

Отчёт по лабораторной работе №13

Операционные системы

Самсонова Мария Ильинична

Содержание

Цель работы	1
Задание	1
Выполнение лабораторной работы	2
Вывод.....	15
Ответы на контрольные вопросы	15

Цель работы

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

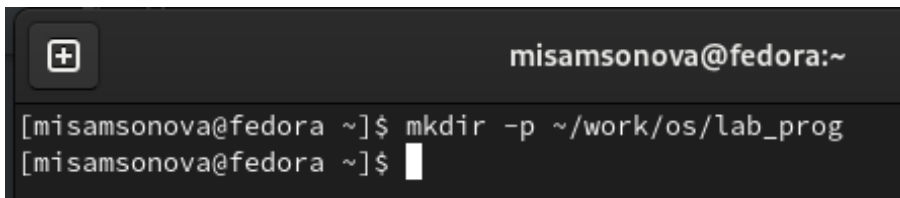
Задание

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`:
 - `gcc -c calculate.c`
 - `gcc -c main.c`
 - `gcc calculate.o main.o -o calcul -lm`
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` с данным в документе содержанием.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`):
 - Запустите отладчик GDB, загрузив в него программу для отладки:
 - `gdb ./calcul`
 - Для запуска программы внутри отладчика введите команду `run`:
 - `run`

- Для постраничного (по 9 строк) просмотра исходного код используйте команду list:
 - list
 - Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:
 - list 12,15
 - Для просмотра определённых строк не основного файла используйте list с параметрами:
 - list calculate.c:20,29
 - Установите точку останова в файле calculate.c на строке номер 21:
 - list calculate.c:20,27
 - break 21
 - Выведите информацию об имеющихся в проекте точка останова:
 - info breakpoints
 - Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:
 - run
 - 5
 - -
 - backtrace
 - Отладчик выдаст следующую информацию:
 - #0 Calculate (Numeral=5, Operation=0x7ffffffd280 "-")
 - at calculate.c:21
 - #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:
 - print Numeral На экран должно быть выведено число 5.
 - Сравните с результатом вывода на экран после использования команды:
 - display Numeral
 - Уберите точки останова:
 - info breakpoints
 - delete 1
7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

Выполнение лабораторной работы

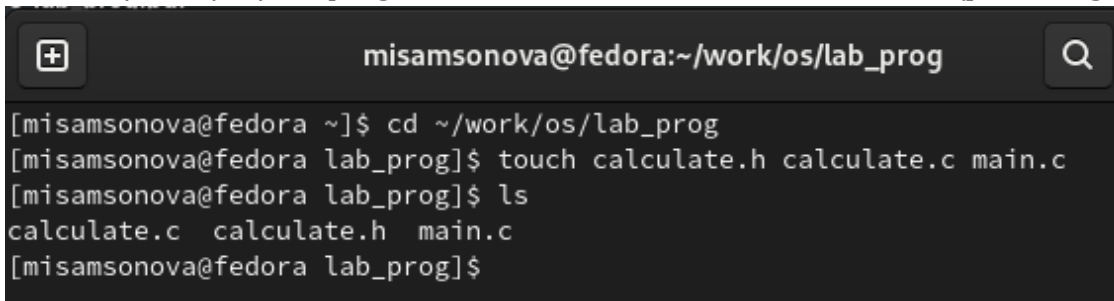
1. В домашнем каталоге создаём подкаталог ~/work/os/lab_progс помощью команды «mkdir -p~/work/os/lab_prog» (Рис. -@fig:001).



```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ mkdir -p ~/work/os/lab_prog  
[misamsonova@fedora ~]$
```

