

Отчёт о выполнении лабораторной работы №14

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

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: *Операционные системы*

Работу выполняла: Живцова Анна

1032201673

НКН68-01-20

Москва. Дисплейный класс РУДН. 2021г.

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.

Выполнение работы

1. Создала файлы: `calculate.h`, `calculate.c`, `main.c`. В них реализовала 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 останавливается. (см рисунки ниже [calculate.c](#) [main.c](#) [calculate.h](#) файлы)



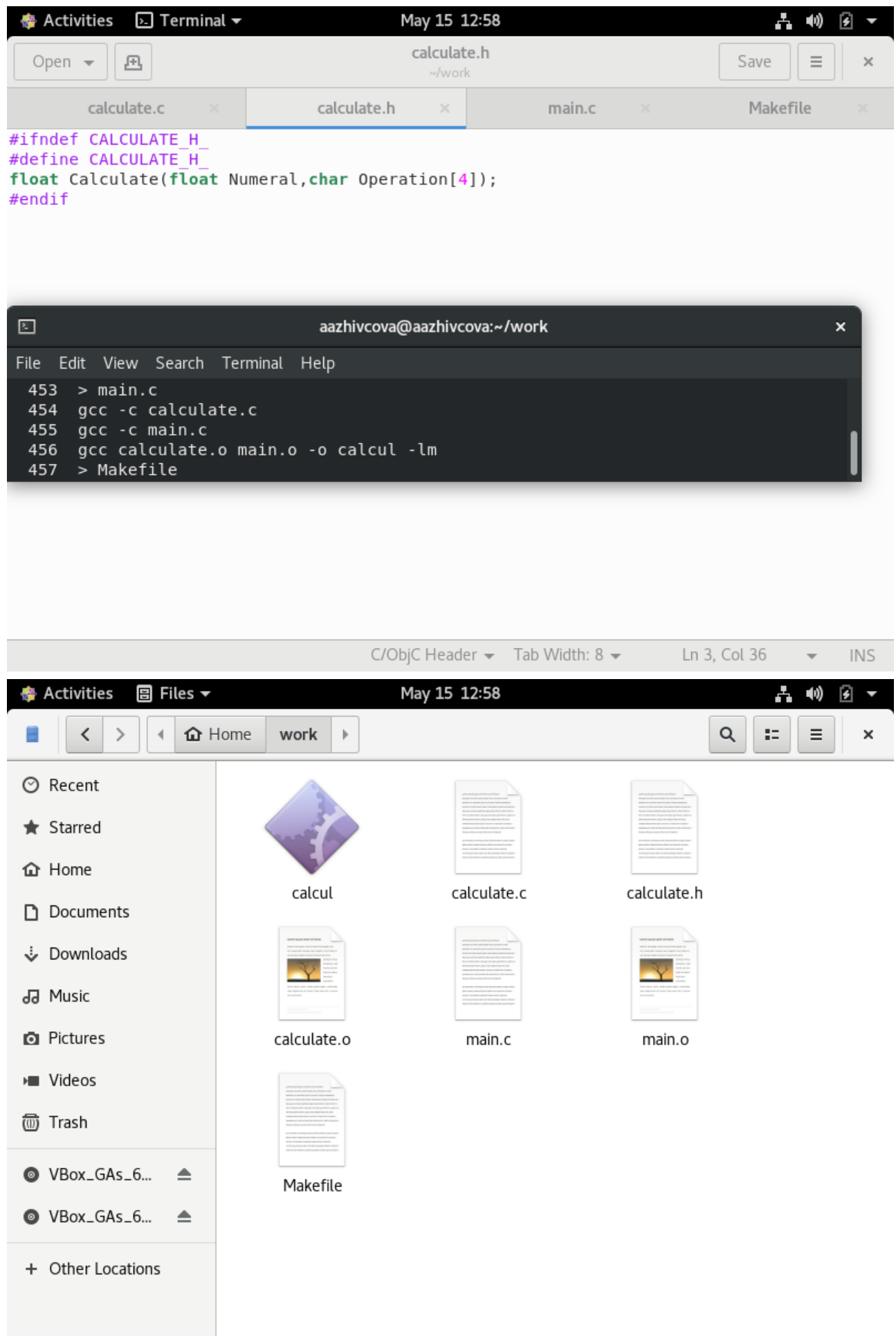
The image shows two screenshots of a text editor window. The top screenshot displays the `calculate.c` file, which defines a `Calculate` function. This function takes a `float` `Numeral` and a `char` array `Operation` of size 4. It uses `strcmp` to check the operation and `scanf` to read a second number. The