Статический анализ кода: современный взгляд

Карпов Андрей Николаевич
к.ф.-м.н., технический директор
ООО «Системы программной верификации»
E-mail: karpov@viva64.com

Program Verification Systems

О чем доклад?

- Андрей Карпов – сотрудник ООО «СиПроВер», разработчик статического анализатора кода PVS-Studio;
- доклад посвящен состоянию дел в области статического анализа Си/Си++ кода;
- это не сравнение инструментов, а взгляд на развитие отрасли.

Статический анализ

это просмотр исходного кода разработчиком в тех местах, где по мнению статического анализатора присутствует неверное оформление или ошибка.

	Время обнаружения дефекта				
Время внесения дефекта	**Выработка требований**	**Проектирование архитектуры**	**Кодирование**	**Тестирование**	**После выпуска ПО**
Выработка требований	1	3	5-10	10	10-100
Проектирование архитектуры	-	1	10	15	25-100
Кодирование	-	-	1	10	10-25

Статический анализ

Два основных направления статического анализа

Поддержка стандарта кодирования, принятого в компании	Поиск ошибок в коде
• отступы; • именование переменных; • правила использование комментариев; • …	• неинициализированные переменные; • всегда ложные условия; • T *p = new T[n]; /* code */ free(p); • …

Часто «дискредитируют» статический анализ.

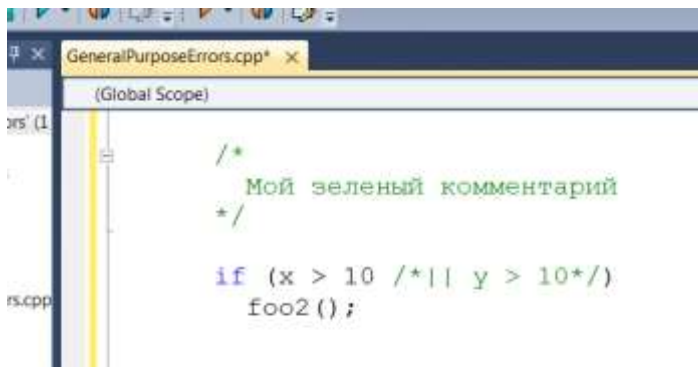
Устаревают стандарты кодирования А все что не развивается, то умирает

- среды разработки подсказывают тип переменных, раскрашивают код, помогают в его форматировании;
- компиляторы выдают все больше предупреждений;
- язык развивается, появляются новые технологии программирования;
- многие правила уже не актуальны.

Application Developer Days

Классическое. Не используйте комментарии в стиле C: /* … */

Стандарт MISRA предостерегает от использования комментариев в стиле /* … */. Комментарии плохо заметны. Сложнее разобраться в коде, легче допустить ошибку.



Вернее **были** плохо заметны.

Не актуально

Application Developer Days

Есть просто устаревшее

А вот еще одна из рекомендаций MISRA, реализованная в Parasoft C++test:

(misra-008) Do not use wide string literals

Не используйте строки в формате wchar_t.

Не рекомендуется:
```
wchar_t* x = L"Fred"; // нарушение
```

Рекомендуется:
```
char* x = "Fred";
```

Пора рекомендовать наоборот!

Application Developer Days

Привет из 1992 года…

В Pasrasoft C++test присутствует правило, взятое из стандарта кодирования 1992 года:

(ellemtel_rule-01)
Include files in C++ always have the file name extension ".hh".

В C++ файлах должны использоваться заголовочные файлы с расширением ".hh". Не сказал бы, что это повсеместно прижилось…

Не рекомендуется:
#include "MyClass.h" // нарушение

Рекомендуется:
#include "MyClass.hh"

Не интересно

С некоторыми правилами не поспоришь - но толку от них никакого

Pasrasoft C++test: (ellemtel_rule-37) Do not use 'magic numbers'

Не используйте магические числа. Вообще не используйте! Делайте enum или константы.

Не рекомендуется:
```
float rgb[3]; // нарушение
n = sizeof(rgb) / sizeof(rgb[0]) // нарушение
```

Рекомендуется:
```
const size_t RgbArraySize = 3;
float rgb[RgbArraySize];
enum color { RED = 0, BLUE = 1, GREEN = 2 };
n = sizeof(rgb) / sizeof(rgb[RED]);
```

Формально все верно.
Но нерациональные трудозатраты при сомнительной пользе.

Application Developer Days

Опасно использовать адресную арифметику - не поспоришь

В C++Test имеется проверка относящаяся к MISRA:

(misra-101) Do not use pointer arithmetic.

Не используйте адресную арифметику.

Не рекомендуется:

```
int* p;
p++; // нарушение
int *x = p+5; // нарушение
```

Рекомендуется:

не использовать ☺

Спасибо за рекомендацию.
А что мне с ней делать?

Application Developer Days

Поиск ошибок тоже устаревает

- ошибки все лучше выявляются компилятором;
- многим разработчикам уже не интересны ошибки, связанные с FAR указателями или массивами больше 64-килобайт;
- мало внимания уделяется использованию библиотек (stl, boost, MFC, Windows API).

Хорошее, но устаревшее правило для диагностики

Неправильно:

```
class ClassX {
  …
  ClassX(const ClassX x) { m_v = x.m_v; } // нарушение
};
```

При вывозе конструктора копирования возникнет вечный цикл.

Правильно:

```
class ClassX {
  …
  ClassX(const ClassX &x) { m_v = x.m_v; }
};
```

Давно реализовано в Visual C++. Даже не компилируется

```
ClassX(const ClassX x) { m_v = x.m_v; }
```

VS2005 сообщает:
error C2652: 'ClassX' :
illegal copy constructor: first parameter must not be a 'ClassX‘

Хорошо, что это правило №1063 есть и в PC-Lint.
На свете еще много неполноценных компиляторов для микроконтроллеров.

