

Отчёт по лабораторной работе №13

Операционные системы

Обрезкова Анастасия Владимировна

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выводы	13
4	Ответы на контрольные вопросы	14

Список иллюстраций

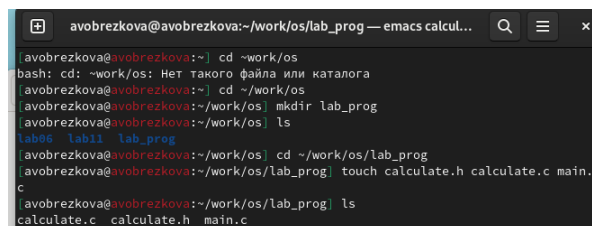
2.1	Создание подкаталога	5
2.2	Создание подкаталога	5
2.3	Файл calculate.c	6
2.4	Файл calculate.c	6
2.5	Файл calculate.h	6
2.6	Файл main.c	7
2.7	Компиляция файла	7
2.8	Makefile	8
2.9	Изменения	8
2.10	Компиляция	8
2.11	Отладчик	9
2.12	Запуск программы	9
2.13	Просмотр кода	9
2.14	Просмотр строк	10
2.15	Просмотр строк	10
2.16	Установка точки	10
2.17	Информация о точках	10
2.18	Остановка программы	11
2.19	Просмотр значения	11
2.20	Удаление точки останова	11
2.21	Анализ файла 1	12
2.22	Анализ файла 2	12

1 Цель работы

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

2 Выполнение лабораторной работы

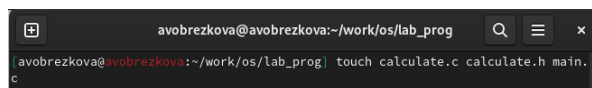
1. В домашнем каталоге создаём подкаталог “~/work/os/lab_prog” с помощью команды “mkdir”.(рис. 2.1).



```
avobrezkova@avobrezkova:~/work/os/lab_prog — emacs calcul...
avobrezkova@avobrezkova:~$ cd ~/work/os
bash: cd: ~/work/os: Нет такого файла или каталога
avobrezkova@avobrezkova:~$ cd ~/work/os
avobrezkova@avobrezkova:~/work/os$ mkdir lab_prog
avobrezkova@avobrezkova:~/work/os$ ls
lab_prog
avobrezkova@avobrezkova:~/work/os$ cd ~/work/os/lab_prog
avobrezkova@avobrezkova:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
avobrezkova@avobrezkova:~/work/os/lab_prog$ ls
calculate.c calculate.h main.c
```

Рис. 2.1: Создание подкаталога

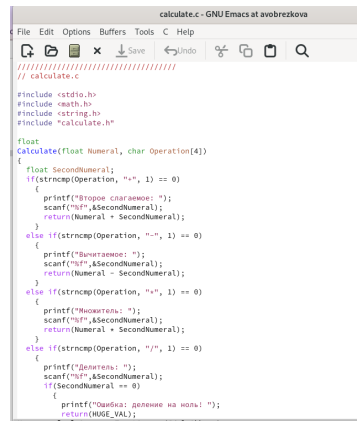
2. Затем перейдём в каталог и создадим файлы: calculate.h, calculate.c, main.c. Делаю это с помощью команды “touch”.(рис. 2.2).



```
avobrezkova@avobrezkova:~/work/os/lab_prog
avobrezkova@avobrezkova:~/work/os/lab_prog$ touch calculate.c calculate.h main.c
```

Рис. 2.2: Создание подкаталога

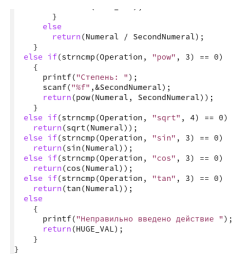
Создадим примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится. Реализация функций калькулятора будет делать в файле calculate.c.(рис. 2.3), (рис. 2.4).



```
calculate.c - GNU Emacs at avobrezkova
File Edit Options Buffers Tools C Help
// calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if (strcmp(Operation, "+", 1) == 0)
    {
        printf("Сложение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if (strcmp(Operation, "-", 1) == 0)
    {
        printf("Вычитание: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if (strcmp(Operation, "*", 1) == 0)
    {
        printf("Умножение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if (strcmp(Operation, "/", 1) == 0)
    {
        printf("Деление: ");
        scanf("%f", &SecondNumeral);
        if (SecondNumeral == 0)
        {
            printf("Ошибка: деление на ноль: ");
            return(MUGE_VAL);
        }
    }
}
```

