

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

дисциплина: Операционные системы

Студент: Хосе Фернадо Леон Атупанья

Группа: НПМбд-02-20

МОСКВА 2021 г.

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.

Задания

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`:
`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`. 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

- Для запуска программы внутри отладчика введите команду `run`
 - Для страничного (по 10 строк) просмотра исходного код используйте команду `list`
 - Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами
 - Для просмотра определённых строк не основного файла используйте `list` с параметрами
 - Установите точку останова в файле `calculate.c` на строке номер 21
 - Выведите информацию об имеющихся в проекте точка останова
 - Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова
 - Отладчик выдаст информацию, а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места
 - Посмотрите, чему равно на этом этапе значение переменной `Numeral`. На экран должно быть выведено число 5
 - Сравните с результатом вывода на экран после использования команды
 - Уберите точки останова
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

Выполнение лабораторной работы

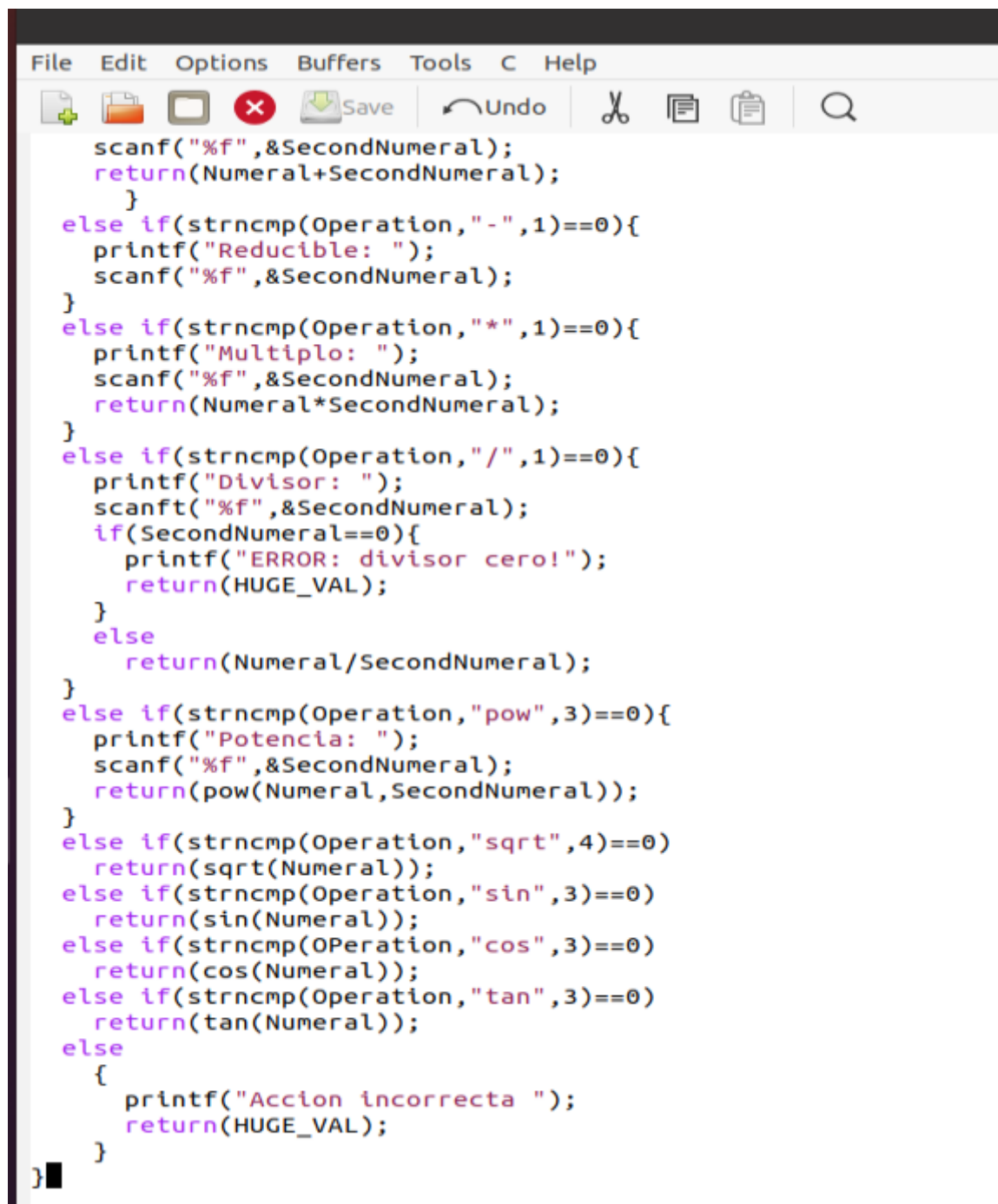
1. В домашнем каталоге создаю подкаталог `calculate` с помощью команды «`mkdir calculate`».
2. Создал в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`, используя команды «`cd calculate`» и «`touch calculate.h calculate.c main.c`» (рис. -@fig:001).

```
fernando@fernando-VirtualBox:~$ cd labor2
fernando@fernando-VirtualBox:~/labor2$ cd 2020-2021
fernando@fernando-VirtualBox:~/labor2/2020-2021$ cd 05
fernando@fernando-VirtualBox:~/labor2/2020-2021/05$ mkdir laboratory14
fernando@fernando-VirtualBox:~/labor2/2020-2021/05$ cd laboratory14
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14$ mkdir calculate
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14$ ls
calculate
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14$ cd calculate
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ touch calculate.h calculate.c main.c
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ ls
calculate.c calculate.h main.c
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$
```

