Examen de Lenguajes de Alto Nivel

Febrero 2003

Cuestiones (4 cuestiones, 4 puntos en total; contestar brevemente de forma razonada)

1) Disponemos del siguiente programa:

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Coche is
   type Color is (Rojo, Verde, Azul);
   type Vehiculo is record
      El_Color : Color;
      Matricula : String(1..7);
   end record;
   V : Vehiculo;
begin
   -- Inicializar
   V.El_Color:=Rojo;
   V.Matricula:="BXX3456";
   -- Mostrar
   Put_Line("El coche matrícula "&V.Matricula&
            " es de color "&Color'Image(V.El_Color));
end Coche;
```

Se pide pasar el tipo `Vehiculo` (pero no la variable `V`) a un paquete compilado separadamente y hacer que sea privado. Para ello, se añadirán las siguientes operaciones:

- inicializar un `Vehiculo` pasándole como parámetros el color y la matrícula
- obtener el color de un vehículo
- obtener la matrícula de un vehículo

Se piden la especificación y cuerpo del paquete. Asimismo, se pide reescribir el programa para que use el nuevo paquete.

2) Se dispone del siguiente paquete:

```
with Elementos; use Elementos; -- definición del tipo Elemento
package Datos is
   procedure Inserta (E : Elemento);
   function Existe (E : Elemento) return Boolean;
end Datos;
```

Se pide convertir este paquete en genérico, usando como parámetros genéricos el tipo `Elemento`, y una función que compare si dos elementos son o no iguales. En otro módulo crear un tipo registro con dos campos: identificador (entero) y nombre (string de hasta 20 caracteres), crear una función para comparar dos de estos registros sólo por su identificador, e instanciar este paquete genérico para ese tipo y esa función.

3) Escribir una tarea Ada que pida por teclado una clave que es un número entero y la compare con una constante también entera. Si coincide, la tarea se termina. Si no, después de una espera se repite la petición, hasta un máximo de 16 veces. La espera será de un segundo la primera vez, el doble la vez siguiente, y así sucesivamente. Cada vez que ocurre un error de formato (`Data_Error`), se reintenta la petición sin esperar ni

incrementar el tiempo de la próxima espera. Si se llega a la iteración 16, se pone un mensaje en pantalla.

4) Escribir un procedimiento al que se le pasa como parámetro el nombre de un fichero de texto, cuenta el número de caracteres ';' que hay en ese fichero, y lo retorna en otro parámetro. Si el fichero no existe retorna cero en ese parámetro. (Nota: Este procedimiento se puede usar para saber cuántas instrucciones tiene un programa Ada, C, o Java, en los que cada instrucción acaba en ';').

Examen de Lenguajes de Alto Nivel

Febrero 2003

Problema (6 puntos)

Se desea implementar un sistema de control de una flota de camiones de transporte. Cada camión dispone de un sistema de localización por GPS, y de un sistema de telefonía móvil que comunica la posición del camión a la central de control de la flota ante una orden de ésta. El software de comunicación con los camiones está ya desarrollado mediante el siguiente paquete:

```
package Camiones is
   Max_Camiones : constant := 126;
   type ID_Camion is range 1..Max_Camiones;
   type Long_Float is digits 15;
   type Coordenadas is record
      Lat : Long_Float range -90.0..90.0;
      Long : Long_Float range -180.0..180.0;
   end record;
   type Estado_Camion is (Fuera_Servicio, En_Ruta, Parado);
   Sin_Conexion : exception;

   function Distancia (C1,C2 : Coordenadas) return Long_Float; -- Km
   function Posicion (C : Id_Camion) return Coordenadas;
   procedure Envia_Mensaje (C : Id_Camion; Mensaje : String);
   function Estado (C : Id_Camion) return Estado_Camion;

end Camiones;
```

Los valores del tipo `ID_Camion` sirven para identificar cada camión. El tipo `Coordenadas` contiene la latitud y longitud del camión. Las operaciones del paquete son:

- `Distancia`. Retorna la distancia en kilómetros entre los puntos `C1` y `C2`.
- `Posicion`. Retorna las coordenadas del camión cuyo identificador es `C`. Si no hay conexión eleva `Sin_Conexion`.
- `Envia_Mensaje`. Envía por telefonía móvil un mensaje al conductor del camión identificado por C; el mensaje se imprime en su hoja de ruta. Si no hay conexión eleva `Sin_Conexion`.
- `Estado`. Obtiene mediante un mensaje de telefonía móvil el estado actual del camión identificado por C, y lo retorna. Si no hay conexión eleva `Sin_Conexion`.

Por otro lado, se dispone del paquete `Ciudades` ya realizado que define dos tipos abstractos de datos: `Ciudad` para representar los datos de una ciudad, y `Ruta` para representar una ruta formada por una lista de ciudades por las que hay que pasar. La especificación es:

```
with Camiones;
package Ciudades is

   type Ciudad is private;
   type Ruta is private;

   No_Cabe, No_Existe : exception;
```

```
   procedure Crea_Ciudad
     (Nombre : String; Pos : Camiones.Coordenadas; C : out Ciudad);

   function Pos_Ciudad (C : Ciudad) return Camiones.Coordenadas;
   function Nombre_De (C : Ciudad) return String;

   procedure Haz_Nula (R : in out Ruta);
   procedure Inserta_Ciudad (C : Ciudad; R : in out Ruta);
   function Num_Ciudades (R : Ruta) return Natural;
   function Ciudad_Num (N : Positive; R : Ruta) return Ciudad;
private
   ...
end Ciudades;
```

Las operaciones de este paquete son:

- `Crea_Ciudad`: Inicializa el nombre y posición de la ciudad `C` según los valores indicados.
- `Pos_Ciudad`: retorna las coordenadas de la ciudad `C`.
- `Nombre_De`: retorna el nombre de la ciudad `C`.
- `Haz_Nula`: Hace nula la ruta `R`.
- `Inserta_Ciudad`: Inserta la ciudad `C` añadiéndola al final de la ruta `R`. Si la ruta no cabe, eleva `No_Cabe`.
- `Num_Ciudades`: Retorna el número de ciudades de la ruta `R`.
- `Ciudad_Num`: Retorna la ciudad almacenada en la posición `N` de la ruta `R`. Si el número `N` es incorrecto, eleva `No_Existe`.

Lo que se pide es implementar el cuerpo del siguiente paquete, que permite almacenar para cada camión una serie de datos, así como hacer operaciones con estos datos.

```
with Camiones, Ciudades;
package Flota is

   No_Cabe_Ruta : exception;

   procedure Actualiza_Posiciones;
   function Posicion_Camion
     (C : Camiones.Id_Camion) return Camiones.Coordenadas;
   function Velocidad
     (C : Camiones.Id_Camion) return Camiones.Long_Float;
   function Tiempo_Restante
     (C : Camiones.Id_Camion) return Duration;
   procedure Asigna_Ruta
     (C : Camiones.Id_Camion; R : Ciudades.Ruta; Enviar : Boolean);
   procedure Quita_Ruta (C : Camiones.Id_Camion);

end Flota;
```

El paquete opera como una máquina de estados abstracta con una variable que almacena para cada camión los siguientes datos:

- Posición actual
- Estado actual
- Velocidad media en kilómetros por hora
- Instante de la lectura de posición anterior
- Un conjunto de hasta 10 rutas para el camión
- El número de rutas almacenadas
- El número de fallos consecutivos de conexión

Las operaciones a implementar son las siguientes:

- `Actualiza_Posiciones`: Para cada camión lee su posición y su estado. Si no hay error, calcula la distancia (*dist*) entre la posición actual y la última leída; calcula la diferencia (*t*) entre la hora actual y el instante de la última lectura de la posición; y calcula la velocidad media según:

$$vmedia = vmedia_{anterior} \cdot 0{,}9 + \frac{dist}{t} \cdot 0{,}1$$

  Luego, también si no hay error, actualiza los valores almacenados de la posición actual, estado actual, instante de lectura de posición, según los datos utilizados. Finalmente, pone a cero el número de fallos consecutivos.

  Si se elevase `Sin_Conexion`, entonces incrementa el número de fallos consecutivos. Si éste llega a ser mayor que 10, cambia el estado a `Fuera_Servicio`. Si no, deja todo como estaba y no hace nada.
- `Posicion_Camion`: Retorna la posición almacenada del camión `C` (no la que indique el GPS).
- `Velocidad`: Retorna la velocidad almacenada del camión `C`. No se requiere ningún cálculo.
- `Tiempo_Restante`: Retorna el tiempo restante (en segundos) para alcanzar el último punto de la primera ruta almacenada para el camión C, calculando el cociente entre la distancia entre la posición almacenada y ese último punto, y la velocidad almacenada. Observar las unidades. Si no hubiera ninguna ruta, retorna 0.0.
- `Asigna_Ruta`: Añade al conjunto de rutas del camión `C` la ruta `R`. Si no caben más, eleva `No_Cabe_Ruta`. Si `Enviar` es `True`, envía un mensaje al camión por cada ciudad de la ruta `R`, con un texto que contiene el número de ciudad (1 para la primera ciudad de la ruta, 2 para la segunda, etc.) y su nombre. Si se elevase `Sin_Conexion` se abandona el envío de mensajes y se pone un mensaje de aviso en la pantalla.
- `Quita_Ruta`: Elimina la ruta primera de las almacenadas en el conjunto de rutas del camión `C`. Si no hubiese rutas, no hace nada.

Nota: Cada parte del paquete se valorará en función de su complejidad.