

Отчёт по лабораторной работе №13

Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Митичкина Екатерина Павловна

Содержание

Цель работы	1
Задача	1
Теоретическое введение:	5
Выполнение лабораторной работы	5
Выводы	10
Ответы на контрольные вопросы	10

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.

Задача

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.
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. Реализация функций калькулятора в файле calculate.h:

```
////////////////////////////////////  
// 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);
    }
}

```

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:

```

////////////////////////
// calculate.h

#ifndef CALCULATE_H_ #define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

```

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:

```

////////////////////////
// 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;
}

```

3. Выполните компиляцию программы посредством gcc:

```

gcc -c calculate.c gcc -c main.c
gcc calculate.o main.o -o calcul -lm

```

4. При необходимости исправьте синтаксические ошибки.

5. Создайте Makefile со следующим содержанием:

```

#
# 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

```

Поясните в отчёте его содержание. 6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): - Запустите отладчик GDB, загрузив в него программу для отладки:

```
gdb ./calcul
```

- Для запуска программы внутри отладчика введите команду run:
run
- Для постраничного (по 9 строк) просмотра исходного код используйте команду list:
list
- Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:
list 12,15
- Для просмотра определённых строк не основного файла используйте list с параметрами:
list calculate.c:20,29
- Установите точку останова в файле calculate.c на строке номер 21:
list calculate.c:20,27
break 21
- Выведите информацию об имеющихся в проекте точка останова:
info breakpoints
- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:
run 5
-
backtrace
- Отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffff280 "-")
at calculate.c:21
#1 0x00000000400b2b in main () at main.c:17
```

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места. - Посмотрите, чему равно на этом этапе значение переменной `Numeral`, введя:

```
print Numeral
```

На экран должно быть выведено число 5. - Сравните с результатом вывода на экран после использования команды:

```
display Numeral
```

- Уберите точки останова:

```
info breakpoints
delete 1
```

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

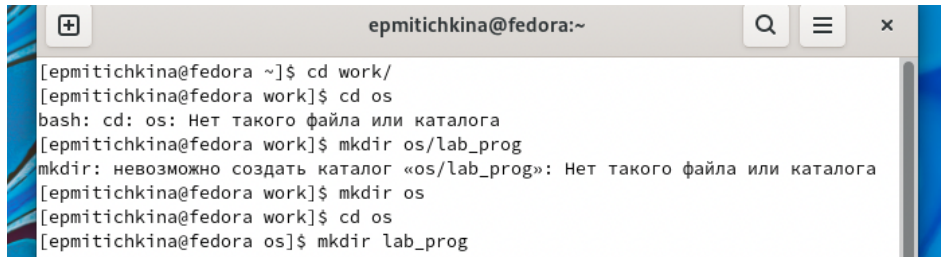
Теоретическое введение:

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.

Выполнение лабораторной работы

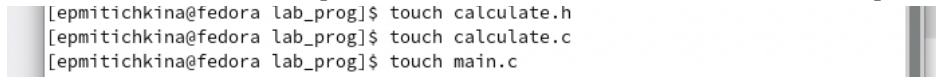
1. Я создала файла подкаталог `~/work/os/lab_prog`.

A terminal window titled 'epmitichkina@fedora:~' with search, menu, and close buttons. It shows a sequence of commands to create a directory structure: 'cd work/', 'cd os' (with an error), 'mkdir os/lab_prog' (with an error), 'mkdir os', 'cd os', and 'mkdir lab_prog'.

```
[epmitichkina@fedora ~]$ cd work/
[epmitichkina@fedora work]$ cd os
bash: cd: os: Нет такого файла или каталога
[epmitichkina@fedora work]$ mkdir os/lab_prog
mkdir: невозможно создать каталог «os/lab_prog»: Нет такого файла или каталога
[epmitichkina@fedora work]$ mkdir os
[epmitichkina@fedora work]$ cd os
[epmitichkina@fedora os]$ mkdir lab_prog
```

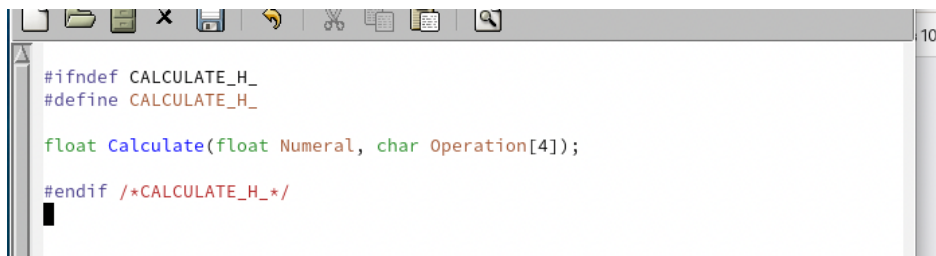
Рис 1. Создание lab_prog

2. Создала в нём файлы: calculate.h, calculate.c, main.c. И реализовала код.

A terminal window showing three 'touch' commands to create files in the 'lab_prog' directory: 'touch calculate.h', 'touch calculate.c', and 'touch main.c'.

