Санитайзеры и стандарт не спасут

План

1. **Непредсказуемость C++**
2. Как избегать UB?
3. Динамический анализ
4. Статический анализ
5. Что говорит стандарт?
6. Автоматизация проверок

Неопределённое поведение (UB)

- Забудем про многопоточность и внешние зависимости
- Одна и та же программа может себя вести по-разному в зависимости от:
  - Компилятора
  - Стандартной библиотеки
  - ОС
  - Процессора
  - Настроек всего перечисленного
- Возникает и при "простых" ошибках, и в (не)очевидном коде
- 20 лет назад: компиляторы "просто генерируют код", UB предсказуемо
  - Разыменование NULL — падение программы
- Сейчас: компиляторы оптимизируют, UB необъяснимо
  - Разыменование NULL — случайная уязвимость

Нельзя предсказать поведение при UB

<table>
<tr><th></th><th>Другие языки</th><th>C++</th></tr>
<tr><td>Выход за границу массива</td><td>Исключение
Печать stack trace</td><td rowspan="5">Результат — мусор
Неверный ответ
При каждом запуске свой (ASLR)
Падение в строке с ошибкой
Падение при выходе из функции
Падение в другой функции
Падение при завершении работы
С верным ответом
С неверным ответом
Вызов не вызываемой функции
С отладочным выводом работает
Путешествия во времени
Невозможное поведение</td></tr>
<tr><td>Забыли return</td><td>Не компилируется
Возвращает None</td></tr>
<tr><td>Висячая ссылка</td><td rowspan="2">Не компилируется
Невозможно</td></tr>
<tr><td>Неинициализированная переменная</td></tr>
<tr><td>Переполнение</td><td>Исключение
Результат по модулю</td></tr>
</table>

Невозможное поведение

Эта функция может упасть с переполнением массива:

```
char destBuffer[16];
void Serialize(bool boolValue) {
    const char* whichString = boolValue ? "true" : "false";
    const size_t len = strlen(whichString);
    memcpy(destBuffer, whichString, len);
}
```

Невозможное поведение

Эта функция может упасть с переполнением массива:

```
char destBuffer[16];
void Serialize(bool boolValue) {
    const char* whichString = boolValue ? "true" : "false";
    const size_t len = strlen(whichString);
    memcpy(destBuffer, whichString, len);
}
int main() {
    bool x; Serialize(x);  // мусор: 0 — false, не-0 — true
}
```

Невозможное поведение

Эта функция может упасть с переполнением массива:

```
char destBuffer[16];
void Serialize(bool boolValue) {
    const char* whichString = boolValue ? "true" : "false";
    const size_t len = 5 -      boolValue;  // оптимизация
    memcpy(destBuffer, whichString, len);
}
int main() {
    bool x; Serialize(x);  // мусор: 0 — false, не-0 — true
}
```

Невозможное поведение

Эта функция может упасть с переполнением массива:

```
char destBuffer[16];
void Serialize(bool boolValue) {   // ABI: bool == 0 или 1
    const char* whichString = boolValue ? "true" : "false";
    const size_t len = 5 - (int)boolValue;   // каст не нужен
    memcpy(destBuffer, whichString, len);
}
int main() {
    bool x; Serialize(x);   // мусор: 0 — false, не-0 — true
}
```

Прочие неочевидности

UB — не событие, а нарушение предположения конкретного компилятора:

1. Процессор = x86-совместимый
2. ОС = Windows, первой вызывается `WinMain`
3. Стандартная библиотека лежит в такой-то DLL
4. `bool` — 1 байт, 0/1
5. Во время выполнения не будет UB (`bool` никогда не 2)

Отключить все оптимизации невозможно

- Что такое оптимизации?
- Можно ли заменить `2*2` на `4`?
- Можно ли заменить `16*x` на `(x << 4)`?

