

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

### Кафедра прикладной информатики и теории вероятностей

#### ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

дисциплина: *Операционные системы*

Студент: **ГЕОРГЕС Геден** Группа: **НПМбд-02-20** Студ. бил.:№ **1032204593**

МОСКВА 2021 г.

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

```
georges@georges-VirtualBox:~$ mkdir -p ~/work/os/lab_prog
georges@georges-VirtualBox:~$
```

(Рисунок 1) 1.

1. Создал в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`, используя команды «`cd ~/work/os/lab_prog`» и «`touch calculate.h calculate.c main.c`» (Рисунок 2)

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

(Рисунок 2)

2) 1. 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` (Рисунки 3, 4)

```
emacs@georges-VirtualBox
File Edit Options Buffers Tools C Help

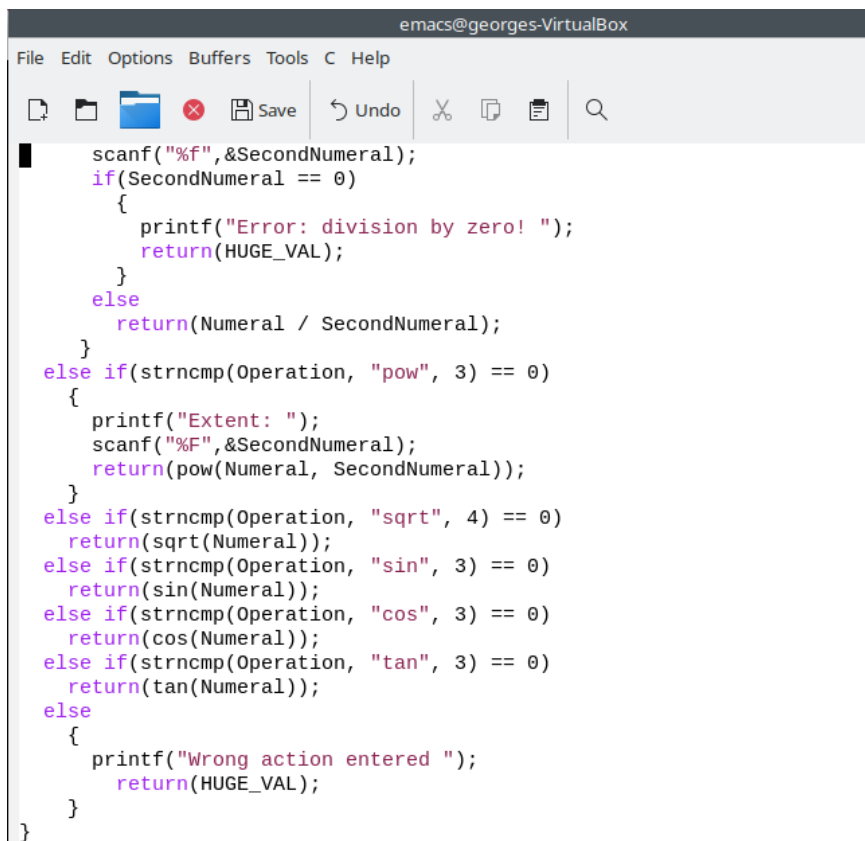
////////////////////
// calculate.c

#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("The second term: ");
        scanf("%f",&SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-", 1) == 0)
    {
        printf("Deductible: ");
        scanf("%f",&SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if (strcmp(Operation, "*", 1) == 0)
    {
        printf("Multiplier: ");
        scanf("%f",&SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strcmp(Operation, "/", 1) == 0)
    {
        printf("Divisor: ");
        scanf("%f",&SecondNumeral);
    }
}

U: ** - calculate.c Top L34 (C/*1 Abbrev)
```

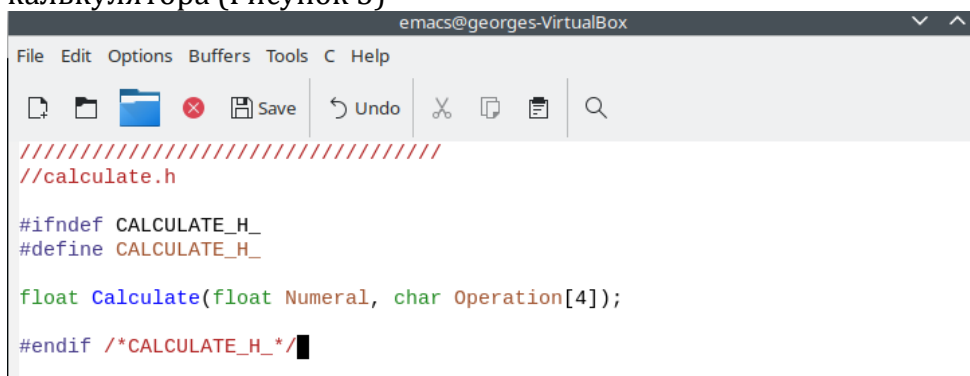
(Рисунок 3)



```
emacs@georges-VirtualBox
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]
scanf("%f",&SecondNumeral);
if(SecondNumeral == 0)
{
    printf("Error: division by zero! ");
    return(HUGE_VAL);
}
else
    return(Numeral / SecondNumeral);
}
else if(strncmp(Operation, "pow", 3) == 0)
{
    printf("Extent: ");
    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("Wrong action entered ");
    return(HUGE_VAL);
}
}
```

(Рисунок 4) 1.

