

Отчёт по лабораторной работе №12

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

Куликов Александр Андреевич

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.

Задание

1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`. Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять `sin`, `cos`, `tan`. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.
3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со следующим содержанием.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`)
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

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.

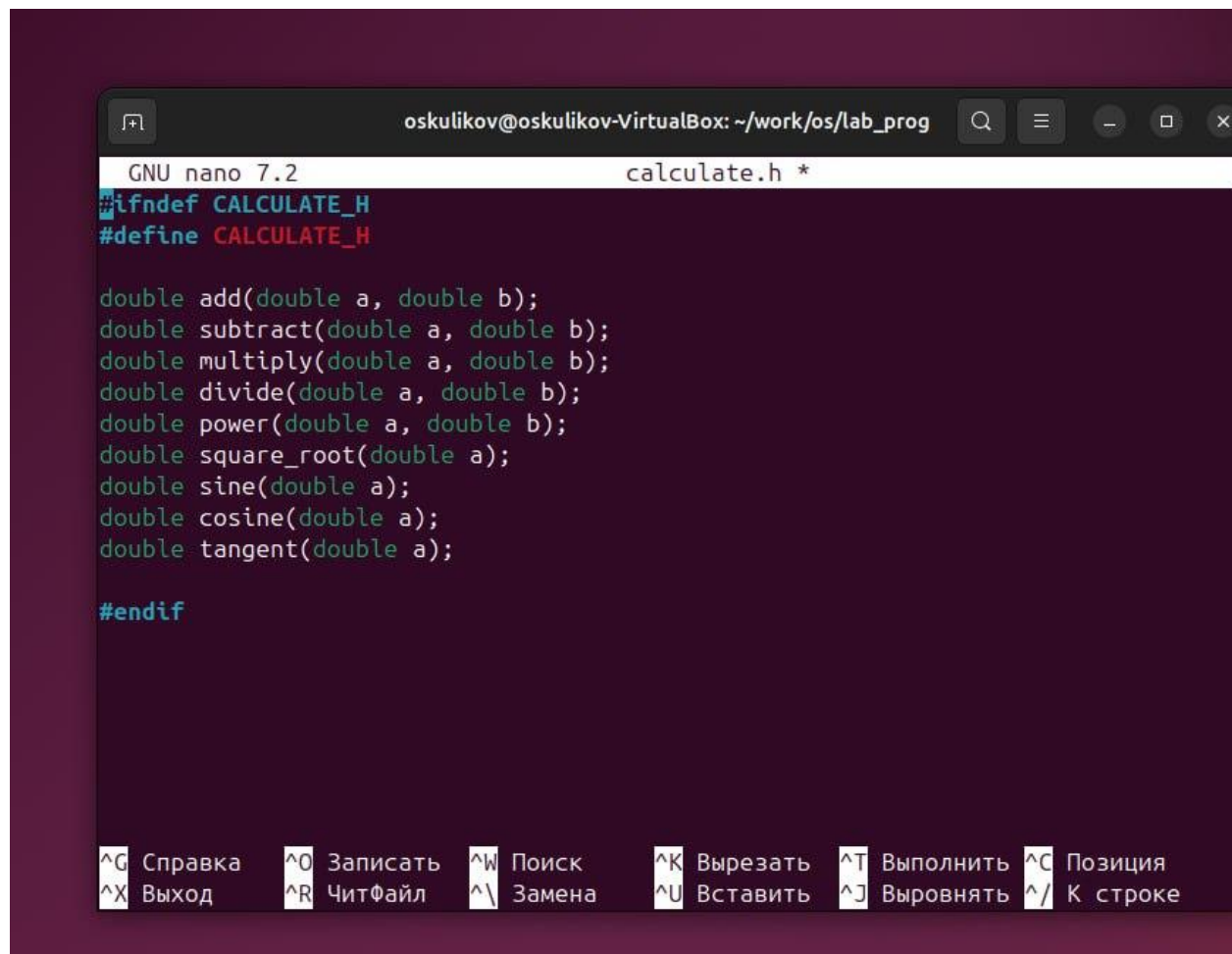
После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла.

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.