function returns the result of the operation: addition, subtraction, or multiplication. The bottom screenshot displays the `main.c` file, which includes `calculate.h` and uses the `Calculate` function. It prompts the user for a number and an operation, then prints the result using `printf`.

```
#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("Второе слагаемое:");
        scanf("%f",&SecondNumeral);
        return(Numeral+SecondNumeral);
    }
    else if(strcmp(Operation,"-",1)==0)
    {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral-SecondNumeral);
    }
    else if(strcmp(Operation,"*",1)==0)
    {
        printf("Множитель: ");
        scanf("%f",&SecondNumeral);
        return(Numeral*SecondNumeral);
    }
}

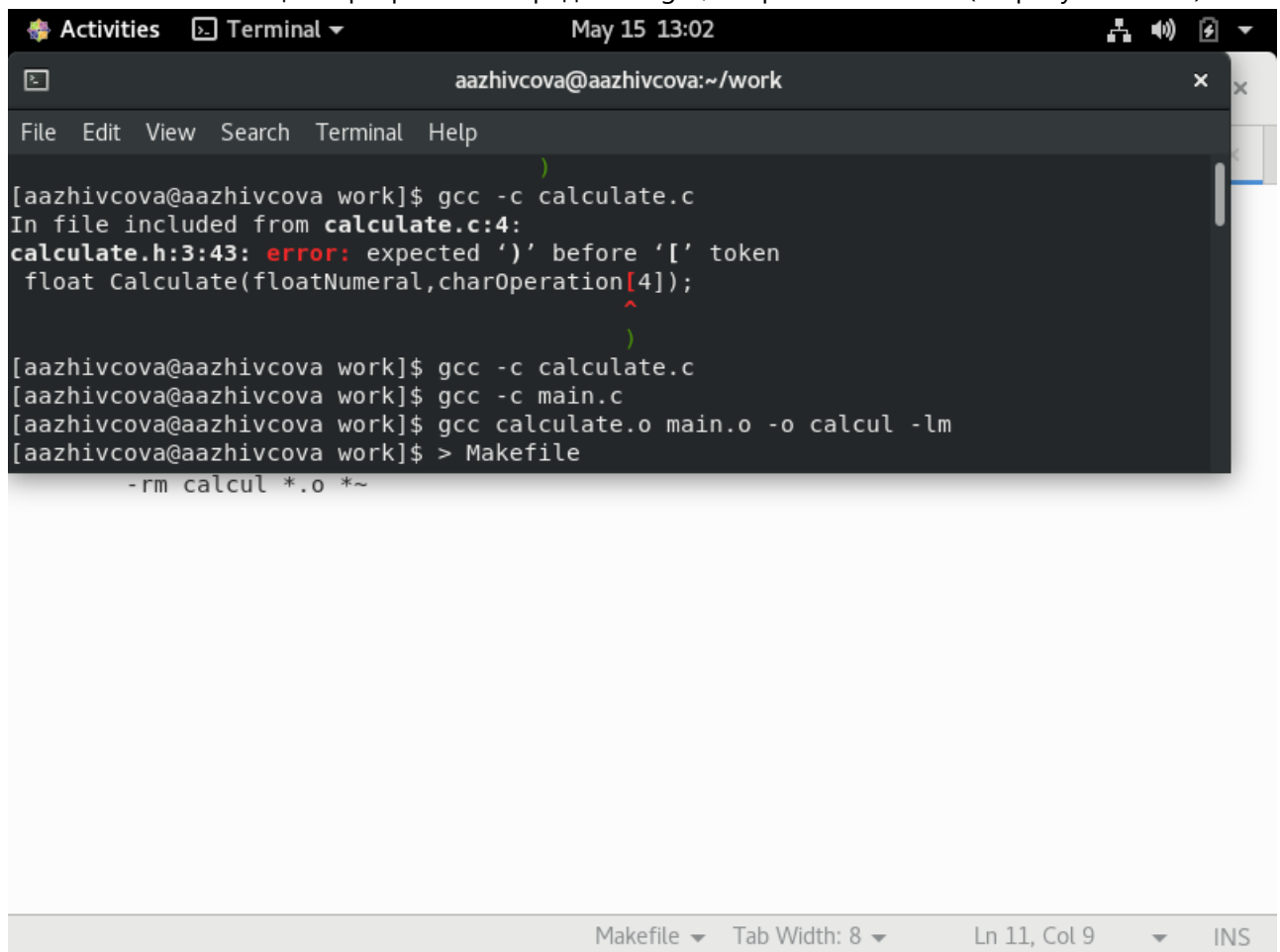
C Tab Width: 8 Ln 41, Col 1 INS
```

```
#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;
}

C Tab Width: 8 Ln 16, Col 2 INS
```



