

Лабораторная работа № 14

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

Маметкадыров Ынтымак

Цель работы

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

Выполнение лабораторной работы

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

```
[itmametskadihrov@itmametskadihrov ~]$ cd work
[itmametskadihrov@itmametskadihrov work]$ mkdir os
[itmametskadihrov@itmametskadihrov work]$ cd os
[itmametskadihrov@itmametskadihrov os]$ mkdir lab_prog
[itmametskadihrov@itmametskadihrov os]$ cd lab_prog
[itmametskadihrov@itmametskadihrov lab_prog]$ █
```

Рис. 1. Создание каталогов

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

```
[itmametskadihrov@itmametskadihrov lab_prog]$ touch calculate.h calculate.c main.c
[itmametskadihrov@itmametskadihrov lab_prog]$ ls
calculate.c calculate.h main.c
[itmametskadihrov@itmametskadihrov lab_prog]$ █
```

Рис. 2. Создание файлов

3. Реализовали примитивнейший калькулятор (рис. 3), 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.

File Edit Options Buffers Tools C Help



```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"
float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strncmp(Operation, "+", 1) == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strncmp(Operation, "-", 1) == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strncmp(Operation, "*", 1) == 0)
    {
        printf("Множитель: ");
        scanf("%f",&SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strncmp(Operation, "/", 1) == 0)
    {
        printf("Делитель: ");
        scanf("%f",&SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка: деление на ноль! ");
            return(HUGE_VAL);
        }
        else
            return(Numeral / SecondNumeral);
    }
}
```

U: --- **calculate.c** Top L33 (C/L Abbrev)

[Emacs Tutorial](#) Learn basic keystroke commands (Учебник Emacs)

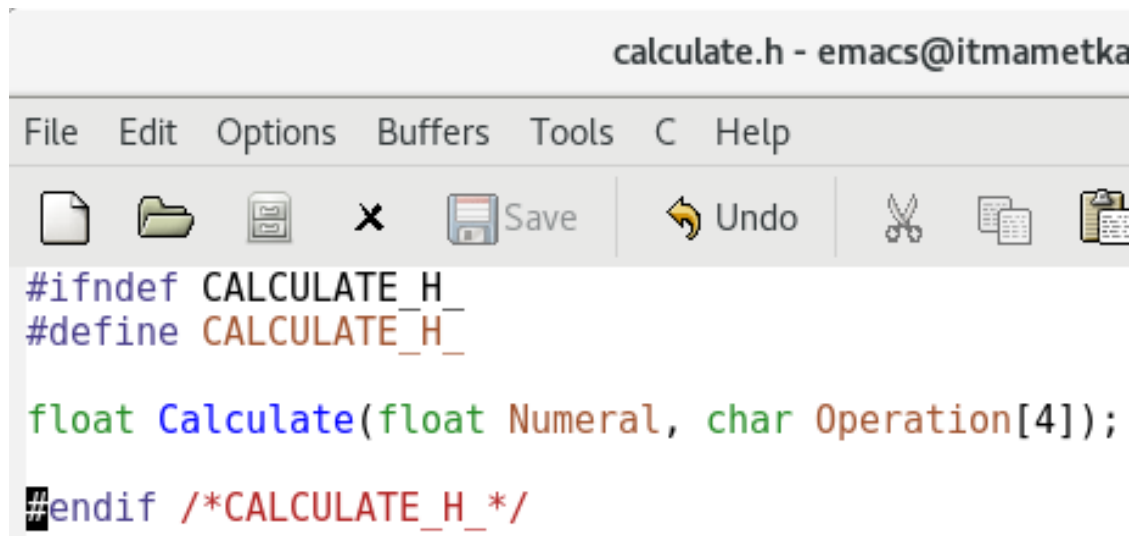
[Emacs Guided Tour](#) Overview of Emacs features at gnu.org

