

Отчет по лабораторной работе №12:

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

Федорова Наталия Артемовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Контрольные вопросы	19
6	Выводы	23

Список иллюстраций

4.1	Создание нового подкаталога и файлов	9
4.2	calculate.h	9
4.3	calculate.c	10
4.4	main.c	11
4.5	Makefile	11
4.6	Компиляция программы посредством make	13
4.7	Запуск отладчика	14
4.8	Просмотр кода и точка останова	15
4.9	Проверка останова и удаление точки останова	16
4.10	Анализ кода файла main.c	17
4.11	Анализ кода файла calculate.c	18

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 Задание

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.
3. Выполните компиляцию программы посредством `gcc`.
4. Создайте `Makefile` и поясните о его содержании.
5. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`)
6. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

3 Теоретическое введение

Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

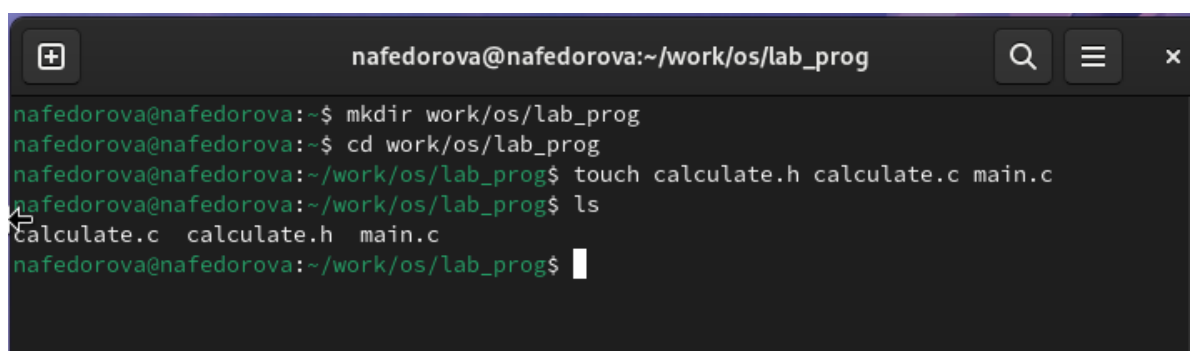
- планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
- проектирование, включающее в себя разработку базовых алгоритмов и спецификаций,
- определение языка программирования;
- непосредственная разработка приложения;
- кодирование — по 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.

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.

4 Выполнение лабораторной работы

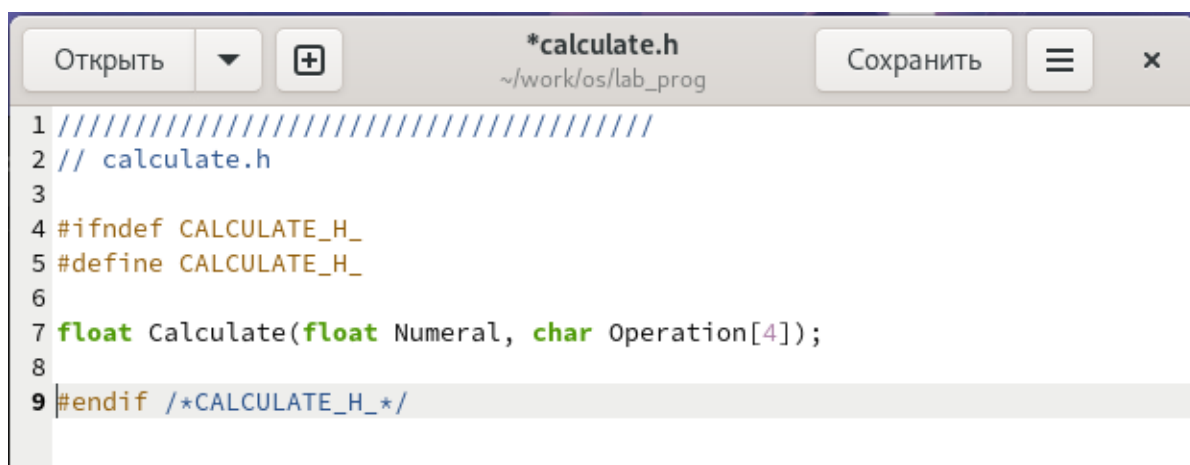
1. В домашнем каталоге создаю новый подкаталог `~/work/os/lab_prog`, перехожу в него и создаю 3 файла: `calculate.h`, `calculate.c`, `main.c` (рис. 4.1):



```
nafedorova@nafedorova:~$ mkdir work/os/lab_prog
nafedorova@nafedorova:~$ cd work/os/lab_prog
nafedorova@nafedorova:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
nafedorova@nafedorova:~/work/os/lab_prog$ ls
calculate.c calculate.h main.c
nafedorova@nafedorova:~/work/os/lab_prog$
```

