Gem #44: Проверка доступности (Часть III)

Автор: Bob Duff, AdaCore

Краткое содержание: Gem Ada #44 - С помощью 'Unchecked_Access можно обходить правила доступа, а контролируемые типы помогают укротить эту опасную функцию.

Давайте начнем…

В Gems Ada #1 и #2 (*прим.* также в Части II – Gem Ada #41) мы показали, как правила доступности помогают предотвратить висячие (оборванные) ссылки, гарантируя, что ссылки не могут указывать с более долгоживущих областей на более короткие. Но что если Вы хотите сделать это?

В некоторых случаях необходимо хранить ссылку на локальный объект в глобальной структуре данных. Вы можете сделать это с помощью "Unchecked_Access" вместо "Access". "Unchecked" в имени напоминает вам, что вы обходите обычные правила доступности. Чтобы предотвратить появление висячих ссылок, необходимо удалить значение ссылки из глобальной структуры данных, прежде чем покинуть область действия объекта.

Что касается любой небезопасной функции, это хорошая идея, чтобы инкапсулировать 'Unchecked_Access, а не рассеивать его вокруг программы. Это можно сделать с помощью ограниченных управляемых типов. Идея заключается в том, что инициализация устанавливает указатель на объект в некоторой глобальной структуре данных, а Finalize удаляет значение ссылки непосредственно перед тем, как она становится висячей ссылкой.

Вот пример. Предположим, что нет никаких задач и объектов, выделенных в куче – в противном случае нам понадобится более сложная структура данных, такая как двусвязный список, с блокировкой. Мы сохраняем стек объектов, реализованных в виде связанного списка через Stack_Top и прикованных через компонент Prev. Все вхождения 'Unchecked_Access инкапсулируются в пакет объектов, и клиенты объектов (таких как Main, ниже в конце текста программы) могут свободно объявлять объекты, не беспокоясь о висячих указателях. Stack_Top никогда не может болтаться, потому что Finalize очищает, даже в случае исключений и прерывает.

Обратите внимание ,что 'Unchecked_Access применяется к формальному параметру типа Object, который является законным, потому что формально типы с тегами определяются как псевдонимы. Обратите внимание, что Print_All_Objects не имеет видимости объектов, которые он печатает.

Эта программа печатает:

```
Inside Nested:
  That_Object
  This_Object
After Nested returns:
  This_Object
```

Обратите внимание, что этот_объект не печатается вторым вызовом Print_All_Objects, потому что он больше не существует в это время.

```
private with Ada.Finalization;
package Objects is

   type Object (Name : access constant String) is limited private;
   --  The Name is just to illustrate what&apos;s going on by printing it out.
```

```
   procedure For_All_Objects (Action : not null access procedure (X : Object));
   --  Iterate through all existing Objects in reverse order of creation,
   --  calling Action for each one.

   procedure Print_All_Objects;
   --  Print out the Names of all Objects in reverse order of creation.

   --  ... other operations

private

   use Ada;

   type Object (Name : access constant String) is new Finalization.Limited_Controlled
with
      record
         --  ... other components
         Prev : access Object := null; -- previous Object on the stack
      end record;

   procedure Initialize (X : in out Object);
   procedure Finalize (X : in out Object);

end Objects;

with Ada.Text_IO;
package body Objects is

   Stack_Top : access Object := null;

   procedure Initialize (X : in out Object) is
   begin
      --  Push X onto the stack:
      X.Prev := Stack_Top;
      Stack_Top := X&amp;apos;Unchecked_Access;
   end Initialize;

   procedure Finalize (X : in out Object) is
   begin
      pragma Assert (Stack_Top = X&amp;apos;Unchecked_Access);
      --  Pop X from the stack:
      Stack_Top := X.Prev;
      X.Prev := null;  --  not really necessary, but safe
   end Finalize;

   procedure For_All_Objects (Action : not null access procedure (X : Object)) is
      --  Loop through the stack from top to bottom.
      Item : access Object := Stack_Top;
   begin
      while Item /= null loop
         Action (Item.all);
         Item := Item.Prev;
      end loop;
   end For_All_Objects;

   procedure Print_All_Objects is
      --  Iterate through the stack using For_All_Objects, passing
      --  Print_One_Object to print each one.
      procedure Print_One_Object(X : Object) is
      begin
         Text_IO.Put_Line ("  " & X.Name.all);
      end Print_One_Object;
   begin
      For_All_Objects (Print_One_Object&amp;apos;Access);
   end Print_All_Objects;
```

```
end Objects;

with Ada.Text_IO; use Ada;
with Objects; use Objects;
procedure Main is

   This_Object : Object (Name => new String&apos;("This_Object"));

   procedure Nested is
      That_Object : Object (Name => new String&apos;("That_Object"));
   begin
      Text_IO.Put_Line ("Inside Nested:");
      Print_All_Objects;
   end Nested;

begin
   Nested;
   Text_IO.Put_Line ("After Nested returns:");
   Print_All_Objects;
end Main;
```

Связанный исходный

Attached Files отсутствуют

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

Об авторе

Нет сведений.

Last Updated: 10/13/2017
Posted on: 9/15/2008

Обсуждение…

3 ответов на "Gem Ada # 44: Проверка доступности (часть III)"

1. 21 октября 2008 года в 15:12

Брайан (Brian) сказал:

Гарантировано ли, что `Finalize` будет вызываться в обратном порядке `Initialize`? Если в одной и той же области было создано несколько объектов, гарантировано ли всегда подтверждение `Assert`?

- Кроме того, поскольку это «Еженедельные статьи Gem Ada», можно ли будет показывать соответствующую дату с каждой из них? Таким образом, я бы знал, насколько далеко позади (по-видимому, по крайней мере, 5 недель).

2. 27 октября 2008 года в 18:34

Кэти Фэрламб (Kathy Fairlamb) сказала:

Я позволю коллегам ответить на первую часть вашего вопроса, Брайан.

Относительно Вашего предложения, чтобы связать дату с каждой статьёй Gem Ada, мы намеренно не датируем статьи Gem Ada, поскольку они, как предполагается, не представляют вида

прогрессивного курса языка программирования Ada, а скорее набор интересных подсказок, представленных без определённого порядка. Поэтому Вы не "отстаёте". :-)

3. 27 октября 2008 года в 22:48

Боб Дафф (Bob Duff) сказал:

> Гарантировано ли, что `Finalize` будет вызываться в обратном порядке

> Инициализировать? Если несколько объектов были созданы в одной и той же области,

> Гарантируется, что всегда будет проходить проверка?

Да, и да.

- Боб