U: %%- *GNU Emacs* 9% L3 (Fundamental)

```
        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. 3. Создание файлов

4. Написали код для интерфейсного файла calculate.h, описывающий формат вызова функции калькулятора (рис. 4).



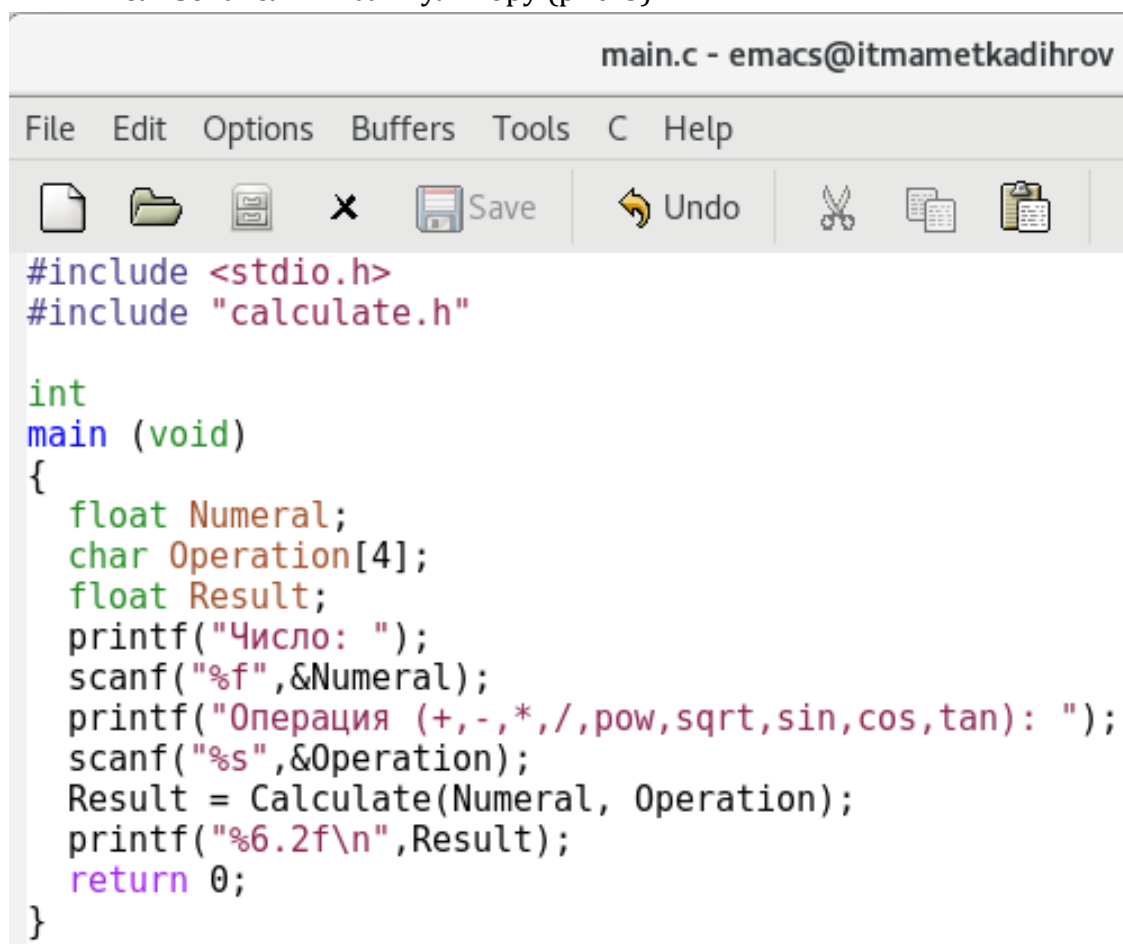
```
calculate.h - emacs@itmametka
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Close, Undo, Cut, Copy, Paste]
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 4. Интерфейсный файл *calculate.h*

5. Написали код для основного файла *main.c*, реализующий интерфейс пользователя к калькулятору (рис. 5).



```
main.c - emacs@itmametakadihrov
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Close, Undo, Cut, Copy, Paste]
#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;
}
```

Рис. 5. Основной файл *main.c*

6. Выполнили компиляцию программы посредством gcc (рис. 6).

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

Рис. 6. Компиляция программы

7. Создали Makefile (рис. 7). Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления 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.