2. Выполнила компиляцию программы посредством gcc, исправила ошибки (см рисунок ниже)

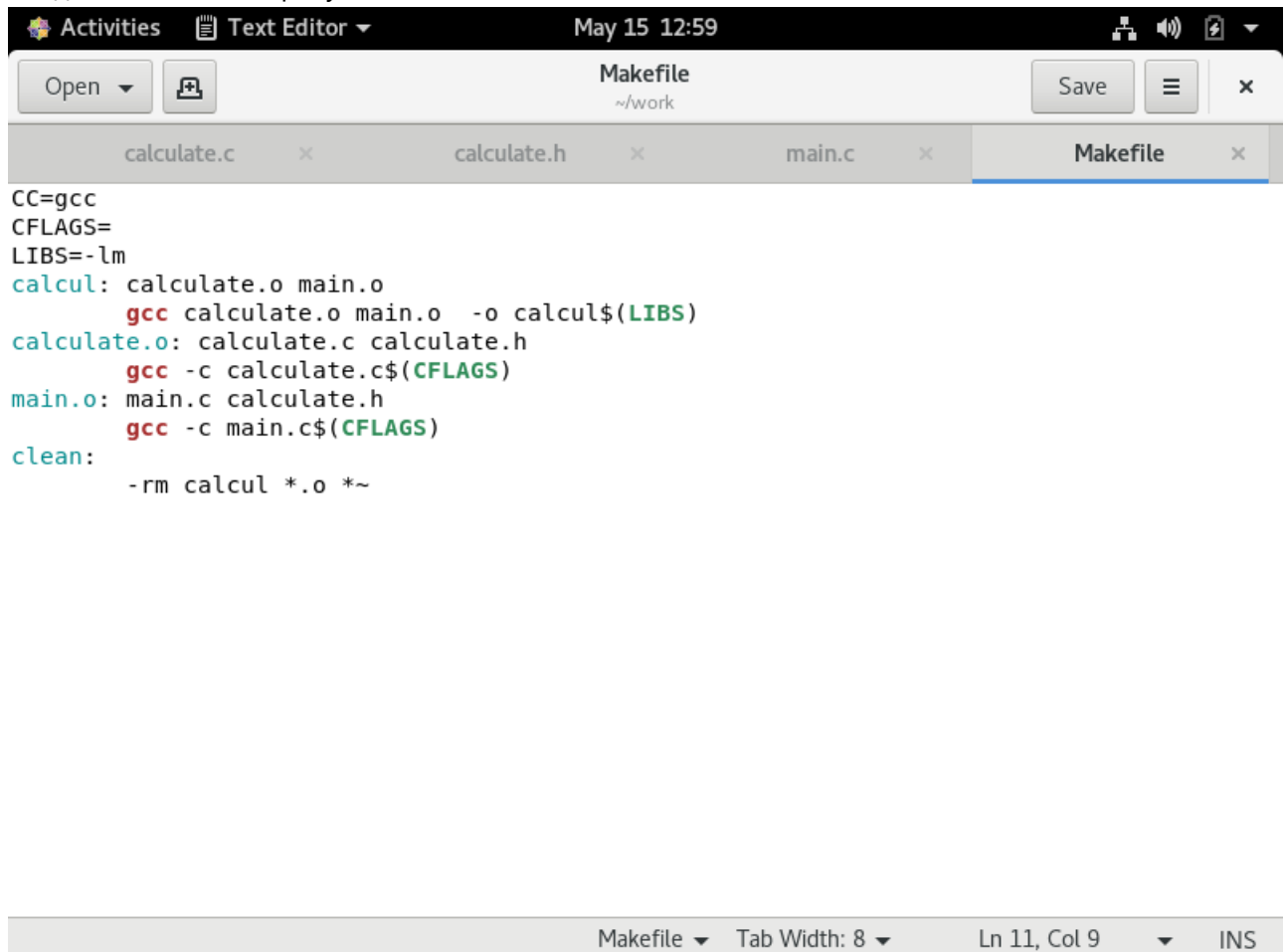


```
Activities Terminal May 15 13:02
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help

[aazhivcova@aazhivcova work]$ gcc -c calculate.c
In file included from calculate.c:4:
calculate.h:3:43: error: expected ')' before '[' token
float Calculate(floatNumeral,charOperation[4]);
                                         ^
[aazhivcova@aazhivcova work]$ gcc -c calculate.c
[aazhivcova@aazhivcova work]$ gcc -c main.c
[aazhivcova@aazhivcova work]$ gcc calculate.o main.o -o calcul -lm
[aazhivcova@aazhivcova work]$ > Makefile
-rm calcul *.o *~

Makefile Tab Width: 8 Ln 11, Col 9 INS
```