{ #fig:001 width=70% }

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.

Открыв редактор Emacs, приступил к редактированию созданных файлов. Реализация функций калькулятора в файле `calculate.c` (рис. -@fig:002).

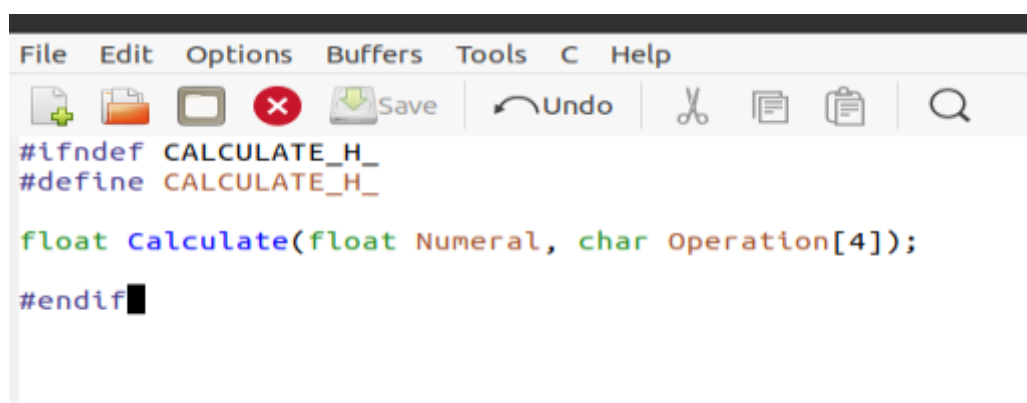


```
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]

scanf("%f",&SecondNumeral);
return(Numeral+SecondNumeral);
}
else if(strncmp(Operation,"-",1)==0){
printf("Reducible: ");
scanf("%f",&SecondNumeral);
}
else if(strncmp(Operation,"*",1)==0){
printf("Multiplio: ");
scanf("%f",&SecondNumeral);
return(Numeral*SecondNumeral);
}
else if(strncmp(Operation,"/",1)==0){
printf("Divisor: ");
scanf("%f",&SecondNumeral);
if(SecondNumeral==0){
printf("ERROR: divisor cero!");
return(HUGE_VAL);
}
else
return(Numeral/SecondNumeral);
}
else if(strncmp(Operation,"pow",3)==0){
printf("Potencia: ");
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("Accion incorrecta ");
return(HUGE_VAL);
}
}
```

{ #fig:002 width=70% }

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис. -@fig:003).



```
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]

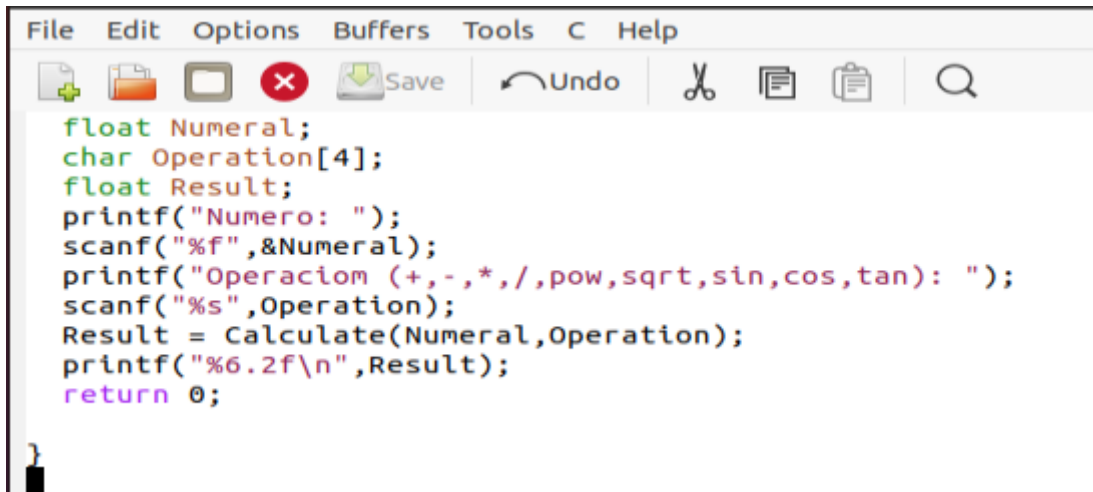
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif
```

