

Отчёт по лабораторной работе №12:

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

Касакьянц Владислав Сергеевич

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
4	Контрольные вопросы	14
5	Выводы	18
	Список литературы	19

Список иллюстраций

3.1	Создание нового подкаталога и файлов	6
3.2	calculate.h	6
3.3	calculate.c	7
3.4	main.c	8
3.5	компиляция программы посредством gcc	8
3.6	Makefile	8
3.7	Запуск отладчика	9
3.8	Просмотр кода и точка останова	10
3.9	Проверка останова и удаление точки останова	11
3.10	Анализ кода файла main.c	12
3.11	Анализ кода файла calculate.c	13

1 Цель работы

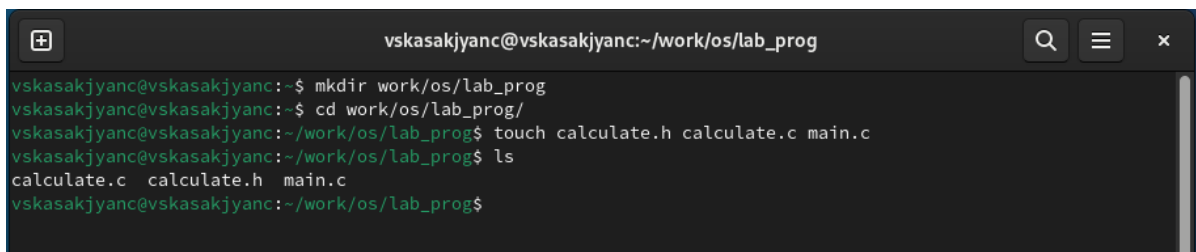
Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

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 Выполнение лабораторной работы

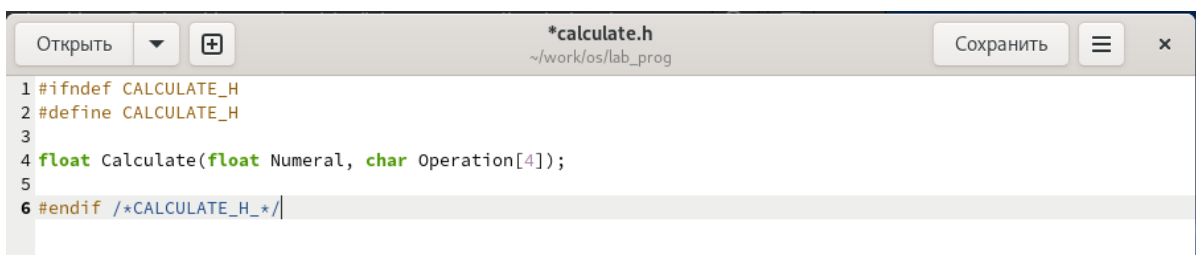
1. В домашнем каталоге создаю новый подкаталог `~/work/os/lab_prog`, перехожу в него и создаю 3 файла: `calculate.h`, `calculate.c`, `main.c` (рис. 3.1):



