

Лабораторная работа №13

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

Хамдамова Айжана

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы:	15
5	Выводы	22
	Список литературы	23

Список иллюстраций

figno:1.	8
figno:2.	9
figno:3.	10
figno:4.	11
figno:5.	12
figno:6.	12
figno:7.	12
figno:8.	13
figno:9.	14
figno:10	14
4.1 Название рисунка	21

Список таблиц

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.

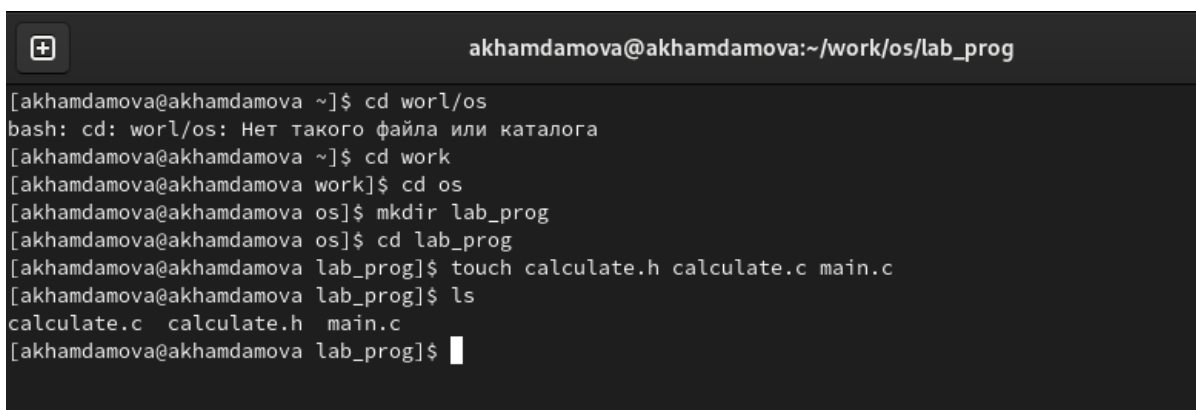
2 Теоретическое введение

Этапы разработки приложений 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. 13.2.2. Компиляция исходного текста и построение исполняемого файла Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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.

Для компиляции файла `main.c`, содержащего написанную на языке C простейшую программу 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`

3 Выполнение лабораторной работы

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.



```
akhamdamova@akhamdamova:~/work/os/lab_prog
[akhamdamova@akhamdamova ~]$ cd worl/os
bash: cd: worl/os: Нет такого файла или каталога
[akhamdamova@akhamdamova ~]$ cd work
[akhamdamova@akhamdamova work]$ cd os
[akhamdamova@akhamdamova os]$ mkdir lab_prog
[akhamdamova@akhamdamova os]$ cd lab_prog
[akhamdamova@akhamdamova lab_prog]$ touch calculate.h calculate.c main.c
[akhamdamova@akhamdamova lab_prog]$ ls
calculate.c calculate.h main.c
[akhamdamova@akhamdamova 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.


```
akhamdamova@akhamdamova:~/work/os/lab_prog

[akhamdamova@akhamdamova ~]$ cd worl/os
bash: cd: worl/os: Нет такого файла или каталога
[akhamdamova@akhamdamova ~]$ cd work
[akhamdamova@akhamdamova work]$ cd os
[akhamdamova@akhamdamova os]$ mkdir lab_prog
[akhamdamova@akhamdamova os]$ cd lab_prog
[akhamdamova@akhamdamova lab_prog]$ touch calculate.h calculate.c main.c
[akhamdamova@akhamdamova lab_prog]$ ls
calculate.c  calculate.h  main.c
[akhamdamova@akhamdamova lab_prog]$
```

Открыть

*calculate.c
~/work/os/lab_prog

Сохранить

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4 #include "calculate.h"
5 float
6 Calculate(float Numeral, char Operation[4])
7 {
8     float SecondNumeral;
9     if(strncmp(Operation, "+", 1) == 0)
10     {
11         printf("Второе слагаемое: ");
12         scanf("%f",&SecondNumeral);
13         return(Numeral + SecondNumeral);
14     }
15     else if(strncmp(Operation, "-", 1) == 0)
16     {
17         printf("Вычитаемое: ");
18         scanf("%f",&SecondNumeral);
19         return(Numeral - SecondNumeral);
20     }
21     else if(strncmp(Operation, "*", 1) == 0)
22     {
23         printf("Множитель: ");
24         scanf("%f",&SecondNumeral);
25         return(Numeral * SecondNumeral);
26     }
27     else if(strncmp(Operation, "/", 1) == 0)
28     {
29         printf("Делитель: ");
30         scanf("%f",&SecondNumeral);
31         if(SecondNumeral == 0)
32         {
33             printf("Ошибка: деление на ноль! ");
34             return(HUGE_VAL);
35         }
36         else
37             return(Numeral / SecondNumeral);
38     }
39     else if(strncmp(Operation, "pow", 3) == 0)
40     {
41         printf("Степень: ");
42         scanf("%f",&SecondNumeral);
43         return(pow(Numeral, SecondNumeral));
44     }
45     else if(strncmp(Operation, "sqrt", 4) == 0)
46     {
47         return(sqrt(Numeral));
48     }
49 }
```

