

# **Лабораторная работа №13**

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

Медникова Екатерина Михайловна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	19
4	Контрольные вопросы	20

## Список иллюстраций

2.1	Создание подкаталога . . . . .	6
2.2	Создание файлов . . . . .	6
2.3	Вставка программы в файл calculate.c . . . . .	7
2.4	Вставка программы в файл calculate.c . . . . .	8
2.5	Вставка программы в файл calculate.h . . . . .	9
2.6	Вставка программы в файл main.c . . . . .	10
2.7	Компиляция программы . . . . .	10
2.8	Создание файла . . . . .	11
2.9	Содержание файла . . . . .	11
2.10	Исправление файла Makefile . . . . .	12
2.11	Компиляция файлов . . . . .	12
2.12	Отладка программы . . . . .	13
2.13	Запуск программы . . . . .	13
2.14	Использование команды list . . . . .	14
2.15	Использование команды list с параметрами . . . . .	14
2.16	Просмотр определённых строк . . . . .	14
2.17	Точка останова . . . . .	15
2.18	Информация об имеющихся точка останова . . . . .	15
2.19	Запуск программы внутри отладчика . . . . .	15
2.20	Вывод информации . . . . .	16
2.21	Значение переменной Numeral . . . . .	16
2.22	Использование команды display Numeral . . . . .	16
2.23	Использование команд info breakpoints и delete 1 . . . . .	16
2.24	splint calculate.c . . . . .	17
2.25	splint calculate.c . . . . .	18
2.26	splint main.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 Выполнение лабораторной работы

1. В домашнем каталоге создала подкаталог ~/work/os/lab\_prog.

```
[emmednikova@fedora ~]$ cd work/os  
[emmednikova@fedora os]$ mkdir lab_prog  
[emmednikova@fedora os]$ cd lab_prog/  
[emmednikova@fedora lab_prog]$
```

Рис. 2.1: Создание подкаталога

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

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

Рис. 2.2: Создание файлов

Написала программы в созданные файлы.



```
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)
```

С ▾ Ширина табуляции: 8 ▾

Рис. 2.3: Вставка программы в файл calculate.c



```
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     else
41         return(Numeral / SecondNumeral);
42 }
43 else if(strncmp(Operation, "pow", 3) == 0)
44 {
45     printf("Степень: ");
46     scanf("%f",&SecondNumeral);
47     return(pow(Numeral, SecondNumeral));
48 }
49 else if(strncmp(Operation, "sqrt", 4) == 0)
50     return(sqrt(Numeral));
51 else if(strncmp(Operation, "sin", 3) == 0)
52     return(sin(Numeral));
53 else if(strncmp(Operation, "cos", 3) == 0)
54     return(cos(Numeral));
55 else if(strncmp(Operation, "tan", 3) == 0)
56     return(tan(Numeral));
57 else
58 {
59     printf("Неправильно введено действие ");
60     return(HUGE_VAL);
61 }
62 }
```

