

---

# Front matter

---

lang: ru-RU

title: "Отчёт по лабораторной работе №14"

subtitle: "Операционные системы"

author: "Бирюкова Анастасия Анатольевна"

## Formatting

---

toc-title: "Содержание"

toc: true # Table of contents

toc\_depth: 2

lof: true # List of figures

lot: true # List of tables

fontsize: 12pt

linestretch: 1.5

papersize: a4paper

documentclass: scrreprt

polyglossia-lang: russian

polyglossia-otherlangs: english

mainfont: PT Serif

romanfont: PT Serif

sansfont: PT Sans

monofont: PT Mono

mainfontoptions: Ligatures=TeX

romanfontoptions: Ligatures=TeX

sansfontoptions: Ligatures=TeX,Scale=MatchLowercase

monofontoptions: Scale=MatchLowercase

indent: true

pdf-engine: lualatex

header-includes:

- \linepenalty=10 # the penalty added to the badness of each line within a paragraph (no associated penalty node) Increasing the value makes tex try to have fewer lines in the paragraph.

- `\interlinepenalty=0` # value of the penalty (node) added after each line of a paragraph.
  - `\hyphenpenalty=50` # the penalty for line breaking at an automatically inserted hyphen
  - `\exhyphenpenalty=50` # the penalty for line breaking at an explicit hyphen
  - `\binoppenalty=700` # the penalty for breaking a line at a binary operator
  - `\relpenalty=500` # the penalty for breaking a line at a relation
  - `\clubpenalty=150` # extra penalty for breaking after first line of a paragraph
  - `\widowpenalty=150` # extra penalty for breaking before last line of a paragraph
  - `\displaywidowpenalty=50` # extra penalty for breaking before last line before a display math
  - `\brokenpenalty=100` # extra penalty for page breaking after a hyphenated line
  - `\predisplaypenalty=10000` # penalty for breaking before a display
  - `\postdisplaypenalty=0` # penalty for breaking after a display
  - `\floatingpenalty = 20000` # penalty for splitting an insertion (can only be split footnote in standard LaTeX)
  - `\raggedbottom` # or `\flushbottom`
  - `\usepackage{float}` # keep figures where there are in the text
  - `\floatplacement{figure}{H}` # keep figures where there are in the text
- 

## 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

---

Создать калькулятор с простейшими функциями.

## Выполнение лабораторной работы

---

В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` с помощью команды «`mkdir -p ~/work/os/lab_prog`» (Рис.1)

```
aabiryukova@aabiryukova-VirtualBox: ~  
aabiryukova@aabiryukova-VirtualBox:~$ cd  
aabiryukova@aabiryukova-VirtualBox:~$ mkdir ~/work/os/lab_prog  
aabiryukova@aabiryukova-VirtualBox:~$ ls  
abc1      file.txt      lab13.sh      reports      script4.sh    Документы  
anastasia_br  file.txt.save laboratory2    script1.sh    script4.sh~   Загрузки  
australia    hello.sh      may           script1.sh~   ski.places     Изображения  
cong.txt     lab013.sh     monthly      script2.sh    text.txt       Музыка  
feathers     '#lab07.sh#'  my_os        script2.sh~   text.txt.save  Общедоступные  
file.cpp     lab07.sh      play         script3.sh    work           'Рабочий стол'  
'#file.txt#' lab130.sh     program.c    script3.sh~   Видео         Шаблоны  
aabiryukova@aabiryukova-VirtualBox:~$
```

Рис.1

Создала 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.2)

```
aabiryukova@aabiryukova-VirtualBox:~$ cd ~/work/os/lab_prog  
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ touch calculate.h  
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ touch calculate.c  
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ touch main.c
```

Рис.2

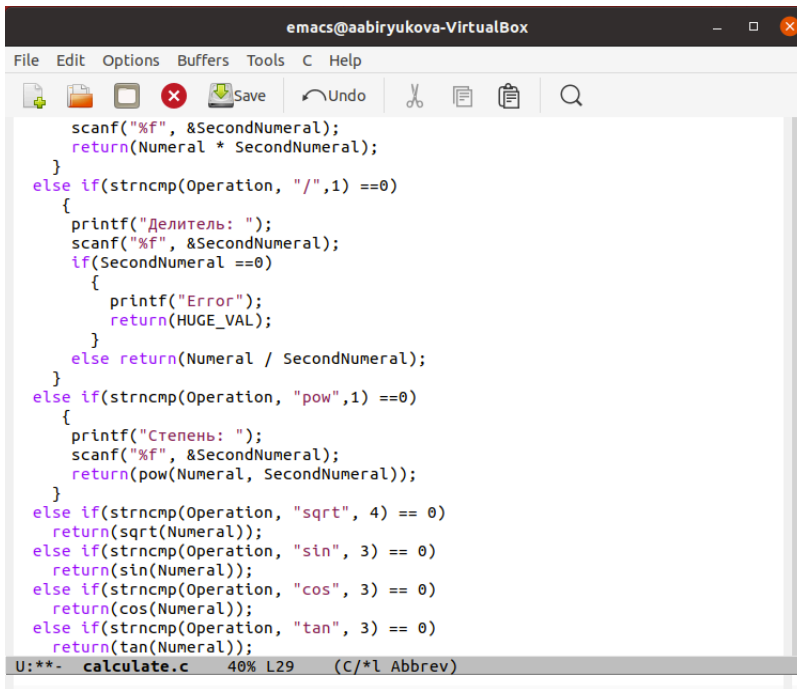
Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.