Наивный взгляд на мир

Хотим: писать безопасные программы (доклад, доклад)

Верим: писать программы по стандарту C++ (без UB) поможет

Предположения:

- Используем стабильные версии компиляторов, библиотек, ОС
- В компиляторах бывают баги, но только в экзотических случаях
- Компиляторы могут добавлять свои расширения, но не сильно противоречащие стандарту C++ (variable length array)

План

1. Непредсказуемость C++
2. **Как избегать UB?**
3. Динамический анализ
4. Статический анализ
5. Что говорит стандарт?
6. Автоматизация проверок

Что в других языках

- Поведение всех конструкций определено, иногда ценой скорости
  - Даже проблемы с компараторами (доклад) будет добрее в JavaScript
- Если компилятору нетривиально доказать корректность, можно запретить синтаксис. В C++ обычно разрешают.

```
int sign(int x) {
    if (x < 0) return -1;
    if (x > 0) return 1;
    if (x == 0) return 0;
}
// a.java:6: error:
// missing return statement
```

```
# Starlark
def fib(n):
    if n <= 1: return n
    return (fib(n - 2) +
            fib(n - 1))
# Error: function fib
# called recursively
```

Динамический анализ

- Анализируется выполнение программы на конкретном вводе
- Может выполняться в "виртуальной машине" (Valgrind)
- Можно изменить код
  - Sanitizers
  - `_GLIBCXX_DEBUG`, `_LIBCPP_DEBUG`, отладочный режим Visual Studio
- Если возникла ошибка:
  - Упасть с пояснением
  - Сообщить и восстановиться
- Можем не найти ввод, где воспроизводится проблема
- Если проблема есть в конкретном запуске — она будет обнаружена
  - На самом деле нет
- Непонятно, как определить "байт памяти" или "корректный указатель"

Статический анализ для поиска UB

- Анализируется код программы целиком, без запуска
- Предупреждения компилятора, clang-tidy, cppcheck, PVS-Studio...
- Можно анализировать синтаксис, граф потока управления
- Всегда есть ложноположительные срабатывание
- Нельзя проверить свойство программы "для всех выполнений"
  - Теорема Райса, задача останова

```
while (true) {
    // Не встречается break, return
    // Нет исключений

}
// Синтаксически бесконечный цикл
```

Статический анализ для поиска UB

- Анализируется код программы целиком, без запуска
- Предупреждения компилятора, clang-tidy, cppcheck, PVS-Studio...
- Можно анализировать синтаксис, граф потока управления
- Всегда есть ложноположительные срабатывание
- Нельзя проверить свойство программы "для всех выполнений"
  - Теорема Райса, задача останова

```
while (true) {
    // Не встречается break, return
    // Нет исключений

}
// Синтаксически бесконечный цикл
```

```
bigint a = 1;
while (!checkCollatz(a)) {
    a++;
}
// Конечность зависит от
// гипотезы Коллатца
```

Статический анализ для поиска UB

- Анализируется код программы целиком, без запуска
- Предупреждения компилятора, clang-tidy, cppcheck, PVS-Studio...
- Можно анализировать синтаксис, граф потока управления
- Всегда есть ложноположительные срабатывание
- Нельзя проверить свойство программы "для всех выполнений"
  - Теорема Райса, задача останова

```
while (true) {
    // Не встречается break, return
    // Нет исключений
    // Нет goto, longjmp, co_return
}
// Синтаксически бесконечный цикл,
// переполнение стека или exit(0)
```

```
bigint a = 1;
while (!checkCollatz(a)) {
    a++;
}
// Конечность зависит от
// гипотезы Коллатца
```

План

1. Непредсказуемость C++
2. Как избегать UB?
3. **Динамический анализ**
4. Статический анализ
5. Что говорит стандарт?
6. Автоматизация проверок

Проверка домашних заданий

До: проверка вручную, code review

После: Portability is Reliability + автотесты

- Жёсткий формат ввода-вывода
- Автоматические тесты: открытые (публичные) и секретные (приватные)
- 0 предупреждений
  - Компиляторы
  - clang-tidy
  - cppcheck
- Code review, если всё прошло:
  - clang-format
  - 2 анализатора
  - 9 режимов компиляции и запуска

