Práctica 3

Tipos abstractos de datos

Con esta práctica se pretende que el estudiante:

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

3.1 Tareas a realizar

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

```
1  generic
2      type elemento_t is private;
3      with function ToString(el_Elemento: elemento_t) return string;
4  package Colas is
5     type cola_t is limited private;
6
7     procedure Poner (el_Elemento: elemento_t; en_la_Cola: in out cola_t);
8     procedure Quitar (un_Elemento: out elemento_t; de_la_Cola: in out cola_t);
9
10    function Esta_Vacia (La_Cola: cola_t) return Boolean;
11    function Esta_Llena (La_Cola: cola_t) return Boolean;
12    procedure MostrarCola (La_Cola: cola_t);
13
14    procedure Copiar ( Origen: cola_t; Destino:in out cola_t);
15    function "="(La_Cola, Con_La_Cola: cola_t) return Boolean;
16 private
17    -- Definición del tipo "cola_t"
18    -- En esta ocasión se implementa una cola dinámica
19    type Nodo;
20    type ref_Nodo is access Nodo;
21    type Nodo is record
22       Datos: elemento_t;
23       ptr_Siguiente: ref_Nodo;
24    end record;
25    type cola_t is record
26       ptr_primero : ref_Nodo;
27       ptr_ultimo : ref_Nodo;
28    end record;
29 end Colas;
```

Fichero 3.1: Especificación del paquete `Colas` (`colas.ads`)

El tipo `Cola` se implementará 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.[1]

Algunos compiladores de Ada disponen de un recolector para la memoria basura, la cual es liberada automáticamente cuando no hay ninguna referencia (puntero) 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 (*Ada95 Reference Manual*) define un procedimiento genérico para liberar memoria en los instantes especificados por la aplicación. Este procedimiento se llama `Ada.Unchecked_Deallocation` . Aunque no se comprenda toda la información ofrecida sobre el procedimiento `Unchecked_Deallocation`, el estudiante debe poder extraer aquella información que le permita construir un procedimiento derivado, de nombre `Liberar`, que sirva para liberar memoria de acuerdo con las necesidades de la práctica.

```
with Ada.Unchecked_Deallocation;

type Nodo is record
    Value : Integer;
end record;

type ref_Nodo is access Nodo;

procedure Free_Node is
    -- Instantiate Unchecked_Deallocation for Nodo and ref_Nodo
    procedure Free is new Ada.Unchecked_Deallocation(Nodo, ref_Nodo);

    P : ref_Nodo := new Nodo'(Value => 42); -- Dynamically allocate memory
begin
    -- Deallocate memory
    Free(P);
    -- P is now null after deallocation
    if P = null then
        Ada.Text_IO.Put_Line("Memory deallocated successfully.");
    end if;
end Free_Node;
```

Fichero 3.2: Unchecked Deallocation in Ada

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, Integer'image);
   use Colas_de_Integer;
   Practica_no_Apta: exception;

   C1, C2, C3: cola_t;
   E: Integer;
begin
   for I in 1..10 loop
      Poner (I, C1);
   end loop;
```

[1] Se llama *memoria basura* a la memoria reservada dinámicamente, cuando deja de ser útil.

