Examen de Lenguajes de Alto Nivel

Febrero 2005

Cuestiones (4 cuestiones, 4 puntos en total)

1) Escribir el cuerpo del paquete cuya especificación se muestra debajo, de modo que el procedimiento `Abre` abra el fichero de texto representado por `Fichero`, con el nombre indicado en `Nombre`. Además, debe leer la primera línea del fichero y compararla con el string fijo `"#datos#"`. Si esa línea no coincide con ese string, o si el fichero no existe, o se acaba antes de poder leer la primera línea, se lanzará la excepción `Incorrecto`.

```
with Ada.Text_IO;
use Ada.Text_IO;

package Datos is

   Incorrecto : exception;

   procedure Abre(Nombre : String; Fichero : in out File_Type);

end Datos;
```

2) Se dispone del siguiente paquete genérico:

```
generic
   type Elemento is private;
package Agenda is

   subtype Fecha is String(1..10); -- formato: mm/dd/aaaa

   -- Anota el elemento en la agenda, para la fecha F
   -- puede lanzar No_Cabe
   procedure Anota(E : Elemento; F : Fecha);

   -- Ver función Siguiente
   procedure Rebobina(F : Fecha);

   -- Retorna el número de elementos anotados en la agenda
   -- para la fecha F
   function Num_Elementos(F : Fecha) return Natural;

   -- Después de Rebobina(F), retorna el primer elemento
   --   anotado en la agenda para la fecha F
   -- En otros casos retorna el siguiente elemento para esa
   --   fecha. Si no queda ninguno, lanza No_Hay_Mas
   function Siguiente (F : Fecha) return Elemento;

   Fecha_Incorrecta, No_Hay_Mas, No_Cabe : exception;
   -- Todas las operaciones pueden lanzar fecha incorrecta

end Agenda;
```

Instanciarlo para crear una agenda para guardar nombres de personas, del subtipo:

```
subtype Persona is String(1..20);
```

Escribir además un procedimiento que muestre en pantalla todas las personas anotadas en la agenda para una fecha `F` que se le pasa como parámetro. No es preciso tratar errores

3) Se dispone del tipo de dato `Persona` que guarda las personas de un árbol genealógico

```
package Arbol_Genealogico is

   subtype Nombre is String(1..20);
   type Persona;
   type Ref_Persona is access Persona;

   type Lista_Hijos is array(1..20) of Ref_Persona;

   type Persona is record
      Nom : Nombre:="                    ";
      Padre, Madre, Conyuge  : Ref_Persona;
      Num_Hijos : Natural:=0;
      Hijo : Lista_Hijos;
   end record;

   ...

   Operacion_Incorrecta : exception;
end Arbol_Genealogico;
```

Crear un procedimiento compilado separadamente al que se le pasa un puntero a una persona y que le divorcia de su cónyuge, haciendo que los punteros `Conyuge` de ambos sean `null`. Si no estuviera casado (es decir si el `Conyuge` ya es `null`), lanza la excepción `Operacion_Incorrecta`.

4) Un sistema utiliza mensajes SMS para enviar mensajes de alarma. Dispone para ello de un paquete con la siguiente especificación

```
package Alarmas is

   type Tipo_Alarma is (Humo, Temperatura, Presencia);

   type Alarma is tagged private;

   -- Almacena Lugar en el objeto
   procedure Pon_Lugar(A : in out Alarma; Lugar : String);

   -- Almacena el tipo de alarma en el objeto
   procedure Pon_Tipo(A : in out Alarma; Tipo : Tipo_Alarma);

   -- Envía un SMS con texto de alarma (obtenido con A_Texto)
   procedure Envia(A : in Alarma);

   -- Retorna un texto con los atributos de la alarma
   function A_Texto (A : Alarma) return String;

private
   ...
end Alarmas;
```

Se desea extender en un nuevo paquete el tipo `Alarma`, para añadir un nuevo atributo del tipo `Ada.Calendar.Time` para almacenar la fecha y hora de generación de la alarma. Para el nuevo tipo se redefinirán `Pon_Lugar` y `A_Texto`.

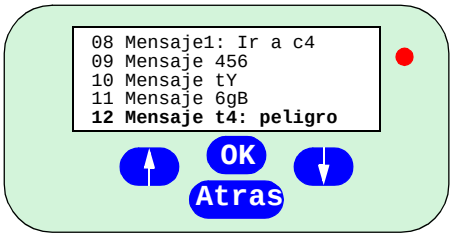
Se pide la especificación del nuevo paquete, así como el cuerpo de la operación `Pon_Lugar`, que debe invocar al `Pon_Lugar` del tipo padre y luego anotar la fecha y hora actual en el nuevo atributo (*Nota*: no se pide el cuerpo de `A_Texto`).