Ubuntu LTS (16.04-22.04)

- g++ 12 (GNU), libstdc++ (GNU)
- clang++ 14 (LLVM), libstdc++ (GNU)
- clang++ 14 (LLVM), libc++ (LLVM)

Всегда `-O2 -Wall -Wextra -Werror`, плюс

- `-DNDEBUG`
- `-fsanitize=undefined -g` + Valgrind
- `-fsanitize=address`
  `-fsanitize=undefined -g`

Вопрос от студента: решение не проходит (2020)

	g++	**clang++ libstdc++**	**clang++ libc++**
Release	OK	OK	OK
Valgrind UBSan	OK	**FAIL**	OK
ASan UBSan	OK	OK	OK

```
// Conditional jump or move
// depends on uninitialised
// value(s)
```

Вопрос от студента: решение не проходит (2020)

	g++	**clang++ libstdc++**	**clang++ libc++**
Release	OK	OK	OK
Valgrind UBSan	OK	**FAIL**	OK
ASan UBSan	OK	OK	OK

```
// Conditional jump or move
// depends on uninitialised
// value(s)
#include <regex>
int main() {
    std::regex regex("x{2,}");
}
```

Вопрос от студента: решение не проходит (2020)

	g++	**clang++ libstdc++**	**clang++ libc++**
Release	OK	OK	OK
Valgrind UBSan	OK	**FAIL**	OK
ASan UBSan	OK	OK	OK

```
// Conditional jump or move
// depends on uninitialised
// value(s)
#include <regex>
int main() {
    std::regex regex("x{2,}");
}
```

- Плохой компилятор?
- Куда репортить баг?
  - Valgrind
  - clang++
  - libstdc++ **(WORKSFORME)**
  - Valgrind, но в Ubuntu
  - clang++, но в Ubuntu
  - libstdc++, но в Ubuntu
- **Не используем std::regex**

Вопрос от проверяющего: решение не проходит (2021)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	OK	OK
Valgrind UBSan	OK/**FAIL**	OK	OK
ASan UBSan	**FAIL**	OK	OK

```
int range_check(int x) {
    throw 0;
}
struct Bar {};
struct Foo {  // former vector<>
    Bar *data = nullptr;
    const Bar &get(int x) const {
        return data[range_check(x)];
    }
};
int main() {
    Foo b;
    try {
        b.get(-1);
    } catch (...) {
    }
}
```

[Баг в GCC (FIXED)](), **переписали код**

Решение студента не работает локально (2021)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	OK	OK
Valgrind UBSan	OK	OK	OK
ASan UBSan	OK	OK	OK

```
#include <cstdlib>
struct TestResultChecker {
    // .....
    ~TestResultChecker() {
        if (failed) {
            std::exit(1);
        }
    }
};
TestResultChecker test1(.....);
TestResultChecker test2(.....);
int main() {
}
```

Бесконечная рекурсия

Решение студента не работает локально (2021)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	OK	OK
Valgrind UBSan	OK	OK	OK
ASan UBSan	OK	OK	OK

```
#include <cstdlib>
struct TestResultChecker {
    // .....
    ~TestResultChecker() {
        if (failed) {
            std::exit(1);
        }
    }
};
TestResultChecker test1(.....);
TestResultChecker test2(.....);
int main() {
}
```

Бесконечная рекурсия

На самом деле undefined behavior

Секретные тесты не работают локально (2021)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	OK	OK
Valgrind UBSan	OK	OK	OK
ASan UBSan	OK	OK	OK

Баг в библиотеке doctest?

```
TEST_CASE("Test thread") {
    std::thread([&]() {
        CHECK(2 * 2 == 4);
    }).join();
}
```

- Windows, msys2 mingw-w64 gcc 12
- Падает с use-after-free
- Только под gdb

Оказывается, в msys2 mingw [сломан thread_local]

```
struct S {
    long long x;
    S() : x(0x1234) {
        // x == 0x1234
    }
    S(const S &) = delete;
    ~S() {
        // x == 0xFEEE...
    }
};
int main() {
  std::thread([&]() {
     thread_local S oss;
     static_cast<void>(oss);
  }).join();
}
```