Рис. 2.3: Файл calculate.c

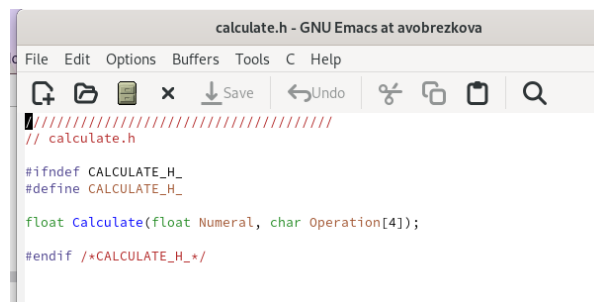


```

    }
    else
    {
        return(Numeral / SecondNumeral);
    }
    else if (strcmp(Operation, "pow", 3) == 0)
    {
        printf("Степень: ");
        scanf("%f", &SecondNumeral);
        return(pow(Numeral, SecondNumeral));
    }
    else if (strcmp(Operation, "sqrt", 4) == 0)
    {
        return(sqrt(Numeral));
    }
    else if (strcmp(Operation, "sin", 3) == 0)
    {
        return(sin(Numeral));
    }
    else if (strcmp(Operation, "cos", 3) == 0)
    {
        return(cos(Numeral));
    }
    else if (strcmp(Operation, "tan", 3) == 0)
    {
        return(tan(Numeral));
    }
    else
    {
        printf("Неправильно введено действие ");
        return(MUGE_VAL);
    }
}
```

Рис. 2.4: Файл calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора.(рис. 2.5).



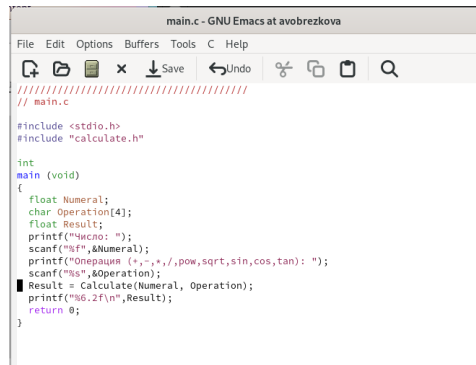
```
calculate.h - GNU Emacs at avobrezkova
File Edit Options Buffers Tools C Help
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 2.5: Файл calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору.(рис. 2.6).

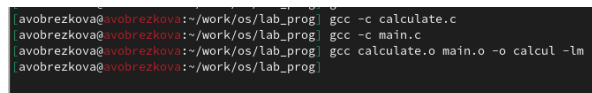


```
main.c - GNU Emacs at avobrezkova
File Edit Options Buffers Tools C Help
// main.c
#include <stdio.h>
#include "calculate.h"

int
main (void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Введите: ");
    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. 2.6: Файл main.c

3. Далее выполним компиляцию программы посредством gcc. (рис. 2.7).



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

Рис. 2.7: Компиляция файла

4. Ошибок не выявлено
5. Создадим Makefile с необходимым содержанием. Он необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а так же их объединения в один исполняемый файл 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.(рис. 2.8).

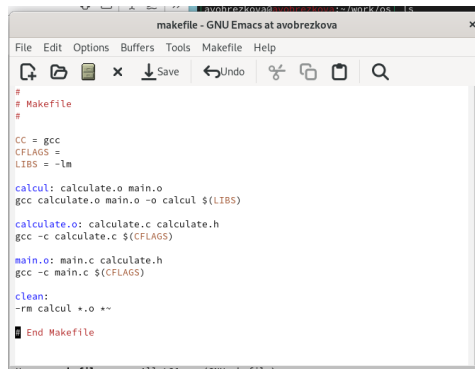


Рис. 2.8: Makefile

- Далее изменим файл. В переменную CFLAGS добавим “-g”, которая необходима для компиляции объектных файлов и их использования в программе отладчика GDB. Также, компиляция выбирается с помощью переменной CC.(рис. 2.9)

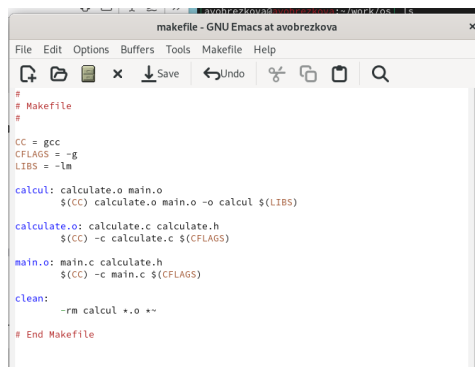


Рис. 2.9: Изменения

После выполняем компиляцию файлов. (рис. 2.10)

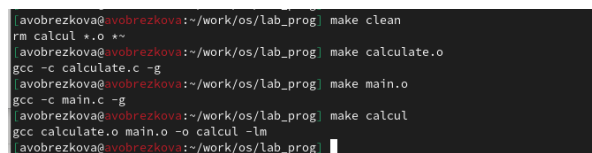


Рис. 2.10: Компиляция

После этого выполняем gdb отладку программы calcul. Запускаем GDB и загружаем в него программу для отладки, используя команду “gdb ./calcul”(рис. 2.11)

```
avobrezkova@avobrezkova:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-2.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)
```

Рис. 2.11: Отладчик

Далее вводим команду “run” для запуска программы внутри отладчика.(рис. 2.12)

```
(gdb) run
Starting program: /home/avobrezkova/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 7.42 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: 10
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 14
24.00
[Inferior 1 (process 4575) exited normally]
(gdb)
```