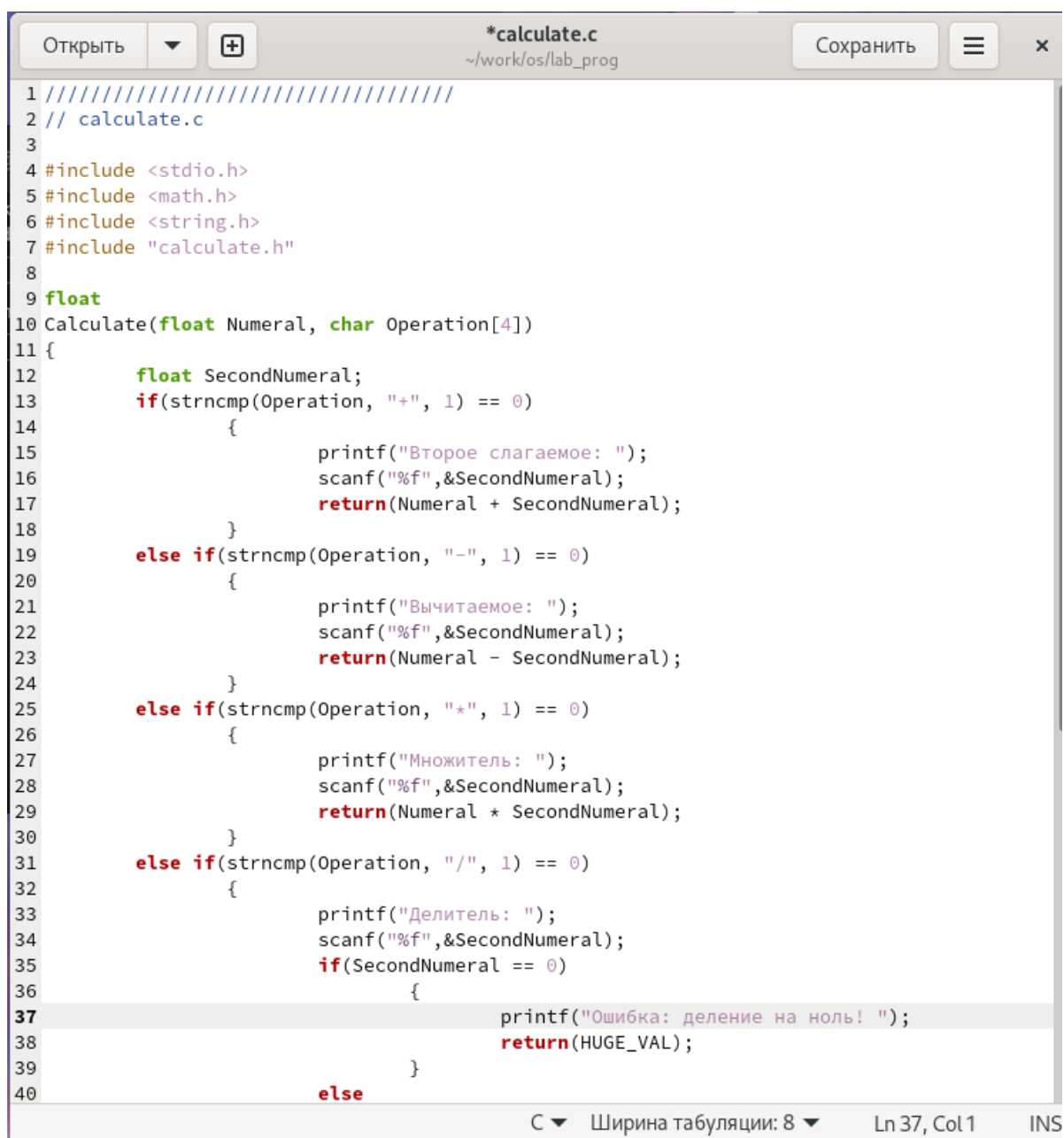
Рис. 4.1: Создание нового подкаталога и файлов