Создание подкаталога

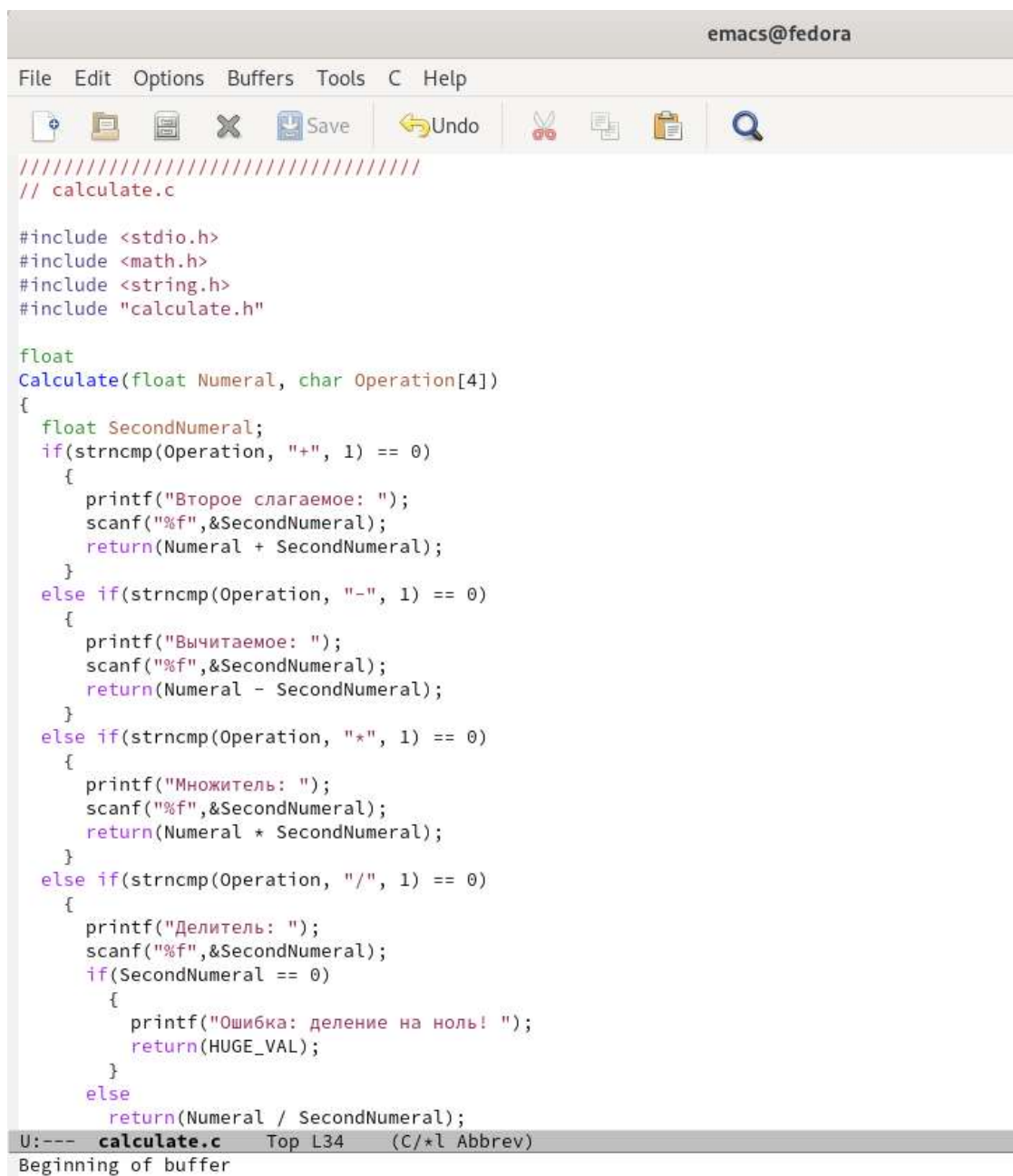
2. Создали в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd ~/work/os/lab_prog» и «touch calculate.h calculate.c main.c» (рис. -@fig:002).



```
misamsonova@fedora:~/work/os/lab_prog  
[misamsonova@fedora ~]$ cd ~/work/os/lab_prog  
[misamsonova@fedora lab_prog]$ touch calculate.h calculate.c main.c  
[misamsonova@fedora lab_prog]$ ls  
calculate.c calculate.h main.c  
[misamsonova@fedora lab_prog]$
```

Создание файлов

- 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. Открыв редактор Emacs, приступили к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.c рис. -@fig:003 , -@fig:004).



