

# **Лабораторная работа №13: отчет.**

**Элементы криптографии. Шифрование (кодирование) различных  
исходных текстов одним ключом.**

Евдокимов Максим Михайлович. Группа - НФИбд-01-20.

# Содержание

<b>Цель работы</b>	<b>4</b>
<b>Задание</b>	<b>5</b>
<b>Указание к работе</b>	<b>6</b>
Тестирование и отладка . . . . .	6
<b>Выполнение лабораторной работы</b>	<b>7</b>
Калькулятор . . . . .	7
Коды . . . . .	10
Работа с gdb . . . . .	14
<b>Контрольные вопросы</b>	<b>19</b>
<b>Выводы</b>	<b>25</b>
<b>Список литературы</b>	<b>26</b>

# Список иллюстраций

1	Создание директории . . . . .	7
2	Создание файлов . . . . .	7
3	Файл calculate.c . . . . .	8
4	Файл main.c . . . . .	9
5	Файл calculate.h . . . . .	9
6	Выполнение кодов . . . . .	9
7	Makefile . . . . .	10
8	Запуск gdb . . . . .	14
9	Run кода . . . . .	15
10	Команда list . . . . .	15
11	Команда list с указанием . . . . .	15
12	Команда list по файлу . . . . .	15
13	Команда list по файлу 2 . . . . .	16
14	Установка точки останова . . . . .	16
15	Просмотр breakpoints . . . . .	16
16	Запуск по условию . . . . .	17
17	Вывод значения . . . . .	17
18	Показ имеющегося в системе значения . . . . .	17
19	Удаление точки останова . . . . .	17
20	Команда splint по calculate . . . . .	18
21	Команда splint по main . . . . .	18

## 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.

# Задание

1. Написать по промеру базовый калькулятор.
2. Изучить примеры создания функций на языке программирования Си.
3. Изучить среду gdb.

# Указание к работе

## Тестирование и отладка

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: “1 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: 1 “gdb file.o”. Затем можно использовать по мере необходимости различные команды gdb. Наиболее часто используемые команды gdb приведены в табл. 13.2. 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.

# Выполнение лабораторной работы

## Калькулятор

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.

```
max@evdokimov:~$ mkdir ~/work/os/lab_prog; cd ~/work/os/lab_prog
max@evdokimov:~/work/os/lab_prog$
```

Рис. 1: Создание директории

2. Создайте в нём файлы: calculate.h, calculate.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.

```
max@evdokimov:~$ mkdir ~/work/os/lab_prog; cd ~/work/os/lab_prog
max@evdokimov:~/work/os/lab_prog$ touch calculate.h calculate.c main.c; ls
calculate.c calculate.h main.c
max@evdokimov:~/work/os/lab_prog$ gedit calculate.c
(gedit:13692): Gtk-WARNING **: 13:49:52.932: Calling org.freedesktop.portal.Inhibit.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: Инт
ерфейс «org.freedesktop.portal.Inhibit» для пути /org/freedesktop/portal/desktop объекта не найден
max@evdokimov:~/work/os/lab_prog$ gedit main.c
(gedit:13750): Gtk-WARNING **: 13:49:52.932: Calling org.freedesktop.portal.Inhibit.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: Инт
ерфейс «org.freedesktop.portal.Inhibit» для пути /org/freedesktop/portal/desktop объекта не найден
max@evdokimov:~/work/os/lab_prog$ gedit calculate.h
(gedit:13798): Gtk-WARNING **: 13:49:52.933: Calling org.freedesktop.portal.Inhibit.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: Инт
ерфейс «org.freedesktop.portal.Inhibit» для пути /org/freedesktop/portal/desktop объекта не найден
max@evdokimov:~/work/os/lab_prog$
```

