Отчёт по лабораторной работе №12

Средства для создания приложений в ОС UNIX

Тагиев Павел Фаикович

Содержание

# 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 [1].

# 2 Задание

1. Провести сборку программы простейшего калькулятора
2. Произвести отладку программы используя gdb
3. Изучить предупреждения линтера splint

# 3 Теоретическое введение

* *GNU Compiler Collection* (обычно используется сокращение GCC) — набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU. GCC является свободным программным обеспечением, распространяется в том числе фондом свободного программного обеспечения (FSF) на условиях GNU GPL и GNU LGPL и является ключевым компонентом GNU toolchain. Он используется как стандартный компилятор для свободных UNIX-подобных операционных систем [2].
* *GNU Debugger* (GDB) — переносимый отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования, включая Си, C++, Free Pascal, FreeBASIC, Ada, Фортран и Rust. GDB — свободное программное обеспечение, распространяемое по лицензии GPL [3].
* *Splint* (Secure Programming Lint) — представляет собой инструмент программирования для статической проверки программ на языке Си на предмет уязвимостей безопасности и ошибок кодирования [4].

# 4 Выполнение лабораторной работы

## 4.1 Набор кода, компиляция и запуск

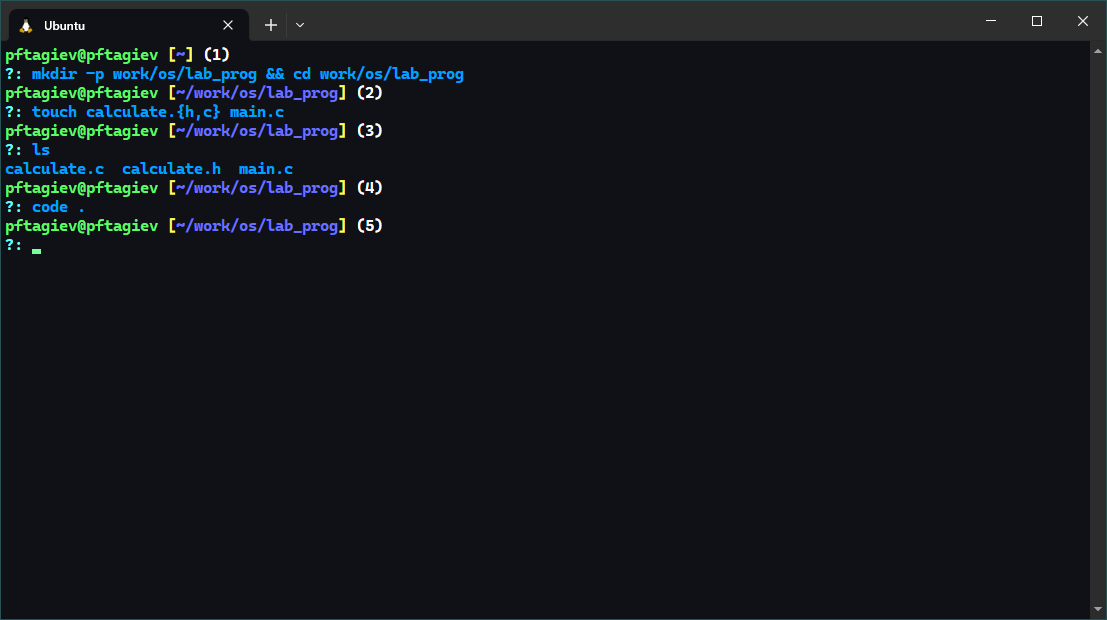


Рис. 1: Создание каталогов и файлов

Создадим каталог ~/work/os/lab\_prog и перейдем в него, как показано на рис. 1 на промпте (1). Затем создадим файлы в которых будет располагаться исходный код нашего калькулятора (промпт (2)). После запустим редактор кода *VSCode* (промпт (4)).

В созданных ранее файлах напишем код простейшего калкулятора, как показано на рис. 2-4.