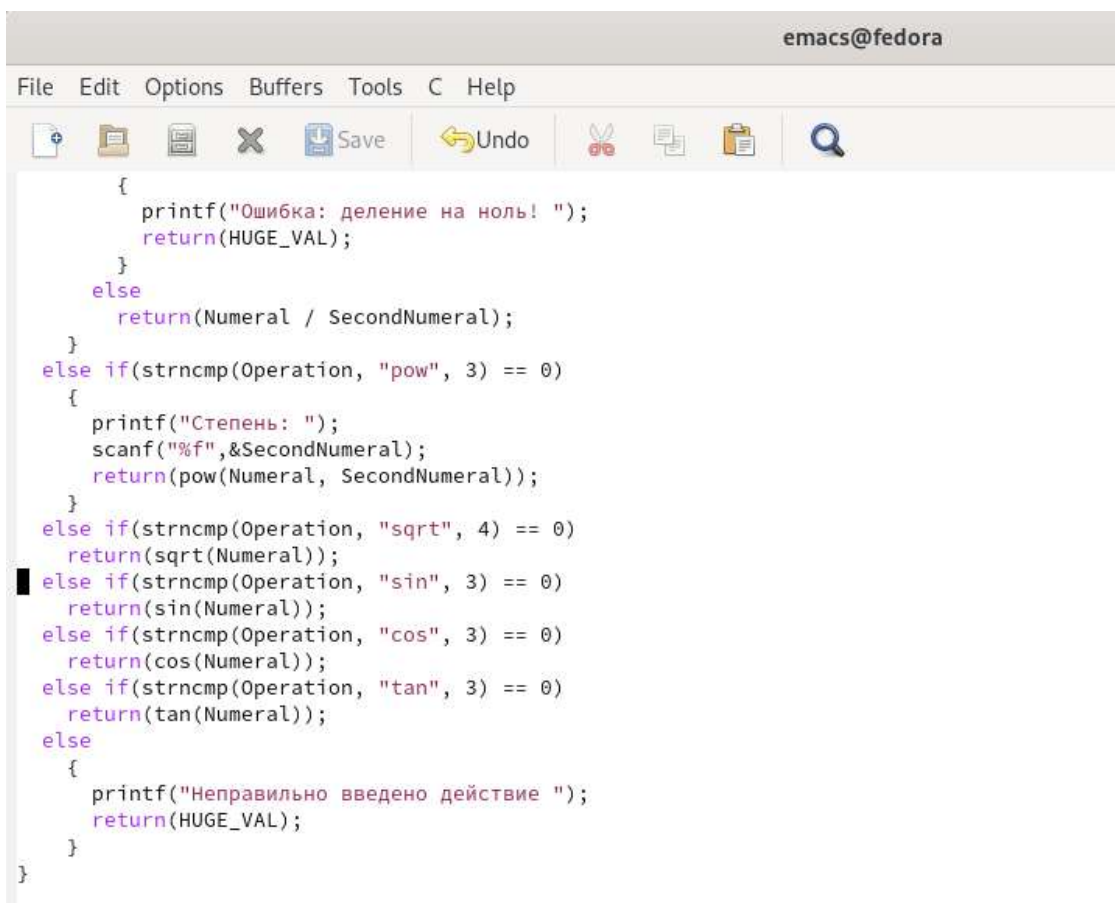
```
emacs@fedora
File Edit Options Buffers Tools C Help
Save Undo
////////////////////////////////////
// calculate.c

#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);
    }
    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);
    }
}
```