3. Создала Makefile (см рисунок ниже [Makefile](#))



```
Activities Text Editor May 15 12:59
Makefile ~/work
Open Save
calculate.c calculate.h main.c Makefile

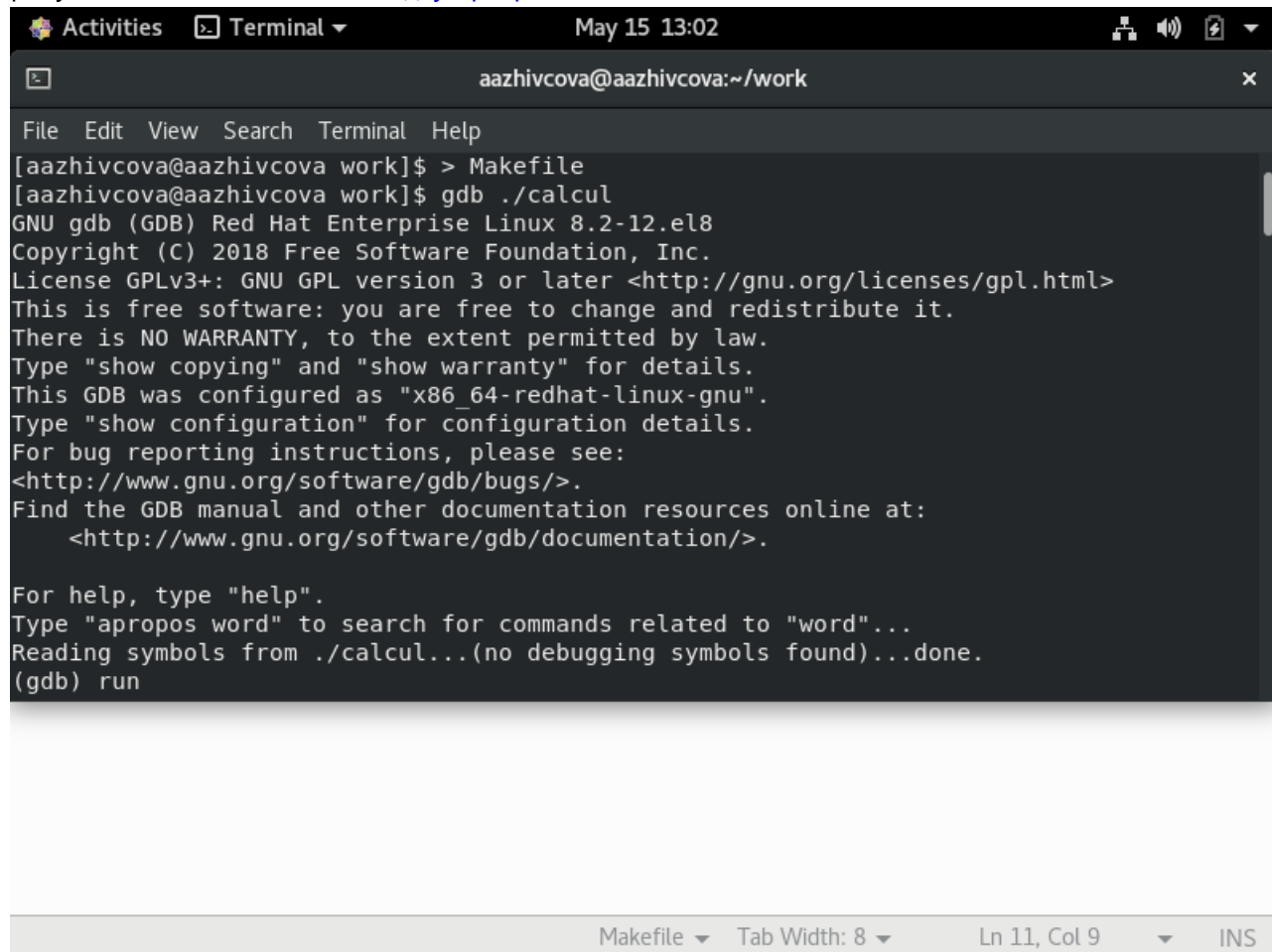
CC=gcc
CFLAGS=
LIBS=-lm
calcul: calculate.o main.o
    gcc calculate.o main.o -o calcul$(LIBS)
calculate.o: calculate.c calculate.h
    gcc -c calculate.c$(CFLAGS)
main.o: main.c calculate.h
    gcc -c main.c$(CFLAGS)
clean:
    -rm calcul *.o *~

Makefile Tab Width: 8 Ln 11, Col 9 INS
```

в начале файла заданы три переменные: CC и CFLAGS и LIBS. 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.

4. С помощью gdb выполнила отладку программы calcul (перед использованием gdb исправила Makefile (добавила -g)) Запустила отладчик GDB, загрузив в него программу для отладки (см рисунок ниже [выполнила отладку программы calcul](#))