Парная скобка найдена в строке: 39

С Ширина табуляции: 8 Стр 28, Стлб 3 ВСТ

Открыть ▾

+

calculate.h
~/work/os/lab_prog

Сохранить

☰

×

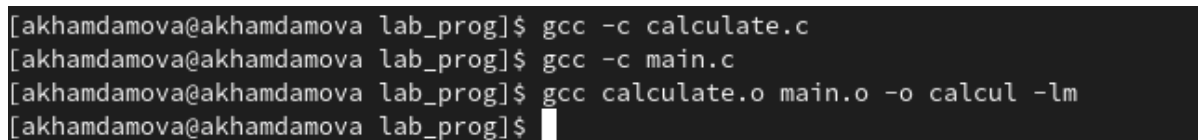
```
1 #ifndef CALCULATE_H_
2 #define CALCULATE_H_
3
4 float Calculate(float Numeral, char Operation[4]);
5
6 #endif /*CALCULATE_H_*/
7
```

Сохранение файла «/home/akhamdamova/work/o... Заголовок C/ObjC ▾ Ширина табуляции: 8 ▾ Стр 3, Стлб 1 ▾ ВСТ



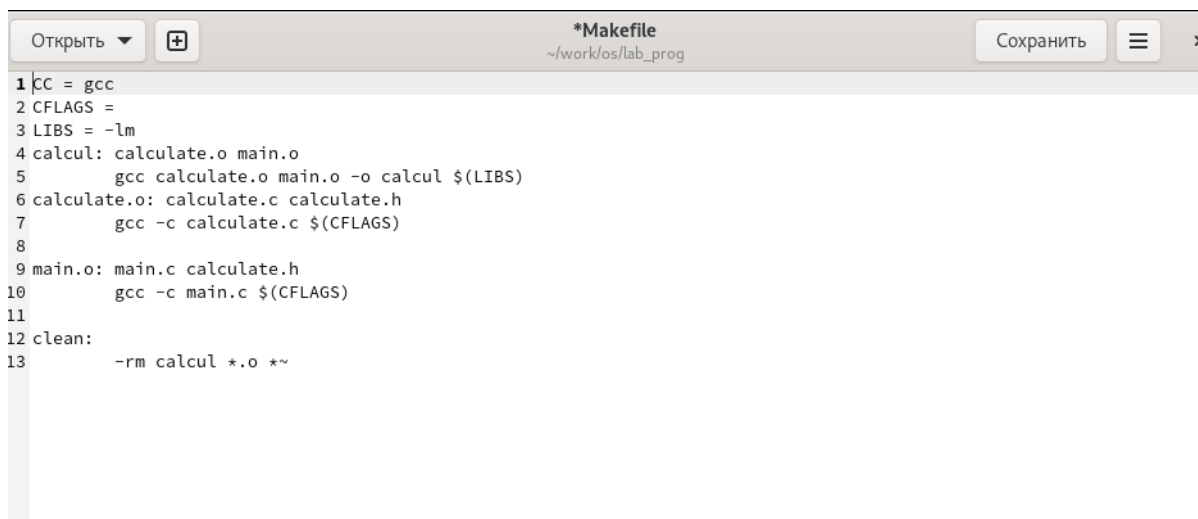
```
1 #include <stdio.h>
2 #include "calculate.h"
3 int
4 main (void)
5 {
6     float Numeral;
7     char Operation[4];
8     float Result;
9     printf("Число: ");
10    scanf("%f",&Numeral);
11    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
12    scanf("%s",&Operation);
13    Result = Calculate(Numeral, Operation);
14    printf("%.2f\n",Result);
15    return 0;
16 }
17
```