### 3.Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (Рисунок 5)



```
emacs@georges-VirtualBox
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]
////////////////////
//calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

(Рисунок 5) 1.

### 4.Основной файл main.c, реализующий интерфейс пользователя к калькулятору

```
emacs@georges-VirtualBox
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("Number: ");
    scanf("%f",&Numeral);
    printf("Operation (+, -, *, /, pow, sqrt, sin, cos, tan): ");
    scanf("%f",&Operation);
    Result = Calculate(Numeral, Operation);
    printf("%6,2f\n",Result);
    return 0;
}
```

(Рисунок 6) 1.

(Рисунок 6) 2.

Выполнил 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» (Рисунок 7)

```
georges@georges-VirtualBox:~/work/os/lab-prog$ gcc -c calculate.c
georges@georges-VirtualBox:~/work/os/lab-prog$ gcc -c main.c
georges@georges-VirtualBox:~/work/os/lab-prog$ gcc calculate.o main.o -o calcul -lm
```

(Рисунок 7)

7) 3. Создал Makefile с необходимым содержанием (Рисунок 8)

```
emacs@georges-VirtualBox
File Edit Options Buffers Tools Makefile Help

C = gcc
CFLAGS =
LIBS = -lm

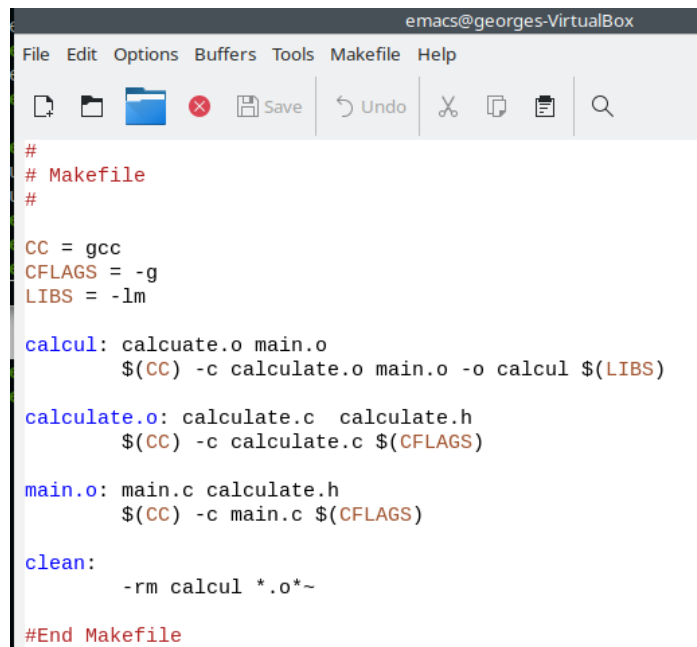
calcul: calculate.o main.o
    gcc -c calculate.o main.o -o calcul $(LIBS)

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 *~
```

(Рисунок 8) 3. 1.Далее исправила



```

#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    $(CC) -c 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~

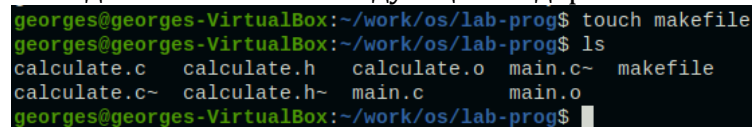
#End Makefile

```

Makefile (Рисунок 9).

(Рисунок 9) 3.

2.Создали Makefile со следующим содержанием:



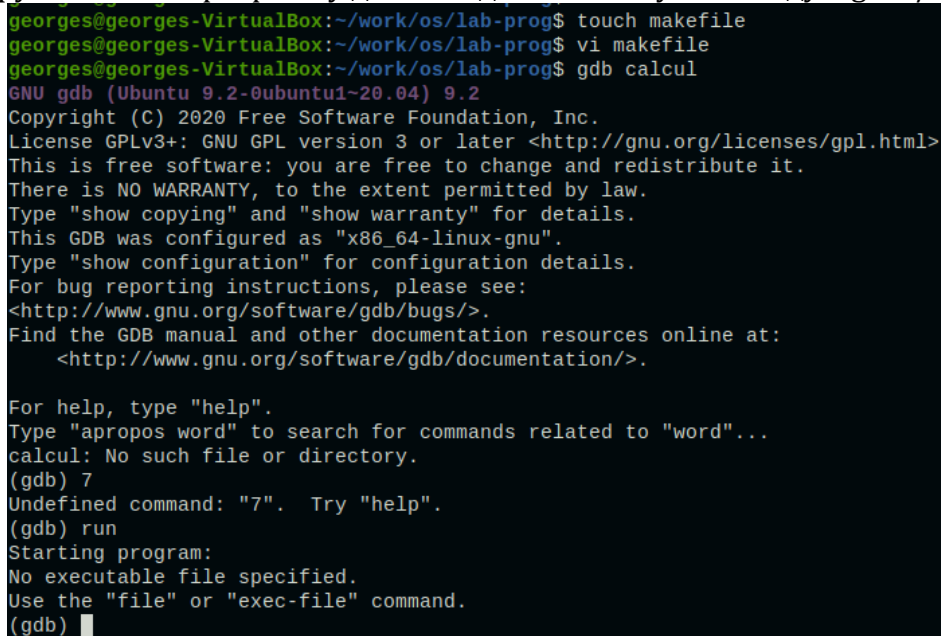
```