Рис. 2: Файл calculate.c

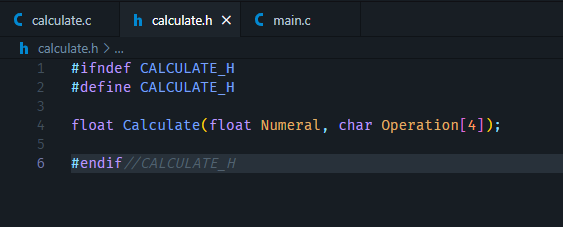


Рис. 3: Файл calculate.h

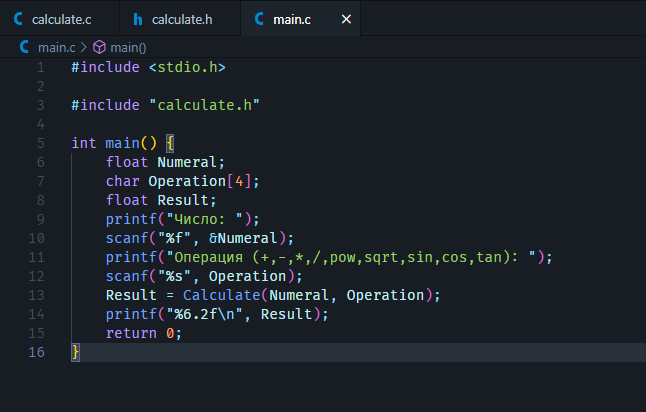


Рис. 4: Файл main.c

Компиляцию набранной программы можно увидеть на рис. 5 в промптах (1)-(3). Проверку собранной программы можно увидеть в промптах (4)-(6), все на том же рис. 5.