3. Выполните компиляцию программы посредством gcc:



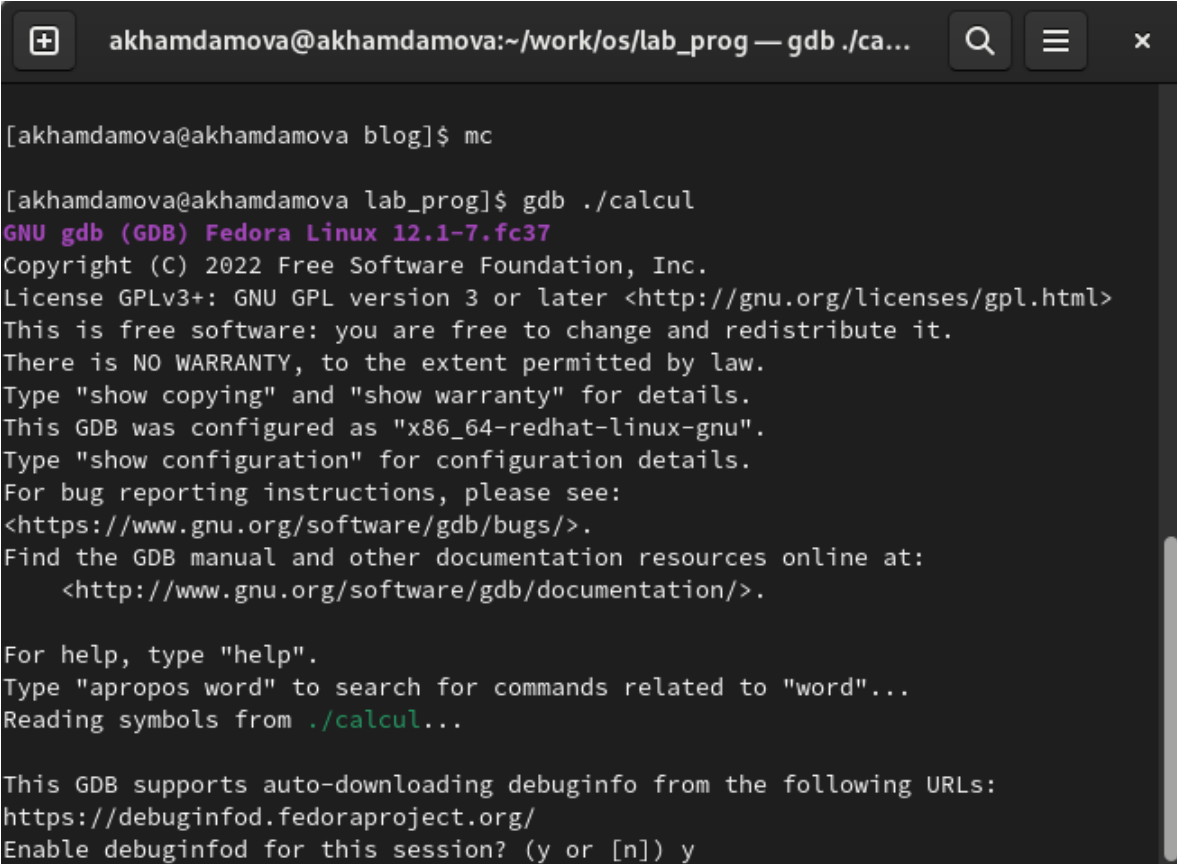
```
[akhamdamova@akhamdamova lab_prog]$ gcc -c calculate.c
[akhamdamova@akhamdamova lab_prog]$ gcc -c main.c
[akhamdamova@akhamdamova lab_prog]$ gcc calculate.o main.o -o calcul -lm
[akhamdamova@akhamdamova lab_prog]$
```

