

Отчет к лабораторной работе №13

Common information

discipline: Операционные системы

group: НПМбд-01-21

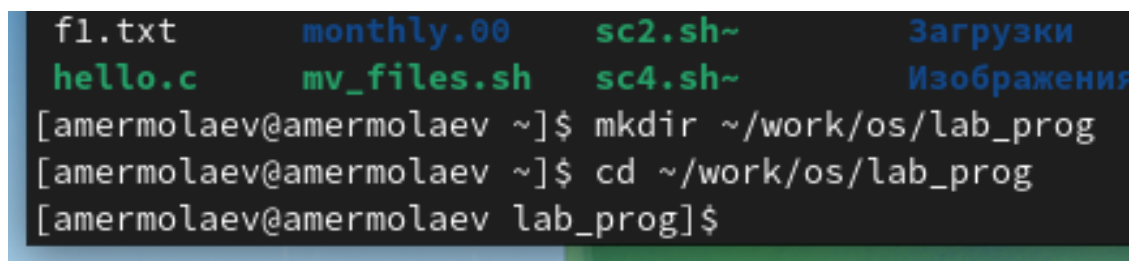
author: Ермолаев А.М.

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.

Выполнение работы

Создадим директорию в ~/work/os/lab_prog.



```
f1.txt      monthly.00  sc2.sh~     Загрузки
hello.c     mv_files.sh  sc4.sh~     Изображения
[amermolaev@amermolaev ~]$ mkdir ~/work/os/lab_prog
[amermolaev@amermolaev ~]$ cd ~/work/os/lab_prog
[amermolaev@amermolaev lab_prog]$
```

создание директории

Создадим в нём файлы: calculate.h, calculate.c, main.c.

Напишем содержимое файла 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);
    }
    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);
    }
}

```

Напишем содержимое файла calculate.h:

```

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

#endif

```

Напишем содержимое файла 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("%6.2f\n",Result);
    return 0;
}

```

Выполним компиляцию программы посредством gcc:

```

[amermolaev@amermolaev lab_prog]$ gcc -c calculate.c
In file included from calculate.c:7:
calculate.h:4: ошибка: незавершённая #ifndef
  4 | #ifndef CALCULATE_H_
    |
calculate.h:9:6: ошибка: expected «;» before «float»
  9 | endif /*CALCULATE_H_*/
    |           ^
    |           ;
[amermolaev@amermolaev lab_prog]$ emacs calculate.c
[amermolaev@amermolaev lab_prog]$ emacs calculate.h
[amermolaev@amermolaev lab_prog]$ gcc -c calculate.c
[amermolaev@amermolaev lab_prog]$ gcc -c main.c
[amermolaev@amermolaev lab_prog]$ gcc calculate.o main.o -o calcul -lm
[amermolaev@amermolaev lab_prog]$ ./calcul
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sin
-0.54
[amermolaev@amermolaev lab_prog]$

```

компиляция

Создадим Makefile со следующим содержанием:

```

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS)

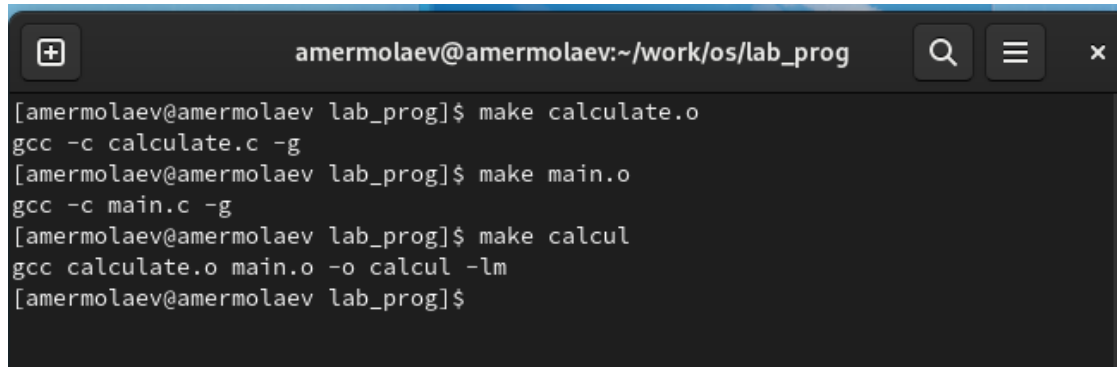
```

```
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 *~
```