2. Записываю в файлы тексты программ, которые даны в руководстве к лабораторной работе (рис. 4.2), (рис. 4.3), (рис. 4.4).



```
1 //////////////////////////////////////////////////
2 // calculate.h
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6
7 float Calculate(float Numeral, char Operation[4]);
8
9 #endif /*CALCULATE_H_*/
```

Рис. 4.2: `calculate.h`



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

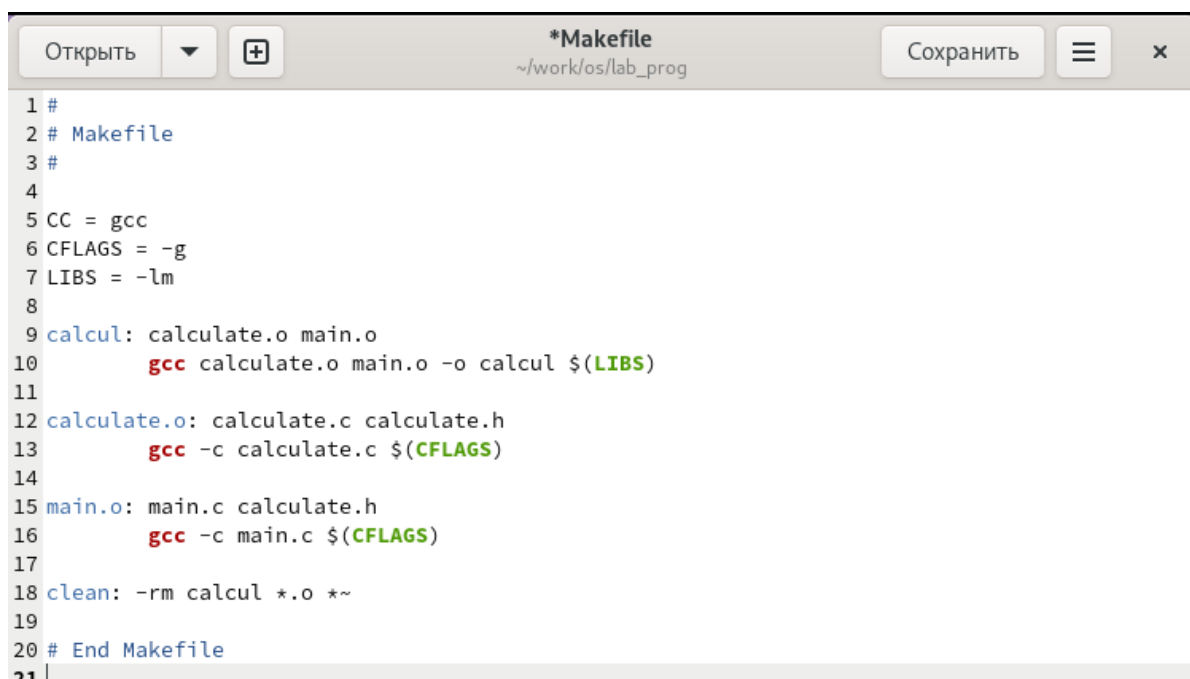
Рис. 4.3: calculate.c



```
1 //////////////////////////////////////////////////
2 // main.c
3
4 #include <stdio.h>
5 #include "calculate.h"
6
7
8 int
9 main (void)
10 {
11     float Numeral;
12     char Operation[4];
13     float Result;
14     printf("Число: ");
15     scanf("%f",&Numeral);
16     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
17     scanf("%s",&Operation);
18     Result = Calculate(Numeral, Operation);
19     printf("%6.2f\n",Result);
20     return 0;
21 }
22
```

