Examen de Lenguajes de Alto Nivel

Febrero 2002

Cuestiones (4 cuestiones, 4 puntos en total)

1) Se dispone de los paquetes Ada que se muestran abajo. Reorganizar los contenidos de estos paquetes en otros, siguiendo los principios del diseño orientado a objetos (sólo escribir las especificaciones). Pensar cuáles de los tipos pueden ser privados. Razonar brevemente sobre la solución propuesta.

```
package Datos is

   type Color is (Rojo, Verde, Azul);

   type Identificador is new String(1..4);

   type  Contenedor is tagged record
       C : Color;
       Id : Identificador;
   end record;

end Datos;

with Datos; use Datos;
package Operaciones is

    procedure Lee (Id : out Identificador);
    procedure Escribe (Id : in Identificador);

    procedure Lee (C : out Contenedor);
    procedure Escribe (C : in Contenedor);

end Operaciones;
```

2) El procedimiento `Opera` del paquete que se muestra abajo puede elevar las excepciones `Demasiado_Trabajo` o `No_Valido`. Se pide escribir un procedimiento compilado separadamente que llame a `Opera` en un lazo hasta que el parámetro `Salir` valga `True`. Si se eleva `Demasiado_Trabajo` se deberá mostrar un mensaje de error y permanecer en el lazo. Si se eleva `No_Valido` se debe salir del lazo, mostrar un mensaje de error, y volver a elevar la misma excepción.

```
package Artilugio is

   Demasiado_Trabajo, No_Valido : exception;

   procedure Opera (Salir : out Boolean);

end Artilugio;
```

3) Escribir una función que indique en su valor de retorno si un fichero de texto cuyo nombre se le pasa como parámetro existe o no.

4) Escribir una tarea periódica Ada que realice lo siguiente cada 10 milisegundos: llamar al procedimiento `P1`; si se eleva `Constraint_Error`, entonces llamar a `P2`.

Examen de Lenguajes de Alto Nivel

Febrero 2002

Problema (6 puntos)

Se desea implementar parte del sistema electrónico portable para ayuda a un excursionista. El sistema se basa en un receptor de GPS capaz de detectar la posición del sistema en el globo terráqueo. El usuario podrá almacenar varias rutas compuestas por secuencias de puntos. Posteriormente, podrá decidir recorrer una ruta, para lo cual el aparato GPS le irá diciendo la dirección a seguir, y la distancia que le queda por recorrer.

El software del receptor de GPS está ya implementado en un paquete cuya especificación es:

```
package Gps is
   type Doble_Precision is digits 15;
   type Punto is private;
   subtype Angulo is Doble_Precision range -360.0..360.0;
   -- en grados
   subtype Distancia is Doble_Precision; -- en metros
   function Crea_Punto (X,Y : Distancia) return Punto;
   function Coordenada_X (P : Punto) return Distancia;
   function Coordenada_Y (P : Punto) return Distancia;
   function Dist (P1,P2: Punto) return Distancia; -- en metros
   function Rumbo (P1,P2: Punto) return Angulo; -- en grados
   function Punto_Actual return Punto;
   Sin_Senal : exception; -- elevada por Punto_Actual
private
   ...
end Gps;
```

La descripción de las operaciones de este paquete es la siguiente:

- `Crea_Punto`: Permite crear un punto, del tipo privado `Punto`, dadas las coordenadas X e Y (en metros) de un punto en el plano, relativas a un origen de coordenadas fijo.
- `Coordenada_X`: Retorna la coordenada X del punto `P`.
- `Coordenada_Y`: Retorna la coordenada Y del punto `P`.
- `Dist`: Retorna la distancia entre los puntos `P1` y `P2`.
- `Rumbo`: Retorna el rumbo a seguir para llegar de `P1` a `P2`. Se retorna en grados, donde 0º es el Norte, y los grados se incrementan en el sentido de las agujas del reloj.
- `Punto_Actual`: Retorna el punto actual en el que se encuentra el sistema, medido por el sistema de GPS. Si no hay señal suficiente de los satélites, la operación eleva `Sin_Senal`.

Además, el software que controla la pantalla y teclado del aparato está también desarrollado, y se encuentra en el paquete:

```
with Gps;
package Pantalla is
   type Pulsacion is (Pulsado, No_Pulsado);
   type Boton is (Izquierda, Derecha, Salir, OK);
   procedure Pinta_Mensaje (Mens : String);
   procedure Pinta_Rumbo (Rumbo : Gps.Angulo;
                          Borrar : Boolean:=False);
   function Estado_Boton (B: Boton) return Pulsacion;
   -- otras operaciones
end Pantalla;
```

La descripción de las operaciones de este paquete es la siguiente:

- `Pinta_Mensaje`: Muestra el texto indicado por `Mens` en la pantalla. Los caracteres posteriores al 20 se ignoran.
- `Pinta_Rumbo`: Si `Borrar` es `False`, muestra el rumbo indicado por `Rumbo` en la pantalla, mediante un dibujo de una flecha. Si `Borrar` es `True`, borra ese dibujo.
- `Estado_Boton`: Devuelve el estado, `Pulsado` o `No_Pulsado`, del botón indicado por `B`. La pantalla tiene cuatro botones, llamados `Izquierda`, `Derecha`, `Salir`, y `OK`.