```
13    Put_Line("En C1 tenemos ");
14    MostrarCola(C1);
15    for I in 11..20 loop
16       Poner (I, C2);
17    end loop;
18    new_line;
19    Put_Line("En C2 tenemos ");
20    MostrarCola(C2);
21
22    new_line;
23    Put_Line("1.- Comprobando si C1 = C1 .... ");
24    if C1 /= C1 then raise Practica_no_Apta; end if; Put("OK!");
25    new_line;
26    Put_Line("2.- Comprobando si C1 /= C2 .... ");
27    if C1 = C2 then raise Practica_no_Apta; end if; Put("OK!");
28
29    Poner (1, C3); Copiar (C2, C3);
30    Put_Line("En C2 tenemos ");
31    MostrarCola(C2);
32    new_line;
33    Put_Line("En C3 tenemos ");
34    MostrarCola(C3);
35    new_line;
36    Put_Line("3.- Comprobando si C2 = C3 .... "); Put("OK!");
37    if C2 /= C3 then raise Practica_no_Apta; end if;
38
39    Put_Line("4.- Comprobando copiar .... ");
40    Poner (100, C3);
41    Poner (200, C2);
42    Put_Line("En C2 tenemos ");
43    MostrarCola(C2);
44    new_line;
45    Put_Line("En C3 tenemos ");
46    MostrarCola(C3);
47    new_line;
48    if C2 = C3 then raise Practica_no_Apta; end if; Put("OK!");
49
50    Quitar (E, C3);
51    Put_Line("En C2 tenemos ");
52    MostrarCola(C2);
53    Put_Line("En C3 tenemos ");
54    MostrarCola(C3);
55    Put_Line("5.- Comprobando si C2 = C3 .... ");
56    if C2 = C3 then raise Practica_no_Apta; end if; Put("OK!");
57
58
59    while not Esta_Vacia (C2) loop
60       Quitar (E, C2); Poner (E, C1);
61    end loop;
62    while not Esta_Vacia (C3) loop
63       Quitar (E, C3);
64    end loop;
65    for I in 1..20 loop
```

```
66       Poner (I, C2);
67    end loop;
68    Poner(200, C2);
69
70    Put_Line("6.- Comprobando quitar .... ");
71    if C1 /= C2 then raise Practica_no_Apta; end if; Put("OK!");
72
73    while not Esta_Vacia (C3) loop
74       Quitar (E, C3);
75    end loop;
76    while not Esta_Vacia (C2) loop
77       Quitar (E, C2);
78    end loop;
79     for I in 1..20 loop
80       Poner (I, C2);
81     end loop;
82     for I in 1..19 loop
83       Poner (I, C3);
84     end loop;
85
86    Put_Line("En C2 tenemos ");
87    MostrarCola(C2);
88    new_line;
89    Put_Line("En C3 tenemos ");
90    MostrarCola(C3);
91    new_line;
92    Put_Line("7.- Comprobando si C2 = C3 .... ");
93    if C2 = C3 then raise Practica_no_Apta; end if; Put("OK!");
94
95
96    Put_Line("8.- Comprobando liberar memoria .... ");
97    for I in 1..1e7 loop
98    begin
99       Poner (1, C1); Quitar (E, C1);
100   exception
101      when Storage_Error =>
102         Put_Line ("Practica no apta:");
103         Put_Line ("La función Quitar no libera memoria.");
104   end;
105   end loop;
106   Put_Line ("Practica apta.");
107exception
108   when Practica_no_Apta =>
109      Put_Line ("Practica no apta:");
110      Put_Line ("Alguna operación no esta bien implementada.");
111   when Storage_Error =>
112      Put_Line ("Practica no apta:");
113      Put_Line ("Posible recursión infinita.");
114end Principal;
```

Programa 3.3: Prueba del paquete `Colas` (`principal.adb`)

En la segunda parte de esta práctica vamos a modificar el paquete `Fracciones` de la práctica anterior añadiéndole una función nueva y vamos a crear una cola de fracciones con el paquete `Cola`

1. Añade a tu paquete `Fracciones` de la práctica anterior una función pública con la siguiente forma `function Imprimir (F: fraccion_t) return string;`

2. Crea un procedimiento principal en un nuevo fichero principal_mix.adb que, haciendo uso del paquete `Fracciones` y el paquete `Cola` cree un paquete para implementar colas de fracciones

3. En el código de principal_mix.adb crea una cola de fracciones y rellénala, utilizando dos bucles for anidados con los valores:

$$\left\{\frac{1}{1},\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{2}{1},\frac{2}{2},\frac{2}{3},\frac{2}{4},\frac{3}{1},\frac{3}{2},\frac{3}{3},\frac{3}{4},\frac{4}{1},\frac{4}{2},\frac{4}{3},\frac{4}{4},\right\}$$

4. Muestra por pantalla los valores de la cola de fracciones