Рис. 4.4: main.c

3. Создаю Makefile (рис. 4.5):



```
1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS = -g
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10     gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13     gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16     gcc -c main.c $(CFLAGS)
17
18 clean: -rm calcul *.o *~
19
20 # End Makefile
21
```

Рис. 4.5: Makefile

Этот Makefile используется для автоматизации процесса компиляции и сборки программы. Вот объяснение каждой его части:

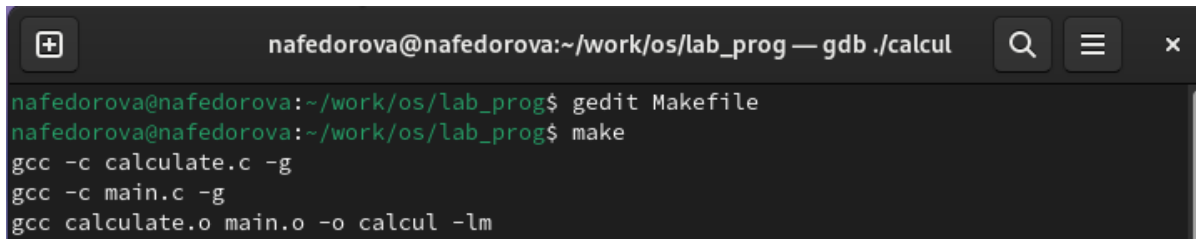
- `CC = gcc`: Эта строка устанавливает переменную `CC` равной `gcc`, что означает, что для компиляции будет использоваться компилятор GNU C.
- `CFLAGS = -g`: Здесь устанавливаются флаги компиляции (`CFLAGS`) для `gcc`. Флаг `-g` добавляет отладочную информацию в исполняемые файлы, что полезно при отладке.
- `LIBS = -lm`: Это определяет переменную `LIBS`, которая содержит флаги для линкера. `-lm` указывает на необходимость подключения математической библиотеки.

Цели и правила в Makefile:

- `calcul`: Это цель для создания исполняемого файла `calcul`. Она зависит от объектных файлов `calculate.o` и `main.o`. Команда `$(CC) calculate.o main.o -o calcul $(LIBS)` компилирует эти объектные файлы вместе с библиотеками, указанными в `LIBS`, для создания исполняемого файла.
- `calculate.o`: Это правило говорит `make`, как создать файл `calculate.o` из `calculate.c` и `calculate.h`. Команда `$(CC) -c calculate.c $(CFLAGS)` компилирует исходный файл `calculate.c` в объектный файл, используя флаги из `CFLAGS`.
- `main.o`: Похоже на предыдущее правило, но для создания `main.o` из `main.c` и `calculate.h`.
- `clean`: Это специальная цель для очистки каталога от файлов, созданных во время сборки. Команда `-rm calcul *.o *` удаляет исполняемый файл `calcul`, все объектные файлы (`.o`) и временные файлы, созданные редакторами (файлы, заканчивающиеся на `~`).

Знак минуса (`-`) перед командой `rm` говорит `make` игнорировать ошибки при удалении файлов (например, если файл уже был удалён).