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

Рис. 3.1: Создание нового подкаталога и файлов

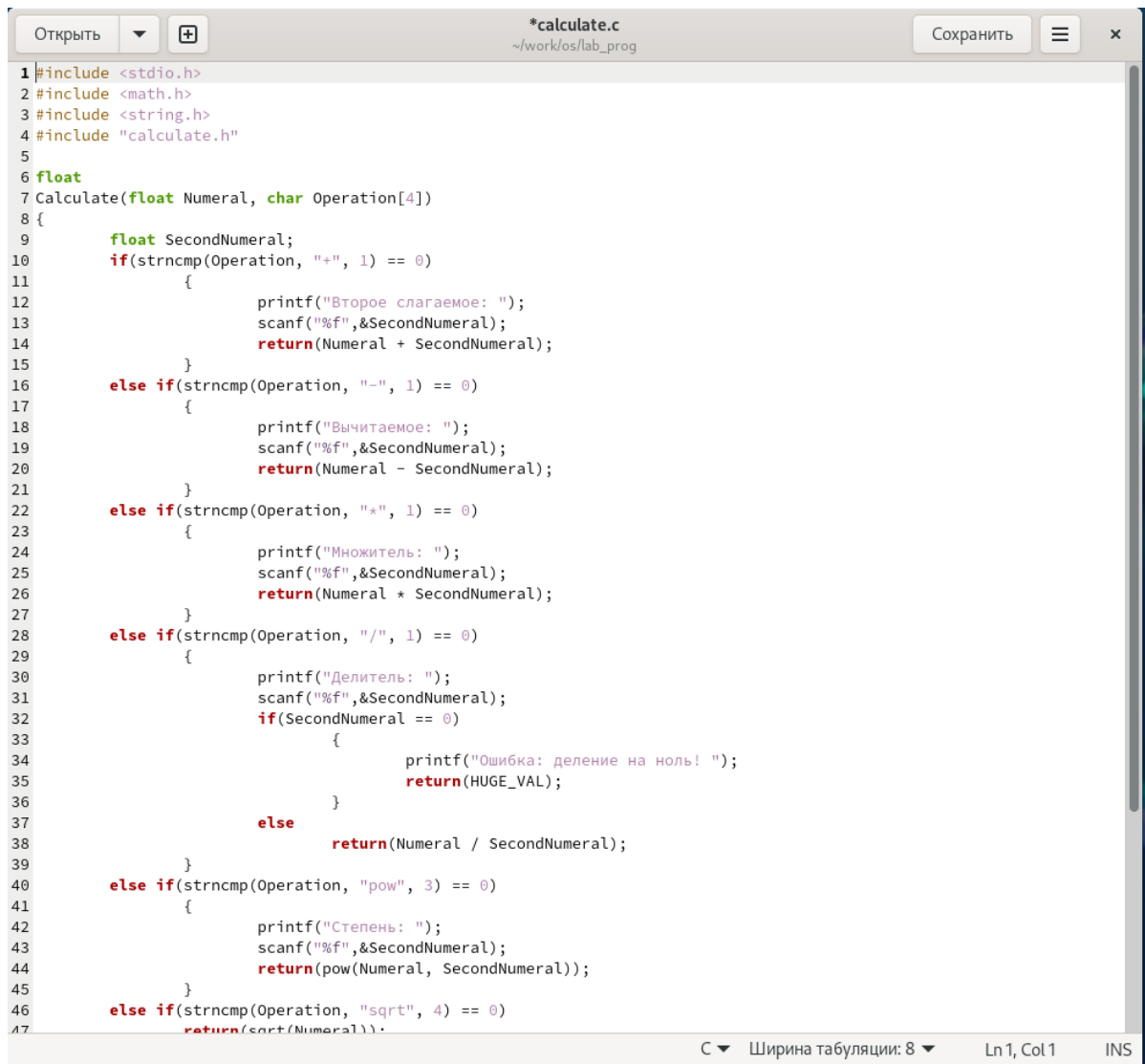
2. Запишем в файлы тексты программ, которые даны в лабораторной работе (рис. 3.2), (рис. 3.3), (рис. 3.4).



```
*calculate.h
~/work/os/lab_prog

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

Рис. 3.2: `calculate.h`