U:--- **calculate.c** Top L34 (C/*l Abbrev)
Beginning of buffer

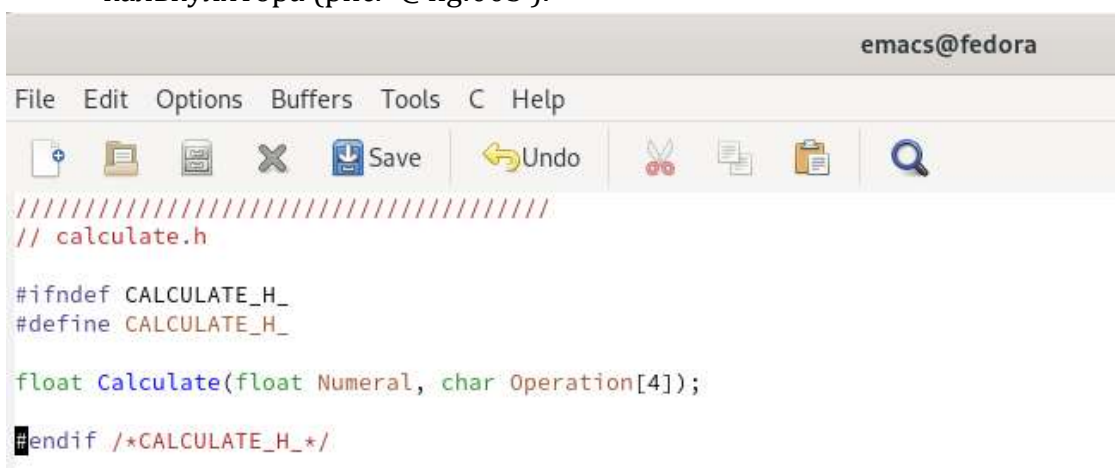
Программа в calculate.c



```
emacs@fedora
File Edit Options Buffers Tools C Help
Save Undo
{
    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);
}
}
```