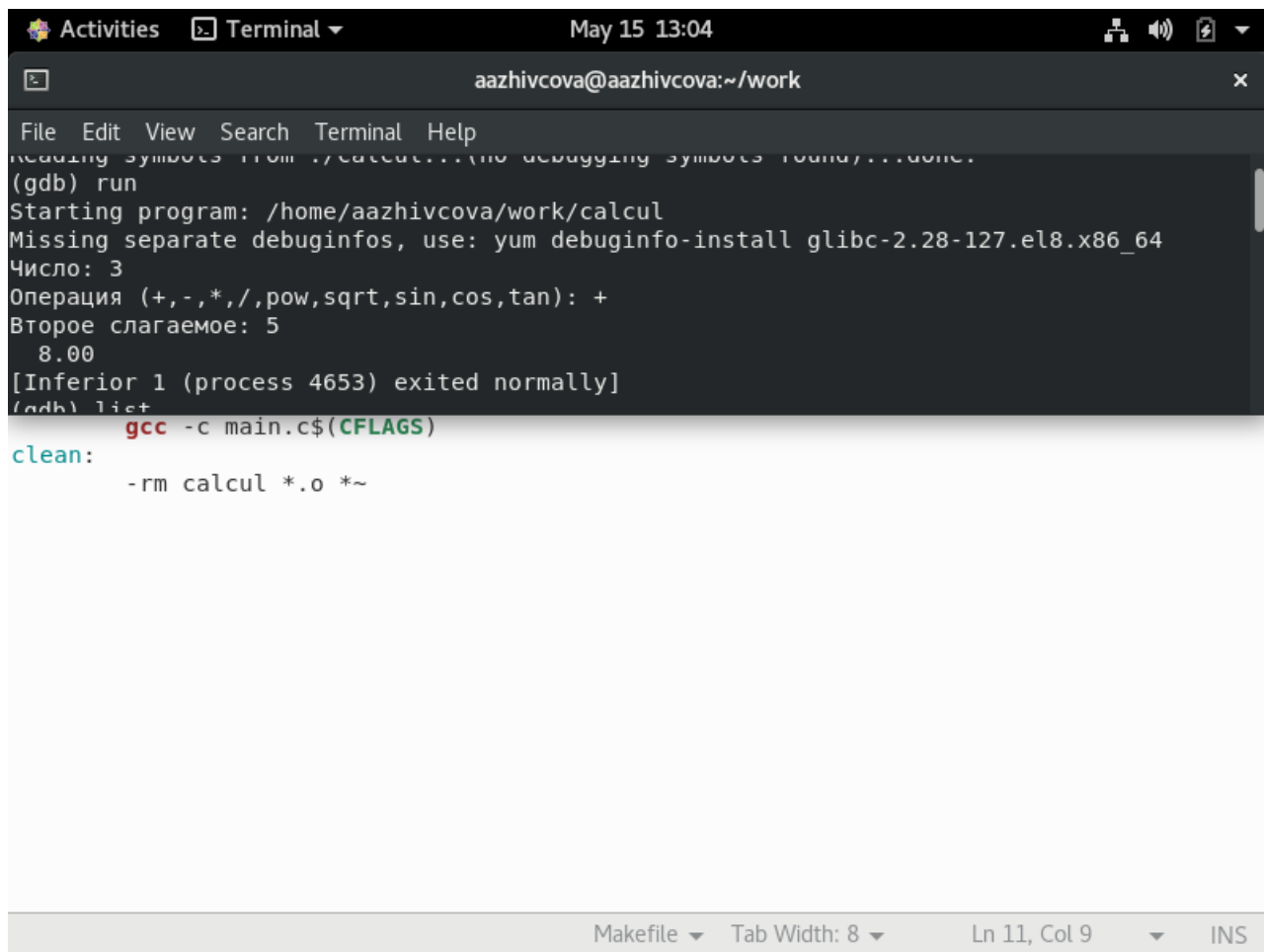


```
Activities Terminal May 15 13:02
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
[aazhivcova@aazhivcova work]$ > Makefile
[aazhivcova@aazhivcova work]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-12.el8
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: 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:
<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"...
Reading symbols from ./calcul...(no debugging symbols found)...done.
(gdb) run
```

Makefile Tab Width: 8 Ln 11, Col 9 INS

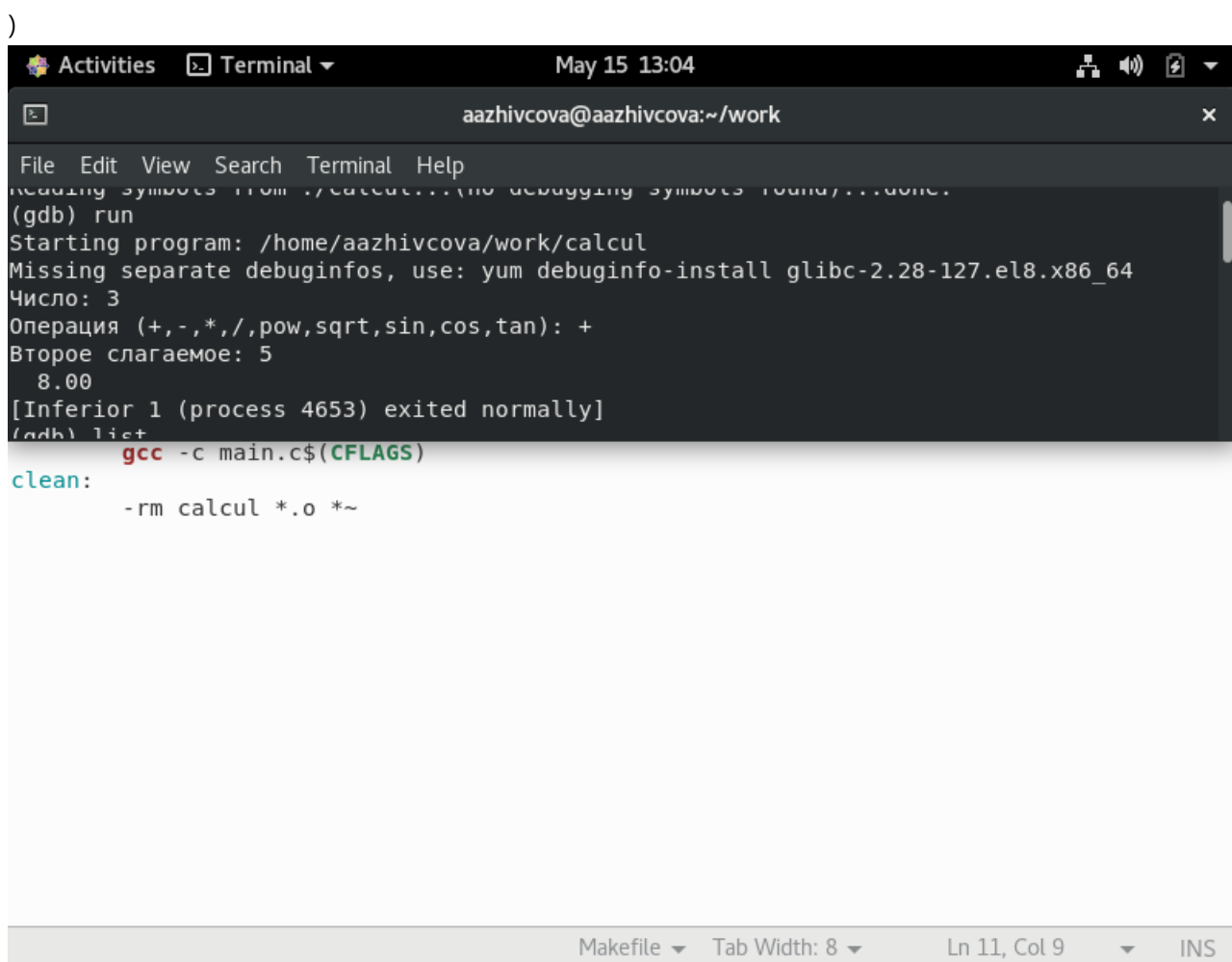
Запустила программу (см рисунок ниже



```
Activities Terminal May 15 13:04
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
loading symbols from ./calcul...no debugging symbols found...done.
(gdb) run
Starting program: /home/aazhivcova/work/calcul
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-127.el8.x86_64
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 5
      8.00
[Inferior 1 (process 4653) exited normally]
(gdb) list
      gcc -c main.c$(CFLAGS)
clean:
      -rm calcul *.o *~
```