Открыв редактор Emacs, приступила к редактированию созданных файлов.

Реализация функций калькулятора в файле calculate.c (Рис.3-5)

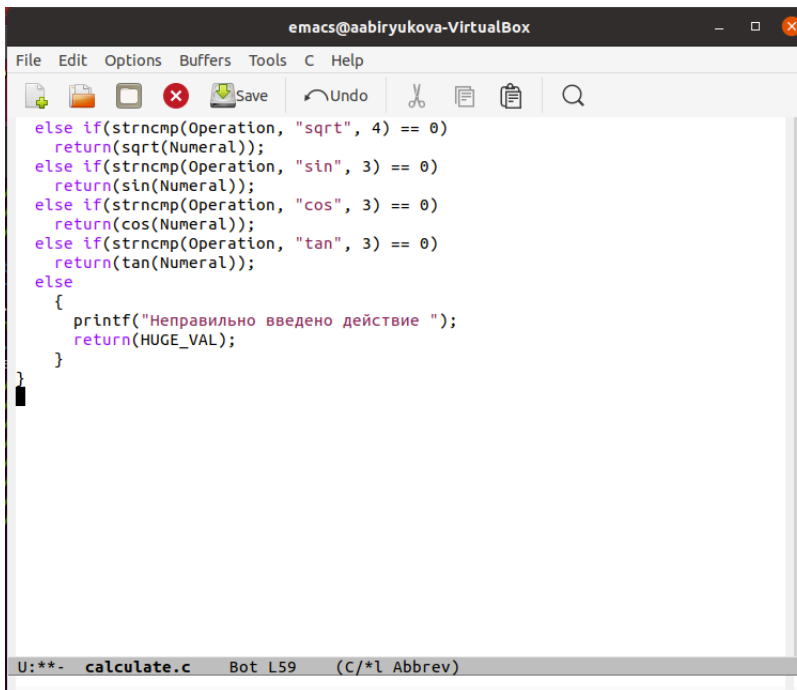
```
emacs@aabiryukova-VirtualBox  
File Edit Options Buffers Tools C Help  
[Icons]  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
#include "calculate.h"  
  
float  
Calculate(float Numeral, char Operation[4])  
{  
    float SecondNumeral;  
    if(strncmp(Operation, "+",1) == 0)  
    {  
        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);  
        return(Numeral / SecondNumeral);  
    }  
}
```

Рис.3



```
scanf("%f", &SecondNumeral);
return(Numeral * SecondNumeral);
}
else if(strncmp(Operation, "/",1) ==0)
{
printf("Делитель: ");
scanf("%f", &SecondNumeral);
if(SecondNumeral ==0)
{
printf("Error");
return(HUGE_VAL);
}
else return(Numeral / SecondNumeral);
}
else if(strncmp(Operation, "pow",1) ==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));
}
U:*** calculate.c 40% L29 (C/*l Abbrev)
```