Программа в *calculate.c*

- Интерфейсный файл *calculate.h*, описывающий формат вызова функции калькулятора (рис. -@fig:005).



```
emacs@fedora
File Edit Options Buffers Tools C Help
Save Undo
////////////////////////////////////
// calculate.h

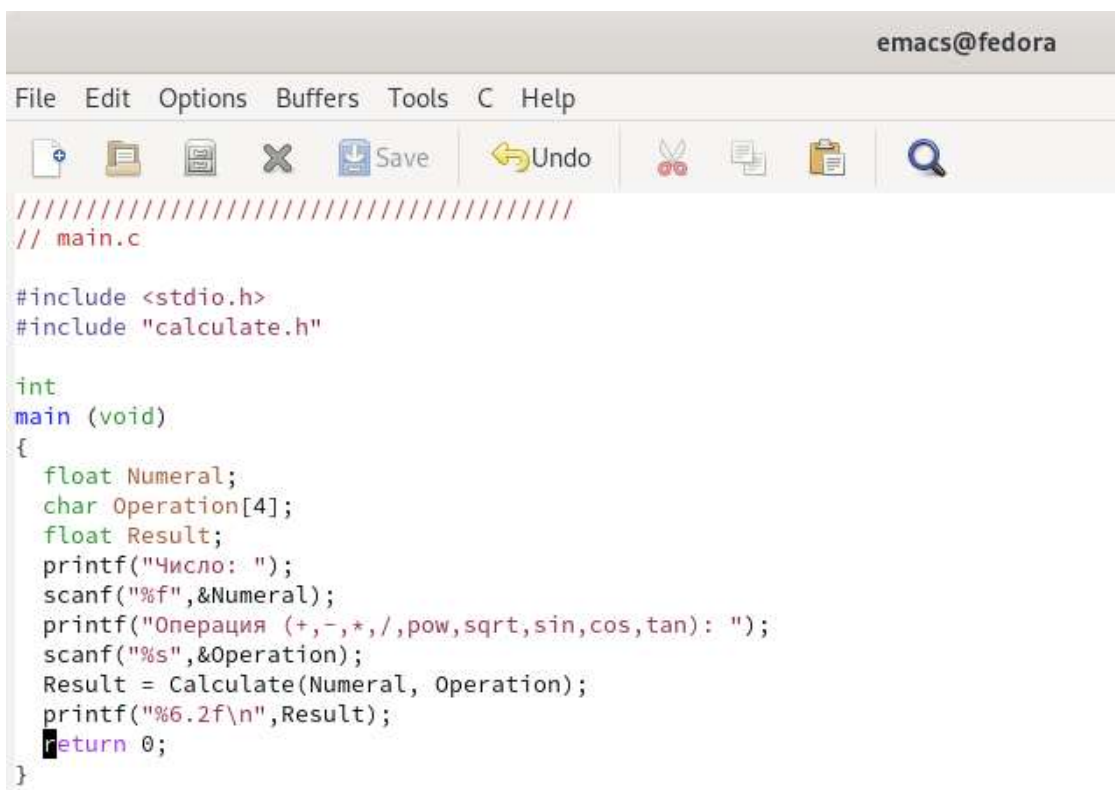
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Программа в *calculate.h*

- Основной файл *main.c*, реализующий интерфейс пользователя к калькулятору (рис. -@fig:006).



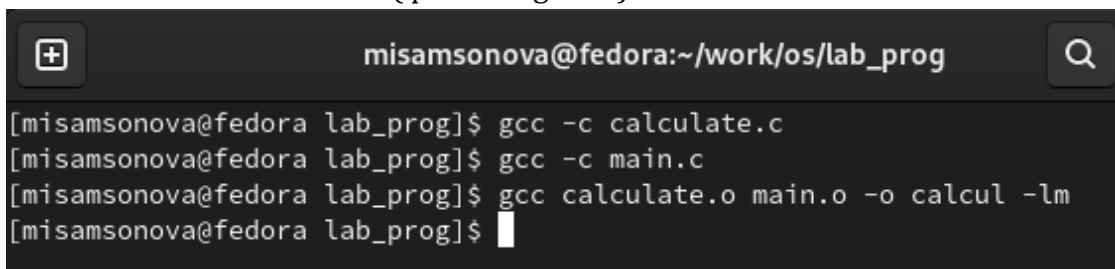
```
//////////////////////
// main.c