```
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(strncmp(Operation, "+", 1) == 0)
11    {
12        printf("Второе слагаемое: ");
13        scanf("%f",&SecondNumeral);
14        return(Numeral + SecondNumeral);
15    }
16    else if(strncmp(Operation, "-", 1) == 0)
17    {
18        printf("Вычитаемое: ");
19        scanf("%f",&SecondNumeral);
20        return(Numeral - SecondNumeral);
21    }
22    else if(strncmp(Operation, "*", 1) == 0)
23    {
24        printf("Множитель: ");
25        scanf("%f",&SecondNumeral);
26        return(Numeral * SecondNumeral);
27    }
28    else if(strncmp(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(strncmp(Operation, "pow", 3) == 0)
41    {
42        printf("Степень: ");
43        scanf("%f",&SecondNumeral);
44        return(pow(Numeral, SecondNumeral));
45    }
46    else if(strncmp(Operation, "sqrt", 4) == 0)
47        return(sqrt(Numeral));
```

Рис. 3.3: calculate.c

Рис. 3.4: main.c

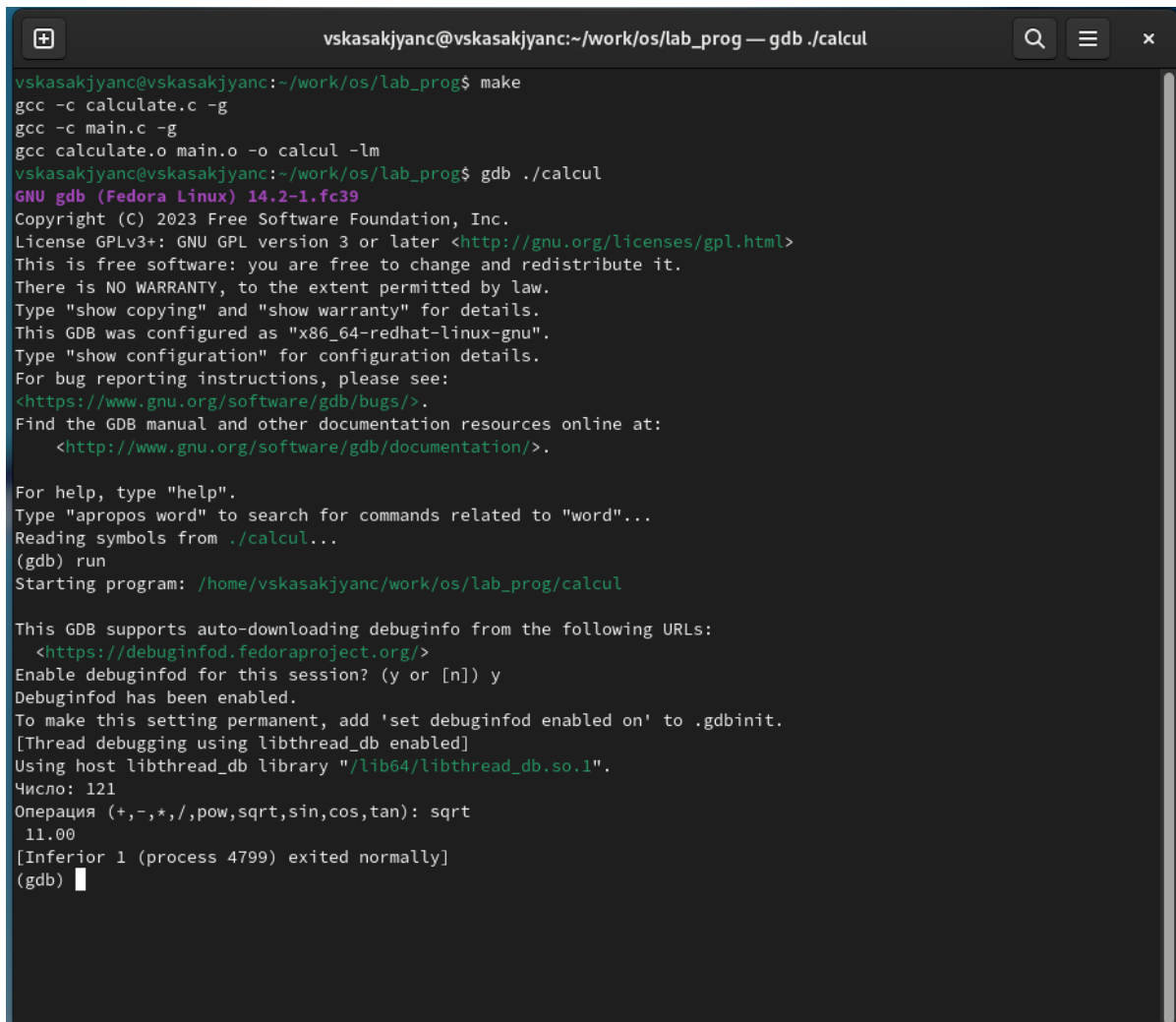
3. Выполним компиляцию программы посредством gcc (рис. 3.5):

Рис. 3.5: компиляция программы посредством gcc

4. Создадим Makefile (рис. 3.6):

Рис. 3.6: Makefile

5. Запустим отладчик GDB, загрузив в него программу для отладки. Запустим программу и посчитаем некое выражение (рис. 3.7)



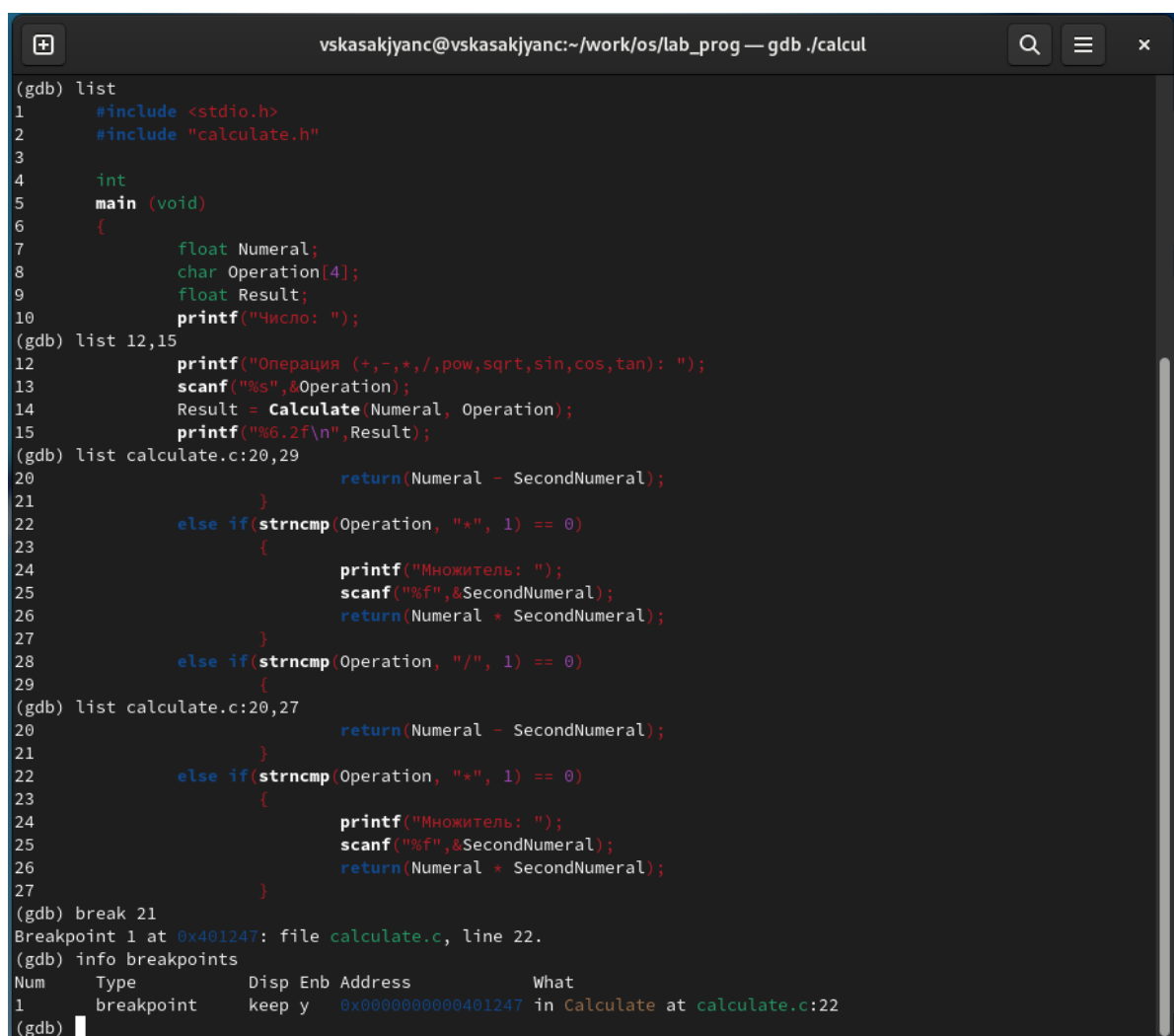
```
vskasakjyanc@vskasakjyanc:~/work/os/lab_prog — gdb ./calcul
vskasakjyanc@vskasakjyanc:~/work/os/lab_prog$ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
vskasakjyanc@vskasakjyanc:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Fedora Linux) 14.2-1.fc39
Copyright (C) 2023 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/vskasakjyanc/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: 121
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sqrt
11.00
[Inferior 1 (process 4799) exited normally]
(gdb) █
```

Рис. 3.7: Запуск отладчика

Для постраничного (по 9 строк) просмотра исходного код используем команду `list`, затем для просмотра строк с 12 по 15 основного файла используем `list 12,15`, посмотрим определённых строк не основного файла, используя `list calculate.c:20,29`, а также установим точку останова в файле `calculate.c` на строке номер 21, использовав `list calculate.c:20,27` и `break 21` (рис. 3.8):