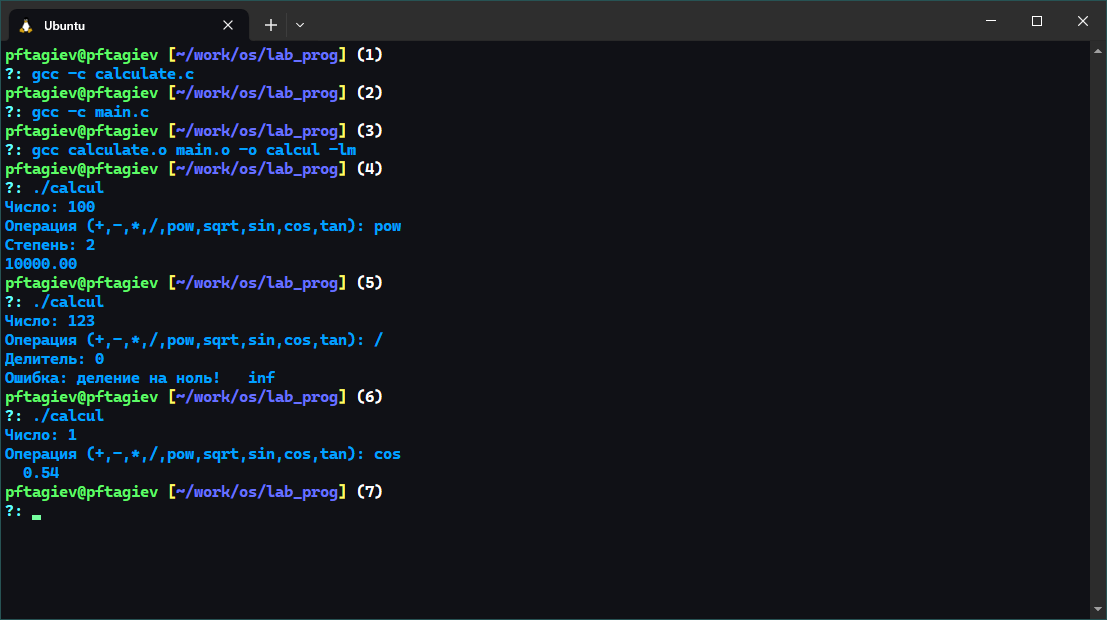


Рис. 5: Компиляция и запуск

## 4.2 Написание и разбор Makefile

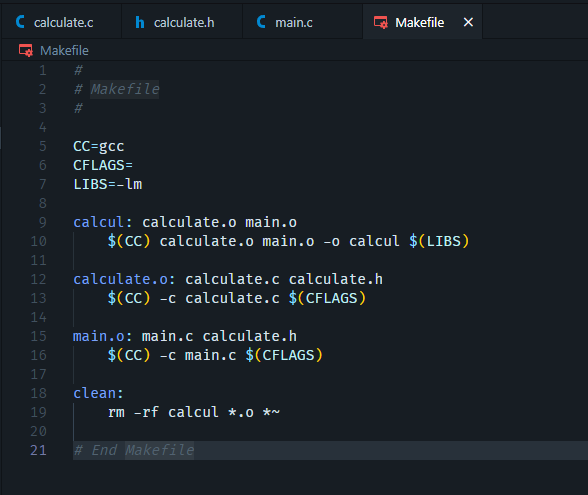


Рис. 6: Написанный Makefile

На рис. 6 можно увидеть написанный Makefile. Давайте разберем его содержимое. В строках *5* - *7* объявляются переменные с помощью которых можно влиять на процесс сборки программы. Например, изменить компилятор (переменная CC) или включить дополнительные опциии (переменные CFLAGS и LIBS).

Далее объявленны таргеты сборки. calculate.o и main.o, создают объектные файлы, а таргет calcul компанует созданные объектные файлы в исполняемый с именем calcul. Последний таргет clean очищает директорию от созданных объектных и временных файлов.