С ▾ Ширина табуляции: 8 ▾

Рис. 2.4: Вставка программы в файл calculate.c

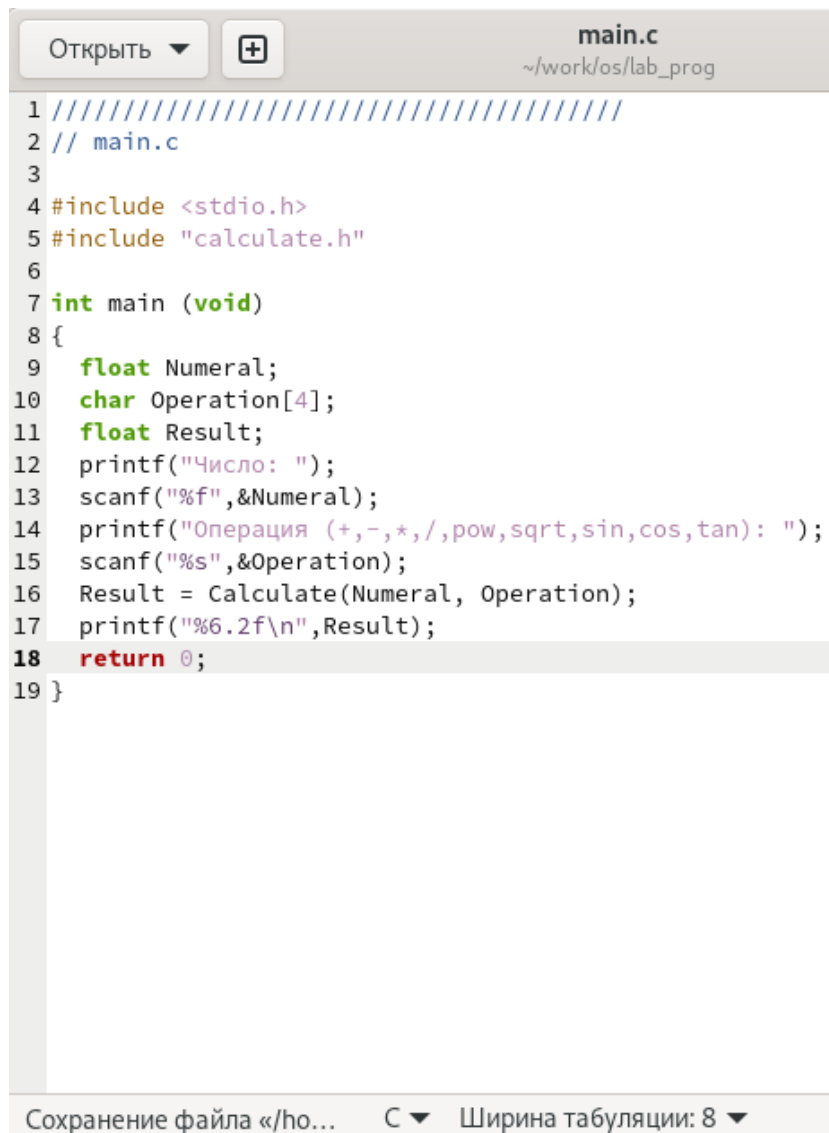


The image shows a code editor window titled "calculate.h" with the path "~/work/os/lab\_prog". The editor contains the following C code:

```
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_*/
```

The status bar at the bottom indicates "Сохране...", "Заголовок C/ObjC", and "Ширина табуляции: 8".

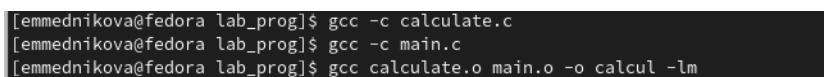
Рис. 2.5: Вставка программы в файл calculate.h



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

Рис. 2.6: Вставка программы в файл main.c

3. Выполнила компиляцию программы посредством gcc:



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

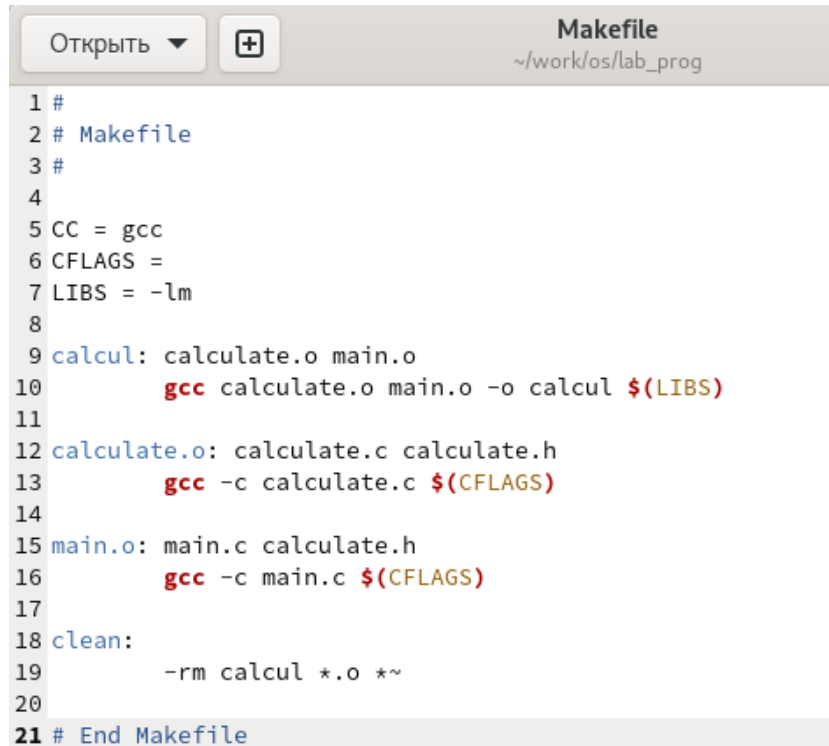
Рис. 2.7: Компиляция программы

4. При компиляции программы синтаксических ошибок выявлено не было.