Выполнение лабораторной работы

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog. (рис. @fig:001)



```
oskulikov@oskulikov-VirtualBox: ~/work/os/lab_prog
GNU nano 7.2 calculate.h *
#ifndef CALCULATE_H
#define CALCULATE_H

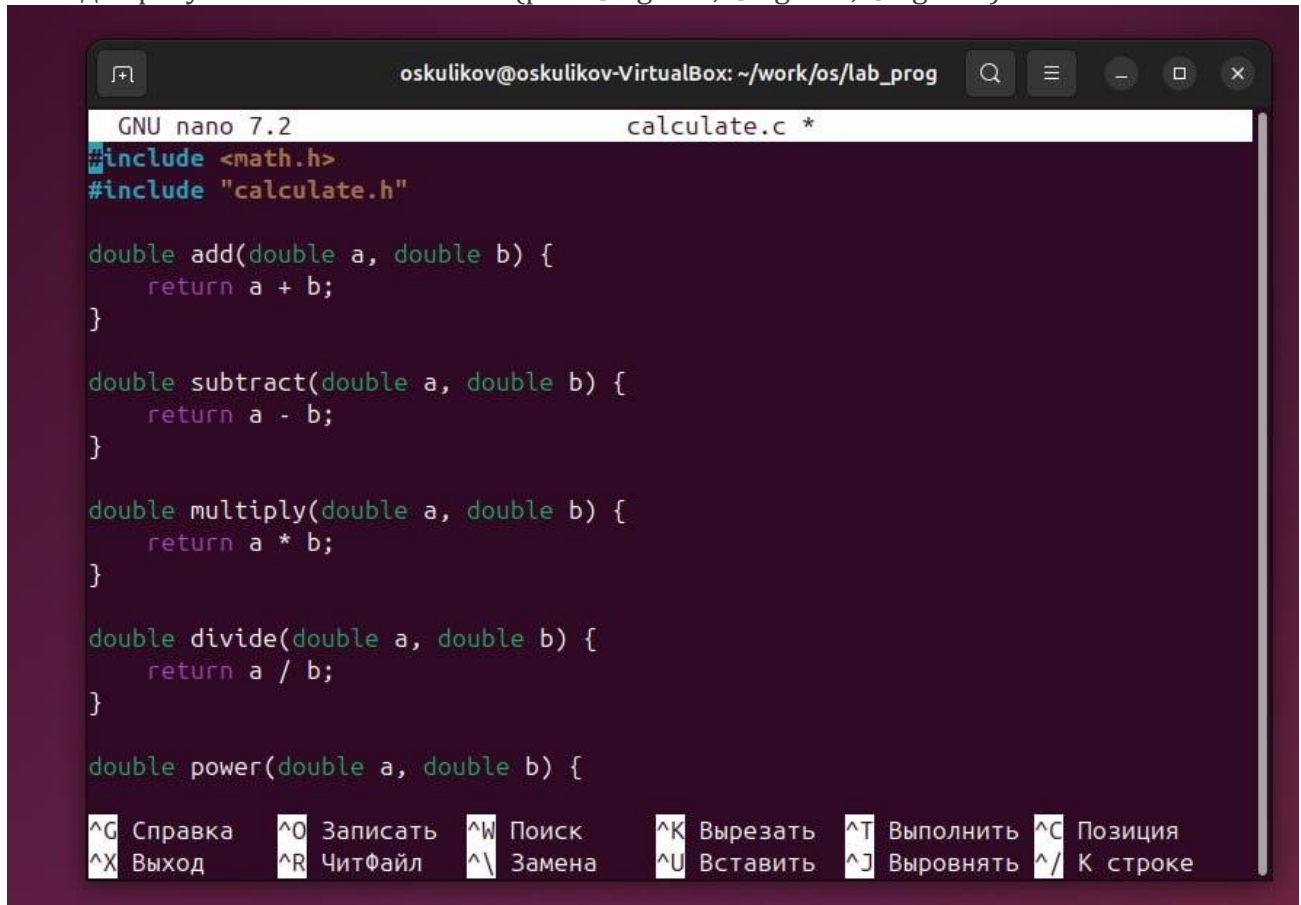
double add(double a, double b);
double subtract(double a, double b);
double multiply(double a, double b);
double divide(double a, double b);
double power(double a, double b);
double square_root(double a);
double sine(double a);
double cosine(double a);
double tangent(double a);

#endif

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^С Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/ К строке
```

- - Создание каталога
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. (рис. @fig:002, @fig:003, @fig:004).



```
oskulikov@oskulikov-VirtualBox: ~/work/os/lab_prog
GNU nano 7.2 calculate.c *
#include <math.h>
#include "calculate.h"

double add(double a, double b) {
    return a + b;
}

double subtract(double a, double b) {
    return a - b;
}

double multiply(double a, double b) {
    return a * b;
}

double divide(double a, double b) {
    return a / b;
}

double power(double a, double b) {
```

