CUARTO CURSO
SISTEMAS EN TIEMPO REAL

Escuela Politécnica Superior
Universidad de Huelva

Departamento de Ing. Electrónica,
Sistemas Informáticos y Automática

Práctica 2: Programación de Tipos Abstractos de Datos

Manuel Sánchez Raya
Versión 0.1
10 de Noviembre de 2014

ÍNDICE

2.1. Introducción.

Con esta práctica se pretende que el alumno:

- Construya una unidad reutilizable e implemente un tipo abstracto de datos con la semántica de las colas.
- Trabaje con tipos puntero y controle la recolección de memoria basura.

2.2.- Tareas a Realizar.

Escribir un paquete de nombre Colas que se ajuste a la especificación siguiente:

```
generic
  type Elementos is private;
package Colas is
  type Cola is limited private;

  procedure Poner(el_Elemento: Elementos; en_la_Cola: in out Cola);
  procedure Quitar(un_Elemento: out Elementos;de_la_Cola:in out Cola);
  function Esta_Vacía (La_Cola: Cola) return Boolean;
  function Esta_Llena (La_Cola: Cola) return Boolean;
  procedure Copiar (Origen: Cola; Destino:in out Cola);
  function "="(La_Cola, Con_La_Cola: Cola) return Boolean;

private
...
end Colas;
```

El tipo Cola se debe implementar como una lista lineal simplemente enlazada. Para ello será necesario trabajar con tipos puntero. Siempre que se trabaja con punteros hay que prestar atención a la recolección de la memoria basura.

Algunos compiladores de Ada disponen de un recolector para la memoria basura, la cual es liberada automáticamente cuando no hay ninguna referencia a ella o cuando el tipo acceso que se utiliza para crearla deja de ser visible. La forma de trabajo del recolector no está definida en la norma por lo que depende de cada compilador.

La periodicidad, no predecible, del funcionamiento del recolector introduce demoras en el funcionamiento global del sistema y esta sobrecarga es inadmisible en algunos sistemas de tiempo real.

Para dar respuesta a este problema, la norma RM95 define un procedimiento genérico para liberar memoria en los instantes especificados por la aplicación. Este procedimiento se llama Ada.Unchecked_Deallocation como se ha indicado en los ejemplos de teoría.

Para probar este paquete puede utilizarse el programa siguiente:

```
with Ada.Text_Io, Colas; use Ada.Text_Io;

procedure Principal is
  package Colas_de_Integer is new Colas(Integer);
```

```
  use Colas_de_Integer;
  Practica_no_Funciona: exception;

  C1, C2, C3: Cola;
  E: Integer;
begin
  for I in 1..10 loop
    Poner(I, C1);
  end loop;
  for I in 11..20 loop
    Poner(I, C2);
  end loop;
  if C1 /= C1 then raise Practica_no_Funciona;
  end if;
  if C1 = C2 then raise Practica_no_ Funciona;
  end if;

  Poner(1, C3); Copiar (C2, C3);
  if C2 /= C3 then raise Practica_no_Funciona;
  end if;

  while not Esta_Vacia (C3) loop
    Quitar(E, C3); Poner(E, C1);
  end loop;
  while not Esta_Vacía (C2) loop
    Quitar(E, C2);
  end loop;
  for I in 1..20 loop
    Poner(I, C2);
  end loop;
  if C1 /= C2 then raise Practica_no_Funciona;
  end if;

  for I in 1..1e7 loop
  begin
    Poner(1, C1); Quitar(E, C1);
  exception
    when Storage_Error =>
      Put_Line("Practica no Funciona.");
      Put_Line ("La funcion Quitar no libera memoria");
    end;
  end loop;
  Put_Line ("Practica correcta");
exception
  when Practica_no_Funciona =>
    Put_Line ("Practica no Funciona");
  when Storage_Error =>
    Put_Line ("Práctica no Funciona.");
    Put_Line  ("Posible  recursión infinita");
end Principal;
```