5. Создала Makefile со следующим содержанием:

```
[emmednikova@fedora lab_prog]$ touch Makefile
[emmednikova@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
[emmednikova@fedora lab_prog]$
```

Рис. 2.8: Создание файла

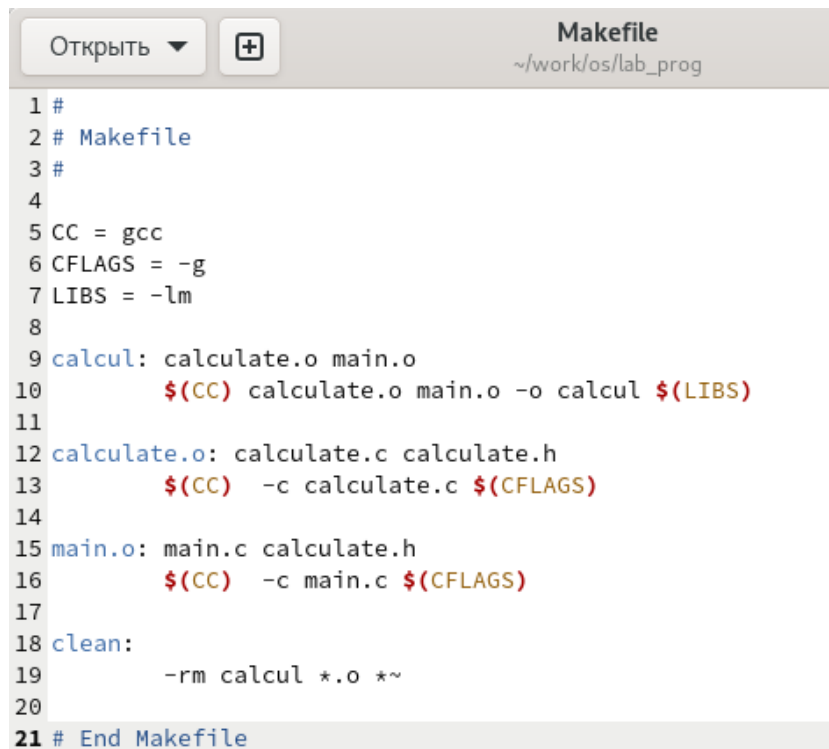


```
1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS =
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10     gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13     gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16     gcc -c main.c $(CFLAGS)
17
18 clean:
19 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

Рис. 2.9: Содержание файла

Данный файл нужен для автоматической компиляции файлов `calculate.c`, `main.c`, а также их объединения в один исполняемый файл `calcul`. Функция `clean` - автоматическое удаление файлов. Переменная `CC` отвечает за утилиту для компиляции. Переменная `CFLAGS` отвечает за опции в данной утилите. Переменная `LIBS` отвечает за опции для объединения объектных файлов в один исполняемый файл.

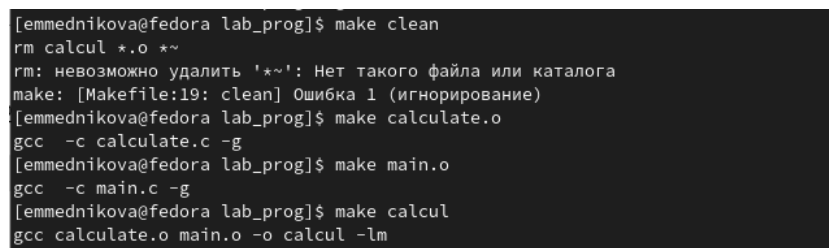
6. Перед использованием `gdb` исправила Makefile:



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

Рис. 2.10: Исправление файла Makefile

Выполнила компиляцию файлов.



```
[emmednikova@fedora lab_prog]$ make clean
rm calcul *.o *~
rm: невозможно удалить '*~': Нет такого файла или каталога
make: [Makefile:19: clean] Ошибка 1 (игнорирование)
[emmednikova@fedora lab_prog]$ make calculate.o
gcc -c calculate.c -g
[emmednikova@fedora lab_prog]$ make main.o
gcc -c main.c -g
[emmednikova@fedora lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
```

Рис. 2.11: Компиляция файлов

С помощью gdb выполнила отладку программы calcul.

```
[emmednikova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 13.1-3.fc37
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)
```

Рис. 2.12: Отладка программы

Для запуска программы внутри отладчика ввела команду run.

```
(gdb) run
Starting program: /home/emmednikova/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.
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
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: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 8
11.00
[Inferior 1 (process 3929) exited normally]
(gdb)
```

Рис. 2.13: Запуск программы

Для постраничного (по 9 строк) просмотра исходного код использовала команду list.

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

Рис. 2.14: Использование команды list

Для просмотра строк с 12 по 15 основного файла использовала list с параметрами.

```
(gdb) list 12,15
12     printf("Число: ");
13     scanf("%f",&Numeral);
14     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15     scanf("%s",&Operation);
(gdb) █
```

Рис. 2.15: Использование команды list с параметрами

Для просмотра определённых строк не основного файла использовала list с параметрами.