^G Справка	^O Записать	^W Поиск	^K Вырезать	^T Выполнить	^C Позиция
^X Выход	^R ЧитФайл	^_\ Замена	^U Вставить	^J Выровнять	^/ К строке

```
oskulikov@oskulikov-VirtualBox: ~/work/os/lab_prog
GNU nano 7.2 main.c *
#include <stdio.h>
#include <math.h>
#include "calculate.h"

int main() {
    double num1, num2, result;
    char op;

    printf("Введите первое число: ");
    scanf("%lf", &num1);

    printf("Введите операцию (+, -, *, /, ^, s (sqrt), S (sin), C (cos), T (tan)");
    scanf(" %c", &op);

    if (op == 's') {
        result = square_root(num1);
    } else if (op == 'S') {
        result = sine(num1);
    } else if (op == 'C') {
        result = cosine(num1);
    }

    printf("Результат: %lf\n", result);
}
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/_ К строке

-
- CC = gcc
- CFLAGS = -Wall -g

LDFLAGS = -lm

TARGET = calcul

OBJ = main.o calculate.o

all: \$(TARGET)

\$(TARGET): \$(OBJ)

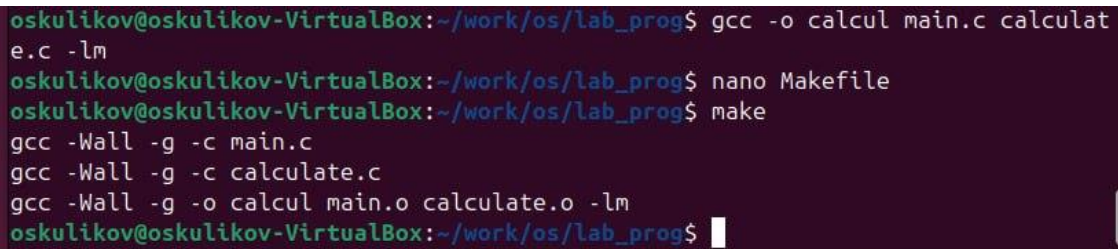
\$(CC) \$(CFLAGS) -o \$@ \$^ \$(LDFLAGS)

%.o: %.c

\$(CC) \$(CFLAGS) -c \$<

clean:

rm -f \$(TARGET) \$(OBJ)

A terminal window with a dark background and light-colored text. The prompt is 'oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog\$'. The user enters 'gcc -o calcul main.c calculate.o -lm', which is followed by a new line. Then the user enters 'nano Makefile', followed by a new line. Then the user enters 'make', which is followed by a new line. The output shows three lines of compilation commands: 'gcc -Wall -g -c main.c', 'gcc -Wall -g -c calculate.c', and 'gcc -Wall -g -o calcul main.o calculate.o -lm'. Finally, the prompt 'oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog\$' is shown again with a cursor.

```
oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog$ gcc -o calcul main.c calculate.o -lm
oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog$ nano Makefile
oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog$ make
gcc -Wall -g -c main.c
gcc -Wall -g -c calculate.c
gcc -Wall -g -o calcul main.o calculate.o -lm
oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog$
```

```
oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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...
```

```
Обрабатываются триггеры для libc-bin (2.39-0ubuntu8.2) ...
oskulikov@oskulikov-VirtualBox:~/work/os/lab_prog$ splint calculate.c
splint main.c
Splint 3.1.2 --- 21 Feb 2021

Finished checking --- no warnings
Splint 3.1.2 --- 21 Feb 2021

main.c: (in function main)
main.c:10:5: Return value (type int) ignored: scanf("%lf", &num1)
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
main.c:13:5: Return value (type int) ignored: scanf(" %c", &op)
main.c:25:9: Return value (type int) ignored: scanf("%lf", &num2)

