

## Отчет к лабораторной работе №13

### Common information

discipline: Операционные системы

author: Бабина Юлия Олеговна

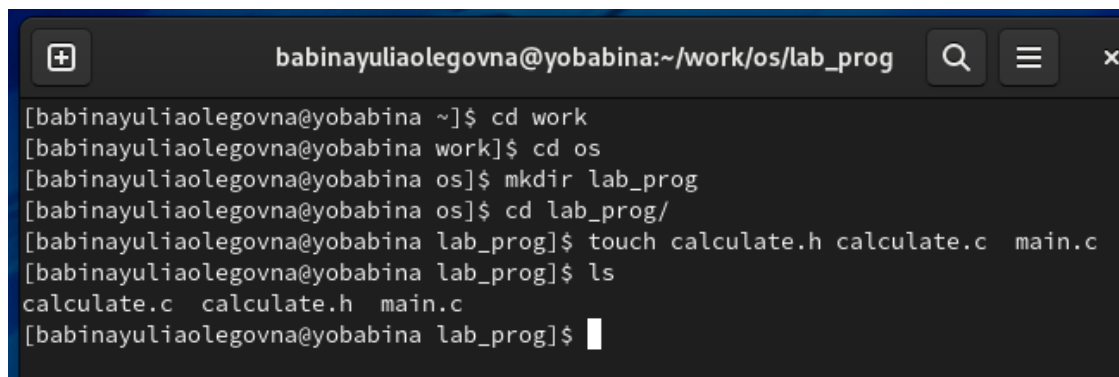
group: НПМбд-01-21

### 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.

### Ход работы

В домашнем каталоге создаем подкаталог `~/work/os/lab_prog` и в нем уже создаем три файла: `calculale.h`, `calculale.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.

A screenshot of a terminal window with a dark background. The window title is "babinayuliaolegovna@yobabina:~/work/os/lab\_prog". The terminal shows a series of commands and their outputs: 

```
[babinayuliaolegovna@yobabina ~]$ cd work
[babinayuliaolegovna@yobabina work]$ cd os
[babinayuliaolegovna@yobabina os]$ mkdir lab_prog
[babinayuliaolegovna@yobabina os]$ cd lab_prog/
[babinayuliaolegovna@yobabina lab_prog]$ touch calculate.h calculate.c main.c
[babinayuliaolegovna@yobabina lab_prog]$ ls
calculate.c calculate.h main.c
[babinayuliaolegovna@yobabina lab_prog]$
```

#### *создание каталога и файла*

В созданных файлах напишем программы для работы калькулятора, которые нам предоставили.

```
emacs@yobabina
File Edit Options Buffers Tools C Help
[Icons: File, Folder, Save, X, Save, Undo, Cut, Copy, Paste, Find]

#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);
        }
    }
}

U:--- calculale.c Top L27 (C/*l Abbrev)

[emacs Tutorial] Learn basic keystroke commands (Учебник Emacs)
[Emacs Guided Tour] Overview of Emacs features at gnu.org
[View Emacs Manual] View the Emacs manual using Info
[11:00%] GNU Emacs 90.1.3 (Fundamental)
```

файл *calculale.c*

```
emacs@yobabina
File Edit Options Buffers Tools C Help
[Icons: File, Folder, Save, X, Save, Undo, Cut, Copy, Paste, Find]

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

#endif /*CALCULATE_H_*/
```

файл *calculale.h*

```
emacs@yobabina
File Edit Options Buffers Tools C Help
[Icons: File, Folder, Disk, X, Save, Undo, Cut, Copy, Paste, Find]

#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;
}
```