## 4.3 Отладка программы с использованием GDB

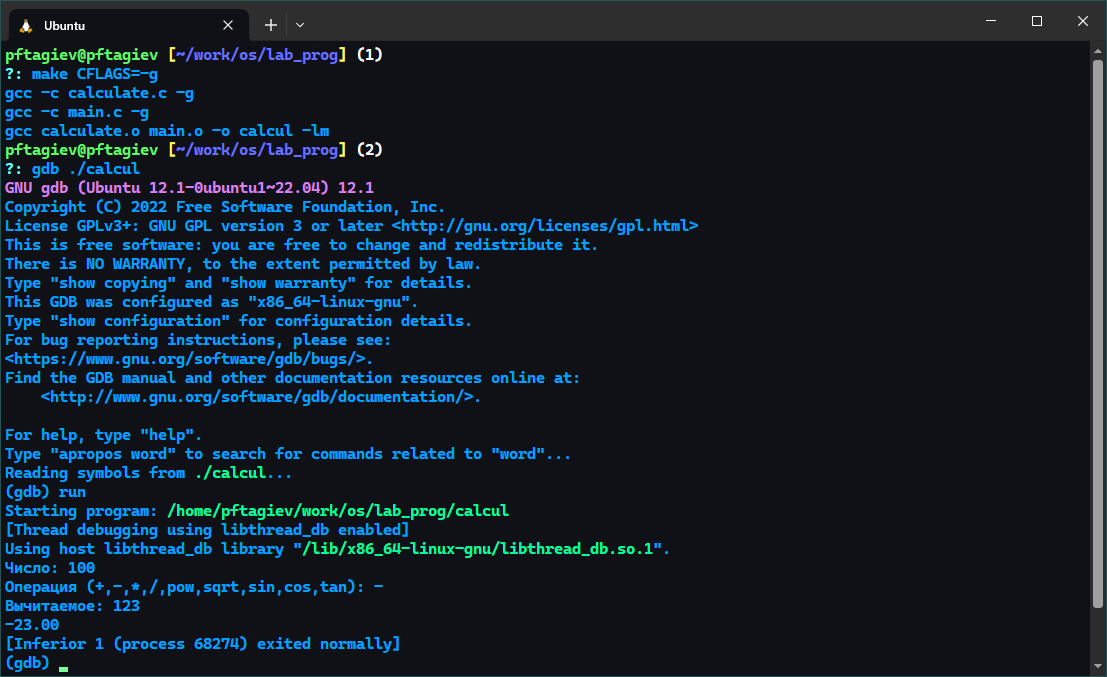


Рис. 7: Запуск отладчика GDB

Соберем написанную программу с флагом -g, чтобы компилятор добавил отладочную информацию. Для этого напишем make CFLAGS=-g, как показано на рис. 7 в промпте (1). Затем запустим отладчик *GDB* передав ему собранную программу. Все на том же рис. 7 можно увидеть использование команды run.

Посмотрим исходный код программы используя команду list в разных вариациях (рис. 8).

Используем точки останова (рис. 9, ориентируйтесь по приглашению оболочки: (gdb)). Выведем кусочек программы командой list, затем установим *breakpoint* на строке *14* (строка *14* соответсвует строке *21* в задании). Выведем информацию о имеющихся точках останова командой info breakpoints. Затем запустим программу командой run. Используя команду backtrace, убедимся что программа остановилась в момент прохождения точки останова.

