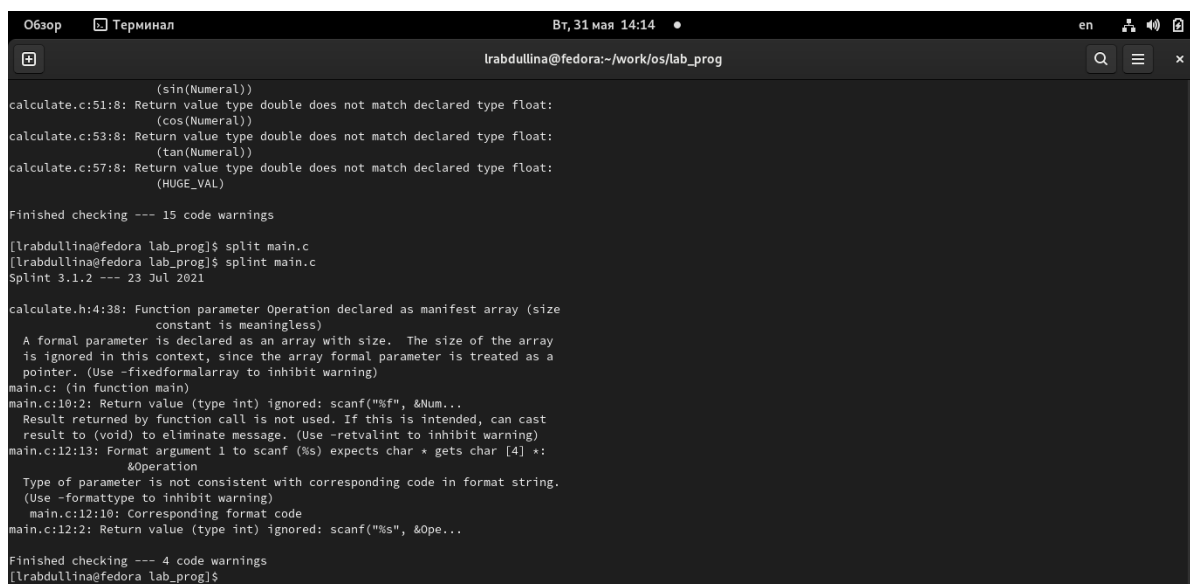
Скриншот 3.21: Загрузка утилиты



```
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:31:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:5: 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:35:8: Return value type double does not match declared type float:
        (HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:43:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:8: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:47:8: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:49:8: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:51:8: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:53:8: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:57:8: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings
[lrabdullina@fedora lab_prog]$
```

Скриншот 3.22: Информация о `calculate.c`



```
Обзор Терминал Вт, 31 мая 14:14 en
lrabduullina@fedora:~/work/os/lab_prog

(sin(Numeral))
calculate.c:51:8: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:53:8: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:57:8: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings

[lrabduullina@fedora lab_prog]$ splint main.c
[lrabduullina@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
[lrabduullina@fedora lab_prog]$
```

Скриншот 3.23: Информация о main.c

С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохраняются. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потере данных.

4 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 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. 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. 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: ... : ... <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. 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. 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. 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:

cscore – исследование 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 С 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 работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

5 Выводы

В ходе лабораторной работы мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

6 Список литературы

<https://esystem.rudn.ru/course/view.php?id=5790> :::