

Отчет по лабораторной работе №12

Операционные системы

Фадин В.В.

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Задача 1	6
3	Выводы	12
4	Ответы на онтрольные вопросы	13

Список иллюстраций

2.1	Создание необходимых файлов и папок	6
2.2	Компиляция программы	7
2.3	Makefile	8
2.4	Отладка команды 1	9
2.5	Отладка команды 2	9
2.6	Вызов команды splint	10

Список таблиц

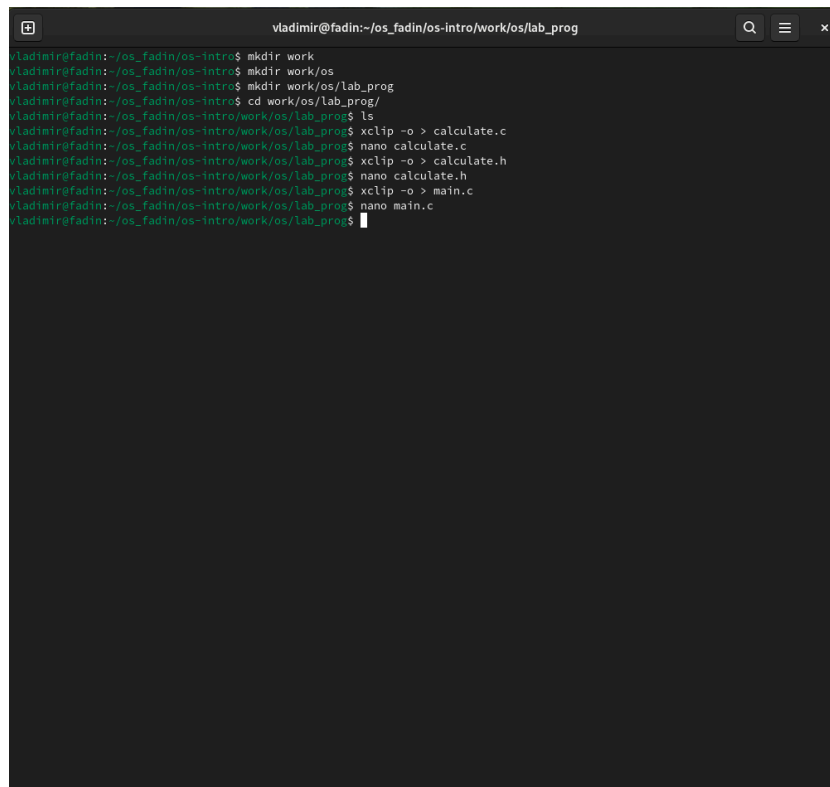
1 Цель работы

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.