georges@georges-VirtualBox:~/work/os/lab-prog$ touch makefile
georges@georges-VirtualBox:~/work/os/lab-prog$ ls
calculate.c  calculate.h  calculate.o  main.c~  makefile
calculate.c~ calculate.h~  main.c      main.o
georges@georges-VirtualBox:~/work/os/lab-prog$

```

(Рисунок 10) 4.

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



```

georges@georges-VirtualBox:~/work/os/lab-prog$ touch makefile
georges@georges-VirtualBox:~/work/os/lab-prog$ vi makefile
georges@georges-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"...
calcul: No such file or directory.
(gdb) 7
Undefined command: "7". Try "help".
(gdb) run
Starting program:
No executable file specified.
Use the "file" or "exec-file" command.
(gdb)

```

11) (Рисунок

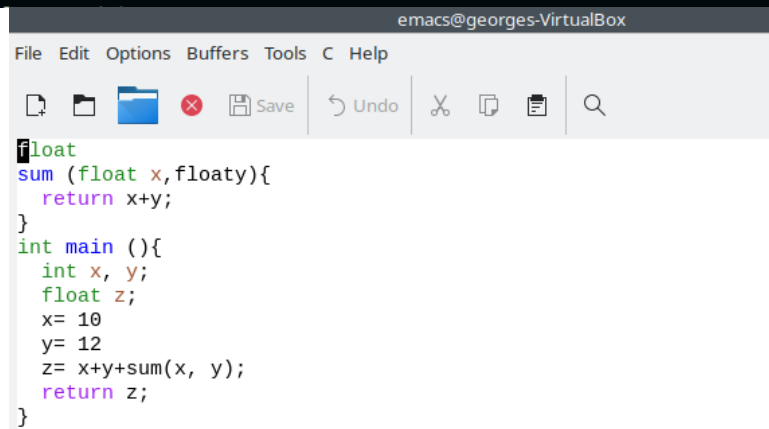
11) 5. Создали файл example.c, записали программу в данный файл (Рисунок 12).

```

georges@georges-VirtualBox:~/work/os/lab-prog$ touch example.c
georges@georges-VirtualBox:~/work/os/lab-prog$ emacs
bc
(emacs:6616): Glib-GObject-WARNING **: The property GtkWidget:use-stock is deprecated and shouldn't be used anymore. It will be removed in a future version.

(emacs:6616): Glib-GObject-WARNING **: The property GtkSettings:gtk-button-images is deprecated and shouldn't be used anymore. It will be removed in a future version
georges@georges-VirtualBox:~/work/os/lab-prog$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.

```



```

float
sum (float x, float y){
    return x+y;
}
int main (){
    int x, y;
    float z;
    x= 10
    y= 12
    z= x+y+sum(x, y);
    return z;
}

```

(Рисунок 12)

(Рисунок 12) С

помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Предварительно я установила данную утилиту с помощью команд «sudo apt update» и «sudo apt install splint» (Рисунок 13)

```

georges@georges-VirtualBox:~/work/os/lab-prog$ sudo apt install splint
[sudo] password for georges:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  distro-info
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  splint-data
Suggested packages:
  splint-doc-html
The following NEW packages will be installed:
  splint splint-data
0 upgraded, 2 newly installed, 0 to remove and 64 not upgraded.
Need to get 740 kB of archives.
After this operation, 2 883 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ru.archive.ubuntu.com/ubuntu focal/universe amd64 splint-data all 1:3.1.2+dfsg-1build1 [57,5 kB]

```

(Рисунок 13) 7. С помощью утилиты splint проанализировали коды файлов calculate.c и main.c. (Рисунок 14)

```

georges@georges-VirtualBox:~/work/os/lab-prog$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

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:10:32: 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:22:15: Unrecognized format code: %F
    Format code in a format string is not valid. (Use -formatcode 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)

```

(Рисунок

```

georges@georges-VirtualBox:~/work/os/lab-prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

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:16:3: Return value (type int) ignored: scanf("%s", &Ope...
main.c:17:31: 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

```

14) georges@georges-VirtualBox:~/work/os/lab-prog\$ (Рисунок 15)

# Вывод: 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. ### Ответы на 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: [1] планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения; [2] проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования; [3] 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; 2) 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: ... : ... <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 normally 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.