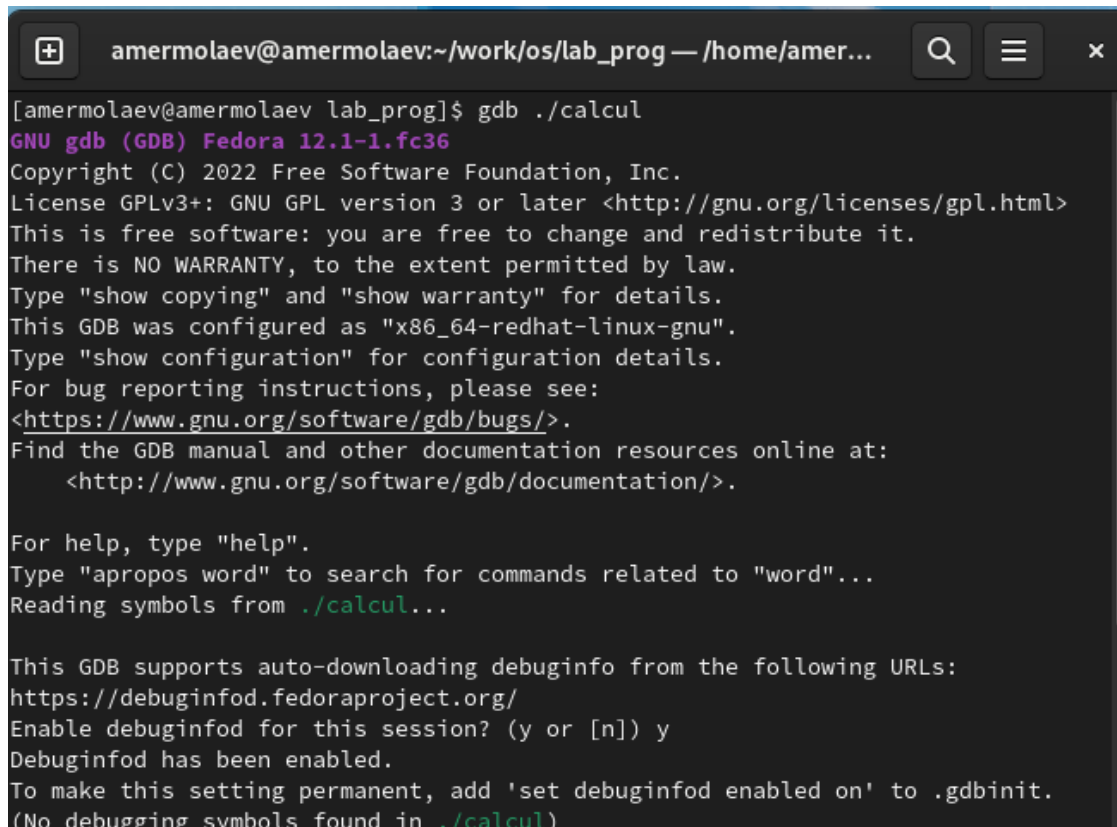
Пересоберем проект с помощью данного файла.



```
amermolaev@amermolaev:~/work/os/lab_prog
[amermolaev@amermolaev lab_prog]$ make calculate.o
gcc -c calculate.c -g
[amermolaev@amermolaev lab_prog]$ make main.o
gcc -c main.c -g
[amermolaev@amermolaev lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
[amermolaev@amermolaev lab_prog]$
```

сборка при помощи файла make

С помощью gdb выполните отладку программы calcul.



```
amermolaev@amermolaev:~/work/os/lab_prog — /home/amer...
[amermolaev@amermolaev lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-1.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...

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.
(No debugging symbols found in ./calcul)
```

запуск команды gdb ./calcul

```
(gdb) run
Starting program: /home/amermolaev/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.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): 1
Неправильно введено действие      inf
[Inferior 1 (process 5002) exited normally]
(gdb) run
Starting program: /home/amermolaev/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 54
66.00
[Inferior 1 (process 5005) exited normally]
```