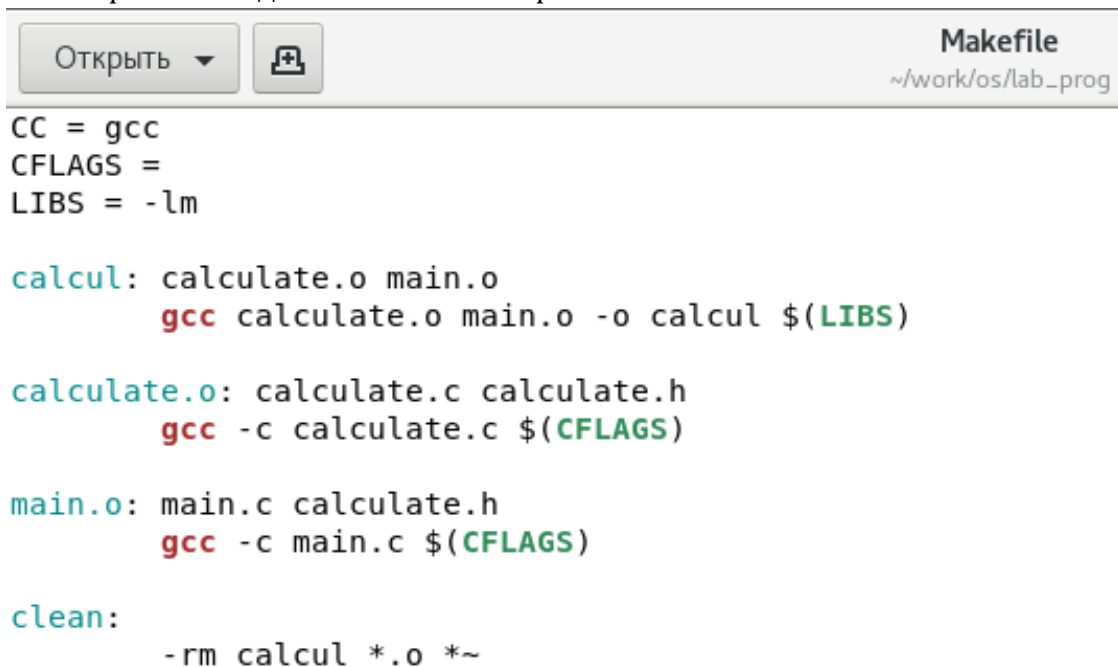
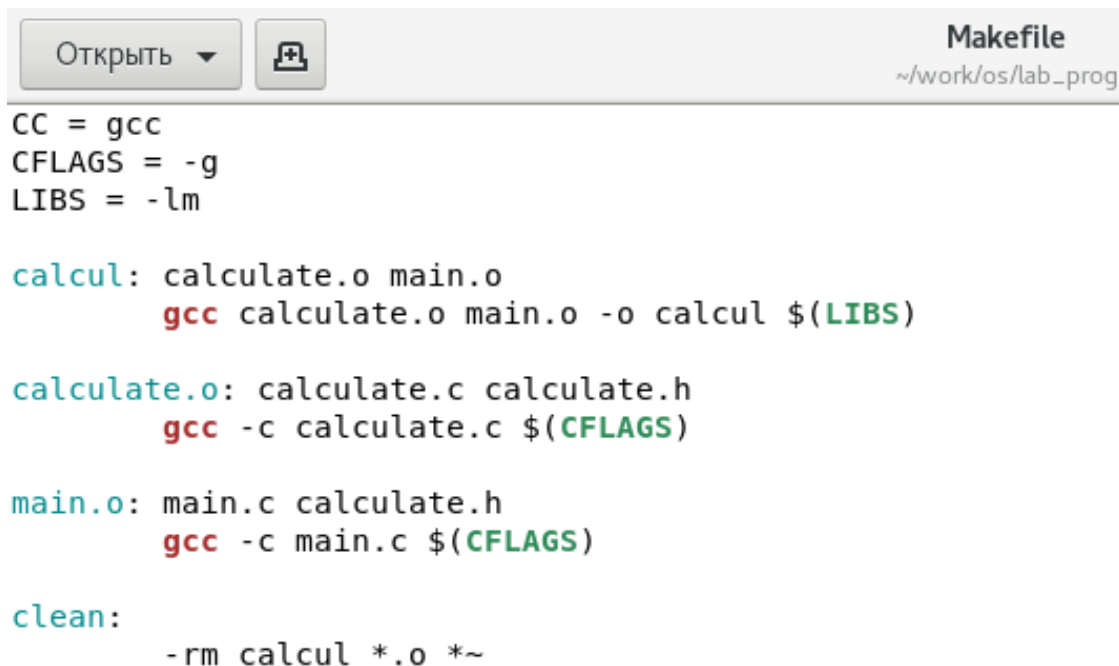