```
(gdb) list calculate.c:20,27
20     {
21         printf("Вычитаемое: ");
22         scanf("%f",&SecondNumeral);
23         return(Numeral - SecondNumeral);
24     }
25     else if(strncmp(Operation, "+", 1) == 0)
26     {
27         printf("Множитель: ");
(gdb) █
```

Рис. 2.16: Просмотр определённых строк

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

```
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb)
```

Рис. 2.17: Точка останова

Вывела информацию об имеющихся в проекте точках останова.

```
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint     keep y   0x000000000040120f in Calculate
                                at calculate.c:21
(gdb)
```

Рис. 2.18: Информация об имеющихся точка останова

Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова.

```
(gdb) run
Starting program: /home/emmednikova/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=0x7fffffffdf24 "-")
  at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:16
(gdb)
```

Рис. 2.19: Запуск программы внутри отладчика

Отладчик выдал следующую информацию:

```
(gdb) run
Starting program: /home/emmednikova/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=0x7fffffffdf24 "-")
at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:16
(gdb)
```

Рис. 2.20: Вывод информации

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду print Numeral. На экран было выведено число 5.

```
(gdb) print Numeral
$1 = 5
(gdb)
```

Рис. 2.21: Значение переменной Numeral

После использования команды display Numeral на экран также было выведено число 5.

```
(gdb) display Numeral
1: Numeral = 5
(gdb)
```

Рис. 2.22: Использование команды display Numeral

Убрала точки останова.

```
(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1     breakpoint keep y  0x000000000040120f in Calculate
                                at calculate.c:21
      breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Рис. 2.23: Использование команд info breakpoints и delete 1



7. С помощью утилиты splint было замечено, что в файлах calculate.c и main.c есть функция scanf, которая возвращает целое число, но данные числа не используются и нигде не сохраняются. Далее утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулём.

```
[emmednikova@fedora lab_prog]$ splint calculate.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-3.1.2-29.fc37.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 --- 23 Jul 2022

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:16:7: 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)
```

Рис. 2.24: splint calculate.c

```

calculate.c:10:31: function parameter operation declared as manifest array
      (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:7: 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:22:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:10: 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:38:17: Return value type double does not match declared type float:
      (HUGE_VAL)
      To allow all numeric types to match, use +relaxtypes.
calculate.c:46:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:13: Return value type double does not match declared type float:
      (pow(Numeral, SecondNumeral))
calculate.c:50:11: Return value type double does not match declared type float:
      (sqrt(Numeral))
calculate.c:52:11: Return value type double does not match declared type float:
      (sin(Numeral))
calculate.c:54:11: Return value type double does not match declared type float:
      (cos(Numeral))
calculate.c:56:11: Return value type double does not match declared type float:
      (tan(Numeral))
calculate.c:60:13: Return value type double does not match declared type float:
      (HUGE_VAL)
Finished checking --- 15 code warnings

```

Рис. 2.25: splint calculate.c

```

[emmednikova@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2022

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:13:3: 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:15:14: 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:15:11: Corresponding format code
main.c:15:3: Return value (type int) ignored: scanf("%s", &Ope...

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

```

Рис. 2.26: splint main.c

## 3 Выводы

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.

## 4 Контрольные вопросы

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

Чтобы получить информацию о возможностях программ gcc, make, gdb и др., нужно использовать команду `man` или `-help (-h)`.

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.*

Для имени входного файла суффикс определяет, какая компиляция требуется. Суффиксы указывают на тип объекта. 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. Например, в команде “gcc-stain.c”: gcc по расширению .c распознаёт тип файла для компиляции и формирует объектный модуль - файл с расширением .o.

#### *4. Каково основное назначение компилятора языка C в UNIX?*

Основное назначение компилятора языка C в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

#### *5. Для чего предназначена утилита make?*

Для сборки разрабатываемого приложения и компиляции лучше всего воспользоваться командой make. Она 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.*

Для работы с утилитой make необходимо в корне рабочего каталога с проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefile имеет следующий синтаксис:

... : ... <команда 1> ...

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. 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?*

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:

```
gcc -c file.c -g
```

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл:

```
gdb file.o
```

Затем можно использовать по мере необходимости различные команды gdb. Для выхода из gdb можно воспользоваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d . Более подробную информацию по работе с gdb можно получить с помощью команд gdb -h и man gdb.

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

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

Схема отладки программы показана в 6 пункте лабораторной работы.

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

При первом запуске компилятор не выдал никаких синтаксических ошибок.

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

Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

score - исследование функций, содержащихся в программе;

lint - критическая проверка программ, написанных на языке C.

12. *Каковы основные задачи, решаемые программой splint?*

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.