#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;
}
```

Программа в main.c

3. Выполнили компиляцию программы посредством gcc (версия компилятора :8.3.0-19), используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (рис. -@fig:007).

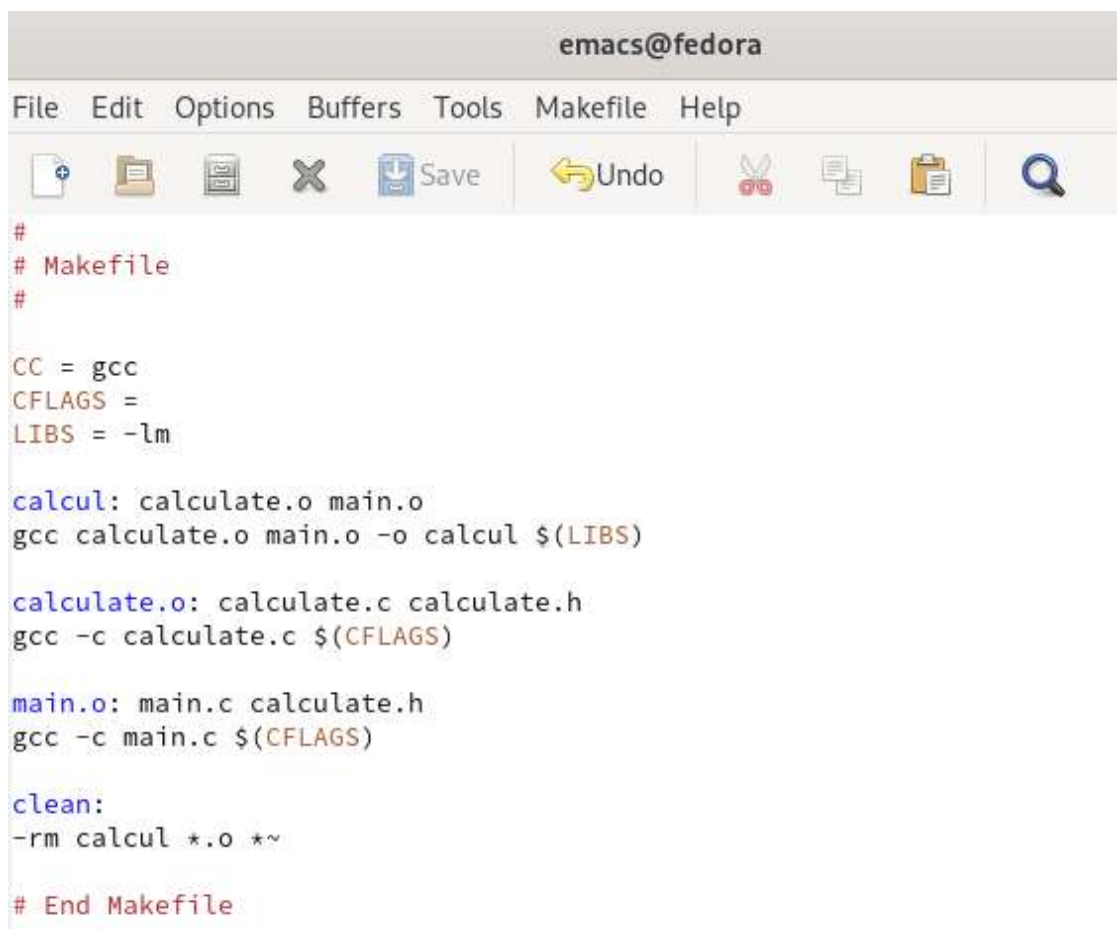


```
misamsonova@fedora:~/work/os/lab_prog

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

Компиляция программы

4. В ходе компиляции программы никаких ошибок выявлено не было.
5. Создали Makefile с необходимым содержанием (рис. -@fig:008).



```
#
# 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)

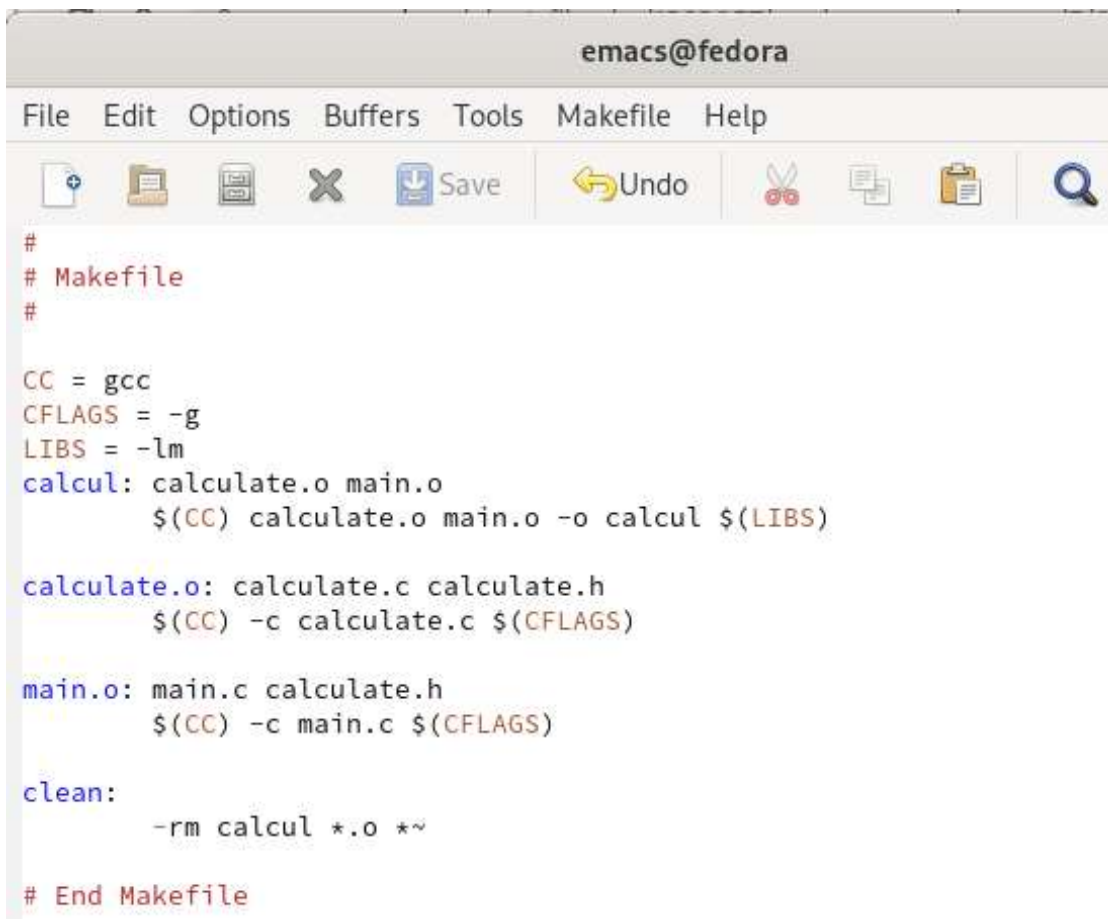
main.o: main.c calculate.h
gcc -c main.c $(CFLAGS)

clean:
-rm calcul *.o *~

# End Makefile
```