4. Выполняю компиляцию программы посредством `make` (рис. 4.6):

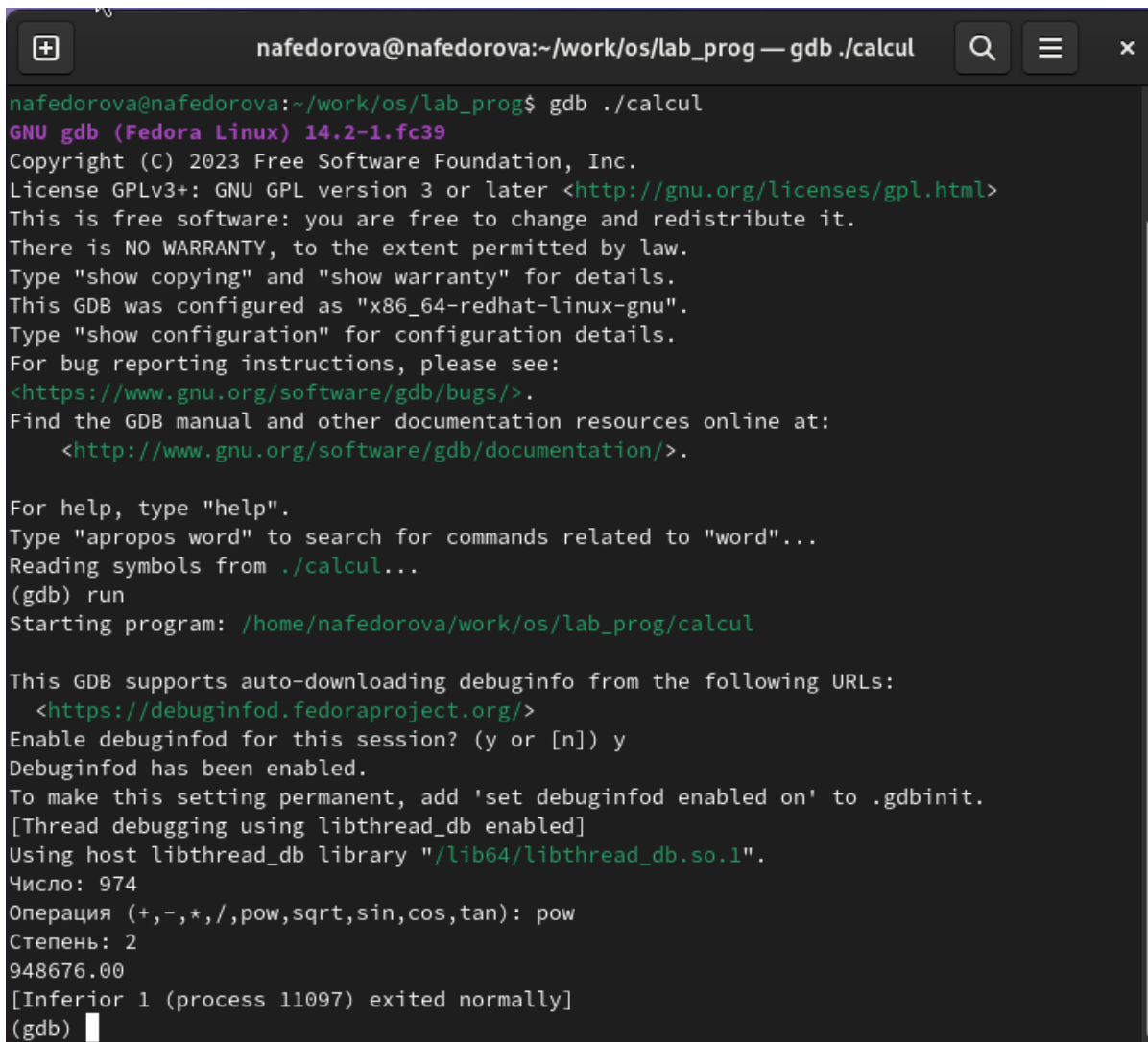
A terminal window with a dark background. The title bar shows the user 'nafedorova' at host 'nafedorova' in directory '~/work/os/lab_prog', with the application 'gdb ./calcul'. The terminal content shows a sequence of commands and their outputs: 'gedit Makefile' is executed; 'make' is executed, resulting in three lines of gcc compilation commands: 'gcc -c calculate.c -g', 'gcc -c main.c -g', and 'gcc calculate.o main.o -o calcul -lm'.

```
nafedorova@nafedorova:~/work/os/lab_prog — gdb ./calcul
nafedorova@nafedorova:~/work/os/lab_prog$ gedit Makefile
nafedorova@nafedorova:~/work/os/lab_prog$ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
```

