Memoria Práctica 2

Sistemas en Tiempo Real

Implementación de un TAD Cola en lenguaje ADA

Almudena García Jurado-Centurión

Índice

Enunciado

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.

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;
```

1. Especificación

La especificación se escribe en un fichero con extensión .ads

En este fichero especificación definiremos las cabeceras del paquete indicado en el enunciado, junto al tipo Cola y los tipos auxiliares necesarios, los cuales irán en la sección *private* del paquete.

Para crear la Cola usaremos dos tipos: *register,* que contendrá los registros de la cola, con el dato y el puntero al siguiente elemento; *link*, que será un apuntador a *register;* y el propio tipo *Cola,* que incluirá los apuntadores al primer registro y último de la cola.

Los tipos *link* y *register* estarán definidos de la siguiente manera:

```
type register;

type link is access register;
type register is tagged
   record
      data: Elementos;
      next: link;
   end record;
```

Para poder definir el tipo *link* como apuntador a *register*, y usar éste en la definición de *register* tenemos que declarar el tipo *register* previamente a su definición.

Una vez hecho esto, podemos definir el tipo *Cola*, con los apuntadores a la primera posición y última de la cola, cada una de las cuales corresponde a un *register.*

```
type Cola is tagged
   record
      first: link;
      last: link;
   end record;
```

Finalmente, definimos una función privada para liberar la memoria de los registros eliminados, usando el paquete *Ada.Unchecked_Deallocation,* y los parámetros correspondientes al puntero y al registro a liberar.

```
procedure Liberar_Register is new Ada.Unchecked_Deallocation
  (register, link);
```

2. Implementación

Una vez definido el paquete en el fichero .ads, pasamos a su implementación.

Para implementar el paquete, copiamos la especificación añadiendo la palabra **body** después de **package,** dentro de la cabecera; y añadiendo las definiciones de las funciones dentro de él.

Para definir las funciones y procedimientos, añadimos las cabeceras terminadas en la palabra **is** y, después de ello, la implementación de dicha función o procedimiento.

Ahora iremos indicando las definiciones de las diferentes funciones y procedimientos:

- **Poner:** Procedimiento para insertar un elemento dentro de la cola indicada por parámetro. El elemento se insertará en la última posición de la cola.

Este procedimiento recibe dos parámetros: *el_Elemento*, correspondiente al elemento a insertar; y *en_la_Cola*, correspondiente a la cola donde se debe insertar el elemento.

Para insertar el elemento, consideraremos que el apuntador *last* de la cola va a apuntar a la primera posición vacía, ubicada al final de la cola; y *first*, a la primera posición, vacía o no, de la misma cola.

En caso de que la cola esté vacía, insertamos el elemento en la primera posición, indicada por *first*, y asignamos *last* a la posición siguiente a *first* (que estará vacía), indicada por el apuntador *next.*

Si la cola no está vacía, insertamos el elemento en la posición indicada por *last,* y asignamos *last* a la siguiente posición.

El procedimiento resultante quedará así:

```
procedure Poner(el_Elemento: Elementos; en_la_Cola: in out Cola) is
begin
   if Esta_Vacia(en_la_Cola) then
     en_la_Cola.first := new register'(data => el_Elemento, next => en_la_Cola.last);
     en_la_Cola.last := en_la_Cola.first.next;
   else
      en_la_Cola.last := new register'(data => el_Elemento, next => null);
      en_la_Cola.last := en_la_Cola.last.next;
   end if;
end Poner;
```

- **Quitar:** Procedimiento para eliminar elementos de la cola, y devolver por parámetro el elemento eliminado. Los elementos se eliminarán de la primera posición de la cola.

Este procedimiento recibe dos parámetros: *un_Elemento*, que es la variable donde se debe devolver el elemento eliminado; y *de_la_Cola*, que es la cola de la que se debe quitar el elemento.

En primer lugar, antes de eliminar el elemento, accedemos a él mediante el apuntador *first*, y asignamos su contenido *(data)* a la variable *un_Elemento.*

Una vez asignado el dato, eliminamos el elemento reasignando el apuntador *first* a la siguiente posición indicada por *next.*
Para poder liberar la memoria una vez reasignado el apuntador, debemos guardar su actual contenido en una apuntador auxiliar, al que llamaremos *aux.*

Este apuntador auxiliar nos servirá para encontrar el siguiente elemento una vez eliminada la posición, y poder reasignar la posición al apuntador *first* de la cola.

Hecho esto, liberamos la memoria de la posición actual indicada por *first,* y reasignamos el apuntador a la posición almacenada en *aux,* correspondiente al siguiente elemento.

El procedimiento resultante quedará así:

```
procedure Quitar(un_Elemento: out Elementos;de_la_Cola:in out Cola) is
   aux : link;
begin
   if Esta_Llena(de_la_Cola) then
      un_Elemento := de_la_Cola.first.data;
      aux := de_la_Cola.first.next;
      Liberar_Register(de_la_Cola.first);
      de_la_Cola.first := aux;
   end if;

end Quitar;
```

- Esta_Vacia y **Esta_Llena:** funciones para indicar si la cola está vacía (no tiene elementos) o está llena (tiene elementos insertados).

Estas funciones reciben sendos parámetros llamados *La_Cola*, correspondientes a la cola a comprobar.

En éste caso, consideramos que la cola está vacía cuando sus dos apuntadores *first* y *last* apuntan a la misma posición, y que esta llena en caso contrario.

Las funciones quedarían así:

```
function Esta_Vacia (La_Cola: Cola) return Boolean is
begin
   return (La_Cola.first = La_Cola.last);
end Esta_Vacia;


function Esta_Llena (La_Cola: Cola) return Boolean is
begin
   return (La_Cola.first /= La_Cola.last);
end Esta_Llena;
```

- Copiar: El procedimiento copiar crea otra cola idéntica a la primera, con los mismos contenidos y en el mismo orden

Recibe dos parámetros: *Origen,* correspondiente a la cola original; y *Destino*, correspondiente a la cola donde se han de copiar los elementos.

Para copiar la cola creamos dos variables auxiliares: *aux,* un puntero a registro que se usará para recorrer la cola original; y *elem,* una variable que se usará para guardar el contenido de la posición correspondiente.

Antes de empezar, la función comprueba si la cola original está llena, caso en en cual empieza a operar.

Para empezar a copiar, igualamos los dos punteros *last* y *first* de la cola *Destino,* para tener posiciones válidas previas a la inserción.

Una vez hecho eso, usamos un bucle while que va recorriendo la cola posición a posición, guardando el contenido de cada una, e insertándolo en la nueva cola. Para realizar la inserción, usaremos el procedimiento *Poner* definido previamente.

El procedimiento, una vez implementado, quedaría así:

```
procedure Copiar (Origen: Cola; Destino: in out Cola) is
   aux: link;
   elem : Elementos;
begin

   if Esta_Llena(Origen) then
      Destino.first := Destino.last;

      aux := Origen.first;

      while aux /= Origen.last loop
         elem := aux.data;
         Poner(elem, Destino);
         aux := aux.next;
      end loop;

   end if;

end Copiar;
```

- **Sobrecarga de operador “=”:** Esta función indica si dos colas son iguales, devolviendo **true** en caso de serlo, y **false** en caso contrario.

La función recibe dos parámetros: *La_Cola* y *Con_La_Cola,* correspondientes a las dos colas a comparar; y devuelve un boolean (true/false)

Hace uso de dos variables auxiliares: *aux1* y *aux2*, que representaran los punteros a cada una de las colas.

Para saber si ambas colas son iguales, se definen 3 casos: ambas colas vacías, en el cual devolveremos verdadero; una cola vacía y una llena, en el cual devolveremos falso; y ambas colas llenas, en el cual habrá que hacer una comprobación posterior, recorriendo la cola posición a posición.

En este último caso, haremos uso de los punteros auxiliares *aux1* y *aux2,* que inicializaremos a las primeras posiciones de cada una de las colas,

Estos punteros se usarán para ir recorriendo las colas mediante un bucle *while*, el cual finalizará cuando se llegue a la última posición de una de las colas.

Durante este bucle, se compararán los contenidos las posiciones de cada cola. Si el contenido de alguna posición es distinto, devolveremos falso; en caso contrario, seguiremos recorriendo la cola.

Una vez terminamos el bucle, comprobamos si ambos punteros han llegado a la última posición de sus respectivas colas. En caso afirmativo devolvemos verdadero, en caso contrario devolvemos falso.

La función, una vez implementada, quedará así:

```
function "="(La_Cola, Con_La_Cola: Cola) return Boolean is
   aux1: link;
   aux2: link;

begin

   if Esta_Vacia(La_Cola) and Esta_Vacia(Con_La_Cola) then
      return true;

   elsif (Esta_Llena(La_Cola) and Esta_Vacia(Con_La_Cola))
     or   (Esta_Vacia(La_Cola) and Esta_Llena(Con_La_Cola)) then
      return false;

   elsif Esta_Llena(La_Cola) and Esta_Llena(Con_La_Cola) then
      aux1 := La_Cola.first;
      aux2 := Con_La_Cola.first;

      while aux1 /= null and aux2 /= null loop
         if aux1.data = aux2.data then
            aux1 := aux1.next;
            aux2 := aux2.next;
         else return false;
         end if;
      end loop;

      return (aux1 = La_Cola.last and aux2 = Con_La_Cola.last);

   end if;

return false;

end "=";
```

Código completo

- **Especificación:** https://github.com/AlmuHS/Practicas_STR/blob/master/Practica2/colas.ads

```
with Ada.Unchecked_Deallocation;

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_Vacia (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

   type register;

   type link is access register;
   type register is tagged
      record
         data: Elementos;
         next: link;
      end record;

   type Cola is tagged
      record
         first: link;
         last: link;
      end record;

   procedure Liberar_Register is new Ada.Unchecked_Deallocation
     (register, link);

end Colas;
```

- **Implementación:** https://github.com/AlmuHS/Practicas_STR/blob/master/Practica2/colas.adb

```
package body Colas is

   procedure Poner(el_Elemento: Elementos; en_la_Cola: in out Cola) is
   begin
      if Esta_Vacia(en_la_Cola) then
        en_la_Cola.first := new register'(data => el_Elemento, next => en_la_Cola.last);
        en_la_Cola.last := en_la_Cola.first.next;
      else
         en_la_Cola.last := new register'(data => el_Elemento, next => null);
         en_la_Cola.last := en_la_Cola.last.next;
      end if;
   end Poner;

   procedure Quitar(un_Elemento: out Elementos;de_la_Cola:in out Cola) is
      aux : link;
   begin
      if Esta_Llena(de_la_Cola) then
         un_Elemento := de_la_Cola.first.data;
         aux := de_la_Cola.first.next;
         Liberar_Register(de_la_Cola.first);
         de_la_Cola.first := aux;
      end if;

   end Quitar;

   function Esta_Vacia (La_Cola: Cola) return Boolean is
   begin
      return (La_Cola.first = La_Cola.last);
   end Esta_Vacia;


   function Esta_Llena (La_Cola: Cola) return Boolean is
   begin
      return (La_Cola.first /= La_Cola.last);
   end Esta_Llena;


    procedure Copiar (Origen: Cola; Destino: in out Cola) is
       aux: link;
       elem : Elementos;
    begin

       if Esta_Llena(Origen) then
          Destino.first := Destino.last;

          aux := Origen.first;

          while aux /= Origen.last loop
             elem := aux.data;
             Poner(elem, Destino);
             aux := aux.next;
          end loop;

       end if;

    end Copiar;
```

```
   function "="(La_Cola, Con_La_Cola: Cola) return Boolean is
      aux1: link;
      aux2: link;

   begin

      if Esta_Vacia(La_Cola) and Esta_Vacia(Con_La_Cola) then
         return true;

      elsif (Esta_Llena(La_Cola) and Esta_Vacia(Con_La_Cola))
        or   (Esta_Vacia(La_Cola) and Esta_Llena(Con_La_Cola)) then
         return false;

      elsif Esta_Llena(La_Cola) and Esta_Llena(Con_La_Cola) then
         aux1 := La_Cola.first;
         aux2 := Con_La_Cola.first;

         while aux1 /= null and aux2 /= null loop
            if aux1.data = aux2.data then
               aux1 := aux1.next;
               aux2 := aux2.next;
            else return false;
            end if;
         end loop;

         return (aux1 = La_Cola.last and aux2 = Con_La_Cola.last);

      end if;

   return false;

   lacola: Cola;

   begin

   lacola.first := lacola.last;


end Colas;
```