Рис. 7. Makefile

8. В Makefile переменной CFLAGS присвоили опцию -g, для компиляции объектных файлов и их использования в программе отладчика GDB. Сделали так, что утилита компиляции выбирается с помощью переменной CC (рис. 8).



```

Открыть ▼
Makefile
~/work/os/lab_prog

CC = gcc
CFLAGS = -g
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)

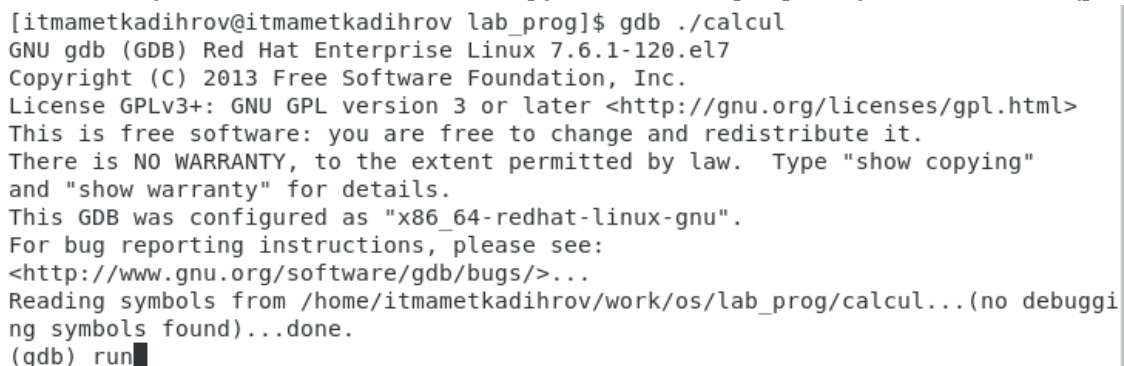
main.o: main.c calculate.h
    gcc -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

```

Рис. 8. Новый Makefile

9. Запустили отладчик GDB, загрузив в него программу для отладки (рис. 9).



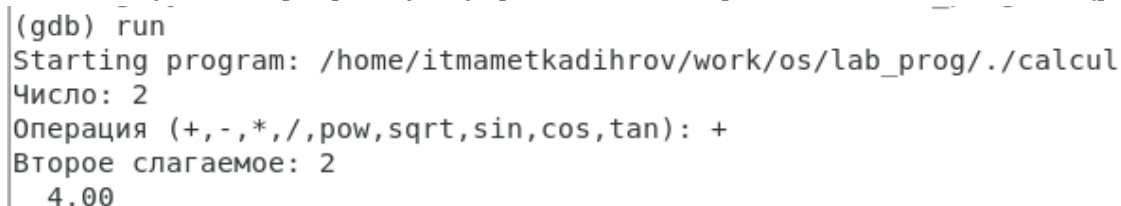
```

[itmametkadihrov@itmametkadihrov lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/itmametkadihrov/work/os/lab_prog/calcul...(no debuggi
ng symbols found)...done.
(gdb) run

```

Рис. 9. Отладчик GDB

10. Запустили программу внутри отладчика при помощи команды run (рис. 10).



```

(gdb) run
Starting program: /home/itmametkadihrov/work/os/lab_prog/./calcul
Число: 2
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 2
4.00