Lo que se pide en primer lugar es implementar el cuerpo del siguiente paquete, que sirve para almacenar las posiciones de los puntos de cada ruta:

```
with Gps;
package Rutas is

   Max_Rutas : constant Integer := 10;
   Max_Puntos : constant Integer :=100;
   subtype Id_Ruta is Integer range 1..Max_Rutas;
   subtype Num_Punto is Integer range 1..Max_Puntos;
   procedure Cambia_Ruta_Siguiente;
   procedure Cambia_Ruta_Anterior;
   function Ruta_Actual return Id_Ruta;
   procedure Borra_Ruta_Actual;
   procedure Inserta_Punto(P : Gps.Punto);
   function Num_Puntos return Natural;
   function Punto_Ruta (Num : Num_Punto) return Gps.Punto;

   No_Existe : exception; -- elevada por Punto_Ruta
   No_Cabe : exception; -- elevada por Inserta_Punto
end Rutas;
```

El cuerpo del paquete contendrá un array que permite almacenar 10 rutas, identificadas mediante los valores del subtipo Id_Ruta. Cada Ruta contendrá en un array hasta Max_Puntos datos del tipo Punto, así como el número de puntos actualmente almacenados. Este número será cero inicialmente. Además contendrá el identificador de la ruta actual, que inicialmente será la ruta número 1. Por último contendrá las implementaciones de las operaciones:

- `Cambia_Ruta_Siguiente`: Cambia la ruta actual a la siguiente. Se considerará que la siguiente a la última es la primera.
- `Cambia_Ruta_Anterior`: Cambia la ruta actual a la anterior. Se considerará que la anterior a la primera es la última.
- `Ruta_Actual`: Retorna el identificador de la ruta actual.
- `Borra_Ruta_Actual` : Pone el número de puntos de la ruta actual a cero.
- `Inserta_Punto`: Inserta el punto `P` al final de la lista de puntos de la ruta actual. Si no cabe, eleva `No_Cabe`.
- `Num_Puntos`: Retorna el número de puntos de la ruta actual.
- `Punto_Ruta`: Retorna el punto número `Num` de la ruta actual. Si `Num` es mayor que el número de puntos de esa ruta, eleva `No_Existe`.

En segundo lugar se pide implementar un procedimiento denominado `Seguir_Ruta`, que a su vez contiene otros dos procedimientos llamados `Seleccionar_Ruta` y `Recalcular_Destino` y una función, llamada `Distancia_Total`. `Seguir_Ruta` facilitará al excursionista el seguimiento de una ruta, punto por punto. Este procedimiento almacena en

una variable el número del punto destino actual de la ruta actual (inicialmente será el punto 1; lo llamaremos `Id_Destino`). Al comenzar, llama a una operación denominada `Seleccionar_Ruta` que hace lo siguiente:

- Entra en un lazo en el que la condición de salida es que el botón `OK` esté pulsado. Dentro de este lazo, si está pulsado el botón `Izquierda`, llama a `Cambia_Ruta_Anterior`; y si está pulsado el botón `Derecha`, llama a `Cambia_Ruta_Siguiente`.

Posteriormente comienza un lazo en el que la condición de permanencia es que el botón `Salir` no esté pulsado. Dentro de este lazo hace lo siguiente:

- Calcula el punto actual mediante la llamada `Gps.Punto_Actual`
- Recalcula el destino llamando a `Recalcular_Destino` (ver abajo)
- Pone en la pantalla un mensaje con la distancia total devuelta por `Distancia_Total` (ver abajo)
- Pinta en la pantalla el rumbo deseado, que se obtiene con la función `Gps.Rumbo` entre el punto actual y el punto `Id_Destino` de la ruta actual
- Si se eleva `Sin_Senal` pone un mensaje en la pantalla y permanece en el lazo
- Si se eleva `No_Existe` (al obtener el punto de una ruta) pone un mensaje en la pantalla y el procedimiento se termina

Las operaciones a las que hemos hecho referencia en este lazo hacen lo siguiente:

- `Distancia_Total`: Calcula la suma de la distancia del punto actual al punto `Id_Destino` de la ruta actual, de ese al siguiente punto de la ruta, y así hasta el último punto.
- `Recalcular_Destino`: Si el punto `Id_Destino` es el último de la ruta no hace nada. Si no, si la distancia del punto actual al punto `Id_Destino` es menor a 100 metros, incrementa el `Id_Destino` en una unidad. Si no es menor a 100 metros, pero la distancia del punto actual a uno de los puntos `N` de la ruta posteriores al punto `Id_Destino` es menor a la distancia al punto `ID_Destino` actual, se cambiará éste por el nuevo punto `N`.