Но разработчику, использующему Visual Studio 2005/2008/2010 это уже не интересно.

Application Developer Days

Пример хорошей проверки

PC-Lint: №682
В примере «array» является указателем , а не массивом.
В 32-битной программе получим значение 1, а не 3.

Неправильно:
```
void Foo26(float array[3])
{
  size_t n = sizeof(array) / sizeof(array[0]); //нарушение
```

Правильно:
```
void Foo26(float *array, size_t array_size)
{
  size_t n = array_size;
```

Пример хорошей проверки

Parasoft C++test: sa-100_DoNotTreatArraysPolymorphically

```
class Class5_Base {
  int a;
};

class Class5 : public Class5_Base {
  int b;
};

void Process5(Class5_Base *p, size_t n) {
  for (size_t i = 0; i != n; ++i)
    p[i].a = 22;
}

Class5 X[5];
Process5(X, 5); //нарушение
```

Через указатель на базовый класс нельзя работать с массивом производных классов, так как классы имеют разный размер.

Пример хорошей проверки

PVS-Studio: V512: A call of the 'memset' function will lead to a buffer overflow or underflow.

Неправильно:

```
MD5Context *ctx;
...
memset(ctx, 0, sizeof(ctx)); // нарушение

if (memcmp(this, &other, sizeof(Matrix4) == 0)) // нарушение
```

Правильно:

```
MD5Context *ctx;
...
memset(ctx, 0, sizeof(*ctx));

if (memcmp(this, &other, sizeof(Matrix4)) == 0)
```

Пример хорошей проверки

PC-Lint: №1546.
Нельзя бросать исключение из деструктора.

```
~ClassX()
 {
   if (!m_init)
     throw std::exception("Error"); //нарушение
   free(m_p1);
   free(m_p2);
 }
```

Пример хорошей проверки

Parasoft C++test: ecpp-25_ZeroConversionProblem

```
void func(float, int *) { }
void func(float, int) { }

void F()
{
Неправильно:
  func(1.0f, 0); //нарушение
  func(1.0f, NULL); //нарушение

Правильно:
  func(1.0f, nullptr);
}
```

Неожиданное поведение кода из-за выбора не той перегруженной функции.

Пример хорошей проверки

PVS-Studio: V501. There are identical sub-expressions to the left and to the right of the '||' operator.

Слева и справа от оператора ||, &&, ==, <=, … находятся одинаковые подвыражения, не имеющие побочных эффектов.

Неправильно:

```
class Foo {
  int iChilds[2];
  ...
  bool hasChilds() const { return(iChilds > 0 || iChilds > 0); } // нарушение
}
```

Правильно:

```
bool hasChilds() const { return(iChilds[0] > 0 || iChilds[1] > 0); }
```

Пример хорошей проверки

Code Analysis for C/C++:

Неправильно:
```
HRESULT hr = CoInitialize(NULL);
if (!hr) //нарушение
```

Правильно:
```
if (FAILED(hr))
```

C6217: неявное приведение между семантически различными целочисленными типами: проверка HRESULT с "not".

Неправильно:
```
HRESULT hr;
hr = TRUE; //нарушение
```

Правильно:
```
#define S_OK ((HRESULT)0L)

hr = S_OK;
```

C6225: неявное приведение между семантически различными целочисленными типами: присвоение HRESULT значения 1 или TRUE.

Application Developer Days

Сам придумал
Нигде аналогичной проверки не видел

PVS-Studio: V517. The use of 'if (A) {...} else if (A) {...}' pattern was detected. There is a probability of logical error presence.

Неправильно:

```
if(radius < THRESH * 5)
  *yOut = THRESH * 10 / radius;
else if (radius < THRESH * 5) //нарушение
  *yOut = -3.0f / (THRESH * 5.0f) * (radius - THRESH * 5.0f) + 3.0f;
else
  *yOut = 0.0f;
```

Повторяющиеся условия в конструкции вида "else if".

Правильно:

Затрудняюсь привести верный вариант.

Состояние дел в области статического анализа

- Многое из нового делается «для галочки»:
  - Qt Best Practices Rules;
  - /Wp64 в Visual C++ – это **10%-15%** от того, что позволяет обнаружить Viva64.
- Хромает поддержка нового и C++0x в частности.

Application Developer Days

Состояние дел в области статического анализа

- Gimpel PC-Lint;
- Parasoft C++test;
- Coverity;
- Klocwork;
- Visual Studio: Code Analysis for C/C++;
- PVS-Studio.

PVS-Studio:

- Viva64 – выявление 64-битных ошибок;
- VivaMP - выявление параллельных ошибок;

- **в разработке современный статический анализатор общего назначения.**

Скачать PVS-Studio:
http://www.viva64.com/ru/pvs-studio/download/

Application Developer Days

Выводы

- многие правила статических анализаторов безнадежно устарели;
- если в новой версии анализатора меняется только интерфейс программы, но не правила анализа, то продукт перестает быть актуальным;
- но развитие языка и библиотек всегда сохраняет актуальность статического анализа;
- самые крутые компиляторы проигрывают самым крутым инструментам анализа.

Application Developer Days

Вопросы ?

Контактная информация:

Карпов Андрей Николаевич
к.ф.-м.н., технический директор
ООО «Системы программной верификации»
Сайт: http://www.viva64.com/ru/main/
E-mail: karpov@viva64.com
Тел.: +7 (4872) 38-59-95 (GMT + 03:00)
Twitter: https://twitter.com/Code_Analysis