Finished checking --- 3 code warnings
```

Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
 - с помощью функций info и man.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.
 - Unix поддерживает следующие основные этапы разработки приложений:
 - 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. Что такое суффикс в контексте языка программирования? Приведите примеры использования.
- В контексте языков программирования, суффикс - это часть имени переменной или значения, которая добавляется к базовому имени или значению, чтобы указать определенный тип, формат или свойство.
 - Примеры использования суффиксов:
 - В C/C++ суффиксы могут использоваться для указания типа литералов. Например, 42 - это целочисленный литерал, а 42.0 - литерал типа с плавающей точкой. В этом случае .0 является суффиксом, указывающим на тип данных. Еще один пример в C/C++ - использование суффиксов f или F для указания типа данных float. Например: 3.14f.
 - В Python суффиксы используются для обозначения размерности числовых литералов. Например, 10_000_000 - это целочисленный литерал, а 10.5 - литерал типа с плавающей точкой. В этом случае .5 является суффиксом. В Python 3.6 и более поздних версиях суффиксы также могут использоваться для указания типа переменных при использовании типовых подсказок. Например, x: int = 10 указывает, что переменная x имеет тип int.
 - В общем, суффиксы в языках программирования используются для явного указания типа данных или свойства значения, что облегчает чтение кода и предотвращает ошибки типов данных.
4. Каково основное назначение компилятора языка C в UNIX?
- Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. Для чего предназначена утилита make?
- При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

#Компилятор

CC=gcc

#Флаги компилятора

CFLAGS=-I.

#Имя исполняемого файла

TARGET=myprogram

#Список исходных файлов

SOURCES=main.c func1.c func2.c

#Генерация объектных файлов

OBJECTS=\$(SOURCES:.c=.o)

#Команда по умолчанию

all: \$(TARGET)

#Сборка исполняемого файла из объектных файлов

\$(TARGET): \$(OBJECTS)

``\$(CC) -o \$(TARGET) \$(OBJECTS)``

#Компиляция каждого исходного файла в объектный файл .c.o:

``\$(CC) \$(CFLAGS) -c \$< -o \$@``

#Очистка временных файлов

clean:

``rm -f \$(OBJECTS) \$(TARGET)``

- В этом примере:
 - Компилятор и флаги компиляции: - CC - переменная, определяющая используемый компилятор (в данном случае gcc). - CFLAGS - переменная, определяющая флаги компилятора (в данном случае -I. добавляет текущий каталог в список каталогов для поиска заголовочных файлов).
- Имя цели (target):
 - TARGET - переменная, определяющая имя исполняемого файла (в данном случае myprogram).
- Список исходных файлов:
 - SOURCES - переменная, содержащая список исходных файлов (в данном случае main.c, func1.c, func2.c).
- Генерация объектных файлов:
 - OBJECTS - переменная, содержащая список объектных файлов, получаемых из исходных файлов (.c файлы заменяются на .o).
- Команда по умолчанию:

- all - указывает, что цель турprogram должна быть создана по умолчанию при вызове make без аргументов.
 - Правила сборки:
 - \$(TARGET) - правило для создания исполняемого файла из объектных файлов.
 - .с.о - правило для компиляции каждого .с файла в объектный .о файл.
 - Очистка временных файлов:
 - clean - правило для удаления временных файлов (используется для очистки проекта от объектных файлов и исполняемого файла).
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
- Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передается ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.
8. Назовите и дайте основную характеристику основным командам отладчика gdb.
- backtrace - выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;
 - break - устанавливает точку останова; параметром может быть номер строки или название функции;
 - clear - удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
 - continue - продолжает выполнение программы от текущей точки до конца;
 - delete - удаляет точку останова или контрольное выражение;
 - display - добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
 - finish - выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
 - info breakpoints - выводит список всех имеющихся точек останова;
 - info watchpoints - выводит список всех имеющихся контрольных выражений;

- splist - выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
 - next - пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;
 - print - выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
 - run - запускает программу на выполнение;
 - set - устанавливает новое значение переменной
 - step - пошаговое выполнение программы;
 - watch - устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
- Выполнили компиляцию программы
 - Не удалось скомпилировать - ошибки
 - Исправили ошибки
 - Запустили отладчик gdb, отладчик выполнил программу, ввели требуемые значения, программа выполнилась без ошибок.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.
- Отладчик выдал ошибку в строке с %s, &Operation, %s - символьный формат, убрали & перед Operation.
11. Назовите основные средства, повышающие понимание исходного кода программы.
- cscope - исследование функций, содержащихся в программе
 - splint - критическая проверка программ, написанных на языке Си.
12. Каковы основные задачи, решаемые программой splint?
- Выявление ошибок в коде: Splint анализирует исходный код на предмет типичных ошибок программирования, таких как доступ к памяти через нулевой указатель, разыменование непроверенного указателя, переполнение буфера, утечки памяти и другие.
 - Поиск потенциальных проблем безопасности: Splint способен обнаруживать уязвимости безопасности, такие как возможные атаки типа "buffer overflow", "format string vulnerabilities", "race conditions" и другие.
 - Проверка соответствия стандартам кодирования: Splint может проверять код на соответствие различным стандартам кодирования и соглашениям о стиле написания кода, таким как MISRA C, POSIX, GNU и другим.
 - Анализ потока управления и данных: Splint проводит анализ потока управления и данных в коде для выявления потенциальных проблем с безопасностью и логикой программы.
 - Поддержка статических типов данных: Splint помогает выявлять ошибки и несоответствия типов данных в программе, что способствует предотвращению некоторых типов ошибок времени выполнения.

- Улучшение читаемости кода: Splint предоставляет рекомендации по улучшению стиля кодирования и читаемости программы, что делает код более понятным и поддерживаемым.

Выводы

В ходе выполнения лабораторной работы я приобрел простейшие навыки разработки и отладки приложений в ОС типа UNIX/Linux на примере создания на языке C калькулятора с простейшими функциями.

Список литературы

1. Практикум по лабораторной работе