

Лабораторная работа №13.

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

ОЗЬЯС Стив Икнэль Дани

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	9
4	Выполнение лабораторной работы	11
5	Выводы	19
6	Контрольные вопросы	20
	Список литературы	23

Список иллюстраций

4.1	Создание подкаталога ~/work/os/lab_prog	11
4.2	Создание файлов calculate.h, calculate.c, main.c	11
4.3	calculate.h	12
4.4	calculate.c	12
4.5	main.c	13
4.6	Выполнение компиляции программы посредством gcc	13
4.7	Создание Makefile	14
4.8	Выполнение отладки программы calcul	15
4.9	Запуск отладчика GDB	15
4.10	Запуск программы внутри отладчика	15
4.11	Просмотр исходного кода	16
4.12	Просмотр строк с 12 по 15 основного файла	16
4.13	Просмотр определённых строк не основного файла	16
4.14	Установка точки останова в файле calculate.c	16
4.15	Вывод информации о точке останова	17
4.16	Запуск программы внутри отладчика при наличии точки останова	17
4.17	backtrace	17
4.18	Значение переменной Numeral на этом этапе	17
4.19	Сравнение с результатом вывода на экран после использования ко- манды display	17
4.20	Удаление точки останова	18
4.21	Анализ кода файла calculate.c	18
4.22	Анализ кода файла main.c	18

Список таблиц

3.1	Описание полезных для выполнения работы команд	10
-----	--	----

1 Цель работы

Цель данной 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. Реализация функций калькулятора в файле `calculate.h`, интерфейсный файл `calculate.h`, описывающий формат вызова функции калькулятора, основной файл `main.c`, реализующий интерфейс пользователя к калькулятору.
3. Выполните компиляцию программы посредством `gcc`:
 - `gcc -c calculate.c`
 - `gcc -c main.c`
 - `gcc calculate.o main.o -o calcul -lm`
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` с заданным содержанием. 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:
 - `gdb ./calcul`
 - Для запуска программы внутри отладчика введите команду `run`:

- run
- Для постраничного (по 9 строк) просмотра исходного код используйте команду list:
 - list
- Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:
 - list 12,15
- Для просмотра определённых строк не основного файла используйте list с параметрами:
 - list calculate.c:20,29
- Установите точку останова в файле calculate.c на строке номер 21:
 - list calculate.c:20,27
 - break 21
- Выведите информацию об имеющихся в проекте точка останова:
 - info breakpoints
- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:
 - run
 - 5
 - backtrace
- Отладчик выдаст следующую информацию:
 - #0 Calculate (Numeral=5, Operation=0x7fffffff280 “-”)
 - at calculate.c:21
 - #1 0x00000000400b2b in main () at main.c:17
- А команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места.
- Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:

- print Numeral
 - На экран должно быть выведено число 5.
 - Сравните с результатом вывода на экран после использования команды:
 - display Numeral
 - Уберите точки останова:
 - info breakpoints
 - delete 1
7. 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.
- В табл. 3.1 приведено краткое описание полезных для выполнения работы команд и клавиш.

Таблица 3.1: Описание полезных для выполнения работы команд

Команда	Описание действия
<code>backtrace</code>	вывод на экран пути к текущей точке останова (по сути вывод названий всех функций)
<code>break</code>	установить точку останова (в качестве параметра может быть указан номер строки или название функции)
<code>continue</code>	продолжить выполнение программы
<code>info</code>	вывести на экран список используемых точек останова
<code>breakpoints</code>	
<code>delete</code>	удалить точку останова
<code>list</code>	вывести на экран исходный код
<code>run</code>	запуск программы на выполнение

- Более подробно об Unix см. в [1–6].

4 Выполнение лабораторной работы

1. В домашнем каталоге создал подкаталог ~/work/os/lab_prog. (рис. 4.1)

```
sozjyas@dk3n60 ~ $ cd work/  
sozjyas@dk3n60 ~/work $ ls  
os  
sozjyas@dk3n60 ~/work $ cd os  
sozjyas@dk3n60 ~/work/os $ mkdir lab_prog  
sozjyas@dk3n60 ~/work/os $ ls  
lab06 lab_prog
```