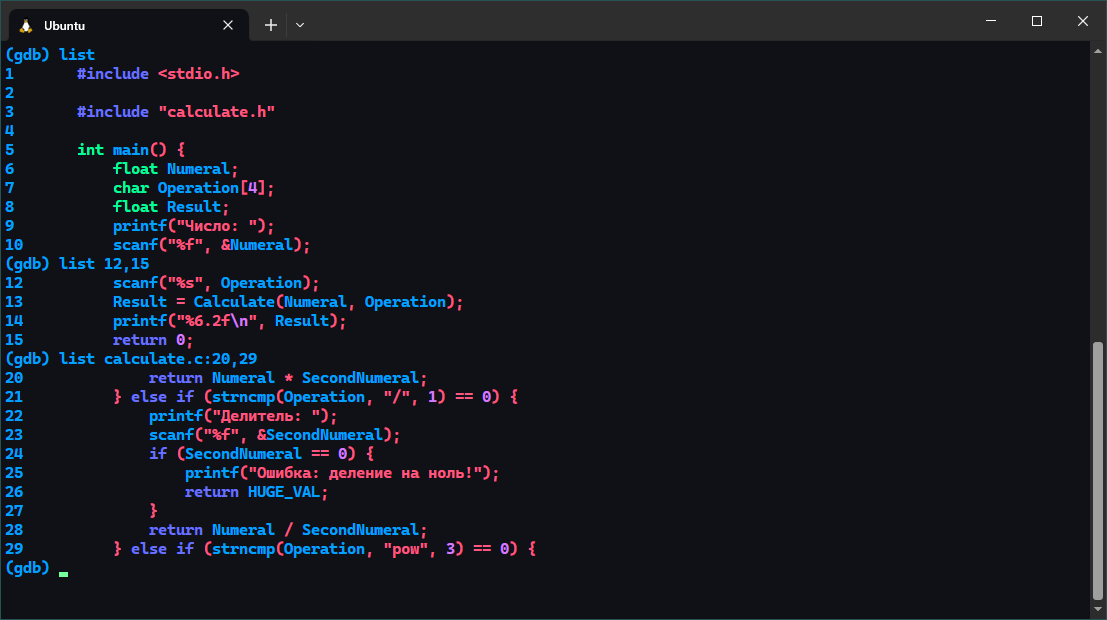


Рис. 8: Команда list

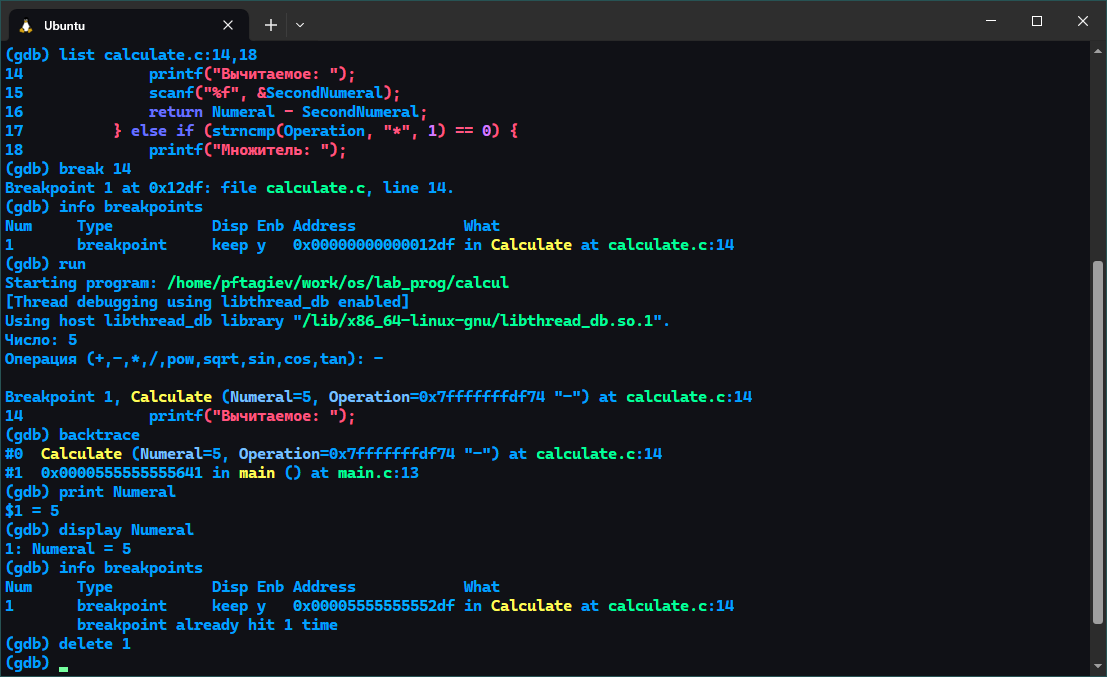


Рис. 9: Точки останова

Теперь выведем значение переменной Numeral на экран, используя команду print, как видно значение переменной равно *5*. Сравним вывод команды print с выводом команды display.

В заключение, удалим созданную точку останова и выйдем из отладчика комбинацией клавиш **CTRL** + **d**.

