Sistemas en Tiempo Real – Práctica ADA 1

Pila acotada

Se trata de terminar la implementación del paquete de pila acotada de números enteros presentado en teoría. Para ello haremos:

- Crear el paquete *pila_acotada* que implemente el objeto protegido *PilaAcotada*.
- Definir dos tareas tipo, una para los productores y otra para los consumidores.
- Crear 5 tareas productoras y 5 tareas consumidoras, cada una de las cuales insertará/leerá 50 elementos de la pila.

El objeto protegido *PilaAcotada* deberá tener un parámetro de inicialización para indicar el tamaño de la pila (por defecto valdrá 10) y constará de los siguientes elementos:

- Elementos privados:
  - Pila de enteros → Vector de enteros que irá desde 1 hasta el tamaño de la pila indicado como parámetro de inicialización del objeto.
  - Cima de la pila → Valor natural (de 0 a $+\infty$) que indicará tanto el número de elementos en la pila como la posición del último elemento insertado (si es >0). Por tanto, para insertar un elemento primero se incrementa la cima y luego se añade y para sacar un elemento, se accede a la posición de la cima y luego se decrementa.
  - Tamaño de la pila → Valor positivo (de 1 a $+\infty$) que indica el número máximo de elementos que puede tener la pila
- Elementos públicos:
  - Entrada PilaAniadir → Tendrá un parámetro de entrada de tipo entero y se encargará de añadir dicho parámetro a la cima de la pila. Esta entrada solo se ejecutará cuando la pila no esté llena.
  - Entrada PilaSacar → Tendrá un parámetro de salida de tipo entero y se encargará guardar en dicho parámetro el valor contenido en la cima de la pila. Esta entrada solo se ejecutará cuando la pila no esté vacía.

Se deberá crear un fichero .ads con la definición del paquete *pila_acotada* (que incluye la clase *PilaAcotada*), un .adb con el código de dicho paquete y otro .adb con el programa principal (descargar los esqueletos de la práctica del Campus Virtual). El programa principal deberá crear 5 productores y 5 consumidores, cada uno de los cuales añadirá/extraerá 50 valores enteros a la/de la pila, mostrando dichos valores. Para los productores/consumidores se deberán definir dos tipos tarea con un parámetro de tipo puntero a *PilaAcotada* que permitirá compartir los datos de la pila entre todas las tareas.

La práctica se comprimirá en un único archivo que será entregado a través del campus virtual usando la tarea creada para tal efecto.

ANEXO – Ayuda para ADA

Vectores

- Definición de una variable de tipo vector: *NombreVariableVector: array (Ini .. Fin) of TipoDatosVector*.
- Definición de un tipo vector: *type NombreTipoVector is array (Ini .. Fin) of TipoDatosVector*. A partir de este momento, se pueden definir variables de tipo *NombreTipoVector*.
- *Ini .. Fin* es el rango de valores de los índices del vector. En ADA, los vectores no van de 0 a N-1, van de *Ini* a *Fin*.
- Acceso al dato i-ésimo de un vector: *NombreVariableVector(i)*.
- Definición de vectores multidimensionales (igual para variables o tipos): Se añaden tantos rangos de valores como dimensiones separados por comas: *(Ini1 .. Fin1, Ini2 .. Fin2, … , IniN .. FinN)*. Para acceder al dato, se ponen los valores de las posiciones separados por comas: *NombreVariableVector(i, j, …, k)*.

Creación de paquetes

Se crearán dos ficheros:

- nombre_paquete.ads: definición del paquete.
- nombre_paquete.adb: implementación del paquete.

Definición de un tipo tarea y creación de una tarea usando ese tipo

- Definición del tipo: *task type NombreTipoTarea[(parámetros de la tarea)]*.
- Definición del cuerpo del tipo tarea:

```
task body NombreTipoTarea is
        [Zona de declaración]
begin
        [código. Al menos debe haber una línea de código. Se puede usar null si no se va a añadir nada]
end NombreTipoTarea;
```

- Creación de una tarea basada en el tipo anterior: *NombreTarea: NombreTipoTarea[(parámetros)]*. La tarea se lanzará al comenzar la zona de código (*begin .. end)* correspondiente a la zona de declaración donde se creó la tarea. Este punto se puede realizar una vez que se defina el tipo tarea, no es necesario esperar a que esté hecho el cuerpo.

Paso de parámetros a tareas

Para poder pasar parámetros a las tareas, deben ser "punteros", que en ADA se traduce en usar la palabra reservada *access* delante del tipo (sería similar al * en C++).

Sin embargo, como lo que queremos es pasar la dirección de una variable estática (lo que en C++ sería el &), usaremos *access all* en lugar de sólo *access* en la definición de los parámetros de la tarea. Se recomienda definir un nuevo tipo puntero.

Además, cuando definamos la variable que queremos pasar por declaración, usaremos la palabra *aliased* delante del nombre del tipo de la variable (esto permite acceder a la dirección de la misma) y, posteriormente usaremos el atributo *'Access* para acceder a la dirección de la variable.

Resumiendo:

```
Variable: aliased TipoNormal;                  -- Variable de tipo no puntero
TipoPuntero is access all TipoNormal;                  -- Nuevo tipo puntero
task type TareaTipo(Parametro: TipoPuntero); -- Tipo tarea con un parámetro
                                                        -- de tipo puntero
task body TareaTipo is begin … end;                  -- Cuerpo de la tarea
Tarea: TareaTipo(Variable'Access);      -- Creación de la tarea pasando por
                                    -- parámetro la dirección de la variable
```

Algunas funciones, atributos y operadores de utilidad

- Put (paquete GNAT.IO): Muestra por pantalla una cadena de caracteres o un entero.
- New_Line (paquete GNAT.IO): Imprime un salto de línea.
- Put_Line (paquete GNAT.IO): Imprime por pantalla una cadena de caracteres y la termina con un salto de línea. No muestra números (ver cómo convertir enteros a cadena a continuación).
- integer'Image(Dato): Devuelve el valor del entero *Dato* en forma de cadena de caracteres (añade un espacio al principio de la cadena devuelta).
- integer'Value(Dato): Devuelve el valor de la cadena de caracteres *Dato* en forma de entero.
- &: Operador de concatenación.