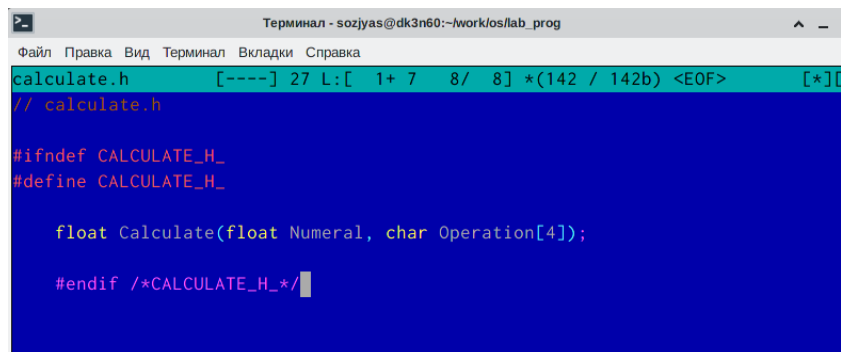
Рис. 4.1: Создание подкаталога ~/work/os/lab_prog

2. Создал в нём файлы: calculate.h, calculate.c, main.c. (рис. 4.2)

```
Терминал - sozjyas@dk3n60:~/work/os/lab_prog  
Файл Правка Вид Терминал Вкладки Справка  
sozjyas@dk3n60 ~/work/os $ cd lab_prog/  
sozjyas@dk3n60 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c  
sozjyas@dk3n60 ~/work/os/lab_prog $ ls  
calculate.c calculate.h main.c  
sozjyas@dk3n60 ~/work/os/lab_prog $
```

Рис. 4.2: Создание файлов calculate.h, calculate.c, main.c

- calculate.h, (рис. 4.3)



```
Терминал - sozjyas@dk3n60:~/work/os/lab_prog
Файл  Правка  Вид  Терминал  Вкладки  Справка
calculate.h [----] 27 L: [ 1+ 7 8/ 8] *(142 / 142b) <EOF> [*]
// calculate.h

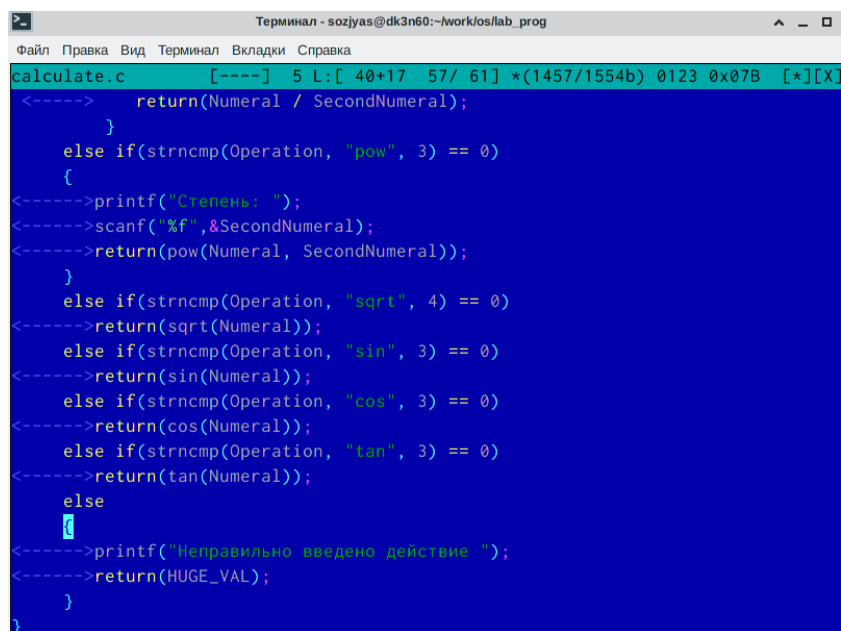
#ifndef CALCULATE_H_
#define CALCULATE_H_

    float Calculate(float Numeral, char Operation[4]);

#endif /*CALCULATE_H_*/
```