Рис. 4.6: Компиляция программы посредством make

5. С помощью gdb выполняю отладку программы calcul:

- Запускаю отладчик GDB, загрузив в него программу для отладки, использовав `gdb ./calcul`. Для запуска программы внутри отладчика ввожу команду `run` и считая некоторое выражение (рис. 4.7)



```
nafedorova@nafedorova:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Fedora Linux) 14.2-1.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...
(gdb) run
Starting program: /home/nafedorova/work/os/lab_prog/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.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 974
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 2
948676.00
[Inferior 1 (process 11097) exited normally]
(gdb)
```

Рис. 4.7: Запуск отладчика

- Для постраничного (по 10 строк) просмотра исходного кода использую команду `list`, затем для просмотра строк с 11 по 20 основного файла использую `list 11,20`, просмотр определённых строк не основного файла использую `list calculate.c:20,25`, а также устанавливаю точку остановки в файле `calculate.c` на строке номер 17, использовав `break 17` (рис. 4.8):

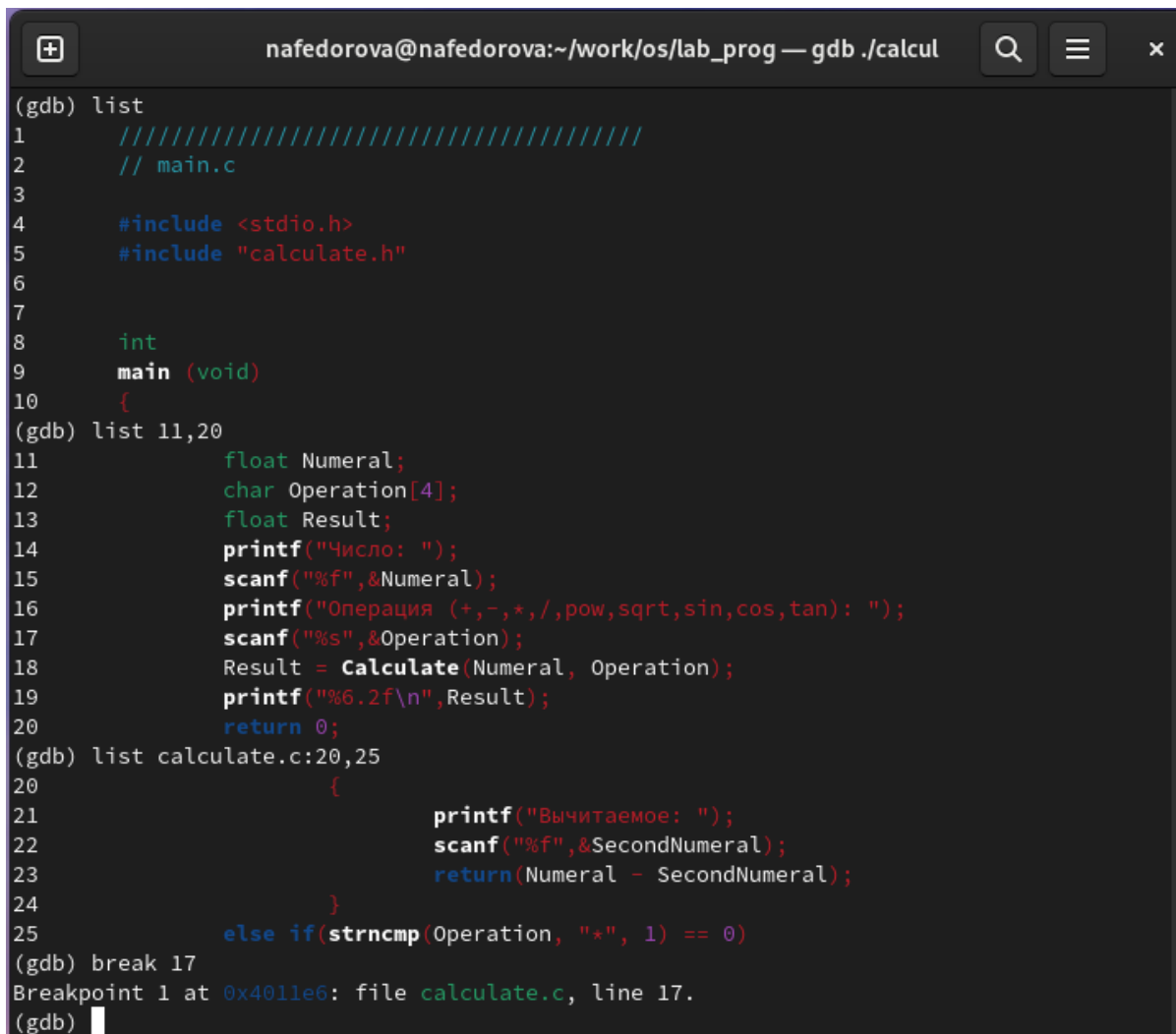
The image shows a GDB terminal window with a dark background. The title bar at the top reads 'nafedorova@nafedorova:~/work/os/lab_prog — gdb ./calcul'. The terminal content shows the following sequence of commands and output:
(gdb) list
1 ///
2 // main.c
3
4 #include <stdio.h>
5 #include "calculate.h"
6
7
8 int
9 main (void)
10 {
(gdb) list 11,20
11 float Numeral;
12 char Operation[4];
13 float Result;
14 printf("Число: ");
15 scanf("%f",&Numeral);
16 printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
17 scanf("%s",&Operation);
18 Result = Calculate(Numeral, Operation);
19 printf("%.2f\n",Result);
20 return 0;
(gdb) list calculate.c:20,25
20 {
21 printf("Вычитаемое: ");
22 scanf("%f",&SecondNumeral);
23 return(Numeral - SecondNumeral);
24 }
25 else if(strncmp(Operation, "*", 1) == 0)
(gdb) break 17
Breakpoint 1 at 0x4011e6: file calculate.c, line 17.
(gdb)
The code is color-coded: comments are green, preprocessor directives are blue, keywords are green, and identifiers/strings are red.