```

Рис. 10. Запуск программы внутри отладчика

11. Для постраничного просмотра исходного код использовали команду list (рис. 11).

```
(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) █
```

Рис. 11. Просмотр командой *list*

12. Для просмотра строк с 12 по 15 основного файла использовали *list* с параметрами 12 и 15 (рис. 12).

```
(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);
```

Рис. 12. Просмотр командой *list* строк с 12 по 15

13. Для просмотра определённых строк не основного файла использовали *list* с параметрами *calculate.c:20,29* (рис. 13).

```
(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)
28         {
29             printf("Делитель: ");
(gdb) █
```

Рис. 13. Просмотр командой *list calculate.c:20,29*

14. Установили точку останова в файле calculate.c на строке номер 21 (рис. 14).

```
(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 0x400810: file calculate.c, line 21.
```

Рис. 14. Установка точки останова

15. Вывели информацию об имеющихся в проекте точка останова (рис. 15).

```
Breakpoint 1 at 0x400810: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type           Disp Enb Address                What
1        breakpoint     keep y   0x0000000000400810    in Calculate
                                     at calculate.c:21
```

Рис. 15. Точки останова

16. Запустили программу внутри отладчика и убедились, что программа остановится в момент прохождения точки останова (рис. 16).

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/itmametkadihrov/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdc70 "-")
  at calculate.c:21
21         printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdc70 "-") at calculate.c:21
#1 0x0000000000400a90 in main () at main.c:17
(gdb) █
```

Рис. 16. Остановка в точке останова

17. Отладчик выдал следующую информацию:

#0 Calculate (Numeral=5, Operation=0x7fffffd280 "-") at calculate.c:21

#1 0x0000000000400b2b in main () at main.c:17

а команда backtrace показала весь стек вызываемых функций от начала программы до текущего места (рис. 16).

18. Посмотрели, чему равно на этом этапе значение переменной Numeral, введя print Numeral (рис. 17), на экран выведено число 5.

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

Рис. 17. print Numeral

19. Сравнили с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (рис. 17).
20. Убрали точки останова (рис. 18).

```
(gdb) info breakpoints
Num    Type           Disp Enb Address              What
1      breakpoint     keep y   0x00000000004007d8  in Calculate
                                           at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
```

Рис. 18. Удаление точек останова

21. С помощью утилиты splint попробовали проанализировать коды файлов calculate.c (рис. 20) и main.c (рис. 21). С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохраняются. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потере данных.

```
[itmametskadihrov@itmametskadihrov lab_prog]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015
```

```
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)

```
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:38:10: 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
```

```
[itmametskadihrov@itmametskadihrov lab_prog]$ █
```

Puc. 20. splint calculate.c

```
[itmametskadihrov@itmametskadihrov lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

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:14:3: 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:16:14: 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:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[itmametskadihrov@itmametskadihrov lab_prog]$ █
```

Рис. 21. splint main.c

Вывод

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.

Ответы на контрольные вопросы

1. Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.
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.
3. 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 -c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль – файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».
 4. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
 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. Для работы с утилитой make необходимо в корне рабочего каталога с Вашим 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.

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.

Пример более сложного синтаксиса Makefile:

```
#  
# Makefile for abcd.c  
  
#  
CC = gcc  
CFLAGS =  
# Compile abcd.c normaly  
abcd: abcd.c  
  
    $(CC) -o abcd $(CFLAGS) abcd.c  
  
clean: -rm abcd.o ~  
  
# End Makefile for abcd.c
```

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. 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

результатирующем 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
```

8. Основные команды 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.

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`.

9. Схема отладки программы показана выше в настоящей лабораторной работе.
10. При первом запуске компилятор не выдал никаких ошибок, но в коде программы `main.c` допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке `scanf("%s", &Operation);` нужно убрать знак `&`, потому что имя массива символов уже является указателем на первый элемент этого массива.
11. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:
 - `cscope` – исследование функций, содержащихся в программе,
 - `lint` – критическая проверка программ, написанных на языке Си.
12. Утилита `splint` 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.