Рис. 4.3: calculate.h

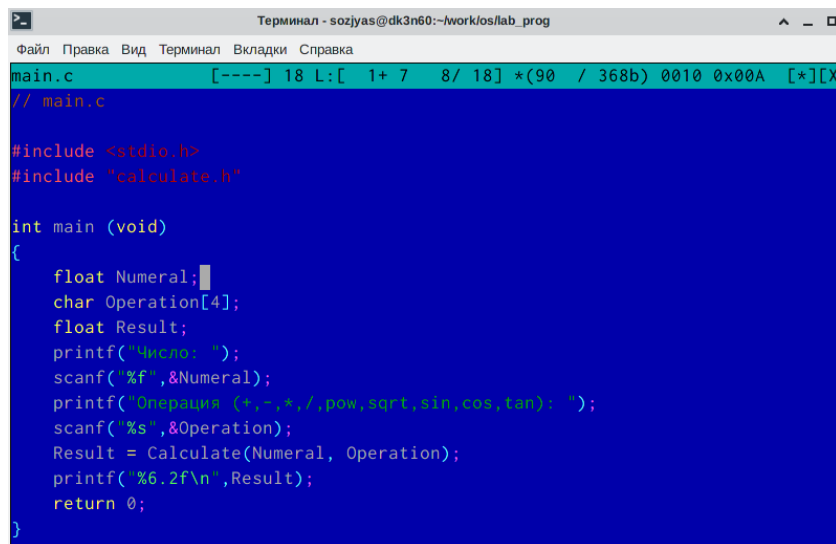
- calculate.c, (рис. 4.4)



```
Терминал - sozjyas@dk3n60:~/work/os/lab_prog
Файл  Правка  Вид  Терминал  Вкладки  Справка
calculate.c [----] 5 L: [ 40+17 57/ 61] *(1457/1554b) 0123 0x07B [*][X]
<-----> return(Numeral / SecondNumeral);
    }
    else if(strncmp(Operation, "pow", 3) == 0)
    {
<-----> printf("Степень: ");
<-----> scanf("%f", &SecondNumeral);
<-----> return(pow(Numeral, SecondNumeral));
    }
    else if(strncmp(Operation, "sqrt", 4) == 0)
<-----> return(sqrt(Numeral));
    else if(strncmp(Operation, "sin", 3) == 0)
<-----> return(sin(Numeral));
    else if(strncmp(Operation, "cos", 3) == 0)
<-----> return(cos(Numeral));
    else if(strncmp(Operation, "tan", 3) == 0)
<-----> return(tan(Numeral));
    else
    {
<-----> printf("Неправильно введено действие ");
<-----> return(HUGE_VAL);
    }
}
```

Рис. 4.4: calculate.c

- main.c (рис. 4.5)



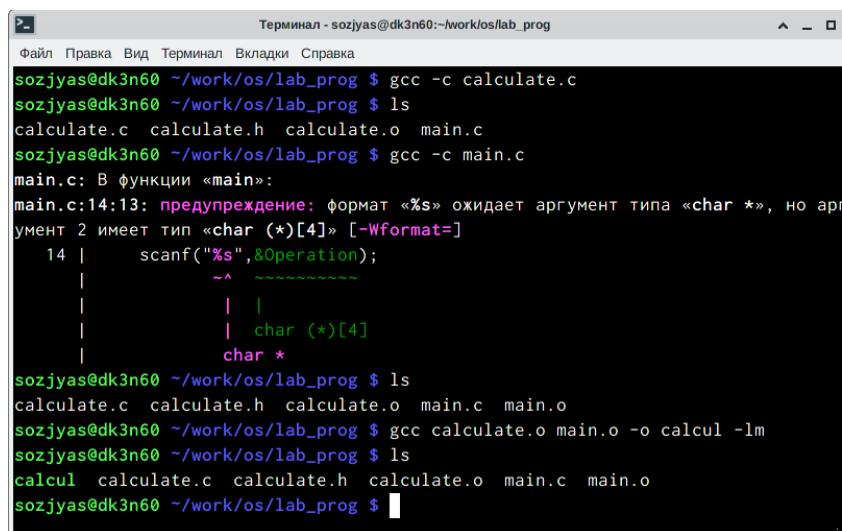
```
main.c [----] 18 L: [ 1+ 7 8/ 18] *(90 / 368b) 0010 0x00A [*][X]
// main.c

#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("%.2f\n",Result);
    return 0;
}
```

Рис. 4.5: main.c

3. Выполнил компиляцию программы посредством gcc (рис. 4.6)



```
sozjyas@dk3n60 ~/work/os/lab_prog $ gcc -c calculate.c
sozjyas@dk3n60 ~/work/os/lab_prog $ ls
calculate.c calculate.h calculate.o main.c
sozjyas@dk3n60 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:14:13: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
   14 |     scanf("%s",&Operation);
      |           ^~ ~~~~~
      |           | |
      |           | char (*)[4]
      |           char *
sozjyas@dk3n60 ~/work/os/lab_prog $ ls
calculate.c calculate.h calculate.o main.c main.o
sozjyas@dk3n60 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
sozjyas@dk3n60 ~/work/os/lab_prog $ ls
calcul calculate.c calculate.h calculate.o main.c main.o
sozjyas@dk3n60 ~/work/os/lab_prog $
```

Рис. 4.6: Выполнение компиляции программы посредством gcc

4. Я исправил незначительную синтаксическую ошибку.