Рис. 2: Создание файлов



```
1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4 #include "calculate.h"
5
6 float
7 Calculate(float Numeral, char Operation[4])
8 {
9     float SecondNumeral;
10    if(strcmp(Operation, "+", 1) == 0)
11    {
12        printf("Второе слагаемое: ");
13        scanf("%f",&SecondNumeral);
14        return(Numeral + SecondNumeral);
15    }
16    else if(strcmp(Operation, "-", 1) == 0)
17    {
18        printf("Вычитаемое: ");
19        scanf("%f",&SecondNumeral);
20        return(Numeral - SecondNumeral);
21    }
22    else if(strcmp(Operation, "*", 1) == 0)
23    {
24        printf("Множитель: ");
25        scanf("%f",&SecondNumeral);
26        return(Numeral * SecondNumeral);
27    }
28    else if(strcmp(Operation, "/", 1) == 0)
29    {
30        printf("Делитель: ");
31        scanf("%f",&SecondNumeral);
32        if(SecondNumeral == 0)
33        {
34            printf("Ошибка: деление на ноль! ");
35            return(HUGE_VAL);
36        }
37        else
38            return(Numeral / SecondNumeral);
39    }
40    else if(strcmp(Operation, "pow", 3) == 0)
41    {
42        printf("Степень: ");
43        scanf("%f",&SecondNumeral);
44        return(pow(Numeral, SecondNumeral));
```

Рис. 3: Файл calculate.c



Рис. 4: Файл main.c

Рис. 5: Файл calculate.h

3. Выполните компиляцию программы посредством gcc:

Рис. 6: Выполнение кодов

4. При необходимости исправьте синтаксические ошибки:

(нет необходимости)

5. Создайте Makefile со следующим содержанием:

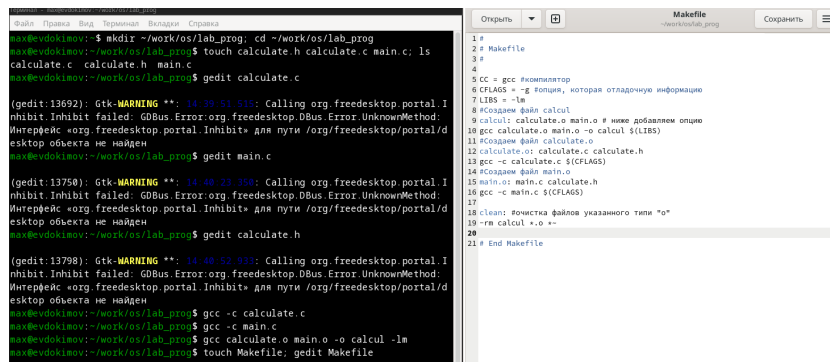


Рис. 7: Makefile

## Коды

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);
}
}

```

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;
}
```

calculate.h:

```
`` ` C
#ifdef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Makefile:

```
#
# Makefile
#

CC = gcc #компилятор
CFLAGS = -g #опция, которая отладочную информацию
LIBS = -lm
#Создаем файл calcul
calcul: calculate.o main.o # ниже добавляем опцию
gcc calculate.o main.o -o calcul $(LIBS)
#Создаем файл calculate.o
calculate.o: calculate.c calculate.h
gcc -c calculate.c $(CFLAGS)
#Создаем файл main.o
main.o: main.c calculate.h
```

```
gcc -c main.c $(CFLAGS)
```

```
clean: #очистка файлов указанного типа "о"
```

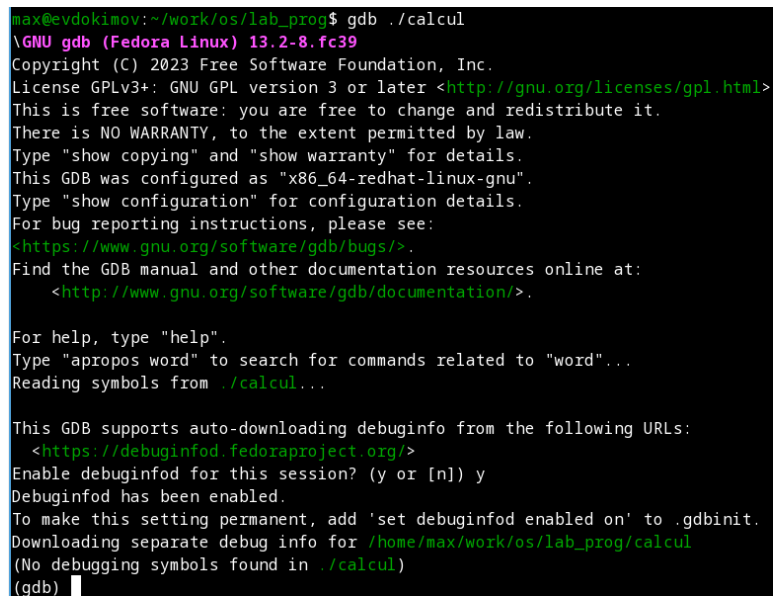
```
-rm calcul *.o *~
```