Рис. 4.8: Просмотр кода и точка остановки

Запускаю программу внутри отладчика с помощью `run` и убеждаюсь, что программа остановится в момент прохождения точки остановки. С помощью команды `backtrace` показываю весь стек вызываемых функций от начала программы до текущего места. Смотрю чему равно на этом этапе значение переменной `Numeral`, введя `print Numeral` и сравниваю с результатом вывода на экран после использования команды, использовав `display Numeral`. Смотрю информацию про точку останова с помощью `info breakpoints` и удаляю эту точку командой `delete 1` (рис. 4.9):

```
nafedorova@nafedorova:~/work/os/lab_prog — gdb ./calcul
(gdb) run
Starting program: /home/nafedorova/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 30
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 30

Breakpoint 1, Calculate (Numeral=30, Operation=0x7fffffffde54 "+") at calculate.c:17
17                                     return(Numeral + SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=30, Operation=0x7fffffffde54 "+") at calculate.c:17
#1 0x00000000004014eb in main () at main.c:18
(gdb) print Numeral
$1 = 30
(gdb) display Numeral
1: Numeral = 30
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint       keep y   0x00000000004011e6 in Calculate at calculate.c:17
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

Рис. 4.9: Проверка остановки и удаление точки остановки

6. С помощью утилиты splint пробую проанализировать коды файлов main.c и calculate.c. Всего 4 предупреждения (рис. 4.10).