Поток Windows

```
(1-4) pthread_create_wrapper();

pthread_create_wrapper(.....) {
   (1) &oss = pthread_key_create
   (2) __cxa_thread_atexit(oss, S::~S)

   (3) // static_cast<void>(oss)

   (4) _pthread_cleanup_dest //free(&oss)
}

(5)    // событие DLL_THREAD_DETACH
(6)    oss.~S();  // обработчик atexit
```

Перерыв на вопросы

Секретные тесты не компилируется на macOS (2021)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	**FAIL**	**FAIL**
Valgrind UBSan	OK	**FAIL**	**FAIL**
ASan UBSan	OK	**FAIL**	**FAIL**

```cpp
#include <iostream>
int main() {
    int arr[]{1, 2};
    auto [x, y] = arr;
    [&]() {
        std::cout << x;   // (!)
    }();
}
```

```
a.cpp:6:22: error: reference to
local binding 'x' declared in
enclosing function 'main'
```

Секретные тесты не компилируется на macOS (2021)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	**FAIL**	**FAIL**
Valgrind UBSan	OK	**FAIL**	**FAIL**
ASan UBSan	OK	**FAIL**	**FAIL**

```
#include <iostream>
int main() {
    int arr[]{1, 2};
    auto [x, y] = arr;
    [&]() {
        std::cout << x;   // (!)
    }();
}
```

```
a.cpp:6:22: error: reference to
local binding 'x' declared in
enclosing function 'main'
```

Прав **только** Clang

Фича стандарта? wg21.link/CWG2313

x — не переменная, а имя

Старое правильное решение не работает (2022)

	g++	clang++ libstdc++	clang++ libc++
Release	OK	OK	OK
Valgrind UBSan	OK	OK	OK
ASan UBSan	OK	OK	**FAIL**

```
#include <stdexcept>
int main()
{
    std::logic_error{ "" };
}
// AddressSanitizer:
// alloc-dealloc-mismatch
// (operator new vs free)
```

Баг в LLVM под Ubuntu (open)

Отключили проверку под libc++

Больше компиляторов и библиотек

- Apple Clang 13
  - Чуть другой libc++ с другими #include
  - При использовании Boost.DLL с address sanitizer падает
- Intel C++ Classic 2021.8
  - По-другому парсит
  - Не может подавить предупреждение из макроса: пришлось отключить
  - Deprecated
- Boost.DLL 1.71 (Ubuntu 20.04): UB, выход за границы enum
- Boost.Asio 1.71 (Ubuntu 20.04): не хватает #include
- Boost.Asio 1.74 (Ubuntu 22.04): для clang отключён move-конструктор
  - Clang притворяется gcc 4, а в нём move ещё не было

Больше компиляторов: Visual Studio 19.32

- Проверка под Windows: `std::ios_base::binary` (переводы строк)
- Более строгие предупреждения про неявные преобразования
- Рекомендуются `scanf_s` из C11, поддержка только у Visual Studio
- В debug-режиме программа зависает с окном "Abort, Retry, Ignore"
- По умолчанию исключения не вызывают деструкторы, CMake в курсе
- По умолчанию по-другому работают макросы, CMake не в курсе
  - Достаточно стандартно, чтобы не заметить на простых примерах
  - Достаточно нестандартно, чтобы студент случайно наткнулся, пока эмулирует в Си параметры функций по умолчанию через макросы

Visual Studio тоже лажает (2022)

```
struct transaction { /* ... */ };
mutex m;
condition_variable c;
vector<transaction> ts;
struct transaction_iterator {
    size_t transaction;
    auto next() {
        unique_lock l(m);
        c.wait(l, [&]() {
            return (transaction < ts.size());   // syntax error: missing ')' before '<'
        });
        return ts[transaction++];
    }
};
```