Makefile Tab Width: 8 Ln 11, Col 9 INS

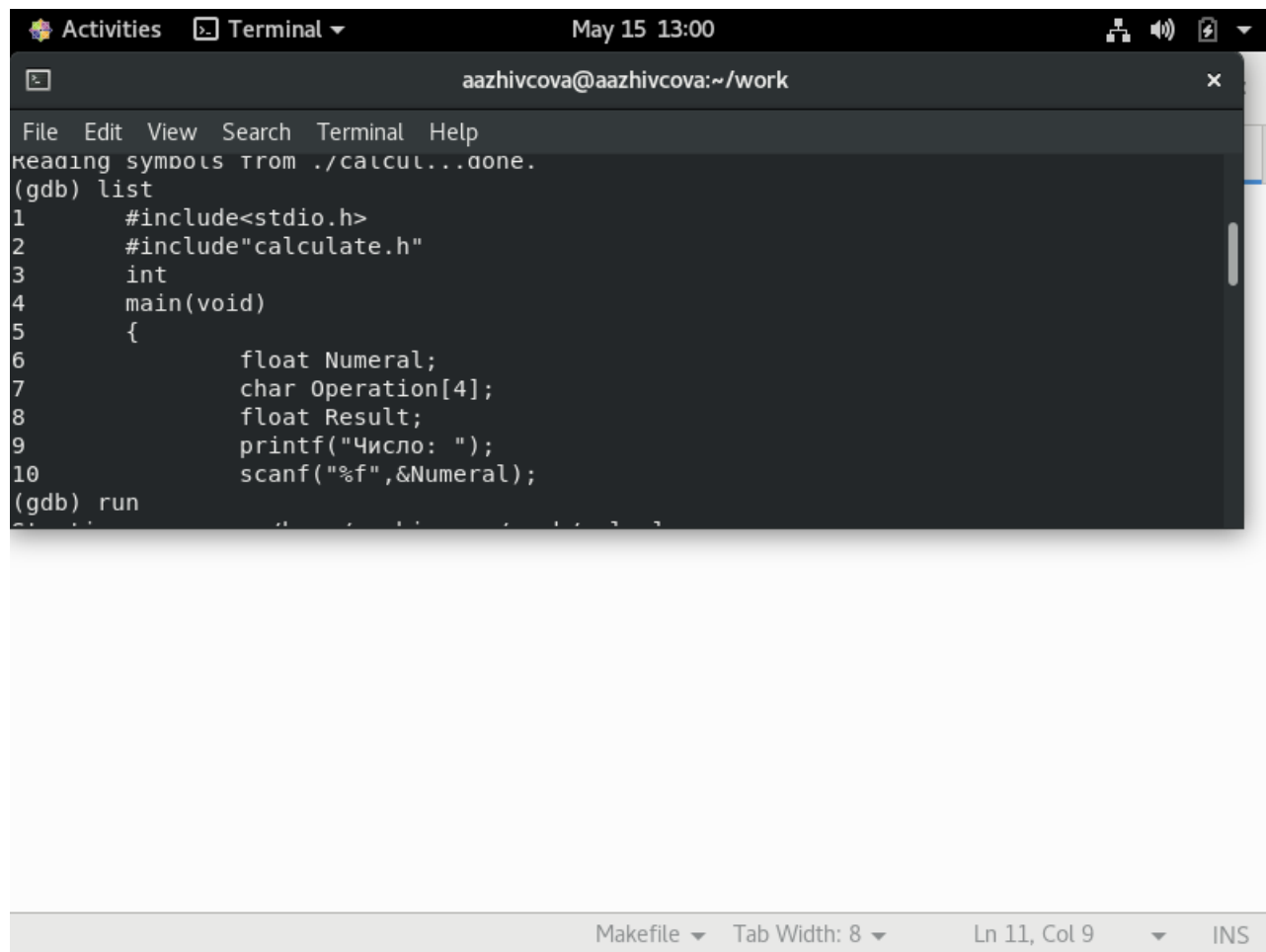
)



```
Activities Terminal May 15 13:04
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
loading symbols from ./calcul...no debugging symbols found...done.
(gdb) run
Starting program: /home/aazhivcova/work/calcul
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-127.el8.x86_64
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 5
      8.00
[Inferior 1 (process 4653) exited normally]
(gdb) list
      gcc -c main.c$(CFLAGS)
clean:
      -rm calcul *.o *~
```