```
[epmitichkina@fedora lab_prog]$ touch calculate.h
[epmitichkina@fedora lab_prog]$ touch calculate.c
[epmitichkina@fedora lab_prog]$ touch main.c
```

Рис 2. Создание файлов

A code editor window with a toolbar and a line number '10' on the right. It displays the content of 'calculate.h', which includes preprocessor directives and a function declaration for 'Calculate'.

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

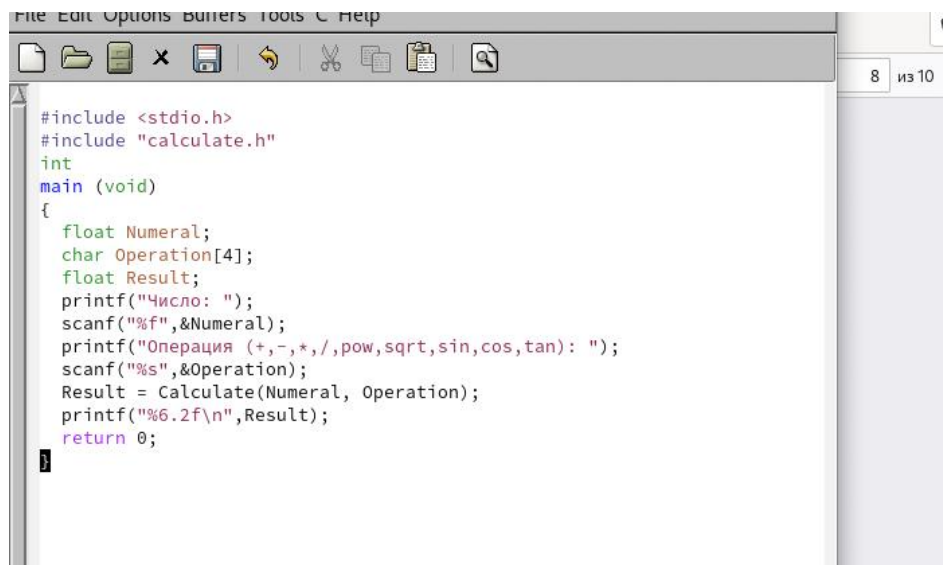
Рис 3. calculate.h

```
File Edit Options Buffers Tools C Help
[Icons]
7 из 10

#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);
    }
}
```

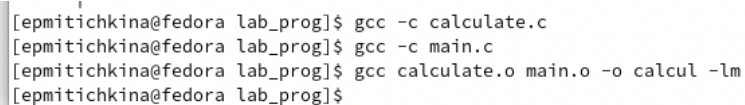
Рис. 4. calculate.c



```
#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("%.2f\n",Result);
    return 0;
}
```

Рис 5. calculate.c

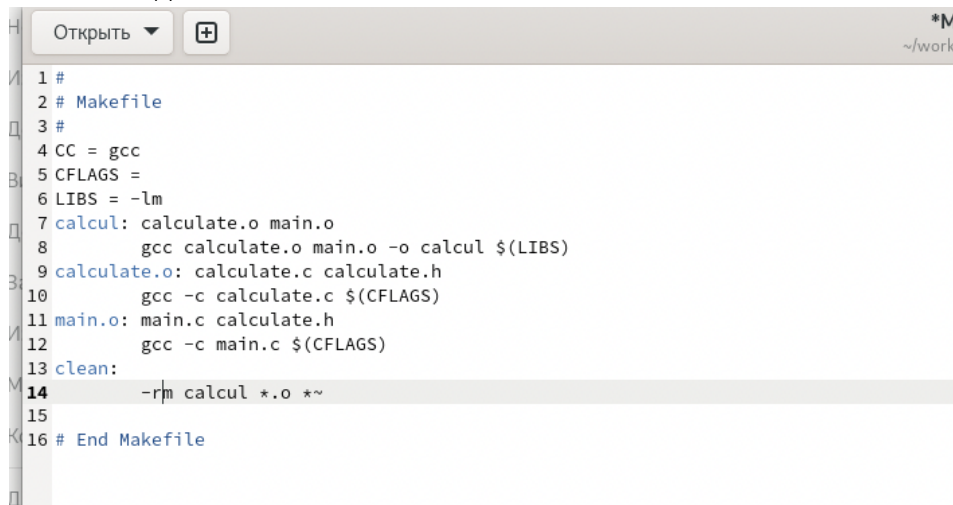
3. Выполнила компиляцию программы