Программа в Makefile

- 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.
6. Далее исправили Makefile (рис. -@fig:009).



```
#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm
calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
    $(CC) -c calculate.c $(CFLAGS)

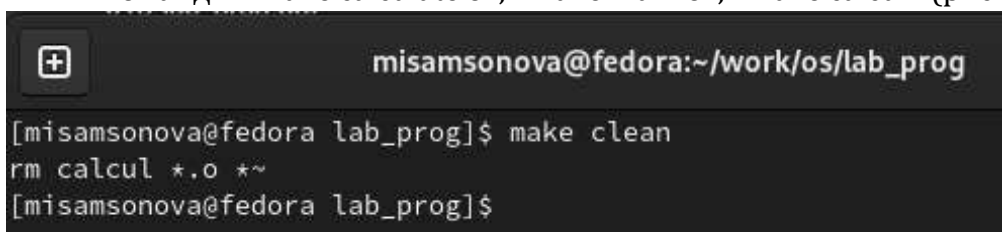
main.o: main.c calculate.h
    $(CC) -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# End Makefile
```

Программа в Makefile

- В переменную CFLAGS добавили опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделали так, что утилита компиляции выбирается с помощью переменной CC. После этого удалили исполняемые и объектные файлы из каталога с помощью команды «make clear» (рис. -@fig:010). Выполнили компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (рис. -@fig:011).



```
misamsonova@fedora:~/work/os/lab_prog

[misamsonova@fedora lab_prog]$ make clean
rm calcul *.o *~
[misamsonova@fedora lab_prog]$
```

Удаление файлов


```
misamsonova@fedora:~/work/os/lab_prog

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

Компиляция файлов

- Далее с помощью gdb выполнили отладку программы calcul. Запустили отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul» (рис. -@fig:012).

```
misamsonova@fedora:~/work/os/lab_prog — gdb ./calcul

[misamsonova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-1.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.
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)
```

Работа с gdb

- Для запуска программы внутри отладчика ввели команду «run» (рис. -@fig:013).

```

(gdb) run
Starting program: /home/misamsonova/work/os/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.
Downloading -0.00 MB separate debug info for system-supplied DSO at 0x7ffff7fc5000
Downloading -0.00 MB separate debug info for /lib64/libm.so.6
Downloading -0.00 MB 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: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 4
24.00
[Inferior 1 (process 6847) exited normally]
(gdb)

```

Работа с gdb - run

- Для постраничного (по 10 строк) просмотра исходного кода использовали команду «list» (рис. -@fig:014).