```
# End Makefile
```

## Работа с gdb

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

– Запустите отладчик GDB, загрузив в него программу для отладки “gdb ./calcul”:



```
max@evdokimov:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Fedora Linux) 13.2-8.fc39
Copyright (C) 2023 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.
Downloading separate debug info for /home/max/work/os/lab_prog/calcul
(No debugging symbols found in ./calcul)
(gdb)
```

Рис. 8: Запуск gdb

– Для запуска программы внутри отладчика введите команду “run”:

```
(gdb) run
Starting program: /home/max/work/os/lab_prog/calcul
Downloading separate debug info for system-supplied DSO at 0x7ffff7c7000
Downloading separate debug info for /lib64/libm.so.6
Downloading separate debug info for /lib64/libc.so.6
Downloading separate debug info for /home/max/.cache/debuginfod_client/7dd93cb9991a89f0ec53d9443a0b78ad952269bc/debuginfo
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Мультипликатор: 7
35.00
[Inferior 1 (process 13924) exited normally]
(gdb)
```

Рис. 9: Run кода

– Для постраничного (по 9 строк) просмотра исходного кода используйте команду “list”:

```
Working directory: /home/max/work/os/lab_prog.
(gdb) list
1      /usr/src/debug/glibc-2.38-7.fc39.x86_64/elf/<built-in>: Файл существует.
(gdb)
```

Рис. 10: Команда list

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами “list 12,15”:

```
(gdb) list 12,15
--Type <RET> for more, q to quit, c to continue without paging--RET
Specified first line '12' is ambiguous:
file: "/usr/src/debug/glibc-2.38-7.fc39.x86_64/elf/<built-in>", line number: 12, symbol: "???"
file: "/usr/src/debug/glibc-2.38-7.fc39.x86_64/elf/<built-in>", line number: 12, symbol: "???"
(gdb) pwd
Working directory: /home/max/work/os/lab_prog.
(gdb)
```

Рис. 11: Команда list с указанием

– Для просмотра определённых строк не основного файла используйте list с параметрами “list calculate.c:20,29”:

```
(gdb) list calculate.c:20,29
No source file named calculate.c.
(gdb) ls
```

Рис. 12: Команда list по файлу

– Установите точку останова в файле calculate.c на строке номер 21 “list calculate.c:20,27” и “break 21”:

```
(gdb) ls
Undefined command: "ls".  Try "help".
(gdb) list calculate.c:20,27
No source file named calculate.c.
(gdb) █
```

Рис. 13: Команда list по файлу 2

```
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +4
Второе слагаемое: 4
8.00
[Inferior 1 (process 13984) exited normally]
(gdb) break 21
No line 21 in the current file.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (21) pending.
(gdb) █
```

Рис. 14: Установка точки останова

– Выведите информацию об имеющихся в проекте точка останова “info breakpoints”:

```
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y   <PENDING>    21
(gdb) █
```

Рис. 15: Просмотр breakpoints

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова “run” “5” “-” “backtrace”:



```
(gdb) run
Starting program: /home/max/work/os/lab_prog/calcul
Downloading source file /usr/src/debug/glibc-2.38-7.fc39.x86_64/stdio-common/errname.c
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: backtrace
5.00
[Inferior 1 (process 13988) exited normally]
(gdb) █
```

Рис. 16: Запуск по условию

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя “print Numeral”, На экран должно быть выведено число 5:

```
[Inferior 1 (process 13988) exited normally]
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) █
```

Рис. 17: Вывод значения

– Сравните с результатом вывода на экран после использования команды “display Numeral”:

```
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) display Numeral
No symbol "Numeral" in current context.
```

Рис. 18: Показ имеющегося в системе значения

– Уберите точки останова “info breakpoints” и “delete 1”:

```
(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1     breakpoint keep  y    <MULTIPLE>
1.1   y          0x00007ffff7d674f0 in __get_errname at errname.c:55
1.2   y          0x00007ffff7fefa60 in __get_errname at errname.c:55
(gdb) delete 1
(gdb) █
```

Рис. 19: Удаление точки останова

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

```
max@evdokimov:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 22 Jul 2023

calculate.h:4:37: 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:7:31: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:13:1: 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:19:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:4: 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:35:7: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:43:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:7: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:47:7: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:49:7: Return value type double does not match declared type float:
```

Рис. 20: Команда splint по calculate

```
max@evdokimov:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 22 Jul 2023

calculate.h:4:37: 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:13:12: 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:13:9: Corresponding format code
main.c:13:1: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
max@evdokimov:~/work/os/lab_prog$
```

Рис. 21: Команда splint по main

# Контрольные вопросы

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

Дополнительную информацию об этих программах можно получить с помощью функций info и man.

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

Unix поддерживает следующие основные этапы разработки приложений:

- создание исходного кода программы;
- представляется в виде файла;
- сохранение различных вариантов исходного текста;
- анализ исходного текста;
- компиляция исходного текста и построение исполняемого модуля;
- тестирование и отладка;
- проверка кода на наличие ошибок;
- сохранение всех изменений, выполняемых при тестировании и отладке.

Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.

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

Использование суффикса “.c” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу “.c” компилятор распознает, что файл `abcd.c` должен компилироваться, а по суффиксу “.o”, что файл `“abcd.o”` является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы `abcd.c` и построения исполняемого модуля `abcd` имеет вид: `“gcc -o abcd abcd.c”`. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых и новых файлов. Опция – `prefix` может быть использована для установки такого префикса. Плюс к этому команда `“bzi diff -p1”` выводит префиксы в форме которая подходит для команды `patch “-p1”`.

4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make?

При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа `make` освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом `make-файле`, который по умолчанию имеет имя `makefile` или `Makefile`.

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

makefile для некой программы расширения “.c” мог бы иметь вид:

```
# Makefile
CC = gcc
CFLAGS =
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
```

В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла.

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

Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен

был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.

8. Назовите и дайте основную характеристику основным командам отладчика gdb.

- `backtrace` – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от `main()`; иными словами, выводит весь стек функций;
- `break` – устанавливает точку останова; параметром может быть номер строки или название функции;
- `clear` – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
- `continue` – продолжает выполнение программы от текущей точки до конца;
- `delete` – удаляет точку останова или контрольное выражение;
- `display` – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;

- `finish` – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
  - `info breakpoints` – выводит список всех имеющихся точек останова;
  - `info watchpoints` – выводит список всех имеющихся контрольных выражений;
  - `splist` – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
  - `nex` – пошаговое выполнение программы, но, в отличие от команды `step`, не выполняет пошагово вызываемые функции;
  - `print` – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
  - `run` – запускает программу на выполнение;
  - `set` – устанавливает новое значение переменной;
  - `step` – пошаговое выполнение программы;
  - `watch` – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится.
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
- Выполнили компиляцию программы;
  - Увидели ошибки в программе;
  - Открыли редактор и исправили программу;

- Загрузили программу в отладчик `gdb run` — отладчик выполнил программу, ввели требуемые значения;
- Программа завершена, `gdb` не видит ошибок.

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

Отладчику не понравился формат “%s” для “&Operation”, так как “%s” — символный формат, а значит необходим только Operation.

11. Назовите основные средства, повышающие понимание исходного кода программы.

Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: `cscope` - исследование функций, содержащихся в программе; `splint` — критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой `splint`?

Проверка корректности задания аргументов всех исполняемых функций, а также типов возвращаемых ими значений; Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; Общая оценка мобильности пользовательской программы.



## Выводы

В ходе выполнения лабораторной работы были изучены новые способы написания программ на языке C, а так же ознакомился с системой gdb.

# Список литературы

1. Лабораторная работа №13