Статический анализ кода C++

Для меня красота C++ заключается в первую очередь во вседозволенности и ужасающей мощи языка. Мы можем работать с памятью так же плотно, как и в C, и в то же время имеем такие средства, как шаблоны и STL, превосходящие по уровню абстрактности все, что я когда-либо видел. Плата за это соответствующая - не всегда вразумительные ошибки компилятора (попробуйте забыть поставить точку с запятой после определения класса), очень большой срок подготовки и обучения программистов, но самое главное - некоторые баги становятся заметными только во время выполнения программы.

О проблеме

Мы хотим больше полезных ворнингов до выполнения наших программ. Одно из средств получения желаемого - статический анализ кода. Статический - значит, не запуская программу. Интересны не только вероятные ошибки, случаи undefined behaviour, утечки памяти, но и вещи вроде недоступности/неиспользуемости кода, рекомендации по повышению интуитивности стиля программирования. Средства получения метрик ПО, добываемых методами статического анализа в этой статье не рассматриваются. Замечания, связанные со стандартами программирования (фигурная скобочка должны находиться на отдельной строчке, ууууу!!) тоже не интересны. Критерии оценки простые - количество и полезность находимых багов, простота использования (в частности, отсутствие требований модификации кода), бесплатность/разумная цена/хороший кряк. Проводи первичный обзор и выдаем на-гора пачку ссылок:

- Static_code_analysis
- List
- Статья Скотта Мейерса об анализаторах
- тема на Stack overflow о бесплатных анализавторах

Найденные штуки

Педантичные ключи gcc

ман-статья про все эти ключи

- *-Wall* включает все ворнинги, среди которых совместимость с новым стандартом, границы массива (по-моему, не пашет, хотя говорят, что будет работать с -O2), valatile/register переменные (сообщит, что ему пофигу на твои умные слова и что *register*, а что нет, он будет решать сам), точки следования (i++ + ++i)
- *-Wextra* - еще пачка ворнингов типа пустых тел в if'ах, сравнение signed и unsigned
- *-pedantic* – следование ISO C++ стандарту
- *-Weffc++* must have опция. Не включается via Wextra или Wall и содержит проверку рекомендаций Скотта Мейерса:

Item 11: Define a copy constructor and an assignment operator for classes with dynamically allocated

memory.

```
Item 12:  Prefer initialization to assignment in constructors.
Item 14:  Make destructors virtual in base classes.
Item 15:  Have "operator=" return a reference to *this.
Item 23:  Don't try to return a reference when you must return an object.
Item 6:  Distinguish between prefix and postfix forms of increment and
decrement operators.
Item 7:  Never overload "&&", "||", or ",".
```

- -Woverloaded-virtual - перегрузка виртуальных фунций
- -Wctor-dtor-privacy - неиспользуемые классы – с приватным конструкторами и деструктором
- -Wnon-virtual-dtor – невиртуальный деструктор! Смотри-ка, сделали!
- -Wold-style-cast - приведение в стиле C - это плохо
- -Werr='тип ворнинга' - воспринимать ворнинг как еррор. Для настроящих самураем -Werr без параметров
- -Winit-self – int i = i;
- -Wunreachable-code - код, который никогда не будет выполнен

Cppcheck

Пожалуй, самая достойна из найденных программ. cpp check и его плагин для эклипсоидов. Распознает довольно много, находит следующие ошибки:

- некоторые memory leaks, например, отсутствие delete и delete[], отсутствие delete в деструкторе
- выход за границу массива
- exception'ы, бросаемые в деструкторе
- разыменование нулевого указателя
- разыменование после очистки памяти
- виртуальность деструктора базового класса
- использование одного и того же итератора для разных контейнеров
- разные более мелкие штуки

Проверялки для C без плюсов

Удивительно, но программисты на чистом Си, похоже, больше заботятся о статическом анализе. Тут мы имеем:

- http://www.cert.org/secure-coding/tools.html – список раз
- http://www.cs.cmu.edu/~aldrich/courses/654/tools/index.html – два

Будут ли они полезны плюсовым разработчикам? Если у вас есть код без классов и вы помните, какие программы на C++ не будут собираться компилятором C, то почему бы и нет?

Splint

- Вот такая штука для чистого C. Собирается не без усилий, но работает чистенько и много чего ищет - смотри мануал

Simian

- similarity analyzer – ищет дублирование кода

Vera++

- Vera++. В отличие от cppcheck, ориентированиа на проверку стиля. Имеет пополняемую базу правил. По умолчанию в базе много реально идиотстких штук типа “перед двоеточием должен быть пробел”. Единственная полезная возможность – запрет на использование using namespace в заголовочных файлах. Правила, однако можно писать и самому на языке Tcl. :)
- RATS -рассказывает страшилки про безопасность и buffer overflow-атаки
- CIL компилирует C в упрощенный C! Упрощенный C уже можно скармливать другим анализаторам

Непонятное, офтопное

- Oink требует Flex и Bison, долго догадывался.. (: А еще надо фиксить кучу ошибок при компиляции. Ниасилил скомпилять, а список фич отнюдь не внушительный
- Mozilla Dehydra – нечто, базирующееся на этом самом Oink'е
- Mozilla Pork ниасилил установку
- Еще есть компиляторы, которые предоставляют больше ворнингов на основе статического анализа. Например Rose
- Rational Purify - прославленная компания IBM Rational имеет свой набор инструментов для статического и динамического анализа. Буду рад, если кто-нибудь расскажем об этой программе, триальник для линукса у них оказался только для x64-архитектуры.

Триальное/крякнутое ПО

А мало его такого. Много анализаторов, стоящих много долларов и не имеют кряков. Вот пример такого анализатора, можно зайти на их сайт и попросить триальник Cleanscape

Что бы еще хотелось

Хотелось бы:

- Политику спецификации исключений как в Java.
- юзанье auto_ptr в контейнерах STL, разыменование auto_ptr после присвоения другому auto_ptr.

- using namespace в h-файлах
- размещающий delete без new или new без delete
- вызов delete[] для немассивов
- = вместо == в if'ах

From:
http://wiki.osll.ru/ - **Open Source & Linux Lab**

Permanent link:
http://wiki.osll.ru/doku.php/etc:users:yuri_v_katkov:cpp_static_anal

Last update: **2016/08/08 20:53**