## 4.4 Анализ кода с помощью линтера splint

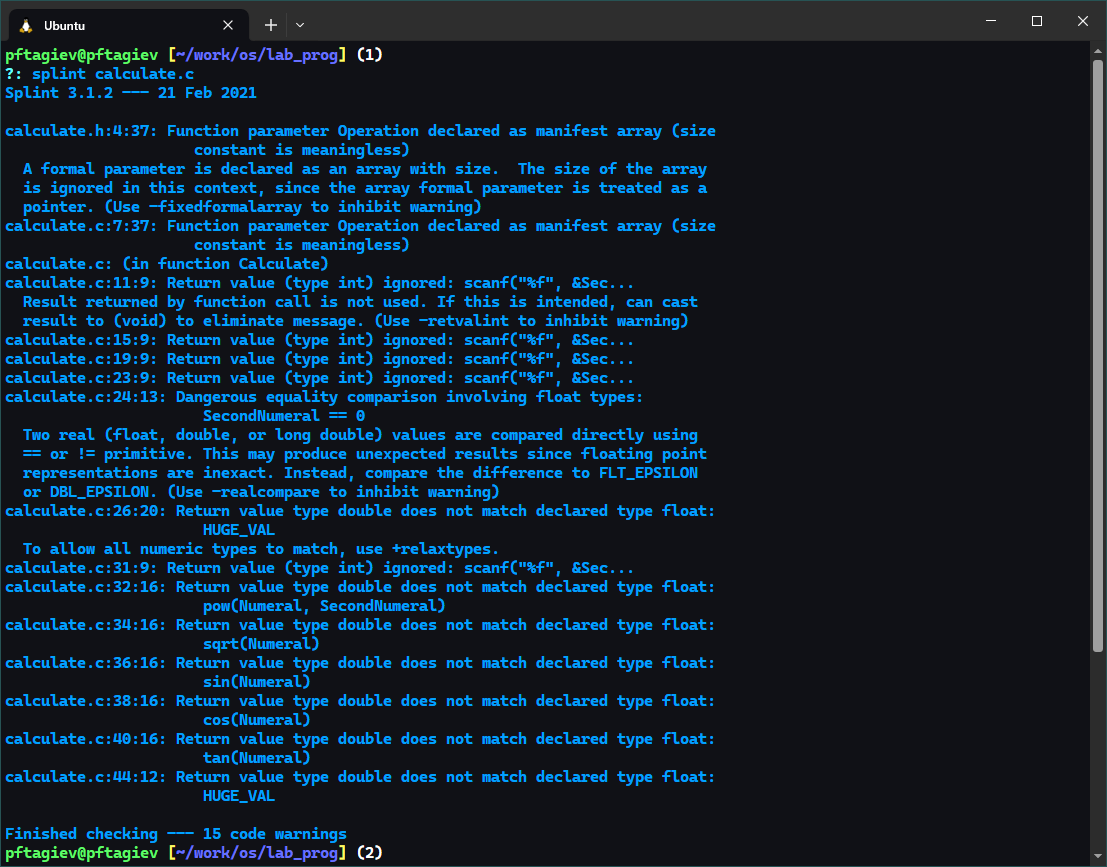


Рис. 10: splint для calculate.c

Разберем вывод утилиты splint для файла calculate.c (рис. 10). Сначала, выводится информация о том, что длина массива указанная в сигнатуре функции Calculate не имеет никакого смысла и игнорируется. Далее несколько раз выводится информация о том что мы игнорируем возвращаемое значение функции scanf. Затем идут предупреждения о неявном преобразовании типа double в тип float. В сумме *15* предупреждений.