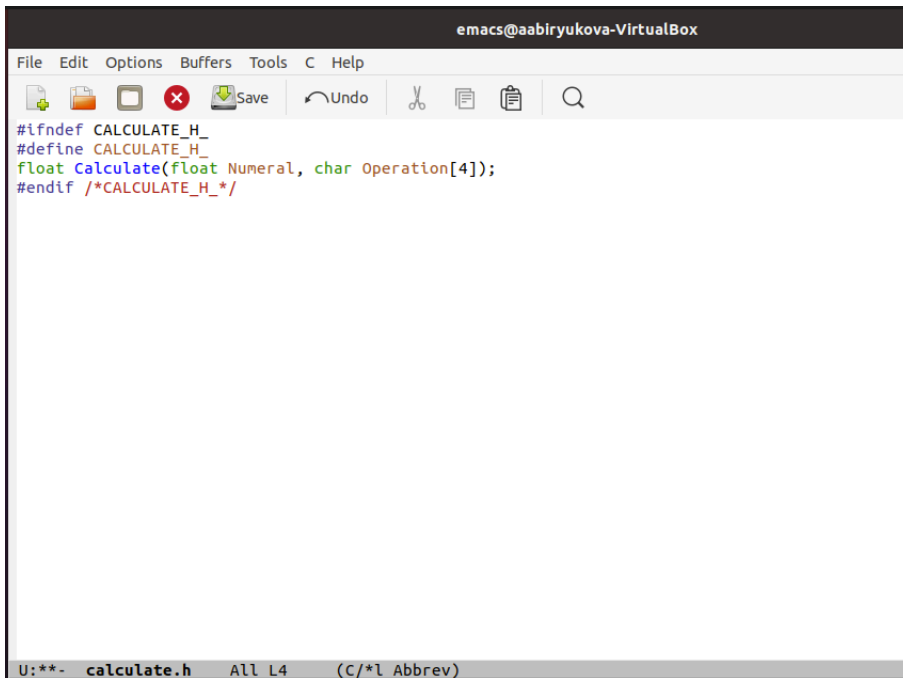
Рис.4



```
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 L59 (C/*l Abbrev)
```

Рис.5

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора(Рис.6)

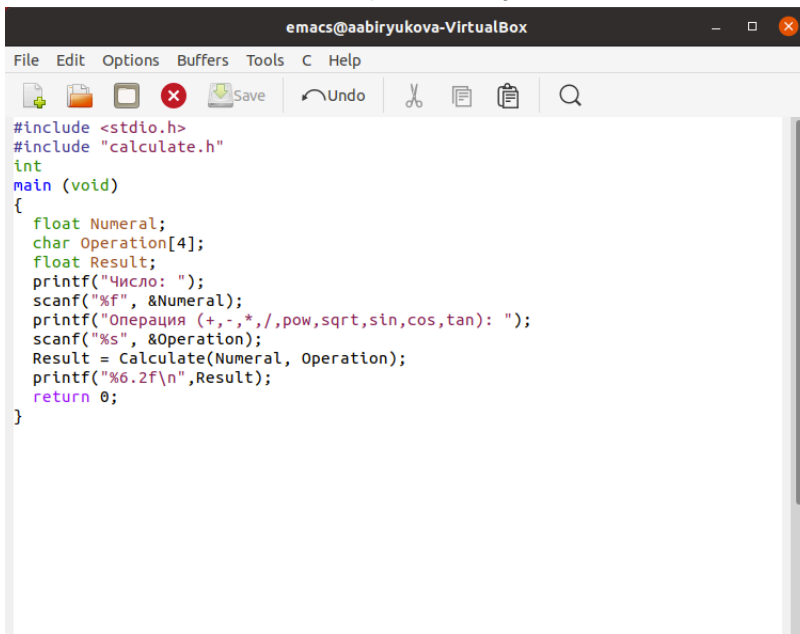


```
File Edit Options Buffers Tools C Help
Save Undo
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/

U:**- calculate.h All L4 (C/*l Abbrev)
```

Рис.6

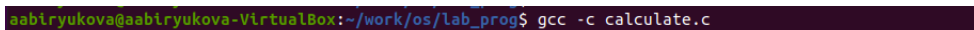
Основной файл main.c, реализующий интерфейс пользователя к калькулятору (Рис.7)



```
File Edit Options Buffers Tools C Help
Save Undo
#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("%6.2f\n",Result);
    return 0;
}
```

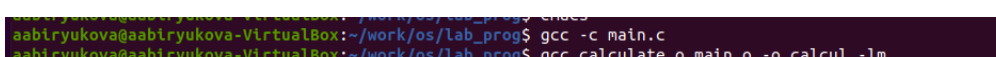
Рис.7

Выполнила компиляцию программы посредством gcc, используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (Рис.8-9)



```
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ gcc -c calculate.c
```

Рис.8



```
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ gcc -c main.c
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
```