2 Выполнение лабораторной работы

2.1 Задача 1

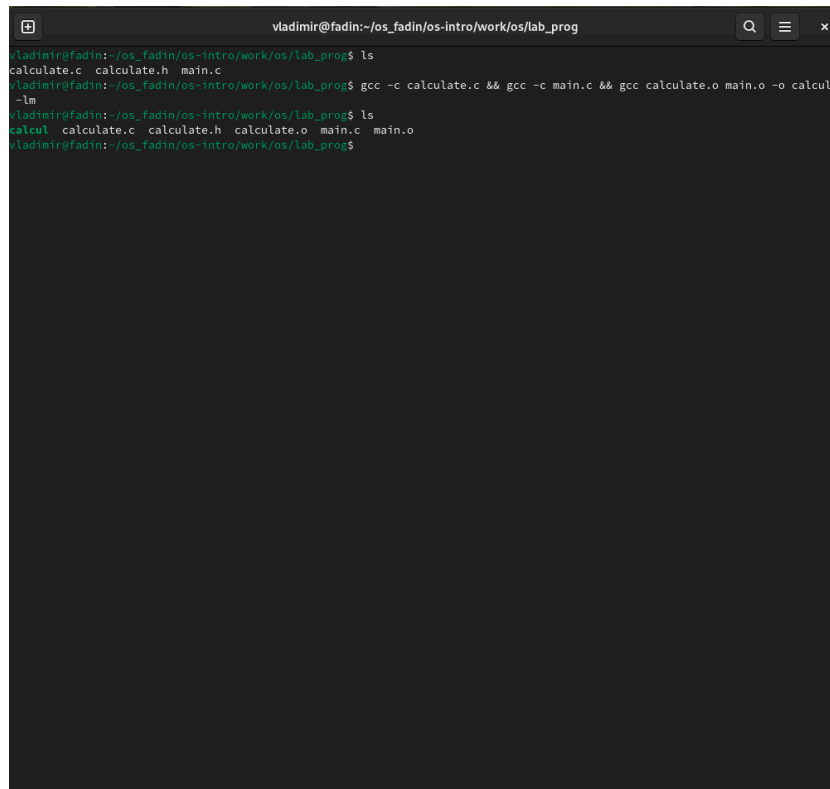
В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`. Создадим в нём файлы: `calculate.h`, `calculate.c`, `main.c` из лабораторной работы.



```
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog
vladimir@fadin:~/os_fadin/os-intro$ mkdir work
vladimir@fadin:~/os_fadin/os-intro$ mkdir work/os
vladimir@fadin:~/os_fadin/os-intro$ mkdir work/os/lab_prog
vladimir@fadin:~/os_fadin/os-intro$ cd work/os/lab_prog/
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ ls
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ xclip -o > calculate.c
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ nano calculate.c
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ xclip -o > calculate.h
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ nano calculate.h
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ xclip -o > main.c
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$ nano main.c
vladimir@fadin:~/os_fadin/os-intro/work/os/lab_prog$
```

Рис. 2.1: Создание необходимых файлов и папок

Выполним компиляцию программы посредством `gcc`:

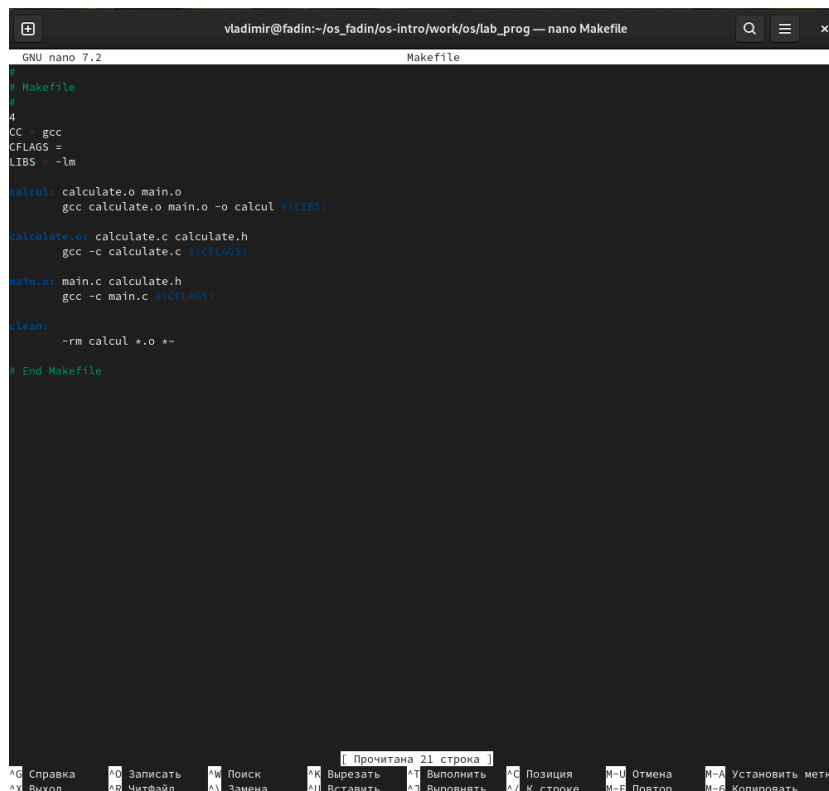
A terminal window with a dark background and light green text. The window title is 'vladimir@fadin:~/os_fadin/os-intro/work/os_lab_prog'. The terminal shows the following commands and output:

```
vladimir@fadin:~/os_fadin/os-intro/work/os_lab_prog$ ls
calculate.c calculate.h main.c
vladimir@fadin:~/os_fadin/os-intro/work/os_lab_prog$ gcc -c calculate.c && gcc -c main.c && gcc calculate.o main.o -o calcul
-lm
vladimir@fadin:~/os_fadin/os-intro/work/os_lab_prog$ ls
calcul calculate.c calculate.h calculate.o main.c main.o
vladimir@fadin:~/os_fadin/os-intro/work/os_lab_prog$
```