{ #fig:003 width=70% }

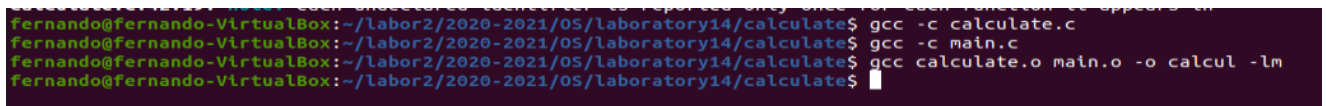
Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. -@fig:004).



```
File Edit Options Buffers Tools C Help
float Numeral;
char Operation[4];
float Result;
printf("Numero: ");
scanf("%f",&Numeral);
printf("Operaciom (+, -, *, /, pow, sqrt, sin, cos, tan): ");
scanf("%s",Operation);
Result = Calculate(Numeral,Operation);
printf("%6.2f\n",Result);
return 0;
}
```

{ #fig:004 width=70% }

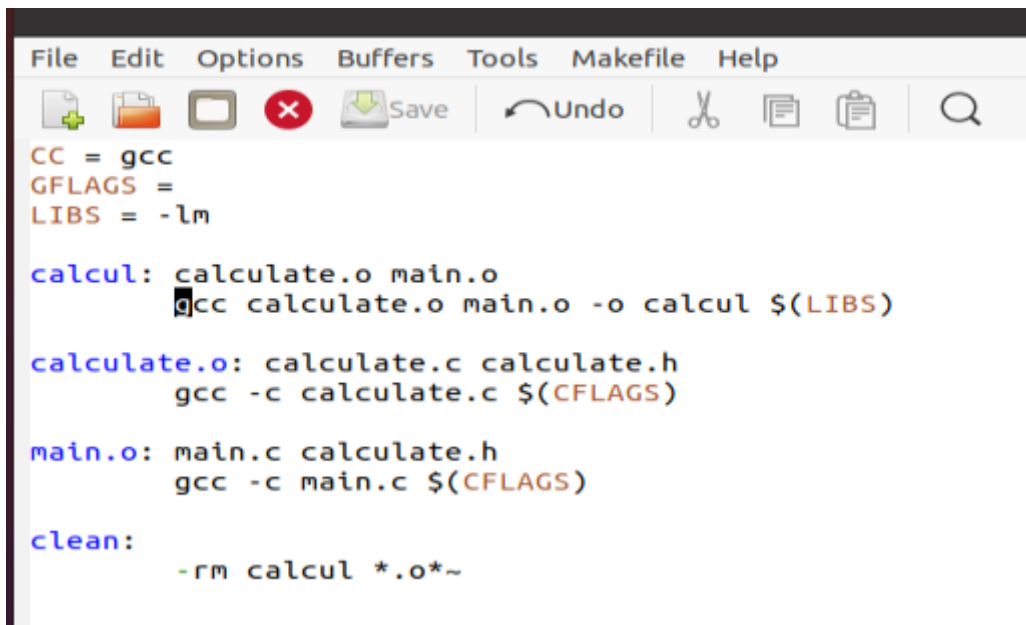
3. Выполнил компиляцию программы посредством gcc, используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (рис. -@fig:005).



```
Fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ gcc -c calculate.c
Fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ gcc -c main.c
Fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ gcc calculate.o main.o -o calcul -lm
Fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$
```