```
nafedorova@nafedorova:~/work/os/lab_prog
nafedorova@nafedorova:~/work/os/lab_prog$ splint main.c
bash: splint: команда не найдена...
Установить пакет «splint», предоставляющий команду «splint»? [N/y] y

* Ожидание в очереди...
* Загрузка списка пакетов....
Следующие пакеты должны быть установлены:
splint-3.1.2-31.fc39.x86_64    An implementation of the lint program
Продолжить с этими изменениями? [N/y] y

* Ожидание в очереди...
* Ожидание аутентификации...
* Ожидание в очереди...
* Загрузка пакетов...
* Запрос данных...
* Проверка изменений...
* Установка пакетов...
Splint 3.1.2 --- 22 Jul 2023

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:15:2: 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:17:13: 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:17:10: Corresponding format code
main.c:17:2: Return value (type int) ignored: scanf("%s", &Ope...

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

Рис. 4.10: Анализ кода файла main.c

Теперь анализирую файл calculate.c. В сумме 15 предупреждений.

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

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:10:31: Function parameter Operation declared as manifest array
        (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:4: 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:22:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:7: 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:38:12: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:46:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:10: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:50:9: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:52:9: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:54:9: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:56:9: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:60:10: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings
nafedorova@nafedorova:~/work/os/lab_prog$
```

Рис. 4.11: Анализ кода файла calculate.c

5 Контрольные вопросы

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

Можно использовать название_программы --help для общей помощи, man название_программы для руководства пользователя или info название_программы для более подробной информации.

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

- **Дизайн:** Определение требований и архитектуры системы.
- **Кодирование:** Написание исходного кода приложения.
- **Компиляция:** Преобразование исходного кода в исполняемый файл.
- **Тестирование:** Проверка функциональности и поиск ошибок.
- **Отладка:** Исправление обнаруженных ошибок.
- **Установка:** Размещение программы в системе для использования.
- **Сопровождение:** Обновление и улучшение программы.

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

Суффикс — это расширение файла, указывающее на тип содержимого. Например, .c для исходных файлов C, .h для заголовочных файлов C.

4. Каково основное назначение компилятора языка C в UNIX?

Компилятор C преобразует исходный код на языке C в машинный код, который может выполняться операционной системой UNIX.

5. Для чего предназначена утилита make?

make автоматизирует процесс компиляции и сборки программы, используя файл Makefile для определения зависимостей между файлами и правил сборки.

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

Пример структуры Makefile:

```
all: program
```

```
program: main.o lib.o
```

```
gcc -o program main.o lib.o
```

```
main.o: main.c
```

```
gcc -c main.c
```

```
lib.o: lib.c
```

```
gcc -c lib.c
```

```
clean:
```

```
rm -f *.o program
```

Элементы Makefile:

- **Цели:** all, program, main.o, lib.o, clean.
- **Зависимости:** Файлы, от которых зависит цель.

- **Правила:** Команды для создания цели из зависимостей.
- **Псевдоцели:** Цели, не связанные с файлами, например `clean`.

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

Возможность остановить выполнение программы, просмотреть и изменить значения переменных. Для использования требуется скомпилировать программу с опцией отладки (например, `gcc -g`).

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

Основные команды `gdb`:

- `run`: Запуск программы.
- `break`: Установка точки останова.
- `next`: Выполнение следующей строки кода.
- `continue`: Продолжение выполнения до следующей точки останова.
- `print`: Вывод значения переменной.
- `quit`: Выход из `gdb`.

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

- Компиляция с опцией `-g`.
- Запуск `gdb`.
- Установка точек останова.
- Запуск программы с помощью `run`.

- Просмотр и изменение переменных.
- Продолжение выполнения и наблюдение за поведением программы.

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

Компилятор выдаст сообщения об ошибках, указывая местоположение и возможную причину ошибки.

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

- Комментарии.
- Читаемые имена переменных и функций.
- Структурирование кода.
- Документация.

12. Каковы основные задачи, решаемые программой splint?

splint выполняет статический анализ кода на C для обнаружения ошибок программирования, уязвимостей безопасности и некачественного кода.

6 Выводы

В данной лабораторной работе 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.