Рис. 2.12: Запуск программы

Для постраничного просмотра исходного кода используем команду “list”.(рис. 2.13)

```
(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)
```

Рис. 2.13: Просмотр кода

Для просмотра строк с 12 по 15 основного файла используем команду “list 12,15”.(рис. 2.14)

```
(gdb) list 12,15
12     float Result;
13     printf("Result: ");
14     scanf("%f", &Numeral);
15     printf("One pause (%f,%f,%f,pow,sqrt,sin,cos,tan): ");
(gdb)
```

Рис. 2.14: Просмотр строк

Для просмотра определённых строк не основного файла используем команду “list calculate.c:20,29”.(рис. 2.15)

```
(gdb) list calculate.c:20,29
20     {
21         printf("Subtraction: ");
22         scanf("%f",&SecondNumeral);
23         return Numeral - SecondNumeral;
24     }
25     else if (strcmp(Operation, "+") != 0)
26     {
27         printf("Multiplication: ");
28         scanf("%f",&SecondNumeral);
29         return Numeral * SecondNumeral;
(gdb)
```

Рис. 2.15: Просмотр строк

Для установки точки в файле “calculate.c” на строке 21 используем команды “list calculate.c:20,27” и “break 21”.(рис. 2.16)

```
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb)
```

Рис. 2.16: Установка точки

Чтобы вывести информацию об имеющихся точках останова используем команду “info breakpoint”(рис. 2.17)

```
(gdb) info breakpoints
Num   Type       Disp Enb Address          What
1     breakpoint keep y  0x000000000040120f in calculate
      at calculate.c:21
(gdb)
```

Рис. 2.17: Информация о точках

Запустим программу внутри отладчика и убедимся, что программа остановилась в момент прохождения точки останова.(рис. 2.18)

```

(gdb) run
Starting program: /home/avobrezkova/work/os/lab_prog/calcul
BFD: warning: /home/avobrezkova/.cache/debuginfod_client/8257ee907646e9b05719753
3d1e4ac8ede7a9c5c/debuginfo has a section extending past end of file
Error while reading shared library symbols for /lib64/libc.so.6:
'/home/avobrezkova/.cache/debuginfod_client/8257ee907646e9b057197533d1e4ac8ede7a
9c5c/debuginfo': can't read symbols: формат файла не распознан.
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, calculate (Numeral=12, Operation=0x7fffffffdf34 "-") at calculate.
c:21
21      printf("Вычисляемое: ");
(gdb)

```

Рис. 2.18: Остановка программы

Посмотрим, чему на этом этапе равно значение переменной Numeral, с помощью команды “print Numeral” и сравним его с результатом вывода на экран после использования команды “display Numeral”. Значения совпадают.(рис. 2.19)

```

(gdb) print Numeral
$1 = 12
(gdb) display Numeral
1: Numeral = 12
(gdb)

```

Рис. 2.19: Просмотр значения

Уберём точки останова с помощью команды “info breakpoints”(рис. 2.20)

```

(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)

```

Рис. 2.20: Удаление точки останова

7. С помощью утилиты splint проанализировала коды файлов calculate.c и main.c.(рис. 2.21), (рис. 2.22)

```
avobrezkova@avobrezkova:~/work/os/lab_prog
[avobrezkova@avobrezkova:~/work/os/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:
```

Рис. 2.21: Анализ файла 1

```
avobrezkova@avobrezkova:~/work/os/lab_prog
[avobrezkova@avobrezkova:~/work/os/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: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
[avobrezkova@avobrezkova:~/work/os/lab_prog]
```

Рис. 2.22: Анализ файла 2

Данные изменения можно проверить по ссылке: https://github.com/avobrezkova/study_2022-2023_os-intro/tree/master/labs/lab13

3 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.

4 Ответы на контрольные вопросы

1. как получить информацию о возможностях программ gcc, make, gdb и др.?

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

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

- планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
- проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
- непосредственная разработка приложения:
 - кодирование – по сути создание исходного текста программы (возможно в нескольких вариантах);
 - анализ разработанного кода;
 - сборка, компиляция и разработка исполняемого модуля;
 - тестирование и отладка, сохранение произведённых изменений;
- документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: `vi`, `vim`, `mceditor`, `emacs`, `geany` и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

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. Каково основное назначение компилятора языка C в UNIX?

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

5. Для чего предназначена утилита make?

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. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

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. 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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

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. Назовите и дайте основную характеристику основным командам отладчика gdb.

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. Опишите по шагам схему отладки программы, которую Вы использовали

при выполнении лабораторной работы.

Схема отладки программы показана в 6 пункте лабораторной работы.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

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. Назовите основные средства, повышающие понимание исходного кода программы.

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. Каковы основные задачи, решаемые программой `splint`?

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 работ программы, переменные с некорректно заданными значениями и типами и многое другое