Рис. 2.2: Компиляция программы

После обнаружил, что необходимо собирать программу с помощью команды, чтобы gdb работал корректно: `gcc -g calculate.c main.c -o calcul -lm`

Создадим необходимый Makefile



```
GNU nano 7.2 Makefile
# Makefile
#
4
CC = gcc
CFLAGS =
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 *~

# End Makefile
```

Рис. 2.3: Makefile

Этот Makefile компилирует программу под названием «calcul» из двух исходных файлов: «calculate.c» и «main.c». Он использует компилятор gcc и связывается с математической библиотекой. Makefile определяет три скрипта:

- `calcul`: связывает `calculate.o` и `main.o` в исполняемый файл.
- `calculate.o` и `main.o`: скомпилирует `calculate.c` и `main.c` в объектные файлы соответственно.
- `clean`: удаляет исполняемый файл, объектные файлы и файлы резервных копий.

После запуска `make`, он проверит зависимости и перенастроит скрипты по мере необходимости.

С помощью `gdb` выполним отладку программы `calcul` по пунктам


```
root@fadin:/home/vladimir/work/os/lab_prog# gdb ./calcul
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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".
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...
(gdb) run
Starting program: /home/vladimir/work/os/lab_prog/calcul

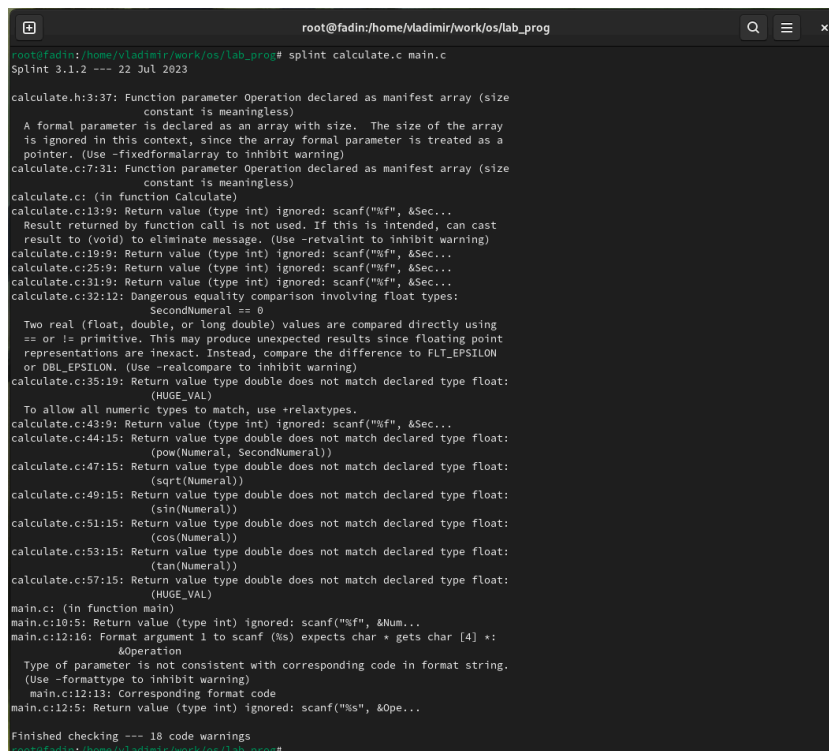
This GDB supports auto-downloading debuginfo from the following URLs:
- <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 1
4.00
[Inferior 1 (process 87488) exited normally]
(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3      int
4      main (void)
5      {
6          float Numeral;
7          char Operation[4];
8          float Result;
9          printf("Numco: ");
10         scanf("%f", &Numeral);
(gdb) list calculate.c:20,29
20         return Numeral - SecondNumeral;
21     }
22     else if (strcmp Operation, "+") == 0)
23     {
24         printf("Newoperation: ");
25         scanf("%f", &SecondNumeral);
```