```
(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3
4      int
5      main (void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Число: ");
(gdb) list 12,15
12         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13         scanf("%s",&Operation);
14         Result = Calculate(Numeral, Operation);
15         printf("%6.2f\n",Result);
(gdb) list calculate.c:20,29
20         return(Numeral - SecondNumeral);
21     }
22     else if(strcmp(Operation, "*") == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strcmp(Operation, "/") == 0)
29     {
(gdb) list calculate.c:20,27
20         return(Numeral - SecondNumeral);
21     }
22     else if(strcmp(Operation, "*") == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
(gdb) break 21
Breakpoint 1 at 0x401247: file calculate.c, line 22.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint      keep y   0x0000000000401247 in Calculate at calculate.c:22
(gdb)
```

Рис. 3.8: Просмотр кода и точка останова

Запустим программу внутри отладчика с помощью `run` и убедимся, что программа остановится в момент прохождения точки останова. С помощью команды `backtrac` покажем весь стек вызываемых функций от начала программы до текущего места. Посмотрим, чему равно на этом этапе значение переменной `Numeral`, введя `print Numeral` и сравним с результатом вывода на экран после использования команды, использовав `display Numeral`. Посмотрим, информацию про точку останова с помощью `info breakpoints` и удалим эту точку командой `delete 1` (рис. 3.9):