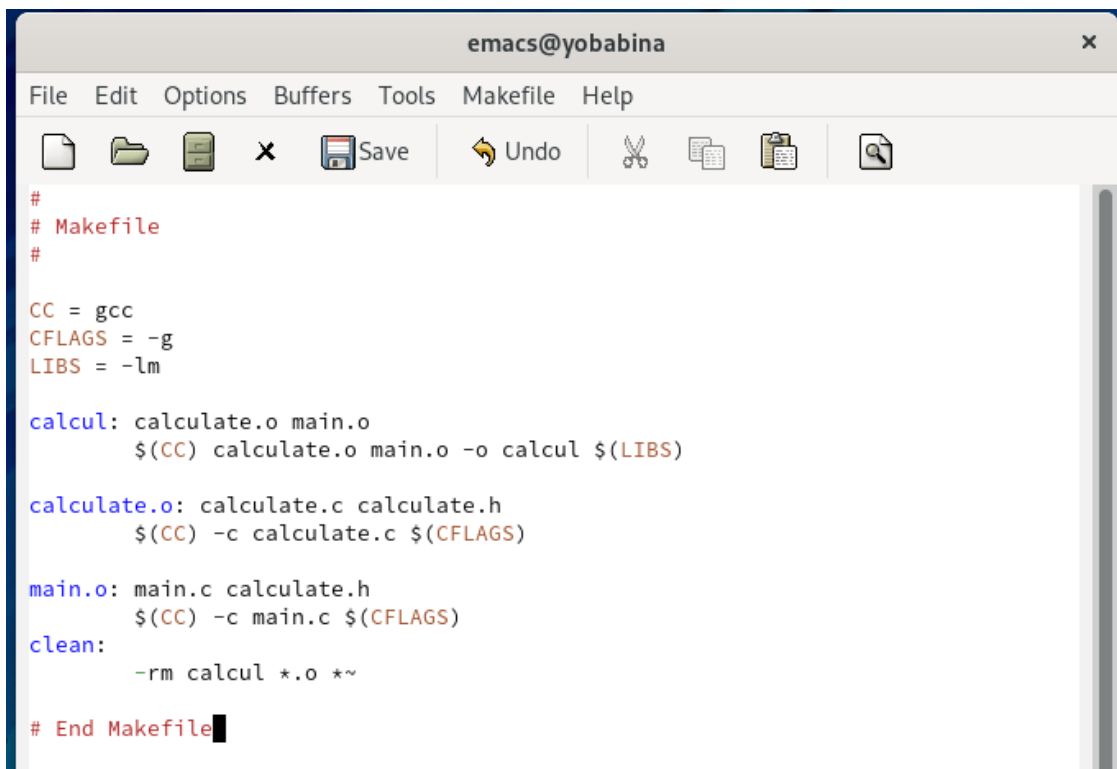
*файл main.c*

Выполним компиляцию программы посредством gcc:

```
компиляция прервана.
[babinayuliaolegovna@yobabina lab_prog]$ gcc -c calculale.c
calculale.c:4:10: фатальная ошибка: calculate.h: Нет такого файла или каталога
 4 | #include "calculate.h"
   |
компиляция прервана.
[babinayuliaolegovna@yobabina lab_prog]$ gcc -c calculale.c
[babinayuliaolegovna@yobabina lab_prog]$ gcc -c main.c
main.c:2:10: фатальная ошибка: calculate.h: Нет такого файла или каталога
 2 | #include "calculate.h"
   |
компиляция прервана.
[babinayuliaolegovna@yobabina lab_prog]$ gcc -c main.c
[babinayuliaolegovna@yobabina lab_prog]$ gcc -cx calculale.o main.o -o calcul -l
m
gcc: ошибка: unrecognized command-line option «-cx»; did you mean «-c»?
[babinayuliaolegovna@yobabina lab_prog]$ gcc -c calculale.o main.o -o calcul -lm
gcc: предупреждение: calculale.o: входные файлы компоновки не использованы, поско
льку компоновка не выполнялась
gcc: предупреждение: main.o: входные файлы компоновки не использованы, поскольку
компоновка не выполнялась
[babinayuliaolegovna@yobabina lab_prog]$ gcc calculale.o main.o -o calcul -lm
[babinayuliaolegovna@yobabina lab_prog]$
```

*компиляция*

Создадим Makefile со следующим содержанием:



```
#
# 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)

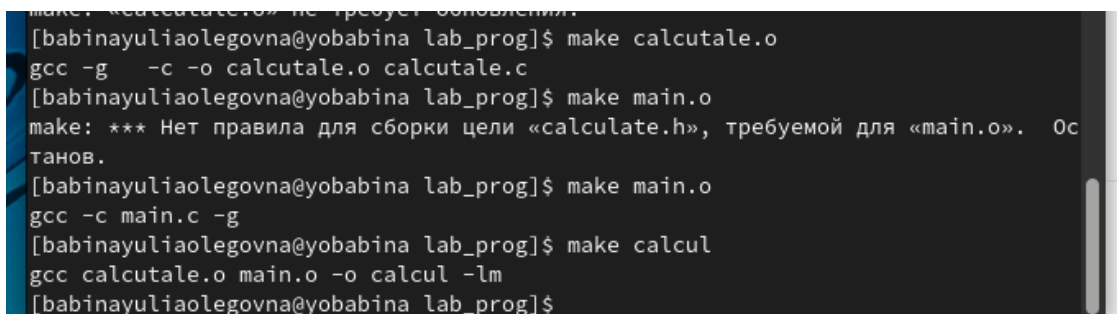
main.o: main.c calculate.h
    $(CC) -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# End Makefile
```