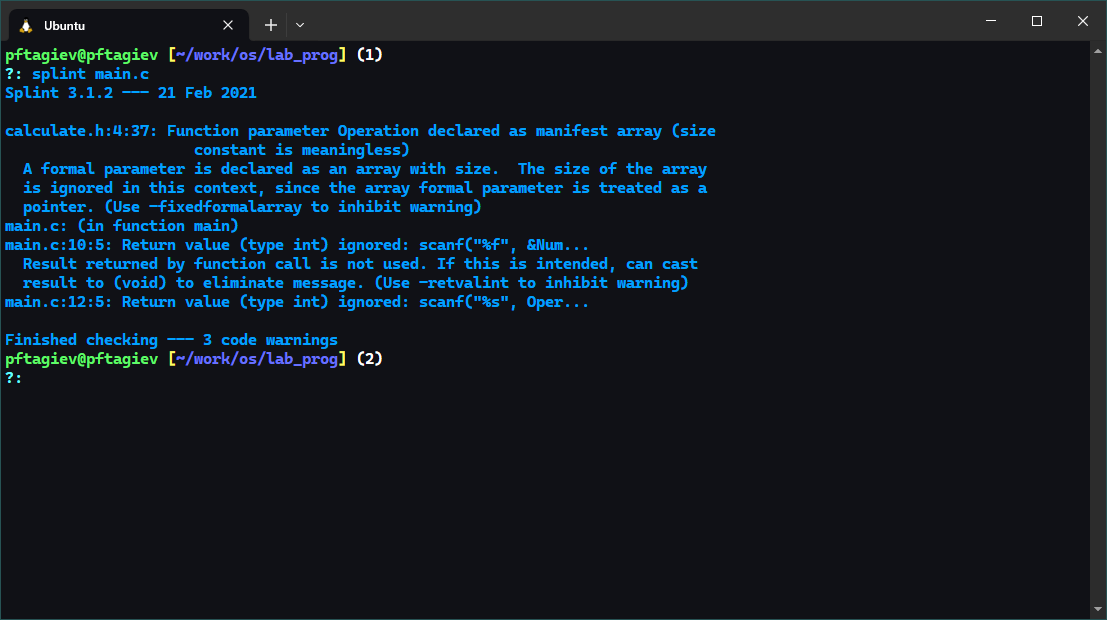


Рис. 11: splint для main.c

Проделаем такой же разбор, но уже для файла main.c. На рис. 11, мы видим уже знакомые нам предупреждения о игнорировании языком Си длины массива в сигнатуре функции Calculate и о игнорировании нами возвращаемого значения функции scanf. Всего *3* предупреждения.

# 5 Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?  
   Можно использовать команду man имя\_программы или посетить сайт проекта *GNU* [5].
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.
   * 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 предоставленной теории [1], суффикс эквивалентен расширению файла. Если же мы говорим о языке Си, в нем суффиксом называется то, что дописывается в конце литерала, например: 3.14f. В этом примере f суффикс, который говорит о том, что литерал будет иметь тип float.
4. Каково основное назначение компилятора языка С в UNIX?  
   Основное назначение компилятора языка Си в UNIX состоит в том, чтобы преобразовывать исходный код, написанный на языке программирования Си, в исполняемый файл, который может быть запущен операционной системой UNIX.
5. Для чего предназначена утилита make? Утилита make является мощным инструментом автоматизации сборки программного обеспечения, который широко используется в UNIX-системах. Ее основное назначение — управление процессом компиляции и сборки программ, обеспечивая эффективное и удобное обновление исполняемых файлов при внесении изменений в исходный код.
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.  
   Пример структуры Makefile и его характеристику можно увидеть на лист. 1.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?  
   Основное свойство, присущее всем программам отладки, — это возможность отслеживать выполнение программы, шаг за шагом, и анализировать ее состояние в любой момент времени. Это свойство называется “отладкой” или “debug mode”. Отладка позволяет разработчикам выявлять и исправлять ошибки в программе, а также понимать логику ее работы. Чтобы использовать это свойство, необходимо выполнить следующие шаги:
   1. Включить отладочную информацию при компиляции программы: Это делается с помощью флагов компилятора, например, -g в компиляторах *GCC*. Флаг -g указывает компилятору включить отладочную информацию в объектные файлы и исполняемый файл. Отладочная информация включает в себя данные о символах (таких как имена переменных и функций), строках кода и расположении переменных в памяти.
   2. Использовать отладчик: Отладчик — это программа, которая позволяет взаимодействовать с выполняющейся целью и контролировать ее выполнение. Примерами отладчиков являются *GDB* (GNU Debugger) для *C/C++* программ и pdb для Python-программ. Отладчик позволяет устанавливать точки останова, просматривать значения переменных, выполнять код пошагово и изучать стеки вызовов функций.
   3. Компилировать программу с отключенными оптимизациями: Некоторые оптимизации компилятора могут усложнить процесс отладки.
8. Назовите и дайте основную характеристику основным командам отладчика gdb.  
   Основные команды отладчика *GDB* (GNU Debugger) включают в себя следующее:
   * break: устанавливает точку останова в указанной строке кода или функции. Когда программа выполняется и достигает точки останова, она приостанавливает свое выполнение, позволяя вам проанализировать ее состояние.
   * run: запускает программу под контролем отладчика. Программа выполняется до первой точки останова или до завершения.
   * continue: продолжает выполнение программы после остановки в точке останова.
   * print: выводит значение выражения или переменной. Это позволяет проверять текущие значения переменных во время выполнения программы.
   * backtrace: отображает стек вызовов функций, показывая последовательность функций, которые были вызваны для достижения текущей точки выполнения. Это помогает понять поток управления в программе.
   * step: выполняет программу пошагово, переходя к следующей строке кода. Если следующая строка содержит вызов функции, отладчик заходит внутрь этой функции.
   * next: выполняет программу пошагово, но в отличие от step, он переходит к следующей строке кода, не заходя внутрь вызываемых функций.
   * finish: продолжает выполнение программы до выхода из текущей функции.
   * info breakpoints: информация о имеющихся точках останова.
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
   1. Собрать программу с ключем -g
   2. Загрузить программу в отладчик *GDB*.
   3. Расставить точки останова.
   4. Запустить загруженную программу командой run.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.  
    К сожалению, я переписал программу калькулятора без них, так как думал что это опечатки, и у меня не возникло никаких ошибок компиляции. Просмотрев код программы из [1], я вижу одну грубую ошибку (файл main.c строка *16*): scanf("%s",&Operation);, здесь не нужно брать адрес переменной Operation, т.к. мы передадим функции scanf char\*\*, а она ожидает char\*.
11. Назовите основные средства, повышающие понимание исходного кода программы.
    * инструменты статического анализа, линтеры (такие как splint)
    * cовременные IDE предлагают различные функции, облегчающие понимание кода, такие как подсветка синтаксиса и автодополнение
    * отладчики (такие как *GDB*)
12. Каковы основные задачи, решаемые программой splint? Программа Splint предназначена для решения следующих основных задач:
    1. Статический анализ кода: Splint выполняет статический анализ кода на языке C, выявляя потенциальные ошибки, проблемы безопасности и нарушения стандартов кодирования. Он проверяет код на соответствие определенным правилам и стандартам, таким как правила из руководства по стилю кодирования MISRA C.
    2. Выявление ошибок времени компиляции: Splint анализирует код на наличие синтаксических и семантических ошибок, которые могут привести к ошибкам во время компиляции. Он проверяет типы переменных, соответствие аргументов функций и соблюдение правил объявления переменных.
    3. Проверка безопасности: Splint специализируется на выявлении потенциальных проблем безопасности в коде, таких как переполнение буфера, использование неинициализированных переменных, ошибки управления памятью и другие распространенные уязвимости. Он помогает разработчикам писать более безопасный и защищенный от атак код.
    4. Подсказки по улучшению кода: Splint предоставляет подсказки и рекомендации по улучшению качества кода.