```
vskasakjanc@vskasakjanc:~/work/os/lab_prog — gdb ./calcul

(gdb) run
Starting program: /home/vskasakjanc/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: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 8
-3.00
[Inferior 1 (process 5221) exited normally]
(gdb) run
Starting program: /home/vskasakjanc/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde14 "*") at calculate.c:22
22         else if(strncmp(Operation, "+", 1) == 0)
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffde14 "*") at calculate.c:22
#1 0x00000000004014eb in main () at main.c:14
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num   Type             Disp Enb Address                      What
1     breakpoint        keep y   0x0000000000401247 in Calculate at calculate.c:22
      breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

Рис. 3.9: Проверка остановки и удаление точки останова

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

В файле `main.c` всего 3 предупреждения (рис. 3.10).

```
vskasakjyanc@vskasakjyanc:~/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: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:11:2: 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:13:13: 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:13:10: Corresponding format code
main.c:13:2: Return value (type int) ignored: scanf("%s", &Ope...

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

Рис. 3.10: Анализ кода файла main.c

А в файлу calculate.c всего 15 предупреждений (рис. 3.11).

```
vskasakjyanc@vskasakjyanc:~/work/os/lab_prog
vskasakjyanc@vskasakjyanc:~/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:13:4: 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:19:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:7: 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:35:12: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:43:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:10: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:47:9: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:49:9: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:51:9: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:53:9: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:57:10: Return value type double does not match declared type float:
        (HUGE_VAL)

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

Рис. 3.11: Анализ кода файла calculate.c

4 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 С в UNIX?

Компилятор С преобразует исходный код на языке С 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.

5 Выводы

В данной лабораторной работе мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

Список литературы