Makefile Tab Width: 8 Ln 11, Col 9 INS

Посмотрела исходный код (10 строк) (см рисунок ниже [Посмотрела исходный код](#))



The screenshot shows a terminal window titled "aazhivcova@aazhivcova:~/work". The window contains the following text:

```
File Edit View Search Terminal Help
reading symbols from ./calcul...done.
(gdb) list
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);
(gdb) run
```

At the bottom of the terminal window, there is a status bar with the following information: "Makefile", "Tab Width: 8", "Ln 11, Col 9", and "INS".

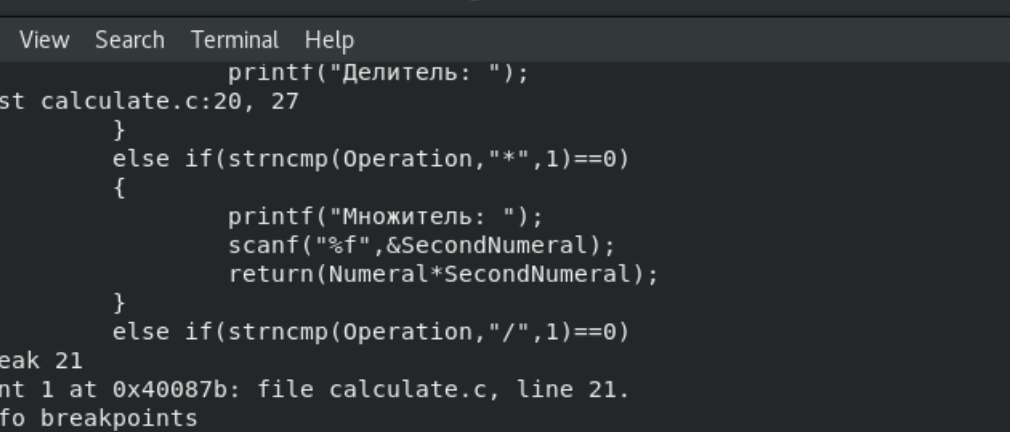
Посмотрела исходный код (строки с 12 по 15) (см рисунок ниже)

Посмотрела исходный код не основного файла (строки с 20 по 29) (см рисунок ниже [строки ОТДЕЛЬНЫЕ](#))

```
Activities Terminal May 15 13:05
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
[Interior 1 (process 5331) exited normally]
(gdb) list 12, 15
12         scanf("%s",&Operation);
13         Result=Calculate(Numeral, Operation);
14         printf("%6.2f\n",Result);
15         return 0;
(gdb) list calculate.c:20,29
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)
--Type <RET> for more, q to quit, c to continue without paging--
28     {
29         printf("Делитель: ");
(gdb) list calculate.c:20, 27
```

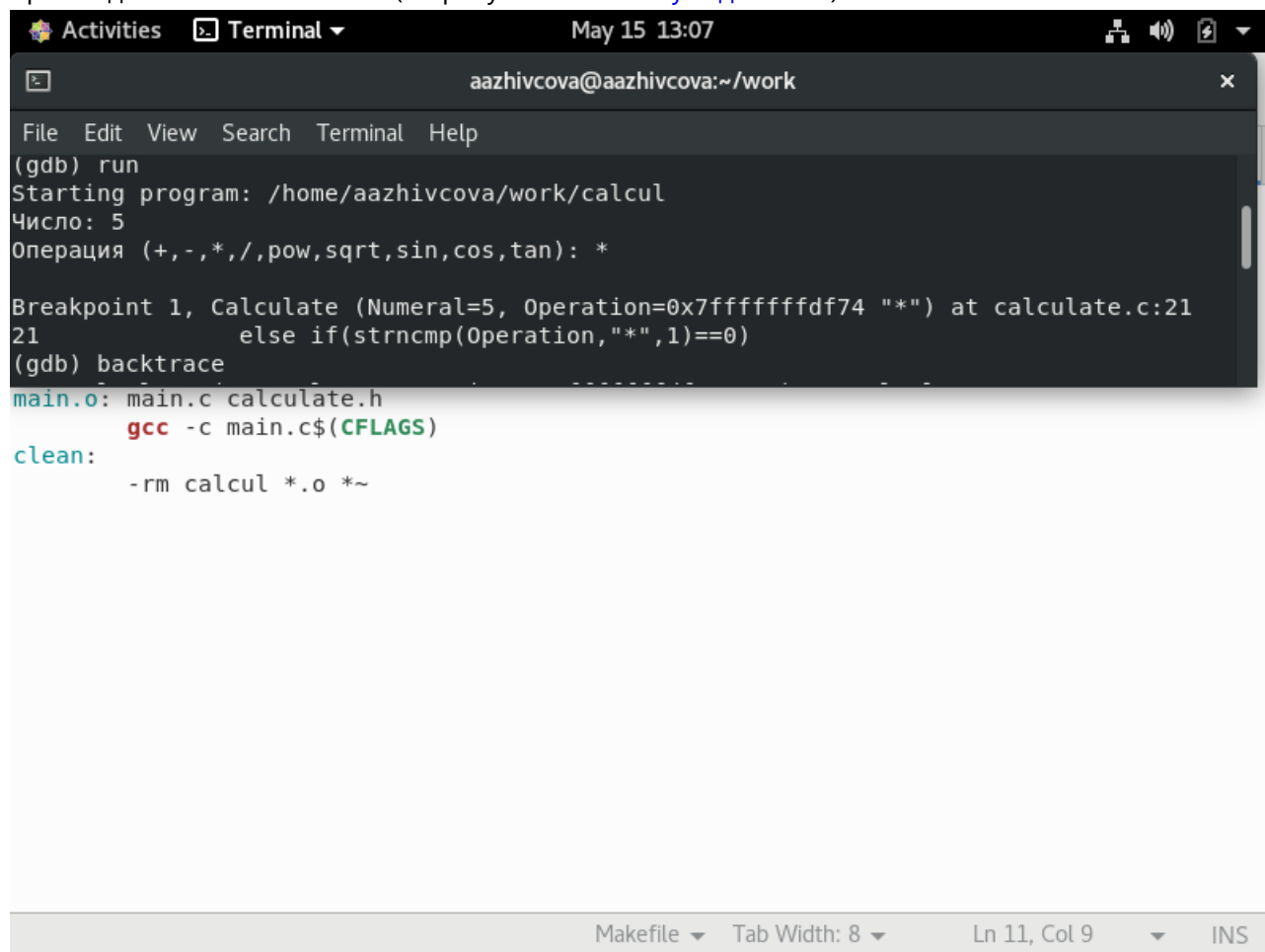
Установила точку останова в файле calculate.c на строке номер 21