```
[epmitichkina@fedora lab_prog]$ gcc -c calculate.c
[epmitichkina@fedora lab_prog]$ gcc -c main.c
[epmitichkina@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[epmitichkina@fedora lab_prog]$
```

Рис 6. Компиляция программы

5. Создала Makefile



```
1 #
2 # Makefile
3 #
4 CC = gcc
5 CFLAGS =
6 LIBS = -lm
7 calcul: calculate.o main.o
8     gcc calculate.o main.o -o calcul $(LIBS)
9 calculate.o: calculate.c calculate.h
10     gcc -c calculate.c $(CFLAGS)
11 main.o: main.c calculate.h
12     gcc -c main.c $(CFLAGS)
13 clean:
14     -rm calcul *.o *~
15
16 # End Makefile
```

Рис 7. Makefile

6. С помощью gdb выполнила отладку программы calcul


```
[1]+  Завершён      emacs
[epmitichkina@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 10.2-9.fc35
Copyright (C) 2021 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...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/epmitichkina/work/os/lab_prog/calcul
Downloading separate debug info for /home/epmitichkina/work/os/lab_prog/system-s
upplied DSO at 0x7ffff7fc9000...
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: 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
21.00
[Inferior 1 (process 4371) exited normally]
(gdb)
```

Рис 8. Отладка

7. Проанализировала коды файла calculate.c и main.c.

```
[epmitichkina@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:5:37: 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:13:7: 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:19:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:10: 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:35:10: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
```

Рис 9. calculate.c

```
Finished checking --- 15 code warnings
[epmitichkina@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:5:37: 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:10:2: 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:12:13: 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:12:10: Corresponding format code
main.c:12:2: Return value (type int) ignored: scanf("%s", &Op...

Finished checking --- 4 code warnings
[epmitichkina@fedora lab_prog]$
```

Рис 10. main.c

Выводы

В результате работы 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.

Ответы на контрольные вопросы

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

Спросить в интернете или использовать утилиту man, также можно использовать опцию -h у gcc для получения дополнительной информации.

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.

Суффикс - это составная часть имени файла, например его расширение.

4. Каково основное назначение компилятора языка C в UNIX?

Преобразование исходного кода программ в объектные файлы.

5. Для чего предназначена утилита make?

Утилита make 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.

В самом простом случае Makefile имеет следующий синтаксис:

```
<цель_1> <цель_2> ... :  
<зависимость_1> <зависимость_2> ...  
<команда 1>  
...  
<команда n>
```

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. Рассмотрим пример Makefile для написанной выше простейшей программы, выводящей на экран приветствие

```
'Hello World!':  
hello: main.c  
gcc -o hello main.c
```

Здесь в первой строке hello — цель, main.c — название файла, который мы хотим скомпилировать; во второй строке, начиная с табуляции, задана команда компиляции gcc с опциями. Для запуска программы необходимо в командной строке набрать команду make:

make

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

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

Возможность останавливать выполнение программы на определенных строчках кода. Для этого нужно установить так называемые брейкпоинты.

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

Установка брейкпоинтов, пошаговое выполнение отлаживаемой программы, вывод исходного кода постранично или построчно, возможность узнать значение переменных.

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
 1. Установка брейкпоинтов
 2. Пошаговое исполнение программы и вывод значения переменных
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

Процесс компиляции аварийно завершается, указывая на ошибки.

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

Комментарии, единый стиль кода, линтеры.

1. Каковы основные задачи, решаемые программой splint?

Увидеть ошибки и предупреждения, указывающие на различные проблемы в коде программы.