Gem #135: Ошибочное выполнение - Часть 4

Автор: Bob Duff—AdaCore

Краткое содержание: Данный Gem - последний в серии публикаций на тему ошибочного выполнения. В нем обсуждается проектирования языка. Зачем нужно в **Ada** вообще ошибочное выполнение?

Давайте начнём…

Многие программисты полагают, что "оптимизаторы кода не должны изменять поведение программы". Многие также полагают, что "оптимизаторы кода не изменяют поведение программы". Оба утверждения неверны в присутствии ошибочности.

Таким образом, если ошибочность так плоха, почему дизайн языка **Ada** имеет ее? Конечно, проектировщик языка должен попытаться минимизировать сумму ошибочности. **Java** является примером языка, который сторонится ошибочности, но за это приходится платить ограничениями в языке. Это означает, что много полезных вещей невозможно или неосуществимо в **Java**: драйверы устройств, например. Существует также стоимость эффективности. **C** является примером языка, который имеет слишком много ошибочности. Каждая операция индексации массива потенциально ошибочна в **C**. (**C** называет его "undefined behavior - неопределенным поведением".)

Ada является где-нибудь промежуточным **Java** и **C** в этом отношении. Можно записать драйверы устройств в **Ada**, и пользовательские пулы хранения данных и другие вещи, которые требуют низкоуровневого доступа к машине.

Но по большей части, вещи, которые могут вызвать ошибочность, могут быть изолированы в пакетах - Вы не должны рассеивать их на всем протяжении программы как в **C**.

Например, для предотвращения висячих указателей попытайтесь держать вместе "`new`" и `Unchecked_Deallocations`, таким образом, чтобы их можно было анализировать локально. Универсальный пакет `Doubly_Linked_List` мог бы иметь ошибки висячего указателя в себе, но он может быть разработан так, чтобы клиенты не могли вызвать висячие указатели.

Другой способ предотвратить висячие указатели состоит в том, чтобы использовать пользовательские пулы хранения данных, которые позволяют освобождение всего пула сразу. Храните объекты "кучи" с подобным временем жизни в том же пуле. Могло бы казаться, что освобождение целого набора объектов, более вероятно, вызовет висячие указатели, но на самом деле совсем противоположное верно. С одной стороны, освобождение целого пула намного более просто, чем обход сложных структур данных, освобождающих отдельные записи одну за другой. С другой стороны, освобождая память в массе, вероятно, вызовет катастрофические отказы, которые могут быть зафиксированы как можно скорее. Наконец, пользовательский пул хранения данных может быть записан для обнаружения висячих указателей, например, при помощи служб операционной системы для маркировки освобожденных регионов как недоступные.

Обратите внимание, что **Ada 2012** имеет "Подпулы" (Subpools), которые делают пользовательские пулы хранения данных более гибкими.

Последний момент об ошибочности, который может быть удивительным, заключается в том, что он может идти назад во времени. Например:

```
if Count = 0 then
   Put_Line ("Zero");
end if;
Something := 1 / Count; -- could divide by zero
```

Если проверки подавлены, весь оператор "`if`", включая `Put_Line`, может быть удален оптимизатором кода. Рассуждение таково: если `Count` ненулевой, мы не хотим печатать "ноль". Если `Count` равен нулю, то он ошибочен, поэтому может произойти что угодно, в том числе не печатать "ноль".

Даже если `Put_Line` не был удален компилятором, он может выглядеть так (*прим.* то есть печати не будет), потому что «ноль» может быть сохранен в буфере, который никогда не сбрасывается, потому что некоторая более поздняя ошибка вызвала сбой программы.

Каждое утверждение об **Ada** должно пониматься как имеющее ", если только исполнение не является ошибочным" после него. В данном случае, "`Count = 0 возвращает True, если Count равен нулю`"», очевидно, истинно, но на самом деле это означает, что «`Count = 0 возвращает True, если Count равен нулю, если только выполнение не ошибочно, и в случае ошибочного выполнения может произойти все что угодно`».

Мораль: старайтесь избегать написания ошибочных (*прим.* подавляя проверки во время исполнения) программ.

Связанный со статьёй текст программы

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Bob Duff

AdaCore

Роберт (Боб) Андерсон Дафф - дизайнер языков программирования. Написал основные части Ada Language Reference Manual. При его участии разработаны и другие языки. Изучил десятки языков программирования.

Разработал, внедрил и/или внес большой вклад в десять компиляторов, три инструмента статического анализа и различные программные системы от реального времени/встроенные до параллельных и распределенных систем.

Эксперт в ведении программных проектов для своевременного их завершения.

Опыт работы в AdaCore:

Senior Software Engineer
AdaCore
2005 – настоящее время 14 лет
New York and Massachusetts

Ранее:

SofCheck, Inc,
Intermetrics,
Oak Tree Software, Inc.

Образование:

Carnegie Mellon University
Bachelor of Science (BS), Applied Mathematics with concentration in Computer Science, 3.6/4.0.
1982
Деятельность и сообщества: Explorer's Club

Last Updated: 10/13/2017
Posted on: 10/29/2012

Обсуждение…