Рис. 2.4: Отладка команды 1

```
21     }
22     else if (strcmp Operation, "+") == 0)
23     {
24         printf("Newoperation: ");
25         scanf("%f", &SecondNumeral);
26         return Numeral + SecondNumeral;
27     }
(gdb) break 21
Breakpoint 1 at 0x4011a7: file calculate.c, line 22.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000004011a7 in calculate at calculate.c:22
(gdb) run
Starting program: /home/vladimir/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 1
4.00
[Inferior 1 (process 87634) exited normally]
(gdb) break 1
Breakpoint 2 at 0x4011a7: file calculate.c, line 10.
(gdb) run
Starting program: /home/vladimir/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 2, Calculate (Numeral=5, Operation=0x7fffffff1a4 "-") at calculate.c:10
10     if (strcmp Operation, "+") == 0)
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffff1a4 "-") at calculate.c:10
#1 0x000000004011a7 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000004011a7 in Calculate at calculate.c:22
2 breakpoint keep y 0x000000004011a7 in Calculate at calculate.c:10
breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
Num Type Disp Enb Address What
2 breakpoint keep y 0x000000004011a7 in Calculate at calculate.c:10
breakpoint already hit 1 time
(gdb)
```

Рис. 2.5: Отладка команды 2

С помощью утилиты splint проанализируем коды файлов calculate.c и main.c:



```
root@fadin:/home/vladimir/work/os/lab_prog# splint calculate.c main.c
Splint 3.1.2 --- 22 Jul 2023

calculate.h:3:37: 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:13:9: 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:19:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:12: 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:35:19: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use -relaxtypes.
calculate.c:43:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:15: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:47:15: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:49:15: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:51:15: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:53:15: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:57:15: Return value type double does not match declared type float:
(HUGE_VAL)
main.c: (in function main)
main.c:10:5: Return value (type int) ignored: scanf("%f", &Num...
main.c:12:16: 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:12:13: Corresponding format code
main.c:12:5: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 18 code warnings
root@fadin:/home/vladimir/work/os/lab_prog#
```

Рис. 2.6: Вызов команды splint

Вывод splint указывает на то, что существует несколько потенциальных проблем с кодом calculate.c и main.c. Вот разбивка предупреждений:

Объявления массива манифеста

- «calculate.h:3:37» и «calculate.c:7:31»: параметр «Operation» объявляется как массив с размером, но размер игнорируется, поскольку массив рассматривается как указатель. Это не ошибка, а скорее предупреждение о том, что указание размера не имеет смысла в данном контексте.

Игнорируемые возвращаемые значения

- calculate.c:13:9, calculate.c:19:9, calculate.c:25:9, calculate.c:31:9, calculate.c:43:9 и main.c:10:5: возвращаемые значения вызовов scanf

игнорируются. Это не обязательно ошибка, но это предупреждение, напоминающее вам, что возвращаемое значение `scanf` может указывать, был ли ввод успешным.

Опасные сравнения с плавающей запятой

- `calculate.c:32:12`: Код сравнивает значение с плавающей запятой (`SecondNumeral`) напрямую, используя `==`, что может привести к неожиданным результатам из-за неточной природы представлений с плавающей запятой. Вместо этого рекомендуется сравнивать разницу с небольшим значением `эпсилон`.

Несоответствие типов

- `calculate.c:35:19`, `calculate.c:44:15`, `calculate.c:47:15`, `calculate.c:49:15`, `calculate.c:51:15` и `calculate.c:53:15`: типы возвращаемых значений некоторых выражений не соответствуют объявленным типам. Например, `HUGE_VAL` — это двойное значение, но оно присваивается переменной с плавающей запятой.

Проблемы со строкой форматирования

- `main.c:12:5`: строка формата `%s` ожидает аргумент `char *`, но вместо этого код передает аргумент `char [4] *`. Это может привести к неопределенному поведению.