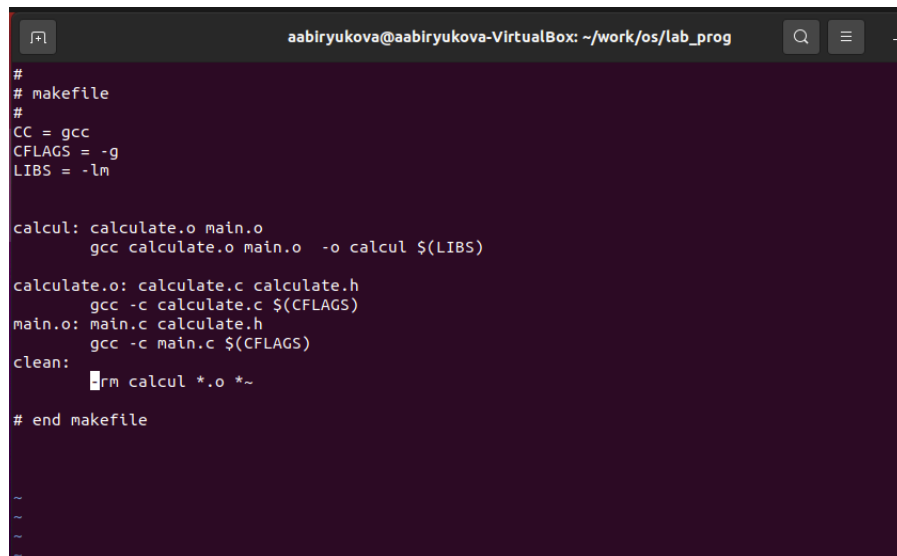
Рис.9

В ходе компиляции программы никаких ошибок выявлено не было.

Создала Makefile с необходимым содержанием (Рис.10-11)

```
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ cd
aabiryukova@aabiryukova-VirtualBox:~$ touch makefile
```

Рис.10



```
#
# 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.11

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.

После этого я удалила исполняемые и объектные файлы из каталога с помощью команды «make clear». Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (Рис.12)

```
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ make clean
rm calcul *.o *~
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ make calculate.o
gcc -c calculate.c -g
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ make main.o
gcc -c main.c -g
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ make calcul
gcc calculate.o main.o -o calcul -lm
```

Рис.12

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul»(Рис.13)

```
aablryukova@aablryukova-VirtualBox:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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...
```

Рис.13

Для запуска программы внутри отладчика ввела команду «run»

Для страничного (по 9 строк) просмотра исходного кода использовала команду «list» (Рис.14)

```
(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);
```

Рис.14

Для просмотра определённых строк не основного файла использовала команду «list calculate.c:20,29» (Рис.15)

```
(gdb) list 12,15
12     scanf("%s", &Operation[4]);
13     Result = Calculate(Numeral, Operation);
14     printf("%6.2f\n", Result);
15     return 0;
```

Рис.15

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (Рис.16-17)

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

Рис.16

```
(gdb) list calculate.c:20,27
20     else if(strncmp(Operation, "*",1) ==0)
21     {
22         printf("Множитель: ");
23         scanf("%f", &SecondNumeral);
24         return(Numeral * SecondNumeral);
25     }
26     else if(strncmp(Operation, "/",1) ==0)
27     {
(gdb) break 21
Breakpoint 1 at 0x1335: file calculate.c, line 22.
```

Рис. 17

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (Рис.18)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address              What
1      breakpoint      keep y   0x0000000000001335 in calculate at calculate.c:22
```

Рис.18

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «-» и «backtrace»

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral»

Убрала точки останова с помощью команд «info breakpoints» и «delete 1» (Рис.19)

```
(gdb) delite 1
Undefined command: "delite". Try "help".
(gdb) delete 1
```

Рис.19

С помощью утилиты splint проанализировала коды файлов calculate.c и main.c.

Предварительно я установила данную утилиту.

Далее воспользовалась командами «splint calculate.c» и «splint main.c».(Рис.20-21)

```
aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:3:37: 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:6:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function calculate)
calculate.c:12:1: 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:17:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:29:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:4: 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:33:7: Return value type double does not match declared type float:
    (HUGE_VAL)
```

Рис.20



```

aabiryukova@aabiryukova-VirtualBox:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:3:37: 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:10:3: Return value (type int) ignored: scanf("%f", &Num...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
main.c:12:3: Return value (type int) ignored: scanf("%s", &Ope...
main.c:13:31: Passed storage Operation not completely defined (*Operation is
                undefined): Calculate (... , Operation)
    Storage derivable from a parameter, return value or global is not defined.
    Use /*out@*/ to denote passed or returned storage which need not be defined.
    (Use -compdef to inhibit warning)

Finished checking --- 4 code warnings

```

Рис.21

## 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:

о кодирование – по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода;

о сборка, компиляция и разработка исполняемого модуля;

о 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: <цель\_1> <цель\_2> ... : <зависимость\_1> <зависимость\_2> ... <команда 1>

...

<команда n>

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.

Пример более сложного синтаксиса Makefile:

---

## Makefile for abcd.c

---

```
CC = gcc CFLAGS =
```

## Compile abcd.c normaly abcd: abcd.c

---

```
$(CC) -o abcd $(CFLAGS) abcd.c clean:
```

```
-rm abcd *.o *~
```

## End Makefile for abcd.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 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:

cscope – 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 C 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 работе программы, переменные с некорректно заданными значениями и типами и многое другое.

## Выводы

---

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 С калькулятора с простейшими функциями.