*файл makefile*

Пересоберем проект с помощью данного файла.



```
make: «calcul.o» не требует обновления.
[babinayuliaolegovna@yobabina lab_prog]$ make calcul.o
gcc -g -c -o calcul.o calcul.c
[babinayuliaolegovna@yobabina lab_prog]$ make main.o
make: *** Нет правила для сборки цели «calculate.h», требуемой для «main.o». Остановка.
[babinayuliaolegovna@yobabina lab_prog]$ make main.o
gcc -c main.c -g
[babinayuliaolegovna@yobabina lab_prog]$ make calcul
gcc calcul.o main.o -o calcul -lm
[babinayuliaolegovna@yobabina lab_prog]$
```

*сборка при помощи файла make*

С помощью команды gdb выполним отладку программы calcul.

```
babinayuliaolegovna@yobabina:~/work/os/lab_prog — gdb ./c...
[babinayuliaolegovna@yobabina ~]$ cd '/home/babinayuliaolegovna/work/os/lab_prog'
[babinayuliaolegovna@yobabina lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-3.fc36
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) █
```

*запуск команды gdb./calcul*

```
Reading symbols from ./calcul...
(gdb) run
Starting program: /home/babinayuliaolegovna/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 separate debug info for /home/babinayuliaolegovna/work/os/lab_prog/system-supplied DSO at 0x7ffff7fc4000...
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: 20
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
100.00
```

*запуск команды run*

```
(gdb) list
1      #include <stdio.h>
2      #include "calcutale.h"
3
4      int
5      main (void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Число: ");
(gdb)
```

использование команды list

```
(gdb) list 12,15
12         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13         scanf("%s", Operation);
14         Result = Calculate(Numeral, Operation);
15         printf("%6.2f\n",Result);
(gdb)
```

использование команды list

```
(gdb) list calculale.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)
```

использование команды list

```
(gdb) list calculale.c:15,22
15         }
16         else if(strncmp(Operation, "-", 1) == 0)
17         {
18             printf("Вычитаемое: ");
19             scanf("%f",&SecondNumeral);
20             return(Numeral - SecondNumeral);
21         }
22         else if(strncmp(Operation, "*", 1) == 0)
(gdb) break 18
Breakpoint 4 at 0x40120f: file calculale.c, line 18.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
4     breakpoint      keep y   0x000000000040120f in Calculate
                                           at calculale.c:18
(gdb)
```

использование команды list

```

at calculate.c:18
(gdb) run
Starting program: /home/babinayuliaolegovna/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 4, calculate (Numeral=5, Operation=0x7fffffffdec4 "-") at calculale.c
:18
18             printf("Вычитаемое: ");
(gdb)

```

*установка точки установка*

```

18             printf("Вычитаемое: ");
(gdb) backtrace
#0 calculate (Numeral=5, Operation=0x7fffffffdec4 "-") at calculale.c:18
#1 0x00000000004014eb in main () at main.c:14
(gdb)

```

*команда backtrace*

```

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

```

*проверка работы*

```

(gdb) info breakpoints
Num   Type           Disp Enb Address                  What
4      breakpoint     keep y   0x000000000040120f in calculate
                                           at calculale.c:18
      breakpoint already hit 1 time
(gdb) delete 4
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)

