Gem #23: Null считается вредным. Предотвращение значение null, когда подобное нежелательно

Автор: Bob Duff, AdaCore

Краткое содержание: Gem Ada #23 - синтаксис «not null» позволяет программам, написанным на языке Ada 2005, предотвращать «null» значение доступа в случаях, когда подобное нежелательно. Этот новый вид синтаксиса позволяет создавать полезную документацию.

Давайте начнем…

Ada, как и многие языки, определяет специальное значение ‘null’ для типов доступа. Все значения типа доступ обозначить какой-то объект указанного типа, кроме NULL, который не определяет ни одного объекта. Значение null может использоваться в качестве специального флага. Например, односвязный список может быть завершен с значением NULL. Функция поиска может возвращать значение NULL означает “не найдено”, предполагая, что результат имеет Тип доступа:

```
type Ref_Element is access all Element;
Not_Found : constant Ref_Element := null;
function Lookup (T : Table) return Ref_Element;
--  Returns Not_Found if not found.
```

Альтернативный дизайн для Lookup будет вызывать исключение:

```
Not_Found : exception;
function Lookup (T : Table) return Ref_Element;
--  Raises Not_Found if not found.
--  Never returns null.                  <--  Ada 95 comment.
```

Никакой проект не лучше во всех ситуациях; это зависит частично от того, полагаем ли мы, что “не найденная” ситуация исключительная.

Очевидно, что клиент, вызывающий Lookup, должен знать, может ли он возвращать null, и если да, то что это значит. В целом, хорошо документировать, могут ли вещи быть нулевыми или нет, особенно для формальных параметров и результатов функции. В Ada 95, мы делаем это с комментариями. В Ada 2005, мы можем использовать “not null” синтаксис:

```
function Lookup (T : Table) return not null Ref_Element;  --  Ada 2005
```

В общем случае лучше использовать язык, соответствующий документации, когда это возможно, а не комментариям, поскольку проверка времени компиляции и/или времени выполнения может помочь убедиться, что “документация” действительно верна. С комментариями, есть большая опасность, что комментарий станет ложным во время обслуживания, и ложная документация, очевидно, угроза.

Во многих, возможно, большинстве случаев, null-это просто опасность отключения. Это хорошая идея, чтобы положить в “not нуль”, когда это возможно. На самом деле, хороший аргумент может быть сделан, что “not null” должно быть значение по умолчанию, с дополнительным синтаксисом требуется, когда null требуется. Так работает SML, например — вы не получите никакого специального значения типа null, если не попросите его. Конечно, Ada 2005 должен быть совместим с Ada 95, так что “not null” не может быть по умолчанию для Ada.

Одно предостережение: объекты доступа по умолчанию-инициализируются в null, так что если у вас “not null” объект (или компонент), вам лучше явно инициализировать его, или вы получите Constraint_Error. По этой причине, “not null" чаще всего полезен для параметров и результатов функции.

Связанный исходный код

Attached Files

```
Title: Null Considered Harmful

Author: Bob Duff, AdaCore

Abstract:
The "not null" syntax allows an Ada 2005 program to prevent access values
from being null in cases where the null value is undesirable. This new syntax
helps provide useful documentation.

Ada, like many languages, defines a special 'null' value for access types.
All values of an access type designate some object of the designated type,
except for null, which does not designate any object.
The null value can be used as a special flag.  For example, a singly-linked
list can be null-terminated.  A Lookup function can return null to mean
"not found", presuming the result is of an access type:


   type Ref_Element is access all Element;
   Not_Found : constant Ref_Element := null;
   function Lookup (T : Table) return Ref_Element;
   --  Returns Not_Found if not found.



An alternative design for Lookup would be to raise an exception:

   Not_Found : exception;
   function Lookup (T : Table) return Ref_Element;
   --  Raises Not_Found if not found.
   --  Never returns null.                 <--  Ada 95 comment.


Neither design is better in all situations; it depends in part on whether we
consider the "not found" situation to be exceptional.

Clearly, the client calling Lookup needs to know whether it can return null,
and if so, what that means.  In general, it's a good idea to document whether
things can be null or not, especially for formal parameters and function
results.  In Ada 95, we do that with comments.  In Ada 2005, we can use the
"not null" syntax:

   function Lookup (T : Table) return not null Ref_Element;  --  Ada 2005


In general, it's better to use the language proper for documentation,
when possible, rather than comments, because compile-time and/or
run-time checks can help ensure that the "documentation" is actually
true.  With comments, there's a greater danger that the comment
will become false during maintenance, and false documentation is obviously
a menace.
```

```
In many, perhaps most, cases, null is just a tripping hazard.
It's a good idea to put in "not null" when possible.
In fact, a good argument can be made that "not null" should be
the default, with extra syntax required when null _is_ wanted.
This is the way SML works, for example -- you don't get any
special null-like value unless you ask for it.
Of course, Ada 2005 needs to be compatible with Ada 95,
so "not null" cannot be the default for Ada.

One word of caution: access objects are default-initialized to null,
so if you have a "not null" object (or component) you had better
initialize it explicitly, or you will get Constraint_Error.
"Not null" is more often useful on parameters and function results,
for this reason.package Example is

   type Element is limited private;
   type Ref_Element is access all Element;

   type Table is limited private;

   Not_Found : constant Ref_Element := null;
   function Lookup (T : Table) return Ref_Element;
   --  Returns Not_Found if not found.

   Not_Found_2 : exception;
   function Lookup_2 (T : Table) return not null Ref_Element;
   --  Raises Not_Found_2 if not found.

   procedure P (X : not null Ref_Element);

   procedure Q (X : not null Ref_Element);

private
   type Element is limited
      record
         Component : Integer;
      end record;
   type Table is limited null record;
end Example;

package body Example is

   An_Element : aliased Element;

   function Lookup (T : Table) return Ref_Element is
   begin
      --  ...
      return Not_Found;
   end Lookup;

   function Lookup_2 (T : Table) return not null Ref_Element is
   begin
      --  ...
      raise Not_Found_2;

      return An_Element'Access;
      -- suppress error: 'missing "return" statement in function body'
   end Lookup_2;

   procedure P (X : not null Ref_Element) is
   begin
      X.all.Component := X.all.Component + 1;
   end P;
```

```
   procedure Q (X : not null Ref_Element) is
   begin
      for I in 1..1000 loop
         P (X);
      end loop;
   end Q;

   procedure R is
   begin
      Q (An_Element'Access);
   end R;

end Example;
```

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

Обсуждение…