# *Gem #11: Лимитированные типы в Ада 2005 – Функции конструктора, часть 2*

**Автор:** : Bob Duff, AdaCore

Краткое содержание: Gem Ада #11 - Данный пример демонстрирует, как функции конструктора можно применять в разных контекстах для создания лимитированных объектов на месте.

**Давайте начнем…**

Мы ранее видели примеры функций конструктора для ограниченных типов, подобных этому:

```
package P is
   type T (<>) is limited private;
   function Make_T (Name : String) return T; -- constructor function
private
   type T is new Limited_Controlled with
      record
         …
      end record;
end P;

package body P is
   function Make_T (Name : String) return T is
   begin
      return (Name => To_Unbounded_String (Name), others => <>);
   end Make_T;
end P;

function Make_Rumplestiltskin return T is
begin
    return Make_T (Name => "Rumplestiltskin");
end Make_Rumplestiltskin;
```

Полезно рассмотреть различные контексты, в которых могут быть вызваны эти функции. Мы уже видели такие примеры как:

```
Rumplestiltskin_Is_My_Name : T := Make_Rumplestiltskin;
```

в этом случае ограниченный объект создан непосредственно в автономном объекте. Этот объект будет завершен каждый раз, когда окружающий контекст оставляют.

Мы также можем:

```
procedure Do_Something (X : T);

Do_Something (X => Make_Rumplestiltskin);
```

Здесь результат функции строится непосредственно в формальном параметре X Do_Something. X будет завершен, как только мы вернемся из Do_Something.

Мы можем выделить инициализированные объекты на "куче":

```
type T_Ref is access all T;
Global : T_Ref;
```

```
procedure Heap_Alloc is
   Local : T_Ref;
begin
   Local := new T'(Make_Rumplestiltskin);
   if … then
      Global := Local;
   end if;
end Heap_Alloc;
```

Результат функции создан непосредственно в выделенном "куче" объекте, который будет завершен, тогда как объект T_Ref сохранится в куче (еще долго после того, как функция Heap_Alloc завершится).

Мы можем создать другой ограниченный тип с компонентом типа T и использовать агрегат:

```
type Outer_Type is limited
   record
      This : T;
      That : T;
   end record;

Outer_Obj : Outer_Type := (This => Make_Rumplestiltskin,
                           That => Make_T (Name => ””));
```

Как обычно, результаты функции строятся на месте, непосредственно в Outer_Obj.This и Outer_Obj.That, без копирования.

Тот самый случай, когда мы не можем вызывать такие функции конструктора в операторе присваивания:

```
Rumplestiltskin_Is_My_Name := Make_T(Name => ""); -- Illegal!
```

Данный код недопустим, потому что операторы присваивания включают копирование (прим. переводчика: в языке C++ таких случаях применяется специальный конструктор объекта). Аналогично, мы не можем скопировать ограниченный объект в некоторый другой объект:

```
Other : T := Rumplestiltskin_Is_My_Name; -- Illegal!
```

**Обсуждение…**