Visual Studio тоже находит баги


```
int operator_priority(char c) {
    static int prio[] = {
        ['+'] = 1,
        ['-'] = 1,
        ['*'] = 2,
        ['/'] = 2,
    };
    return prio[c];
}
// operator_priority(' ') == 0
// operator_priority('*') == 1
```

Visual Studio тоже находит баги


```
int operator_priority(char c) {
    static int prio[128] = {
        ['+'] = 1,
        ['-'] = 1,
        ['*'] = 2,
        ['/'] = 2,
    };
    return prio[c];
}
// operator_priority(' ') == 0
// operator_priority('*') == 1
// operator_priority('0') is UB
```

Никто не гарантирует проверку границ массива (даже санитайзеры или отладочный режим: тут чистые массивы)

План

1. Непредсказуемость C++
2. Как избегать UB?
3. Динамический анализ
4. **Статический анализ**
5. Что говорит стандарт?
6. Автоматизация проверок

clang-tidy: шаблоны мешаются (2020, FIXED)

```
template<typename> struct T {   // almost vector<>
    int *p = nullptr;
    T(const T &rhs) : p(rhs.p ? new int(*rhs.p) : nullptr) {}
    ~T() { delete p; }
    T& operator=(const T &rhs) {
        // operator=() does not handle self-assignment properly
        if (this == &rhs)
            return *this;
        delete p; p = new int(*rhs.p);
        return *this;
    }
};
```

- Только clang-tidy 10, только под Windows (не под Linux)

clang-tidy: шаблоны мешаются (2022, NEW)

```
template <typename T>
struct vector {
    int len;
    void at(int k) {
        if (k >= len) {
            throw std::out_of_range(std::to_string(len));
            // C-style casts are discouraged
        }
    }
};
```

Решение не компилируется g++ Release (2022)

```
struct shared_ptr {
    int *ctr_ = nullptr;
    int *data_ = nullptr;
    shared_ptr(int *d) : ctr_(data ? new int(1) : nullptr), data_(data) {
    }
    shared_ptr(const shared_ptr &other)
        : ctr_(other.ctr_), data_(other.data_) {
        if (ctr_ != nullptr) { ++*ctr_; }
    }
    ~shared_ptr() {
        if (ctr_ != nullptr) {
            --*ctr_;  // pointer used after 'delete' [-Wuse-after-free]
            if (*ctr_ == 0) { delete ctr_; delete data_; }
        }
    }
};
```

Баг GCC (OPEN), я не понимаю что там происходит

План

1. Непредсказуемость C++
2. Как избегать UB?
3. Динамический анализ
4. Статический анализ
5. **Что говорит стандарт?** (но это неточно)
6. Автоматизация проверок

Что если компилировать не с -std=c++11?

```
#include <vector>
#include <iostream>
using namespace std;
int main() {
    vector<int> a;
    for (int aa : a)
        cout << aa << endl;
}
```

Что если компилировать не с -std=c++11?

```
#include <vector>
#include <iostream>
using namespace std;
int main() {
    vector<int> a;
    for (int aa : a)
        cout << aa << endl;
}
```

```
warning: range-based ‘for’ loops only available with -std=c++11 or -std=gnu++11
```

If a program contains a violation of any diagnosable rule or an occurrence of a construct described in this Standard as “conditionally-supported” when the implementation does not support that construct, a conforming implementation shall issue **at least one diagnostic message**.

This **diagnostics** is produced. I'd like to point out that clang behaves similarly (albeit this is not really relevant when we talk about the correctness).

