Линтеры и санитайзеры

Или как стрелять себе в ногу чуть реже

Санкт-Петербургский
государственный университет

Вспомним уже известное

- Что такое линтеры?
- Зачем они нужны?
- А давайте покроем всё тестами!

Анализ кода

- Хотим искать ошибки в коде
- Как это можно делать?

Анализ кода

- Хотим искать ошибки в коде
- Как это можно делать?
- Тесты
  - Призваны «ловить» ошибки в логике
  - Сложно ловить «системные» ошибки, вроде работы с памятью

Анализ кода

- Хотим искать ошибки в коде
- Как это можно делать?
- Тесты
  - Призваны «ловить» ошибки в логике
  - Сложно ловить «системные» ошибки, вроде работы с памятью
- Статически — без запуска кода
  - Статические анализаторы
  - Линтер — частный случай статического анализатора
  - Современные компиляторы подрабатывают статическими анализаторами
- Динамически — во время исполнения
  - Динамические анализаторы

clang-tidy

clang-tidy[1] — линтер для C/C++ на основе экосистемы LLVM

- Требует информации о том, как собирался проект
  - Конфигурируйте CMake с флагом `-DCMAKE_EXPORT_COMPILE_COMMANDS=ON`
- Поддерживает как проверки вида «предупреждения компилятора» (в частности clang), так и отдельные проверки, например
  - вызовы `memset(ptr, 0, n * sizeof(ptr))` вместо `memset(ptr, 0, n * sizeof(*ptr))`
  - использование магических констант
  - правильное использование нейминга
- Пример использования (в основном в CI)
  - https://github.com/WoWaster/spbu-c-ci-example

[1] Страница: https://clang.llvm.org/extra/clang-tidy/

Sanitizers

Определение

Инструментирование — модификация (исходного или бинарного) кода программы с целью дальнейшего проведения анализа

Sanitizers (санитайзеры)[2] — набор динамических анализаторов, разработанный Google

- Инструментирование программы на этапе компиляции
- Довольно сложно устроены для рассказа сейчас
  - Подробности можно найти в wiki репозитория
  - В видео: https://youtu.be/7WyBAUJ8UA8

[2] Репозиторий: https://github.com/google/sanitizers

Виды санитайзеров

ASan находит ошибки работы с памятью: выход за границы массива, use-after-free, double free

- В среднем замедляет программу в 2 раза

MSan находит использование неинициализированной памяти

- В среднем замедляет программу в 3 раза

UBSan находит ошибки, связанные с undefined behavior

- Работает на этапе компиляции, поэтому «бесплатный»
- Есть опциональная библиотека времени исполнения «немного влияющая на скорость исполнения»

LSan находит утечки памяти

- Отрабатывает перед завершением программы, поэтому почти «бесплатный»

другие ThreadSanitizer, DataFlowSanitizer, TypeSanitizer, RealtimeSanitizer

Valgrind

Valgrind[3] — фреймворк для создания динамических анализаторов, а также набор инструментов

- Не требует пересборки программы, но лучше иметь отладочную информацию
- Программа исполняется на «виртуальном» процессоре, что позволяет собирать огромное количество разной информации о программе

Наиболее интересный нам инструмент — Memcheck, который позволяет отлавливать ошибки работы с памятью
Замедляет программу в 10–30 раз

[3] Сайт: https://valgrind.org/

Практика

Разберём примеры отсюда:
https://github.com/spbu-coding-2025/workshop-analyzers

Посмотрим на то, как добавить себе clang-format и clang-tidy в репозиторий: https://github.com/WoWaster/spbu-c-ci-example