## *Gem #132: Ошибочное выполнение - Часть 1*

**Автор:** Bob Duff—AdaCore

Краткое содержание: Многие разработчики, работающие с языком Ada, приходят в замешательство при виде термина «ошибочный», так как в контексте Ada данный термин означает не совсем то же, что в нормальном языке. «Ошибочный» в обыденности означает «неправильный». Но по отношению к Ada данный термин обозначает определенный тип «неправильности». В данном Gem будет прояснено значение данного термина в контексте работы с Ada.

**Давайте начнём…**

Ada очень хорошо справляется с требованием компиляторов обнаруживать ошибки во время компиляции или с ошибками во время выполнения. Тем не менее, есть некоторые виды ошибок, которые невозможно обнаружить. Справочное руководство называет такие ошибки «ошибочным исполнением».

RM-1.1.5 (10) определяет термин:

```
...[T]he implementation need not detect such errors either prior to or
during run time. ...[T]here is no language-specified bound on the
possible effect of erroneous execution; the effect is in general not
predictable.

... [T] его реализация не должна обнаруживать такие ошибки ни до, ни во
время выполнения. ... [T] здесь не указана языковая граница возможного
эффекта ошибочного выполнения; эффект в целом не предсказуем.
```

Примером "ошибочного выполнения" является подавление проверки результата выполнения участка кода, в котором возникла ошибка, и выполнение которого завершается неудачно. Например:

```
pragma Suppress (All_Checks); -- Or use the -gnatp switch
...
A (X) := A (X) + 1;
```

Если X выходит за рамки индексов масива, то приведенное выше приводит к «ошибочному выполнению». Это означает, что программа может делать все, что угодно. Объясняя значение ошибки, люди часто любят говорить о впечатляющих катастрофах: "это может стереть ваш системный диск!- Твоя клавиатура может загореться!" "Порождение демонов!"

Я думаю, что это несколько вводит в заблуждение. Во-первых, если вы работаете под управлением операционной системы с надлежащей защитой, ошибочное выполнение не приведет к стиранию системного диска. Дело в том, что Ada не гарантирует этого, но операционная система делает. Аналогично, Ada не мешает вашей клавиатуре загораться, но мы надеемся, что производитель компьютера будет Вас защищать от этого.

Одна из катастроф, которая на самом деле может произойти, заключается в том, что приведенный выше код перезапишет произвольное расположение памяти. Какая бы переменная там ни хранилась, она может быть уничтожена. Это катастрофа, потому что для отслеживания таких ошибок могут потребоваться часы или даже дни. Если Вам повезет, вы сразу же получите ошибку использования не существующего сегмента памяти, что значительно облегчит поиск ошибки.

Но самое худшее - это не пожары на клавиатуре, не разрушенные переменные и не что-то еще впечатляющее. Самое худшее, что может вызвать ошибочное выполнение, - это то, что программа будет вести себя именно так, как вы хотели, возможно, потому, что уничтоженное место памяти не

использовалось ни для чего важного. Так в чем проблема? Если программа работает, почему мы должны беспокоиться, если какой-то педантичный языковой юрист говорит, что это ошибочно?

Чтобы ответить на этот вопрос, обратите внимание на этот общий метод отладки: у вас есть большая программа. Вы вносите небольшие изменения (чтобы исправить ошибку, добавить новую функцию или просто сделать код чище- это естественно приведет к перераспределению выделенной памяти для большинства объектов программы). Вы проводите регрессионные тесты, и что-то не получается. Вы выводите, что причиной новой ошибки является небольшое изменение, которое вы сделали. Поскольку изменение мало по сравнению с размером всей программы, легко понять, в чем проблема.

С ошибками этот метод отладки не работает. Кто-то написал вышеуказанную ошибочную программу (ошибочную, если X выходит за рамки допустимых значений индекса). Это сработало просто отлично. Затем, год спустя, вы делаете некоторые изменения, совершенно не связанные с утверждением `" A (X) := A (X) + 1;".` Это заставляет вещи перемещаться в памяти, так что теперь важные данные уничтожаются. Вы больше не можете предполагать, что ваше изменение вызвало ошибку; вы должны рассмотреть весь текст программы.

Мораль этой истории такова: не пишите ошибочных программ, не подавляйте обработку ошибок – производите их собственную обработку.

**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: 9/10/2012*

**Обсуждение…**