```

*удаление точки останова*

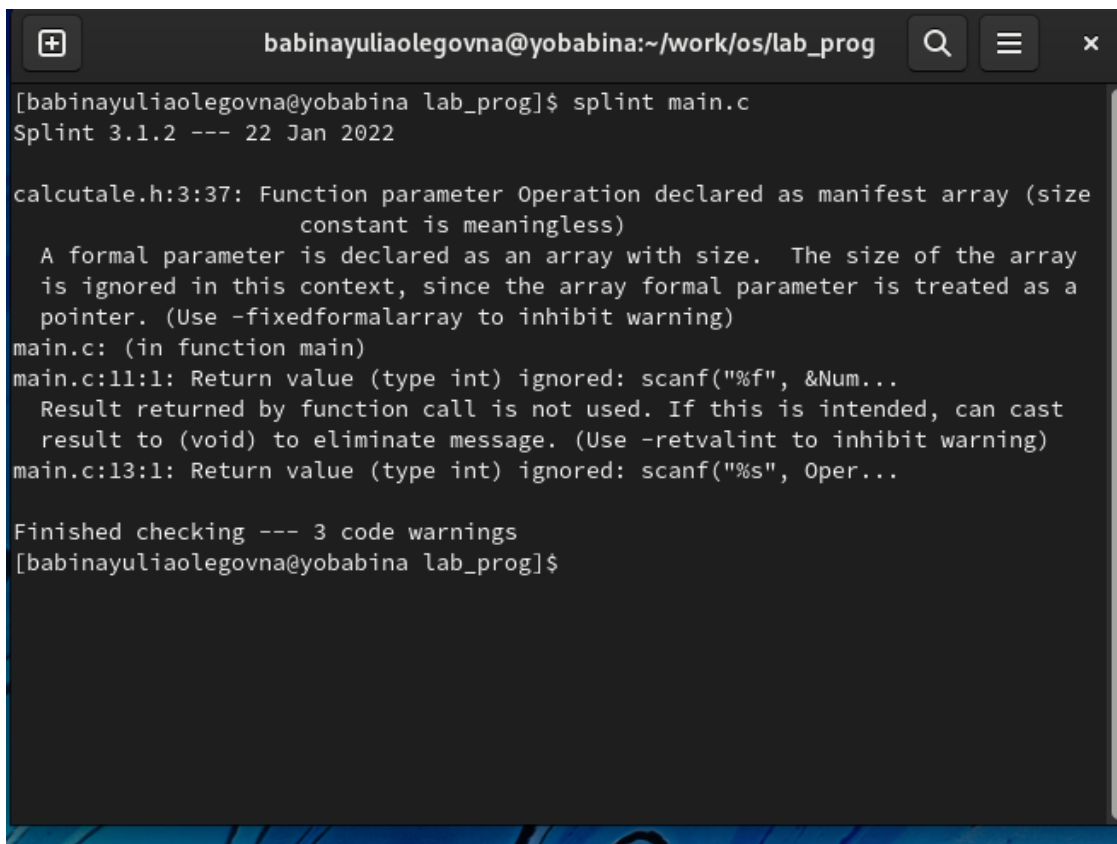
```
babinayuliaolegovna@yobabina:~/work/os/lab_prog
[ babinayuliaolegovna@yobabina lab_prog ]$ splint calculale.c
Splint 3.1.2 --- 22 Jan 2022

calculale.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)
calculale.c:7:31: Function parameter Operation declared as manifest array (size
      constant is meaningless)
calculale.c: (in function Calculate)
calculale.c:13:6: 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)
calculale.c:19:6: Return value (type int) ignored: scanf("%f", &Sec...
calculale.c:25:6: Return value (type int) ignored: scanf("%f", &Sec...
calculale.c:31:6: Return value (type int) ignored: scanf("%f", &Sec...
calculale.c:32:9: 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)
calculale.c:35:13: Return value type double does not match declared type float:
```

*удаление точки останова*

В конце с помощью утилиты `splint` попробуем проанализировать коды файлов `calculale.c` и `main.c`





```
[babinayuliaolegovna@yobabina lab_prog]$ splint main.c
Splint 3.1.2 --- 22 Jan 2022

calcutale.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:11:1: 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:13:1: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
[babinayuliaolegovna@yobabina lab_prog]$
```

утилита *splint*

## Ответы на контрольные вопросы

### Вопрос 1

Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.

### Вопрос 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.

### Вопрос 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».

### Вопрос 4

Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

### Вопрос 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.

### Вопрос 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.

### Вопрос 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

### **Вопрос 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.

### **Вопрос 9**

Схема отладки программы показана в 6 пункте лабораторной работы.

### **Вопрос 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.

### **Вопрос 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.

### **Вопрос 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.

## **Вывод**

В ходе 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.