Вывела информацию об имеющихся в проекте точка останова (см рисунок ниже [точка останова](#))



```
29         printf("Делитель: ");
(gdb) list calculate.c:20, 27
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)
(gdb) break 21
Breakpoint 1 at 0x40087b: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint        keep y   0x000000000040087b in Calculate at calculate.c:21
(gdb) run
```


Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова (см рисунок ниже [запуск до точки](#))



```
Activities Terminal May 15 13:07
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
(gdb) run
Starting program: /home/aazhivcova/work/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf74 "*") at calculate.c:21
21         else if(strncmp(Operation,"*",1)==0)
(gdb) backtrace
main.o: main.c calculate.h
gcc -c main.c$(CFLAGS)
clean:
-rm calcul *.o *~
Makefile Tab Width: 8 Ln 11, Col 9 INS
```

Посмотрела, чему равно на этом этапе значение переменной, используя print и display (см рисунок ниже [значение переменной](#))

```

Activities Terminal May 15 13:07
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
21         else if(strncmp(Operation,"*",1)==0)
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf74 "*") at calculate.c:21
#1  0x0000000000400ad4 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
main.o: main.c calculate.h
gcc -c main.c$(CFLAGS)
clean:
-rm calcul *.o *~

Makefile Tab Width: 8 Ln 11, Col 9 INS

```

Убрала точки останова (см рисунок ниже [удалила точку останова](#))

```

Activities Terminal May 15 13:08
aazhivcova@aazhivcova:~/work
File Edit View Search Terminal Help
1: Numeral = 5
(gdb) info breackpoints
Undefined info command: "breackpoints". Try "help info".
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint      keep y  0x000000000040087b in Calculate at calculate.c:21
        breakpoint already hit 1 time
(gdb) delete 1
main.o: main.c calculate.h
gcc -c main.c$(CFLAGS)
clean:
-rm calcul *.o *~

Makefile Tab Width: 8 Ln 11, Col 9 INS

```

Контрольные вопросы

1. Дополнительную информацию о этих программах можно получить с помощью функций `info` и `man` и `help`
2. 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. Суффикс - расширение. Например `c`.
4. `gcc` по расширению (суффиксу) `.c` распознает тип файла для компиляции и формирует объектный модуль — файл с расширением `.o`.
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. `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`:

```
CC = gcc
CFLAGS =
abcd: abcd.c
$(CC) -o abcd $(CFLAGS) abcd.c
clean:
-rm abcd *.o *~
```

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. 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. `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
9.
 1. Выполнили компиляцию программы
 - 2) Увидели ошибки в программе
 3. Открыли редактор и исправили программу
 4. Загрузили программу в отладчик `gdb`
 5. `run` — отладчик выполнил программу, мы ввели требуемые значения.
 6. программа завершена, `gdb` не видит ошибок.
10. в сообщении указывался файл и строка с ошибкой, а также её характер.
11. `cscope` - исследование функций, содержащихся в программе; `split` — критическая проверка программ, написанных на языке Си.

12. Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита 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.

Библиография

<https://web-profi.by/process-razrabotki-programmnogo-obespecheniya/>

<https://habr.com/ru/post/255991/>

https://ru.wikipedia.org/wiki/Процесс_разработки_программного_обеспечения

Вывод

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