5. Создал Makefile с заданным содержанием (рис. 4.7)

- CC = gcc замена слова gcc на CC
- LIBS = -lm дополнительные опции

- `calcul: calculate.o main.o`
 - `gcc calculate.o main.o -o calcul $(LIBS)` команда для создания исполняемого файла `calcul`
- `calculate.o: calculate.c calculate.h`
 - `gcc -c calculate.c $(CFLAGS)` команда для создания объектного файла `calculate.o`
- `main.o: main.c calculate.h`
 - `gcc -c main.c $(CFLAGS)` команда для создания объектного файла `main.o`
- `clean:`
 - `-rm calcul .o ~` команда для удаления всех объектных файлов и файлов с знаком `~` в конце

```

Терминал - sozjyas@dk5n52:~/work/os/lab_prog
Файл Правка Вид Терминал Вкладки Справка
Makefile [----] 0 L: [ 1+19 20/ 20] *(274 / 288b) 0035 0x023 [*][X]
# Makefile

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
<-----><----->$(CC) calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
<-----><----->$(CC) -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
<-----><----->$(CC) -c main.c $(CFLAGS)

clean:
<-----><----->-rm calcul *.o *~

End Makefile

```

Рис. 4.7: Создание Makefile

6. С помощью `gdb` выполнил отладку программы `calcul` (перед использованием `gdb` исправил Makefile)(рис. 4.8

```
Терминал - sozjyas@dk5n52:~/work/os/lab_prog
Файл Правка Вид Терминал Вкладки Справка
sozjyas@dk5n52 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c Makefile
sozjyas@dk5n52 ~/work/os/lab_prog $ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
sozjyas@dk5n52 ~/work/os/lab_prog $ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
sozjyas@dk5n52 ~/work/os/lab_prog $
```

Рис. 4.8: Выполнение отладки программы calcul

- Запустил отладчик GDB, загрузив в него программу для отладки:(рис. 4.9)

```
Терминал - sozjyas@dk3n60:~/work/os/lab_prog
Файл Правка Вид Терминал Вкладки Справка
sozjyas@dk3n60 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.2 vanilla) 10.2
Copyright (C) 2021 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 in ./calcul)
(gdb)
```

Рис. 4.9: Запуск отладчика GDB

- Запустил программу внутри отладчика (рис. 4.10)

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/o/sozjyas/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
11.00
[Inferior 1 (process 20865) exited normally]
```

Рис. 4.10: Запуск программы внутри отладчика

- Просмотрел исходный код используя команду list (рис. 4.11)

```
(gdb) list
warning: Source file is more recent than executable.
1      // main.c
2
3      #include <stdio.h>
4      #include "calculate.h"
5
6      int main (void)
7      {
8          float Numeral;
9          char Operation[4];
10         float Result;
```

Рис. 4.11: Просмотр исходного кода

- Просмотрел строки с 12 по 15 основного файла (рис. 4.12)

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

Рис. 4.12: Просмотр строк с 12 по 15 основного файла

- Просмотрел определённые строки не основного файла (рис. 4.13)

```
(gdb) list calculate.c:20,29
20     printf("Вычитаемое: ");
21     scanf("%f",&SecondNumeral);
22     return(Numeral - SecondNumeral);
23 }
24 else if(strncmp(Operation, "*", 1) == 0)
25 {
26     printf("Множитель: ");
27     scanf("%f",&SecondNumeral);
28     return(Numeral * SecondNumeral);
29 }
(gdb)
```

Рис. 4.13: Просмотр определённых строк не основного файла

- Установил точку останова в файле calculate.c на строке номер 21: (рис. 4.14)

```
(gdb) b 21
Breakpoint 1 at 0x1252: file calculate.c, line 21.
(gdb)
```

Рис. 4.14: Установка точки останова в файле calculate.c

- Вывел информацию об имеющихся в проекте точка останова: (рис. 4.15)