{ #fig:005 width=70% }

4. В ходе компиляции программы никаких ошибок выявлено не было.
5. Создал Makefile с необходимым содержанием (рис. -@fig:006). 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.



```
File Edit Options Buffers Tools Makefile Help
+ Save Undo
CC = gcc
GFLAGS =
LIBS = -lm

calcul: calculate.o main.o
    gcc calculate.o main.o -o calcul $(LIBS)

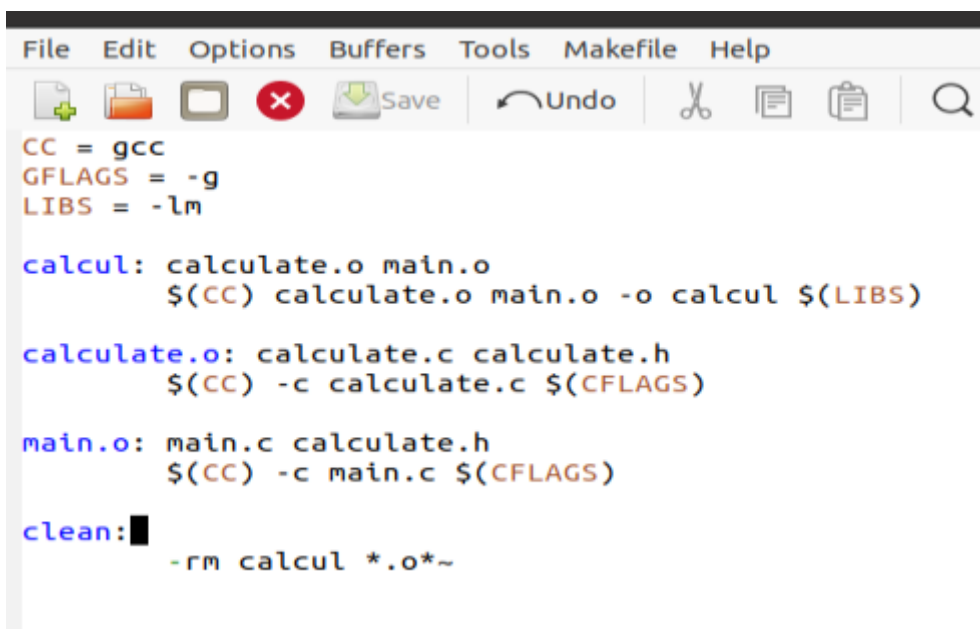
calculate.o: calculate.c calculate.h
    gcc -c calculate.c $(CFLAGS)

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

clean:
    -rm calcul *.o*~
```

{ #fig:006 width=70% }

6. Далее исправил Makefile (рис. -@fig:007). В переменную CFLAGS добавил опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделал так, что утилита компиляции выбирается с помощью переменной CC.



```
File Edit Options Buffers Tools Makefile Help
+ Save Undo
CC = gcc
GFLAGS = -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*~
```

{ #fig:007 width=70% }

После этого я удалил исполняемые и объектные файлы из каталога с помощью команды «make clean». Выполнил компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (рис. -@fig:008).

```
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ make clean
rm calcul *.o *~
rm: cannot remove 'calcul': No such file or directory
make: [Makefile:15: clean] Error 1 (ignored)
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ make calculate.o
gcc -c calculate.c
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ make main.o
gcc -c main.c
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ make calcul
gcc calculate.o main.o -o calcul -lm
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ █
```

{ #fig:008 width=70% }

Далее с помощью gdb выполнил отладку программы calcul. Запустил отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul» (рис. -@fig:009).

```
gcc calculate.o main.o -o calcul -lm
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory14/calculate$ gdb ./calcul
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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...
(No debugging symbols found in ./calcul)
(gdb) █
```

{ #fig:009 width=70% }

Для запуска программы внутри отладчика ввёл команду «run» (рис. -@fig:010).

```
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/fernando/labor2/2020-2021/05/laboratory14/calculate/calcul
Numero: 7
Operacion (+,-,*,/,pow,sqrt,sin,cos,tan): +
Segundo termino: 15
22.00
[Inferior 1 (process 2904) exited normally]
(gdb) █
```

{ #fig:010 width=70% }

7. С помощью утилиты splint проанализировал коды файлов calculate.c и main.c. Воспользовался командами «splint calculate.c» и «splint main.c» (рис. -@fig:011) (рис. -@fig:012).

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.

```
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/fernando/labor2/2020-2021/OS/laboratory14/calculate/calcul
Numero: 7
Operacion (+,-,*,/,pow,sqrt,sin,cos,tan): +
Segundo termino: 15
22.00
[Inferior 1 (process 2904) exited normally]
(gdb) █
```

{ #fig:010 width=70% }

```
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory14/calculate$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

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:11:5: 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:16:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:20:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:26:8: 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:28:13: Return value type double does not match declared type float:
    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:35:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:36:11: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:39:11: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:41:11: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:43:11: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:45:11: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:49:13: Return value type double does not match declared type float:
    (HUGE_VAL)
calculate.c:51:2: Path with no return in function declared to return float
    There is a path through a function declared to return a value on which there
    is no return statement. This means the execution may fall through without
    returning a meaningful result to the caller. (Use -noret to inhibit warning)

Finished checking --- 16 code warnings
█
```

{ #fig:011 width=70% }

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.

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. 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:
`<цель_1> <цель_2> ... : <зависимость_1> <зависимость_2> ...`
`<команда 1>`

...

`<команда n>`

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.

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 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
```

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 в 6 пункте лабораторной работы.
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 работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

Выводы

В ходе этой лабораторной работы я приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в операционных системах UNIX/Linux, создав язык программирования с калькулятором с простейшими функциями.