```

(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) list
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16     scanf("%s",&Operation);
17     Result = Calculate (Numeral, Operation);
18     printf("%6.2f\n",Result);
19     return 0;
20 }
(gdb)

```

Работа с gdb - list

- Для просмотра строк с 12 по 15 основного файла использовали команду «list 12,15» (рис. -@fig:015).

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

Работа с gdb - list 12,15

- Для просмотра определённых строк не основного файла использовали команду «list calculate.c:20,29» (рис. -@fig:016).

```
(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)
```

Работа с gdb - list calculate.c:20,29

- Установили точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (рис. -@fig:017).

```
(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 0x40129f: file calculate.c, line 21.
(gdb)
```

Работа с gdb - list calculate.c:20,27

- Вывели информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (рис. -@fig:018).

```
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint     keep y   0x00000000000040120f in Calculate
                                                at calculate.c:21
(gdb)
```

Работа с gdb - info breakpoints

- Запустили программу внутри отладчика и убедились, что программа остановилась в момент прохождения точки останова. Использовали команды «run», «5», «*» и «backtrace» (рис. -@fig:019).

```
(gdb) run
Starting program: /home/misamsonova/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=0x7fffffffdf24 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1 0x0000000000004014eb in main () at main.c:17
(gdb)
```

Работа с gdb - run

- Посмотрели, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (рис. -@fig:020).

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

Работа с gdb - print Numeral

- Сравнили с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (рис. -@fig:021).

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

Работа с gdb - display Numeral

- Убрали точки останова с помощью команд «info breakpoints» и «delete 1» (рис. -@fig:022).

```

(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x00000000000040120f in calculate
                                                at calculate.c:21

        breakpoint already hit 1 time
(gdb) delete 1
(gdb)

```

Работа с gdb - info breakpoints

7. Далее воспользовались командами «splint calculate.c» и «splint main.c» (рис. - @fig:023 , -@fig:024). 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.

```
misamsonova@fedora:~/work/os/lab_prog

[misamsonova@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
    representations are inexact. Instead, compare the difference to FLT_EPSILON
    or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:38:10: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:46:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:13: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:50:11: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:52:11: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:54:11: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:56:11: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:60:13: Return value type double does not match declared type float:
        (HUGE_VAL)

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

Результат команды splint calculate.c

```
misamsonova@fedora:~/work/os/lab_prog
[misamsonova@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:16:14: 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:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &Ope...

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

Результат конмады splint main.c

Вывод

В процессе выполнения данной 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.

Ответы на контрольные вопросы

1. Чтобы получить информацию о возможностях программ gcc, make, gdbи др. нужно воспользоваться командой тапили 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: окодирование –по сути создание исходного текста программы (возможно в нескольких вариантах); –анализ разработанного кода; осборка, компиляция и разработка исполняемого модуля; отестирование и отладка, сохранение произведённых изменений;
 - документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geanyи 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) .swsпринимаются гсскак программы на языке С, файлы с расширением .ssили .С–как файлы на языке С++, а файлы срасширением .осчитаются объектными. Например, в команде «gcc-stain.c» : gccпо расширению (суффиксу) .сраспознает 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 -oi в качестве параметра задать имя создаваемого файла: «gcc-ohellomaiВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями.п.с».
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 makeнеобходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefileили 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 Makefileможет 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 Makefileимеет 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.Пример более сложного синтаксиса Makefile:## Makefile for abcd.c#CC = gccCFLAGS =# Compile abcd.c normalyabcd: abcd.c\$(CC) -o abcd \$(CFLAGS) abcd.cclean:-rm abcd .o ~# EndMakefileforabcd.c. В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем

clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

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 ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями. 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-ни 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: `ccscope` – 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 Санализатор `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.