Баг в 2015 (gcc 5), исправлено в 2021 (gcc 9)

А можно ли использовать синтаксис из Си?

```
#include <iostream>
int main() {
    int arr[] = {
        [10] = 1,
        [20] = 2
    };
    std::cout << arr[10];
}
```

- gcc: ошибка компиляции
- clang: ошибка компиляции
- msvc: ошибка компиляции
- Apple Clang: **работает**

Внимание: LLVM Clang != Apple Clang

Баг в библиотеке?

```
// a.cpp
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"

// b.cpp
#include <string>
#include "doctest.h"

void foo() {
    std::string a, b;
    CHECK(a == b);
}
```

```
/tmp/b-c494e1.o: In function `doctest::...':
b.cpp:(.text...basic_string...+0x23):
undefined reference to `operator<<'
```

Баг в LLVM? (NEW)

```
// a.cpp
#include <iosfwd>
#include <string>
void f(std::ostream &os) { os << std::string("Hello world!\n"); }

// b.cpp
#include <iostream>
void f(std::ostream &);
int main() { f(std::cout); }
```

```
Undefined symbols for architecture x86_64:
  "std::__1::basic_ostream<....>& std::__1::operator<<", referenced from:
      f(std::__1::basic_ostream<....>&) in f-e729ff.o
```

Пример с лекции: баг в GCC?

```
struct A {
    explicit A(int = 42);
};
int main() {
    A a1 = { };      // not explicit
    A a2 = { 24 };   // not explicit
}
```

```
a.cpp: In function 'int main()':
a.cpp:5:12: warning: converting to 'A' from initializer list would use
                     explicit constructor 'A::A(int)'
a.cpp:5:12: note: in C++11 and above a default constructor can be explicit
a.cpp:6:15: error: converting to 'A' from initializer list would use
                   explicit constructor 'A::A(int)'
```

Пример почти из GTA Online (обсуждение)

```
int sum_all(const char *s) {
    int sum = 0;
    std::size_t pos = 0;
    while (s[pos]) {
        int value, read;
        if (std::sscanf(s + pos,
            "%d%n", &value, &read)
            != 1) {
            break;
        }
        pos += read;
        sum += value;
    }
    return sum;
}
```

- `%d` — число
- `%n` — количество считанных символов
- Считываем числа из строки, суммируем
- **Не** пишем `pos < strlen(s)`
- На 10МБ строчке работает долго

Пример почти из GTA Online (обсуждение)

```
int sum_all(const char *s) {
    int sum = 0;
    std::size_t pos = 0;
    while (s[pos]) {
        int value, read;
        if (std::sscanf(s + pos,
            "%d%n", &value, &read)
            != 1) {
            break;
        }
        pos += read;
        sum += value;
    }
    return sum;
}
```

```
int sscanf(const char *s, args...) {
    return fscanf(
        _IO_strfile_read(f), args...);
}
FILE *_IO_strfile_read(const char *s) {
    char *end = strchr(s, '\0');
    // ....
}
```

- sscanf, очевидно, работает за линейное время
  - Стандарту тоже очевидно, молчит
- Линейное от чего?
  - Количество прочитанных символов
  - Длина строки

Не UB, но падает

```
#include <iostream>
int read;  // "прочитано"
int readInt() {
    int x;
    std::cin >> x;
    read++;
    return x;
}
int main() {
    std::ios_base::sync_with_stdio(false);
    readInt();
    readInt();
    std::cout << read << "\n";
}
```

```
g++ a.cpp -static -o a
```

- Статическая линковка
- Падает под Linux
- Падает под Windows
- Не падает под Visual Studio

```
malloc: built-in function
'malloc' declared as
non-function
```

Не UB, но падает

```cpp
#include <iostream>
int read;  // "прочитано"
int readInt() {
    int x;
    std::cin >> x;
    read++;
    return x;
}
int main() {
    std::ios_base::sync_with_stdio(false);
    readInt();
    readInt();
    std::cout << read << "\n";
}
```

`g++ a.cpp -static -o a`

- Статическая линковка
- Падает под Linux
- Падает под Windows
- Не падает под Visual Studio

```
malloc: built-in function
'malloc' declared as
non-function
```

- В POSIX есть функция `read`
- По факту нарушение ODR
- По стандарту всё корректно

План

1. Непредсказуемость C++
2. Как избегать UB?
3. Динамический анализ
4. Статический анализ
5. Что говорит стандарт?
6. **Автоматизация проверок**

Запуск открытых тестов

Первая попытка на Travis:

- В pull request #7 скачивается код из #79 — не могли воспроизвести три месяца

Вторая попытка на GitHub Classroom:

- Не создаются PR (pull request) у 25 из 65 студентов, надо руками

Третья попытка на чистых GitHub Actions:

- С лета 2021: гонка между открытием pull request и его доступностью в API
- Студенты случайно коммитят в master => надо отдельно проверять хэш master
- При обновлении master PR меняется, но не приходит webhook
- 1-3 раз в год в день дедлайна лежит
- Тикеты в поддержку мне уже недоступны

Как сломать self-hosted GitHub Actions Runner

```
git clone https://github.com/org/rep1
cd repo
git clone https://github.com/org/rep2
git add .  # создали submodule
git commit
git push
# редактирование
git add . && git commit
git push
```

- Рабочая папка не чистится
- Проблема с 2020 года
- Первый запуск ок
- Повторный запуск падает
- Надо руками чистить папку

Ещё можно очистить в workflow

diff 3.3 (Ubuntu 16.04)

```
$ echo -n a > file1.txt
$ echo -n b > file2.txt
$ cat file1.txt
a$ cat file2.txt
b$ diff -B file1.txt file2.txt
$ echo $?
0
```

- -n означает "без перевода строки"
- Ключ -B игнорирует пустые строки
- Проблема из 2014 года
- Исправлено в diff 3.4

Итоги

- Баги в анализаторах можно найти случайно на учебных примерах
  - ~2000 решений домашних заданий, несколько десятков багов
  - Есть стабильные баги: конструктор std::out_of_range + address sanitizer
  - Есть популярные баги: ложное срабатывание анализатора GCC (~5% студентов)
- Использование стабильных версий ничего не гарантирует
  - Репортите баги!
  - В новых версиях — новые баги, баги везде
- UB всё ещё сильно вероятнее, чем баг компилятора
  - Минимизируйте пример!
- Компиляторы иногда нарочно не следует стандарту
- Стандарт прописывает не всё

Никогда нельзя обновить компилятор и быть уверенным в успехе

Бонусы

Спасёт ли нас constexpr?

```
static_assert(2 == []() {
    vector<int> v{1, 2, 3};
    v.push_back(4);
    // v.resize(1);
    v.pop_back();
    v.pop_back();
    return v.size();
}());
```

- С C++20 много чего constexpr
- Гарантия: UB — ошибка компиляции
- Под капотом всё ещё эмулирует и выходит за границу?

```
C:/tools/msys64/mingw64/include/c++/12.2.0/bits/stl_cons
truct.h:162:11: error: 'constexpr' loop iteration count
exceeds limit of 262144 (use '-fconstexpr-loop-limit='
to increase the limit)
  162 |         for (; __first != __last; ++__first)
      |         ^~~
```

Miscompilation (из чата t.me/ProCxx/444317, FIXED)

```
#include <iostream>
int main() {
    for (
        int i = 0;
        int x = i < 2 ? 1 : 0;
        i++)
    {
        std::cout << x;
    }
}
// Ожидается: 11
// Реально: 000000000000...
```

Исправление:

```
+unsigned QuestionColonDepth = 0;
 while (true) {
-  unsigned QuestionColonDepth = 0;
   // .....
   if (P.Tok.is(tok::question))
     ++QuestionColonDepth;
   else if (P.Tok.is(tok::colon)) {
     if (QuestionColonDepth)
       --QuestionColonDepth;
     else
       CanBeCondition = false;
   }
   // .....
 }
```

Что считать "использованием"? (2020, FIXED)

```
#include <concepts>
[[nodiscard]] int foo() {
    return 0;
}
[[maybe_unused]]
constexpr bool b = requires {
    { foo() } -> std::same_as<int>;
};
[[maybe_unused]]
constexpr auto x = sizeof(foo());

<source>:6:10: warning: ignoring return value of 'int foo()', declared with
attribute 'nodiscard' [-Wunused-result]
    6 |  { foo() } -> std::same_as<int>;
```