В целом эти предупреждения предполагают, что в коде могут быть некоторые проблемы с объявлениями массивов, игнорированием возвращаемых значений, опасными сравнениями с плавающей запятой, несовпадениями типов и несогласованностью строк формата.

3 Выводы

Мы 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.

4 Ответы на онтрольные вопросы

1. Чтобы получить информацию о возможностях таких программ, как `gcc`, `make`, `gdb` и других, вы можете использовать опцию `--help` или `-h`, за которой следует имя программы. Например, `gcc --help` или `gdb -h`. Это отобразит использование и параметры для каждой программы.
2. Основными этапами разработки приложений в UNIX являются:
 - **Редактирование:** написание исходного кода с помощью текстового редактора.
 - **Компиляция:** перевод исходного кода в машинный код с использованием компилятора, такого как `gcc`.
 - **Связывание:** объединение объектных файлов в исполняемый файл с помощью компоновщика, такого как `ld`.
 - **Отладка:** выявление и исправление ошибок в программе с помощью отладчика, такого как `gdb`.
 - **Тестирование:** проверка правильности работы программы.
3. В контексте языков программирования суффикс — это символ или набор символов, добавляемый в конец имени переменной или функции для обозначения ее типа или назначения. Например, `myVariable_i` может указывать на целочисленную переменную, а `myFunction_f` может указывать на функцию с плавающей запятой.
4. Основная цель компилятора C в UNIX — преобразовать исходный код C в машинный код, который может выполняться компьютером.

5. Утилита `make` используется для автоматизации процесса сборки путем выполнения серии команд, указанных в `Makefile`. Он проверяет зависимости между файлами и перестраивает только то, что необходимо.
6. Базовая структура `Makefile` состоит из:
 - **Цели:** файлы, которые необходимо создать, например исполняемые файлы или объектные файлы.
 - **Зависимости:** файлы, необходимые для сборки целевых объектов, например исходные файлы или библиотеки.
 - **Команды:** действия, которые необходимо предпринять для создания целей, например компиляция или связывание.
7. Главным свойством, общим для всех программ-отладчиков, является возможность **пошагового выполнения кода**, проверяя состояние переменных и регистров на каждом этапе. Чтобы использовать эту функцию, вам необходимо скомпилировать программу с включенными символами отладки (флаг «-g») и запустить ее под отладчиком, например «gdb».
8. Основные команды `gdb`:
 - `run`: запускает выполнение программы.
 - `break`: устанавливает точку останова в определенном месте.
 - `next`: выполняет следующую строку кода.
 - `step`: выполняет следующую инструкцию, переходя к функциям.
 - `print`: отображает значение переменной.
 - `backtrace`: отображает стек вызовов.
9. Схема отладки, которую я использовал в этой лабораторной работе, включала:
 - Компиляция программы с включенными символами отладки (`gcc -g`).
 - Запуск программы под `gdb` (`gdb ./program`).
 - Установка точек останова в определенных местах («`break main`»).

- Прохождение кода, проверка переменных и регистров («next», «step», «print»).
 - Анализ стека вызовов («backtrace»).
10. Когда компилятор обнаруживает синтаксическую ошибку, он обычно отображает сообщение об ошибке с указанием номера строки и характера ошибки. Компилятор не создаст исполняемый файл, пока не будут исправлены все синтаксические ошибки.
11. Основными инструментами для понимания исходного кода являются:
- **Отладчики:** такие как gdb, которые позволяют пошагово выполнять код и проверять переменные.
 - **Инструменты анализа кода:** например, «сплент», который проверяет наличие потенциальных проблем и предупреждает о возможных ошибках.
 - **Документация:** комментарии и документация внутри самого кода.
12. Основными задачами, решаемыми “шиной”, являются:
- **Обнаружение утечек памяти:** выявление потенциальных утечек памяти.
 - **Обнаружение разыменования нулевого указателя:** предупреждение о потенциальных разыменованиях нулевого указателя.
 - **Проверка границ массива:** проверка доступа к массиву за пределами границ.
 - **Обнаружение несоответствия типов:** выявление несоответствия типов между переменными и параметрами функции.