Examen de Lenguajes de Alto Nivel

Febrero 2005

Problema (6 puntos)

Se desea desarrollar un programa para la gestión de un terminal portátil para la recepción de mensajes enviados por radio desde un servidor remoto. Como se muestra en la figura, el terminal dispone de una pantalla de 5 líneas, una luz de dos colores, y cuatro botones. El software de acceso al hardware está ya realizado en el siguiente paquete:




```
package Terminal is

   Max : constant Integer:=50;

   type Tipo_Boton is (Sube, Baja, OK, Atras);
   type Tipo_Luz is (Bien, Mal);
   subtype Num_Linea is Integer range 1..5;
   subtype Num_Mensaje is Integer range 1..Max;
   subtype Contenido is String(1..40);

   -- Retorna True si el boton ha pasado de estar no pulsado a
   -- pulsado desde la última vez que se llamo a la función
   function Se_Ha_Pulsado (Boton : Tipo_Boton) return Boolean;

   -- Pone el color de la luz de estado del terminal
   procedure Pon_Luz(Valor : Tipo_Luz);

   -- Borra la pantalla
   procedure Borra_Pantalla;

   -- Pone la línea L con el número de mensaje y mensaje indicados
   -- si Resaltar es True, el mensaje se escribe resaltado
   procedure Pon_Linea
     (L : Num_Linea;
      N: Num_Mensaje;
      Mensaje : Contenido;
      Resaltar : Boolean:=False);

   -- Si se ha recibido un mensaje por radio pone Hay_Mensaje a
   -- true, y lo retorna en Mensaje.
   -- si no, pone Hay_Mensaje a false
   -- puede lanzar Sin_Conexion si no detecta portadora
   procedure Recibe_Mensaje
     (Hay_Mensaje : out Boolean;
      Mensaje : out Contenido);

   Sin_Conexion : exception;

end Terminal;
```

Se dispone además de un tipo de datos abstracto ya implementado que permite almacenar datos en una cola. Es similar a las colas vistas en clase, pero está ampliado con las siguientes funciones:

- `Esta_Lleno`: retorna `true` si la cola está llena, y `false` en caso contrario

- `Num_Elementos`: retorna el número de elementos actualmente guardados en la cola
- `Elemento_De`: permite obtener un elemento cualquiera de la cola, dada su posición numérica. El uno es el más antiguo.
- `Elimina`: permite eliminar un elemento cualquiera de la cola, dada su posición.

La especificación del nuevo paquete `Colas` se muestra a continuación:

```
with Excepciones_Colas;
generic
   type Elemento is private;
   Max : Integer := 100;
package Colas is
   type Cola is private;

   No_Hay  : exception renames Excepciones_Colas.No_Hay;
   No_Cabe : exception renames Excepciones_Colas.No_Cabe;

   procedure Haz_Nula (La_Cola : in out Cola);

   procedure Inserta
     (El_Elemento : in  Elemento; La_Cola : in out Cola);
   -- puede elevar No_Cabe

   procedure Extrae
     (El_Elemento : out Elemento; La_Cola : in out Cola);
   -- puede elevar No_Hay

   procedure Elimina
     (Posicion : in Positive; La_Cola  : in out Cola);
   -- puede elevar No_Hay

   function Esta_Vacia (La_Cola : in Cola) return Boolean;

   function Esta_Llena (La_Cola : in Cola) return Boolean;

   function Num_Elementos (La_Cola : in Cola) return Natural;

   function Elemento_De
     (Posicion : Positive; La_Cola : in Cola) return Elemento;
   -- puede elevar no_hay

private
   ...
end Colas;
```

Lo que se pide es hacer el programa principal, que gestiona el terminal de manera que represente los mensajes en la pantalla del terminal, e interprete sus botones.

El programa instancia el paquete `Colas` para elementos del tipo `Terminal.Contenido`, en un número igual a `Terminal.Max`. Además, declara al menos las siguientes variables:

- `Estado`: representa el estado del terminal, que puede ser `Normal` o `Borrando` mensaje (inicialmente, `Normal`)
- `Cola`: es la variable del tipo `Cola` donde se almacenan los mensajes
- `Seleccionado`: indica cuál es el mensaje actualmente seleccionado (inicialmente, ninguno)

El programa hace la cola nula, y luego entra en un lazo infinito repitiendo constantemente lo

siguiente, en este orden:

- Procesar mensajes: Llama a recibe mensaje y si hay un mensaje lo inserta en la cola. Si la cola estuviese llena, antes de insertar hay que eliminar el mensaje más viejo. Si se eleva `Sin_Conexion`, hay que encender la luz al valor `Mal` (con `Pon_Luz`), y en caso contrario al valor `Bien`.
- Si el estado es `Normal`: Si se ha pulsado `Sube` y `Seleccionado` es mayor que 1, decrementar `Seleccionado` en una unidad. Similarmente, si se ha pulsado `Baja` y `Seleccionado` es menor que el número de mensajes almacenados en la cola, incrementar `Seleccionado` en una unidad. Si se ha pulsado `OK` y `Seleccionado` no es cero, pasar al estado `Borrando`. En todos los casos, pintar hasta cinco mensajes consecutivamente en las cinco líneas de la pantalla. El primer mensaje a pintar es el que ocupa el lugar igual al máximo de `1` y `Seleccionado-4`. El último es el `Seleccionado`, que se pintará resaltado (salvo que no haya ninguno seleccionado). Al pintar cada mensaje, se le pone su número de orden en la cola, pues lo requiere el procedimiento `Pon_Linea`.
- Si el estado es `Borrando`: Se borra la pantalla. Luego se pinta en la línea 3 el mensaje almacenado en la posición `Seleccionado`. En la línea 5 se pone el texto `"Borrar?"`. Luego:
  - Si se pulsa `OK` el mensaje `Seleccionado` se elimina de la cola, y si `Seleccionado` no es 1 se decrementa; además, si no hay mensajes en la cola `Seleccionado` se pone a cero; por último se cambia el estado a `Normal`.
  - Si se pulsa `Atras`, entonces cambiar el estado a `Normal`, sin borrar el mensaje.
  - En otros casos no se hace nada (sigue el estado `Borrando`)