Листинг 1: Пример Makefile

# Определение переменных  
CC = gcc  
CFLAGS = -Wall -O2  
  
# Определение цели по умолчанию  
all: программа  
  
# Правило для сборки исполняемого файла  
программа: программа.o функция.o  
 $(CC) $(CFLAGS) -o программа программа.o функция.o  
  
# Правило для компиляции исходного файла в объектный файл  
.c.o:  
 $(CC) $(CFLAGS) -c $< -o $@  
  
# Правило для удаления объектных файлов и исполняемого файла  
clean:  
 rm -f программа.o функция.o программа

# 6 Выводы

В этой лабораторной работе мы приобрели базовые навыки разработки, тестирования и отладки приложения в UNIX-подобных операционных системах.

# Список литературы

1. Кулябов. Операционные системы. Москва: РУДН, 2016. 118 с.

2. GNU Compiler Collection [Электронный ресурс]. 2024. URL: <https://ru.wikipedia.org/wiki/GNU_Compiler_Collection>.

3. GNU Debugger [Электронный ресурс]. 2024. URL: <https://en.wikipedia.org/wiki/GNU_Debugger>.

4. Splint (programming tool) [Электронный ресурс]. 2024. URL: <https://en.wikipedia.org/wiki/Splint_(programming_tool)>.

5. GNU Manuals Online [Электронный ресурс]. 2022. URL: <https://www.gnu.org/manual/manual.html>.