```
(gdb) i b
Num Type      Disp Enb Address      What
1 breakpoint keep y 0x0000000000001252 in Calculate at calculate.c:21
(gdb)
```

Рис. 4.15: Вывод информации о точке останова

- Запустил программу внутри отладчика и убедился, что программа остановилась в момент прохождения точки останова: (рис. 4.16)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/o/sozjyas/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffff0c4 "-") at calculate.c:21
21 scanf("%f",&SecondNumeral);
(gdb)
```

Рис. 4.16: Запуск программы внутри отладчика при наличии точки останова

- backtrace (рис. 4.17)

```
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffff0c4 "-") at calculate.c:21
#1 0x0000555555555566 in main () at main.c:15
(gdb)
```

Рис. 4.17: backtrace

- Посмотрел, чему равно на этом этапе значение переменной Numeral (рис. 4.18)

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

Рис. 4.18: Значение переменной Numeral на этом этапе

- Сравнил с результатом вывода на экран (рис. 4.19)

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

Рис. 4.19: Сравнение с результатом вывода на экран после использования команды display

- Убрал точку останова: (рис. 4.20)

```
(gdb) delete 1
(gdb) i b
No breakpoints or watchpoints.
```

Рис. 4.20: Удаление точки останова

7. С помощью утилиты splint попробуйте проанализировать коды файлов:

- calculate.c (рис. 4.21)

```
Numeral, char Operation[4])
{
    if (Operation[0] != '+' || Operation[1] != '\0')
        return 0;
    if (Operation[0] == '+')
        SecondNumeral = SecondNumeral + FirstNumeral;
    else
        return 0;
    printf("Результат: %d\n", SecondNumeral);
    return 1;
}

int main()
{
    char Operation[4];
    int FirstNumeral, SecondNumeral;
    printf("Введите первое число: ");
    scanf("%d", &FirstNumeral);
    printf("Введите операцию: ");
    scanf("%s", Operation);
    calculate(FirstNumeral, Operation);
    return 0;
}
```

```
sozjyas@dk3n60 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:41: 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:9:37: Function parameter Operation declared as manifest array (size
constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:15:2: Return value (type int) ignored: scanf("%s", Operation);
```

Рис. 4.21: Анализ кода файла calculate.c

- и main.c. (рис. 4.22)

```
sozjyas@dk3n60 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:41: 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:12:5: Return value (type int) ignored: scanf("%f", &Numeral);
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:14:5: Return value (type int) ignored: scanf("%s", Operation);
Finished checking --- 3 code warnings
```

Рис. 4.22: Анализ кода файла main.c

5 Выводы

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

6 Контрольные вопросы

1. Получить информацию о возможностях программ gcc, make, gdb и др. можно с помощью команд man.
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 в контексте языка программирования означает расширение файла программы.

- Пример использования - `gcc -c main.c`

4. Компилятор языка C в UNIX предназначен для сборки разрабатываемой программы, написанной на языке C.

5. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

6. Пример структуры Makefile

- `hello: main.c`
- `gcc -o hello main.c`

7. Любой отладчик имеет способность искать и устранять ошибки в программе. Для его использования необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле.

8. Основные команды GDB

Команда	Описание действия
<code>backtrace</code>	вывод на экран пути к текущей точке останова (по сути вывод названий всех функций)
<code>break</code>	установить точку останова (в качестве параметра может быть указан номер строки или название функции)
<code>continue</code>	продолжить выполнение программы
<code>info breakpoints</code>	вывести на экран список используемых точек останова
<code>delete</code>	удалить точку останова
<code>list</code>	вывести на экран исходный код

Команда	Описание действия
run	запуск программы на выполнение

9. Опысание схемы отладки

- calcul: calculate.o main.o
 - gcc calculate.o main.o -o calcul \$(LIBS) команда для создания исполняемого файла calcul
- calculate.o: calculate.c calculate.h
 - gcc -c calculate.c \$(CFLAGS) команда для создания объектного файла calculate.o
- main.o: main.c calculate.h
 - gcc -c main.c \$(CFLAGS) команда для создания объектного файла main.o

10. Он обнаруживает синтаксические и семантические ошибки.

11. Основные средства, повышающие понимание исходного кода программы - GDB и splint.

12. Splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.

Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016.
URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.