запуск команды run

```

(gdb) list
1  //////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int main (void){
8      float Numeral;
9      char Operation[4];
10     float Result;
(gdb) list 12,15
12     scanf("%f",&Numeral);
13     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
14     scanf("%s",&Operation);
15     Result = Calculate(Numeral, Operation);
(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)

```

использование команды list

```

(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint     keep y   0x000000000040120f in Calculate
                                                at calculate.c:21

```

установка точки останова

```

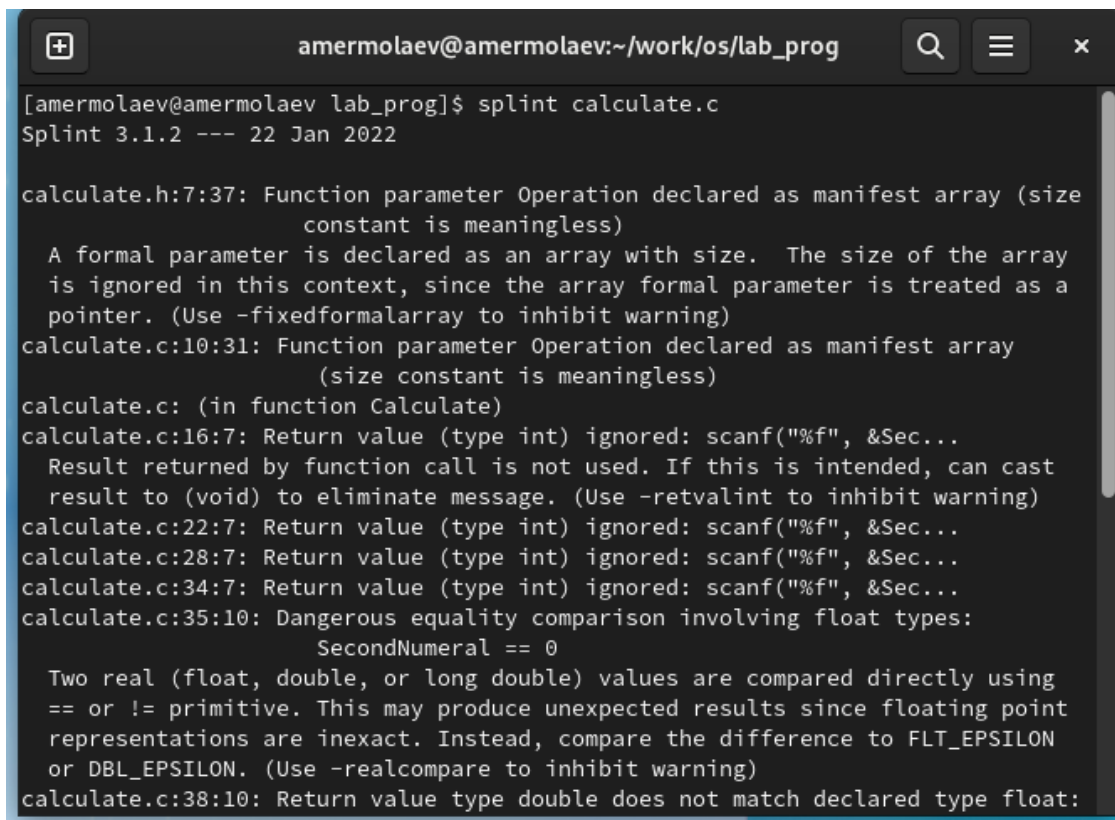
(gdb) run
Starting program: /home/amercolaev/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

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 0x00000000004014eb in main () at main.c:15
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num   Type             Disp Enb Address                  What
1     breakpoint       keep y   0x000000000040120f in Calculate
                                           at calculate.c:21
      breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)

```

проверка корректности работы и удаление точки останова

В конце с помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.



```

[amercolaev@amercolaev lab_prog]$ splint calculate.c
Splint 3.1.2 --- 22 Jan 2022

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:

```

утилита splint

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

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:12:3: 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:14:14: 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:14:11: Corresponding format code
main.c:14:3: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[amermolaev@amermolaev 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 -h <команда>`.

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.

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 рамках выполнения работы я приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.