5. Создайте Makefile со следующим содержанием



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

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



```
akhamdamova@akhamdamova:~/work/os/lab_prog — gdb ./ca...
[akhamdamova@akhamdamova blog]$ mc
[akhamdamova@akhamdamova lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 12.1-7.fc37
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
```

```

Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/akhamdamova/work/os/lab_prog/calcul
Downloading 0.01 MB separate debug info for system-supplied DSO at 0x7ffff7fc6000
0
Downloading 2.28 MB separate debug info for /lib64/libm.so.6
Downloading 7.35 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: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 9
54.00
[Inferior 1 (process 3370) exited normally]
(gdb)

```

```

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 9
54.00
[Inferior 1 (process 3370) exited normally]
(gdb) list
Downloading 0.00 MB source file /usr/src/debug/glibc-2.36-9.fc37.x86_64/elf/sofi
ni.c
1      /* Terminate the frame unwind info section with a 4byte 0 as a sentinel;
2         this would be the 'length' field in a real FDE.  */
3
4      typedef unsigned int ui32 __attribute__ ((mode (SI)));
5      static const ui32 __FRAME_END__[1]
6          __attribute__ ((used, section (".eh_frame")))
7          = { 0 };
(gdb)

```

4 Контрольные вопросы:

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
Дополнительную информацию о 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; 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.
 - компиляция исходного текста и построение исполняемого модуля;
 - тестирование и отладка;
 - проверка кода на наличие ошибок
 - сохранение всех изменений, выполняемых при тестировании и отладке.
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 (old) и новых (new) файлов. Опция `-prefix` может быть использована для установки такого префикса. Плюс к этому команда `bzr 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 программы abcd.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. Таким образом, спецификация взаимосвязей имеет формат: `target1 [target2...]: [:] [dependment1...] [(tab)commands] [#commentary] [(tab)commands] [#commentary]`, где `#` — специфицирует начало комментария, так как содержимое строки, начиная с `#` и до конца строки, не будет обрабатываться командой `make`; `:` — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд `()`, но она считается как одна строка; `::` — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний. Приведённый выше make-файл для программы `abcd.c` включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем `abcd`. Вторым способом позволяет включать в исполняемый модуль `testabcd` возможность

выполнить процесс отладки на уровне исходного текста.

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;
 - `next` – 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 2) Увидели ошибки в программе Открыли редактор и исправили

программу Загрузили программу в отладчик `gdb run` — отладчик выполнил программу, мы 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`, т.к `%s` — символьный 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`?
13. Проверка корректности задания аргументов всех исполняемых функций , а также типов возвращаемых ими значений;
14. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
15. Общая оценка мобильности пользовательской программы

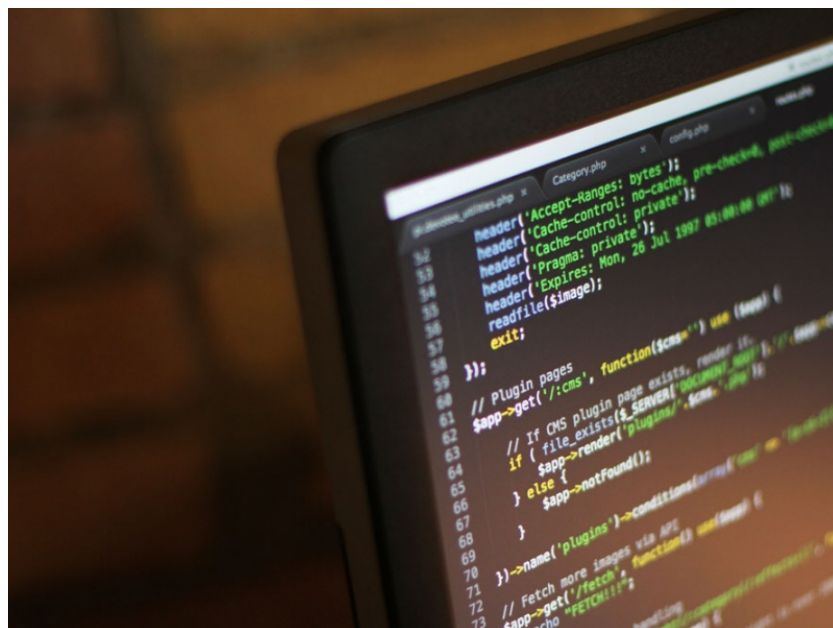


Рис. 4.1: Название рисунка

5 Выводы

Здесь кратко описываются итоги проделанной работы.

Список литературы